brigante's e-book

Въведение в програмен език C++

наръчник за младия пишман програмист:)

от brigante
mailto:to6ev@yahoo.com
http://e-down.hit.bg
30май2008
Ще разгледам:

I.

1. **Въведение в C++**
2. **Съхраняване на данни**
3. **Извършване на операции**
4. **Създаване на конструкции**
5. **Низове**
6. **Четене и запис на файлове**
7. **Използване на функции**
8. **Създаване на класове и обекти**
9. **Указатели към данни**
10. **Наследяване**
11. **Полиморфизъм**
12. **Макроси**

II.

1. **Програмиране с Microsoft Visual C++ версия 6.0**
2. **Visual C++ 2008 Express Edition (v 9.0) (code name Orcas) – орките:) т'ва е остров:)**

1. Въведение:

Езикът за програмиране C++ (си плюс плюс се произнася,а плюсчетата са за това,че в програмирането оператора за нарастване е ++ :))) и оттам е името на езика, който е разширение на езика C , който пък за пръв път е имплементиран в операционната система UNIX през 1972г. от Денис Ричи.
Та за създатели на езика C се считат Д.Ричи и Кен Томпсън, които са работели в AT&T Bell Labs в Ню Джързи. По-късно там започва работа и добрият стар Бьорн Страуструп и той взел че се прехласнал мнооо по езика C и решил да го разшири:) Абе изобщо в такива лаборатории се развиват нещата:)
То в къщи е трудна работа:) Само Линус Торвалдс ги може теа неща ха-ха:) ама той пък също е взаимствал от ядрото на мини Unix системката Minix :) едно време програмисчетата яко са си разменяли сорсчетата и така едни от други ВЗАИМСТВАТ (така се учи програмирането!)
Операционни с-ми се пишат на езици C,C++,Асемблер, а за мазохистите е Паскал:))))))))))))
Minix3 е писана от Андрю Таненбаум за учебна цел кат' Паскалчето хахаха:) ОК спирам да се шегувам!

C++ е разработен от Бьорнчо м/у 1983 и 1985г. Той добавя възможности към езика C и създава така наречения „ C с класове“. Тези класове дефинират обекти със специални възможности!
C++ е платформено независим и програмките могат да се създават на всякаква операционна с-ма!
Тук ще показвам прости примерчета,които са за WINDOWS , но могат да бъдат създадени и за Линукс.
Сигурно си задавате въпроса „'що да го уча този език, баси:)“

- **за да можете безпроблемно да боравите с каквито и да е програми като цяло, за да си обогатите малко информационно технологичната култура:)**
- **за да не стоите като индиянец с пушка пред компа:)**
- **за да се научите след време как да си създадете своя операционна с-ма:) свой компилатор :)**
- **за да се фукате пред другарчетата си,че изучавате най-мощния програмен език**
- **за да можете да краквате софт**
- **за да можете да пишете мощни програми било то за уин или юникс базираните с-ми**
- **за да можете да се справяте безпроблемно с линукс ос и други UNIX like OS**
- **абе 'щот така:)))))))**

Най-добре е да се ползват програмните среди на Майкрософт и Борланд...
Използвайки съвременните интегрирани среди за разработка(IDE= Integrated Development Environment) като Visual C++ на Майкрософт и Borland C++ Builder ,програмистът ще може много по-бързо да създава своите сложни приложения.

Но първо трябва да се научи същността на езика C++ и след това да започнете с визуалното програмиране,което си има своите специфики и работа със самата програмна среда(IDE)...
Първо трябва да се научите да изпълнявате програмките си в конзолен прозорец(cmd prompt) за уиндоус,а в линукс се казва терминален прозорец при X Window (графичен интерфейс за юникс=UNIX,UNIX-like OS)

Тук просто ще Ви въведа в програмирането на C++ и от части с Microsoft Visual C++

Както знаете езика C++ вече е стандартизиран...това е много важно:)
Ще Ви подхвърля няколко компилатора и програмни среди:

C++ Компилатори:
Програмите на C++ първо се пишат като текстови файлове (Notepad за Windows) (Vi или Emacs и др. за UNIX-like с-ми), но после се записват с файлово разширение .cpp (за C++) или .c (за C)
За да може да бъде изпълнена програмката тя първо трябва да бъде компилирана в байт код,за да я разбере компа. Компилаторът прочита текстовата версия и я транслира във втори файл-машинно четивен. Ако има синтактични грешки компилатора ще ги отчете и няма да бъде създаден втори файл.
Визуалните интегрирани среди за разработка са добри за създаване на сложни програми,но създават множество работни файлове и отначало може да Ви е проблемно,но с течение на времето ще свикнете да разпознавате кой файл за какво е:)

http://www.microsoft.com/express
това е визуалната програмна среда,която е безплатна ... изисква се да имате регистрация...става бързо сайта е обновен и се изисква да имате инсталиран Silverlight,за да го разгледате
Ако ще се занимавате с продуктите на Майкрософт, то трябва най-редовно да посещавате сайта им!
Знам,че не Ви е приятно:) Задължително е да си даунлоудните и всички .NET Framework

http://www.codegear.com/products/cppbuilder
http://www.codegear.com/downloads/free/cppbuilder
това е на Борланд компилатора и развойната среда,която е платена,но има и билдер 6 за c++ , който не е!

http://www.bloodshed.net/devcpp.html
още една добра среда

http://www.eclipse.org/downloads/

Eclipse IDE for C/C++ Developers (68 MB)

Популярният GNU(guh-new)(gnu not unix) C++ compiler е достъпен безплатно и си е включен в операционната с-ма Линукс,но може и да не е включен:)
За да проверите дали имате GNU C++ compiler на машината Ви просто въведете c++ -v в команден промпт(за windows os от start->run напишете cmd и после бутон ОК) ;Terminal е за юникс и подобие:)))
Ако сте го инсталирали правилно компилатора ще отговори като покаже версията си...

Хубаво е да си добавите компилера към системната променлива Path(т'ва е един вид пътя до указан файл в с-та Ви директория:) В Windows отворете прозорец Environment Variables като кликнете в/у икона System в Control Panel и изберете Advanced
Намирате променлива Path и редактирате да включва адреса до компилера:)

Ако си инсталирате среда от Майкрософт , то автоматично си конфигурира пътя...

От компилатор до компилатор има разлика и всеки се извиква специфично.
Всеки компилатор на C++ трябва да може да разпознава код на C и си го разпознава де:)

GCC,the GNU Compiler се взема от http://gcc.gnu.org
http://www.codeblocks.org/
още една много добра оупън сорс средичка:) линукс

А за Linux програмните среди са:

http://www.kdevelop.org
KDevelop е официалната среда за разработка на KDE...Подобно на комерсиалните продукти тук имате достъп до много Wizzard-и, инструменти за визуално генериране на приложения, вграден дебъгер, парсер на класове, който ви създава дърво на методите и ви позволява да избирате метода от там, вместо да го търсите из целият файл

http://wxstudio.sourceforge.net
http://www.jedit.org е Java текстов редактор и могат да се редактират над 60 вида програмни езици и скриптове

http://www.netbeans.org едно напълно функционално и мощно IDE особенно подходящо за Java програмисти

http://www.eclipse.org написано на Java, Eclipse се базира не на AWT или SWING, а на SWT библиотеката...можете да ползвате Eclipse както за разработка на C/C++, Java, така и WEB базирани проекти

http://anjuta.sourceforge.net

Anjuta се базира на Scintilla. Идеята на авторите е да създадат една добра среда за писане на C/C++, която да се интегрира безпроблемно в GNOME

http://quanta.sourceforge.net

за сериозните WEB developer-и

Стандартната библиотека на C++

(Standard Template Library of C++)

Това са всички хедър файлове,включени в c++ компилатора и предназначението им:

<algorithm>	**дефинира някои полезни алгоритми**
<bitset>	**администрира множества от битове**
<complex>	**извършва сложни аритметични изчисления**
<deque>	**реализира контейнер-опашка с два края (deque)**
<exception>	**управлява обработката на изключения**
<fstream>	**обработва външни потоци**
<functional>	**конструира обекти-функции**
<iomanip>	**iostream манипулатори, които приемат аргумент**
<ios>	**базиран на шаблон клас за iostream класове**
<iosfwd>	**позволява недефинирани iostream шаблонни класове**
<iostream>	**iostream обекти, обработващи стандартни потоци**
<istream>	**извършва извличане на стойности**
<iterator>	**шаблони, които улесняват дефинирането и обработката на итератори**
<limits>	**тества свойствата на числови типове**
<list>	**реализира контейнер-списък**
<locale>	**управлява поведението на базата на локалното местоположение**
<map>	**реализира асоциативни контейнери**
<memory>	**заделя и освобождава памет за съхранение на различни контейнери**
<new>	**функции, които заделят и освобождават място за съхранение**
<numeric>	**предоставя полезни числови функции**
<ostream>	**извършва вмъквания**
<queue>	**реализира контейнер-опашка**
<set>	**реализира контейнер с уникални елементи**
<sstream>	**обработва низови контейнери**
<stack>	**реализира контейнер стек**

<stdexcept>	**докладва за възникването на изключения**
<streambuf>	**предоставя буфер за iostream операции**
<string>	**реализира контейнер-низ**
<typeinfo>	**предоставя резултата от оператора typeid**
<utility>	**предоставя основни стандартни операции**
<valarray>	**поддържа ориентирани към стойности масиви**
<vector>	**реализира контейнер-вектор**
<cassert>	**извършва assert проверки при изпълнението на функции**
<cctype>	**класифицира знакове**
<cerrno>	**тества кодовете на грешките, отчетени от функциите на библиотеката**
<cfloat>	**тества свойствата на типове с плаваща запетая**
<ciso646>	**поддържа ISO 646 варианта на кодови таблици**

<climits>	**тества свойствата на целочислени типове**
<clocale>	**поддържа различни конвенции за културата**
<cmath>	**предоставя често използвани математически функции**
<csetjmp>	**изпълнява нелокални конструкции goto**
<csignal>	**управлява различните условия за изключения**
<cstdarg>	**осъществява достъп до вариращ брой аргументи**
<cstddef>	**дефинира няколко полезни типа и макроси**
<cstdio>	**предоставя входни и изходни операции**
<cstdllb>	**предоставя различни оснвни операции**
<cstrlng>	**обработва няколко вида низове**
<ctlmo> **<cwchar>** **<cwotypt>**	**предоставя различни формати за часа и датата** **обработва широки потоци** **класифицира широки знакове**

В стандартния C++ , цялата информация отнасяща се до стандартната библиотека е дефинирана в именованото пространство *std* . Така , за получаване на директен достъп до тези имена , вие ще трябва да включите следващия *using* израз след включването на необходимите хедъри

using namespace std ;

Нека започнем с програмирането...

Една програма може да съдържа една или много функции,но винаги трябва да има функция с името main
Тази функция main е началната точка на всички програми на C++ и компилатора няма да компилира кода иначе.
На другите функции в програмата може да се зададат к'вито имена желаете ,но трябва да съдържат букви, цифри и подчертаващо тире. Не може да започва с цифра!!!
В C++ има ключови думи и трябва да избягвате да ги слагате като имена на функциите.

Нека разгледаме първата програма,която по традиция се пише от начинаещите програмисчета:)

```
#include <iostream>                    // може да е < iostream. h> но вече хедъра не се изписва:) това е коментар
using  namespace std;

int main()
{
  cout « "tova e moiata parva programa\n“;                    // това ни е конструкцийката и винаги завършва с ;
return 0;
}
```

сега да разясня:

след тези две налконени черти // всичко е коментар към кода и се игнорира от компилатора
```
/*
```
Това е другия начин за коментар ... ако е на няколко реда,но рядко се ползва и е стар идва от C езика
```
*/
```

значи празните редове се игнорират от комилатора т.е. 3-тия ред, а можеше и да създадем 4-ти празен ред:)принципно си се слагат за четливост на кода и коментари
Кодът на програмката ни започва с инструкцията за включване(include) инфо от входно-изходната библиотека iostream
Но това е заглавен хедър всъщност <iostream.h>
.h идва от header file заглавен файл...
Редът започва винаги със знак #(диез както при телефоните:) Това е предпроцесорната инструкция.
Името на библиотеката трябва да е в тези скобки < >
Трябва да се знае към кой заглавен файл да се обърнем! Това ще разберете като научите документацията на стандартната библиотека:) По-надолу ще я обясня...
Никога не включвайте един и същ заглавен файл(.h) повече от един път!
След като се запознаете със стандартната библиотека на C++ ,ще можете да предвиждате кои заглавни файлове ще са нужни:)

Ето едни от срещаните заглавни файлове за включване в C++ ...разбира се това е една малка част:)

заглавен файл	директивата #include	съдържа декларации за тези функции
stdio.h	#include <stdio.h>	стандартни входни и изходни функции и ф-ции,извършващи файлови операции
iostream.h	#include <iostream.h>	стандартни оператори за работа с потоци (за C++) използват се вместо printf и scanf (в C)
string.h	#include <string.h>	string=низ Функции за манипулиране на низове.Например:копиране на низ в друг
ctype.h	#include <ctype.h>	Функции за тестване и промяна типа на отделни символи в низ "tova e moiata parva programa\n“

на това се казва низ:) а наклонена черта и буквата n е за преминаване към нов ред

math.h	**#include <math.h>**	**Тригонометрични ф-ции,логаритми когато ще се пресмятат**
malloc.h	**#include <malloc.h>**	**Функции за динамично заемане и освобождаване на блокове в паметта от операционната система**

Вторият ред от кода прави всички функции от библиотеката iostream достъпни посредством стандартните им имена,които са в пространството от имена std
Една от тях е функцията cout (си аут) , която се използва за извеждане на резултата от програмата!
cout е съкращение от „character-oriented output stream“=“символно-ориентиран изходен поток“
В декларацията на функцията типът данни (типовете данни съм разгледал по-надолце:)
е зададен като int (integer) , което означава
цяло число.
Това означава,че след като изпълни конструкциите, (конструкциите тва е всичко между скобките { }
тази функция трябва да върне целочислена стойност тези дето все едно се цункаме в чата:)
на операционната с-ма. ВСЯКА КОНСТРУКЦИЯ ТРЯБВА ДА ЗАВЪРШВА С ТОЧКА И ЗАПЕТАЯ ;

Нашата програмка в момента съдържа само една функция!
main
Тя е задължителна! Обикновените скобки ( ) след main са празни понеже не са зададени аргументи...

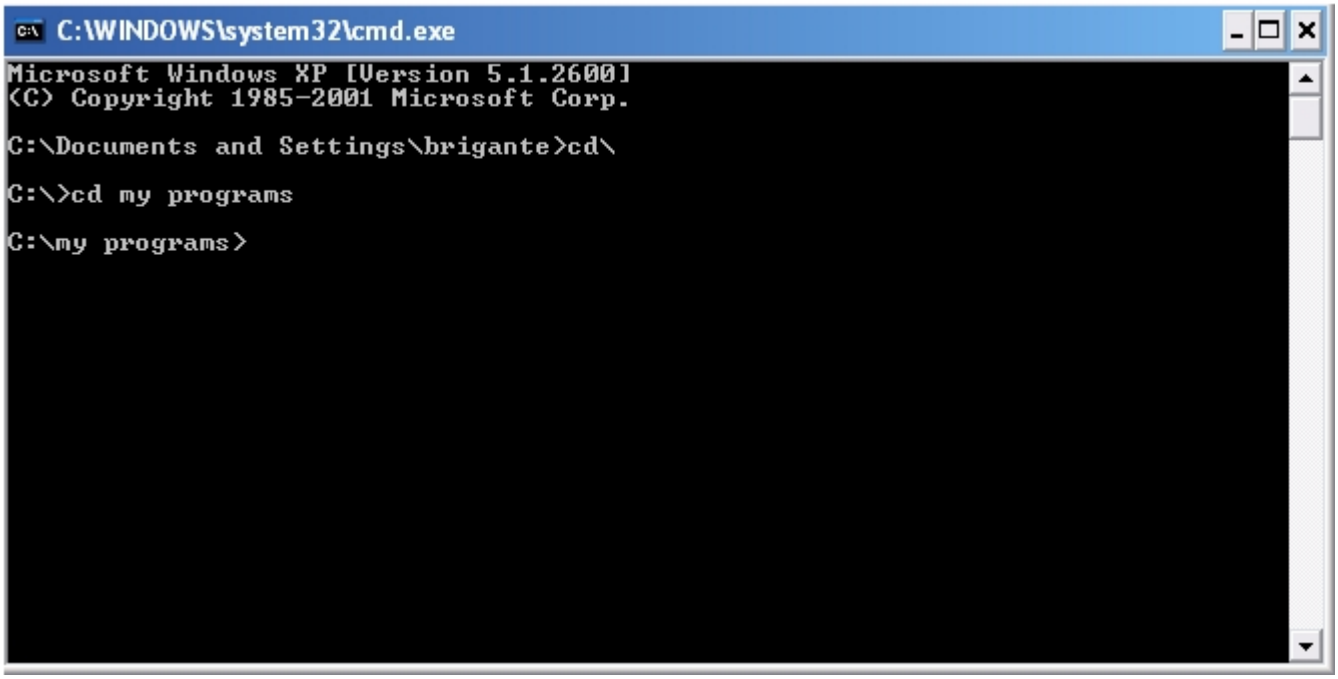
Сега да разгледаме фигурните скобки { }
Те съдържат конструкциите,които трябва да бъдат изпълнени от програмата.
Първата ни <u>конструкция</u> извиква функцията cout (си аут)
Тя е дефинирана в стандартната входно-изходна библиотека iostream.h

Операторът <<указва, че cout трябва да изведе текстовия низ в стандартния изход.
Низовете винаги трябва да ги поставяте в кавички.
Нашият низ съдържа текста tova e moiata parva programa и контролния знак \n
който е за нов ред – този знак премества печатащата глава в лявото поле на реда,след като изпише текста ни (в команд промпта,когато сме в конзолен режим:)

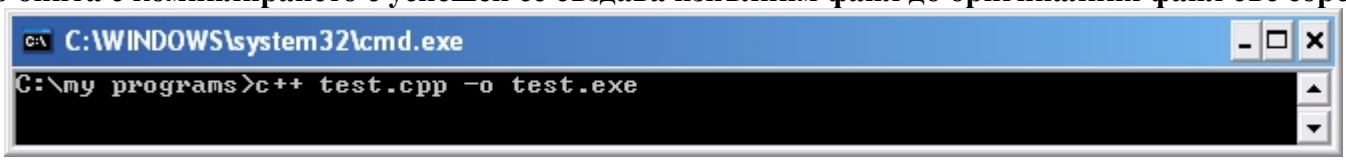
Последната конструкция във функцията main използва ключовата дума return , за да върне стойност нула на операционната с-ма.Това ще означава,че програмата е изпълнена правилно.

Нека сега да опитате да компилирате програмата:
За целта си създайте примерно една директория=папка озаглавена My Programs
в c:\ My Programs
и там ще си съхранявате работните файлчета.
В Линукс е /home/MyPrograms

Файлът със сорс(изходен,изворен) кода на примерната програма по-горе дето я писах може да го озаглавите test.cpp и го записвате в директорията My Programs
Сега от команден промпт ще намерим тази директория,за да компилираме файла...
От Start->Run и написвате cmd -> OK
Ще се отвори команден промпт и там ще видите следния ред:
C:\Documents and Settings\вашето име на компа>
трябва да напишете cd\ т.е. реда да стане ето така:
C:\Documents and Settings\вашето име на компа>cd
и реда става:
C:\>
и за да влезете в директорията My Programs , която сте създали напишете cd my programs и реда става:
C:\>cd my programs натискате Enter и реда трябва да изглежда така:
C:\ my programs > т.е. Вече сте влезли в директорията:)



Когато опита с компилирането е успешен се създава изпълним файл до оригиналния файл със сорса:)



В Уиндоус е изпълним файл .exe ,а в Линукс е .out
За да накарате компилатора да генерира изпълним файл със специфично име трябва да се използва опцията -o последвана от предпочитаното име на файла ... тук в случая е test.exe
Както казах от компилатор до компилатор има разлика и всеки си е със своите специфики...
и се извиква по различни начини...затова трябва да прочитате хелпа на всеки такъв:)
за да знаете как да компилирате с него...
Тук аз компилирам с компилатора на борланд 5 и със Microsoft Visual C++ 6.0 и gnu c++

За Линукс е необходимо преди името на файла да се ползва префикс ./ име на файла

<u>Именовани пространства: (namespace)</u>
Когато включите нов вид хедър (.h) в програмата,то неговото съдържание се намира в именованото пространство std
Именованото пространство представлява декларативна област и ограничава имената на идентификаторите,за да се избегнат колизии м/у тях или иначе казано тяхната цел е да локализират имената на идентификаторите, за да се избегне дублиране на имена.При C++ съществуват много имена на променливи,функции и много класове...едно време е нямало такива именовани пространства и са възниквали множество конфликти...примерно си създавате дефиниранка функцийка някаква и тя може да замени в зависимост от списъка на параметрите си стандартна такава функция:)и двете имена щяха да са записани в глобалното пространство...става дублиране на някои идентификатори от различни библиотеки...и ето защо се създава по-късно тази дума namespace
Съдържанието на новия вид хедъри се поставя в именованото пространство std
Вие също можете да създадете свое именовано пространство в рамките на вашата програма, за да локализирате областта на видимост на някои имена, за които мислите, че могат да причинят конфликти. Това е особено важно, ако създавате библиотеки с класове или функции.
Ключовата дума namespace ви разрешава да обособите част от глобалното пространство, като създадете декларативна област. По същество, именованото пространство дефинира област на валидност. Общият вид на namespace е показан по-долу:

```
namespace име {
  // декларации
  }
```

Всичко, дефинирано в рамките на конструкцията namespace, се намира в областта на видимост на това именовано пространство.

Ето и пример за такова пространство:

```
namespace MyNameSpace {
int i , k;
void myfunc(int j) { cout << j ; }
class myclass {
int i;
public:
void seti (int x) { i = x; }
int geti () { return i; }
};
}
```

Тук i, k, myfunc() и класът myclass са част от областта на видимост, дефинирана от именованото пространство MyNameSpace(МоетоИменованоПространство)
Идентификаторите, дефинирани в рамките на дадено именовано пространство, могат да бъдат директно указвани в неговите рамки. Например в MyNameSpace, конструкцията return i се обръща директно към i. Все пак тъй като namespace дефинира област на видимост, то за обръщението към декларираните в него обекти, от някое място извън него, трябва да използвате оператора за определяне на област на видимост. Например, за да присвоите стойност 10 на i в програмен код извън MyNameSpace трябва да използвате следната конструкция:

```
MyNameSpace::i = 10;
```

А за деклариране на обект от тип myclass извън MyNameSpace се използва конструкция като тази:

```
MyNameSpace::myclass ob;
```

Изобщо, за да получите достъп до даден член на именовано пространство, от място извън него, пред името му трябва да добавите името на пространството, следвано от оператора за принадлежност.
Ако програмата съдържа множество обръщения към членовете на дадено именовано пространство то указването на пространството за всеки от тях се превръща в отегчителна и неприятна задача.
Конструкцията using е създадена с цел да улесни това. Тя има следните две общи форми:

```
using namespace  име;
  using име::член;
```

В първата форма *име* указва името на именованото пространство,до което искате да имате достъп и когато се позлва тази форма всички членове дефинирани в указаното пространство се пренасят в текущото пространство и се ползват директно...
Ако използвате втората форма на using само определен член на именованото пространство става видим...например, ако сме дефинирали MyNameSpace както по-горе, то следните конструкции на using са валидни:

```
using MyNameSpace::k; // само k е видим
k = 10; //  k е видим

using namespace MyNameSpace; // всички членове са си видими
i = 10; // всички членове са видими
```

Може да има повече от една декларация за именовано ппространство с едно и също име. Това позволява пространство да се раздели в няколко файла или дори в рамките на един файл...
пример:

```
namespace NS {
int i;
namespace NS {
int j ;
```

Тук NS е разделено на две части. Съдържанието на всяка от частите спада към едно и също пространство NS
Именованото пространство трябва да бъде декларирано извън всякакви други области на видимост с едно изключение- пространствата могат да се вложат едно в друго. Това означава, че не можете да декларирате именовано пространство, което да е локално за функция...
има един особен вид именовани пространства, наречени *именовани пространства без имена,* които позволяват да създаваме идентификатори, уникални в рамките на даден файл...вида им е:

```
namespace {
// декларации
}
```

Пространствата без имена ви позволяват да създадете уникални идентификатори, които са достъпни само в рамките на определен файл...т.е. в рамките на файла, който съдържа имеиованото пространство без име можете да се обръщате директно към членовете на това пространтсво,но извън файла те са недостъпни...за малки програмки(с кратък код) няма нужда да се създават пространства,но ако пишете библиотеки или се осигурява преносимост се огражда вече в именовано пространство.

2.Съхраняване на данни:

Ще Ви покажа как да съхранявате променливи и константни данни,и много елементи в масиви...

Създаване на променливи(Variables)

Програмистът може да избере произволно име за променливата,но то трябва да отговаря на следните изисквания все пак: обикновено променливите се пишат с малки букви!

Правила за именуване на променлива:	*примерно*
НЕ МОЖЕ ДА ЗАПОЧВА С ЦИФРА	3name
МОЖЕ ДА СЪДЪРЖА ЦИФРА	name3
НЕ МОЖЕ ДА СЪДЪРЖА АРИТМЕТИЧНИ ОПЕРАТОРИ	a+b*c
НЕ МОЖЕ ДА СЪДЪРЖА ПУНКТУАЦИОННИ ЗНАЦИ	#!!% и др.
МОЖЕ ДА СЪДЪРЖА ПОДЧЕРТАВАЩО ТИРЕ	_name_name
НЕ МОЖЕ ДА СЪДЪРЖА ИНТЕРВАЛИ	name name
МОЖЕ ДА Е СЪС СМЕСЕН РЕГИСТЪР	NAme
НЕ МОЖЕ ДА СЪДЪРЖА КЛЮЧОВИ ДУМИ	class или double или while и др.

Ключовите думи в езика C++ са по-надолу ...

Трябва да се измислят смислени имена на променливите! Така ще е по-лесно за Вас.
За да създадете променлива в една програма,тя трябва да бъде декларирана!
Синтаксисът е следния:

тип данни име на променлива ;
int name3 ; примерно е това:)
правилно е да се запише във вида:
int name3;
Първо в декларацията се указва какъв тип данни ще съдържа променливата.
Както виждате типът(int) е последван от интервал и името на променливата.Съобразявайте се с горните правила за именуване:) Накрая декларацията завършва с точка и запетая ;
Важно е да се отбележи,че с една декларация може да се създадат много променливи от един и същи тип имената обаче се отделят със запетайка!

тип данни променлива1, променлива2, променлива3 ;

Типове променливи
Има 5 основни типа данни! Дефинират се с ключови думи...

тип данни	описание	
char	байт,който съдържа един знак	A
int	цяло число	100
float	число с плаваща запетая с точност до шест десетици	0.123456
double	число с плаваща запетая с точност до десет десетици	0.0123456789
bool	булева с-ст (истина,неистина=true,false)	false(0) true(-1)
	всяка ненулева стойност представлява true(истина)	

C++ предоставя специален тип данни string(низ)
Петте типа данни заделят различно количество машинна памет за съхраняване на данните!
Типът char е най-малкият, а double е най-големия...той е два пъти по-голям от типа float и затова го използвайте само когато е необходимо точно дълго число с плаваща запетая!

Стойностите на типа char трябва да бъдат поставяни в апострофи, а не в кавички!!!
Декларациите на променливи се правят преди изпълнимия код...
Когато на променлива бъде присвоена стойност(става със знака =) се казва,че тази променлива е „инициализирана“... но да не Ви обърквам с термини:)
знакът за присвояване на стойност е = (това не означава равно)

ето няколко примерчета:

```
int nomer1, nomer2 ;  //декларираме в случая 2 променливи от типа integer
char bukva;               //декларирана променлива от тип char
double dalgo4islo;
float deseti4no = 9.5;  //декларирано е и инициализирано=присвоена е стойност 9.5 и типа става float:)
bool ime = true;         //задаваме булева стойност истина е инициализирана...
nomer1 = 100;
nomer2 = 200;           //инициализираме си променлива от integer
bukva = 'A';             //инициализира прменлива от типа char  и забележете слага се в апостроф!
Dalgo4islo = 1.987654321;    // тип double
ime = false;                        // присвояваме нова булева стойност на променливата
```

накрая се поставя точка и запетая!
преди да присвоявате стойност на променлива,то тя трябва да се декларира:)
не е задължително променливите да бъдат инициализирани при декларирането

Нека сега да <u>визуализираме стойностите на променливите</u>:
Това ще рече да се види всичко в команд промпта(cmd)...ще работи в конзолен режим...а при визуалното програмиране се работи в специалните програмни среди(IDE) и всичко е един вид визуално:)
но и там може да си програмирате в конзолен режим(избира се конзолна апликация проект)ще обясня по-надолу...

Стойностите на променливите могат да се визуализират с помощта на функция cout (си аут)
тя служи за извеждане(<u>изход</u>)...както горе изведохме низа tova e moiata parva programa
Стойността,която е съхранена във всяка променлива заменя името на променливата в изхода!
Ето примерче:

```
#include <iostream>
using namespace std;

int main ()
{
    int nomer1 = 100;
    char bukva = 'E';
     float deseti4no = 9.5;
     double pi = 3.14 ;
     bool ime = false;

cout « “nomer:\t“ « nomer1 « endl;
cout « “bukva:\t“ « bukva « endl;
cout « “deseti4no:\t“ « deseti4no « endl;
cout « “pito:\t“ « pi « endl;
cout « “imeto:\t“ « ime « endl;
return 0;                          //изход от програмата
}
```

Тук използвам контролен знак за табулация(Tab) , за да има подравняване в изхода...повече за тези контролни знаци по-надолце:)
Този пример показва как програмата създава и инициализира променливи за 5-те основни типа данни и после визуализира съхранените в тях стойности...В края на всеки ред от изхода виждате,че има конструкция endl Това е алтернатива на контролния знак \n ,което значеше нов ред нали си спомняте буквата след end е l (ел) да не си помислите,че е числото едно:)))))))

Компилирайте си примерчето с компилатора на борланд:)

```
// друг пример за изход(output)
#include <iostream>
using namespace std;

int main()
{
   int num1 = 1234, num2 = 5678;
   cout << endl;                                  //старт от нова линия(първи ред е празен:)
   cout << num1 << num2;                          //извежда две стойности
   cout << endl;                                  //край на новата линия(ред)
   return 0;                                      //изход
}
```

и ще се изведе 12345678 в конзолата

Въвеждане на стойности на променливи (вход) input

Функцията cin (си ин) използва двата знака за по-голямо от >>
и те обозначават вход
А по-нагоре използвахме cout (си аут) <<изход
Та cin се използва за получаване на вход от потребителя...т.е. когато в конзолата(команд промпта) се появи съобщение,което сме извели с cout и то ни приканва да въведем да кажем някакво число...
Сега ще дам примерче със събиране на две числа като ще бъде приканен потребителя да въвежда числа
Разбира се това може да се направи по няколко различни начина,но номера е да разберете в случая що е то вход:)нещо като да си влезете у вас,ама треало да имате ключ:)

```
#include <iostream>                 // <iostream.h>   хедърчето=заглавен файл нали се сетихте:)
using namespace std;                // ползваме именованите пространства...всеки компилатор различно:)

int main()
{
int nomer1, nomer2, ob6to;

cout « “Vavedete 4islo: “;      // това съобщение ще се изведе в промпта(не можете да пишете на кирилица)
                                //когато програмирате във визуална среда тогава може във Form апликация
                                //да си пишете на кирилка и методийка в лейбълчетата  Label ,но има време:)
cin » nomer1;                   //ето тук вече е за въвеждането      входче
cout « “Vavedete drugo 4islo: “;
cin » nomer2;
ob6to = nomer1 + nomer2;            //поставя се интервал преди и след на равното и плюсчето:)и на другите
cout « “Ob6to 4isloto e: “ « ob6to « endl;

return 0;
}
```

Компилирате и ще стане:) сбор на числата...

Ето и още примерчета относно конзолен вход/ изход (input/output)
cin>> / cout<<
или scanf() / printf(“тук е низа); това е при езика C,но може да пишете и тези функции в C++
За въвеждане на стойности от клавиатурата,използвате тази форма на оператора »
cin » променлива;
И понеже ползваме операторите на C++ за вход/изход ,затова включваме хедъра <iostream>

Долу давам още примерчета за вход/изход:

Тази програмка ще изведе низ, две целочислени стойности и ст-ст с плаваща запетая:

```
#include <iostream>
using namespace std;

int main()
{
int i, j;
double d;              //число с плаваща запетайка:)

i = 10;
j = 20;
d = 99.101;

cout « “eto stoinostite: “;
cout « i;
cout « ' ';
cout « j;
cout « ' ';
cout « d;

return 0
}
```

Изходът от програмката е следния: eto stoinostite: 10 20 99.101
забележете,че са на един ред с интервал само!

Или този пример може да се направи и по следния начин:

```
#include <iostream>
using namespace std;

int main()
{
      int i, j;
      double d;

       i = 10;                    // цяло число е нали виждате:)
       j = 20;
       d = 99.101;

       cout « “eto stoinostite : “;
        cout « i « ' ' « j « ' ' « d;              //извежда ги:)

return 0;
}
```

Следващата програмка подканва потребителя да въведе целочислена стойност:

```
#include <iostream>
using namespace std;

int main()
{
int i;

cout « “Vavedi stoinost: “;
cin » i;
cout « “Eto q stoinostta:“ « i « “\n“;
return 0;
}
```

Ето и резултата от изпълнението на програмката:

```
Vavedi stoinost: 100
Eto q stoinostta: 100
```

т.е. въведената стойност е присвоена на i

Следващата програма подканва потребителя да въведе целочислено число,число с плаваща запетая и низ,но ще използваме един единствен израз,за да прочетем и трите стойности:

```
#include <iostream>
using namespace std;
int main()
{
int i;
float f;
char s [80];

cout « “Vavedi integer, float, string: “;
cin » i » f » s;
cout « “Eti gi dannite: “;
cout « i « ' ' « f « ' ' « s;
return 0;
}
```

Можете да въвеждате колкото си искате елементи в един израз.Елементите от данни трябва да са разделени с празни полета(интервали,табулации,празен ред).
Ако се чете низ(както в случая с таз програмка) то въвеждането от клавиатурата ще приключи при първия празен символ: пример
като въведете 10 100.14 vavejdam niza eto go kade e
ще се изведе само 10 100.14 vavejdam
низа е незавършен понеже прочитането му е спряло след vavejdam
останалата част е останала във входния буфер като изчаква следваща операция за вход

Когато се ползва оператора » всичко въведено се буферира поредово...няма нужда от натискане на команда...но ето и програмка за поредово буфериране:

```
#include <iostream>
using namespace std;
int main()
{
char ch;
cout « “vavedi, x za da spre \n“;
do {
cout « “: “;
cin » ch;
} while (ch != 'x') ;
return 0;
}
```

тук вече ще се изисква да се натиска клавиша ENTER след всеки натиснат клавиш:)
ползва се и цикъла do-while,който ще разгледам по-надолу:)

Сега ще разгледам определителите на типа данни:
все още говорим за типовете данни: char int float double bool
нали помните:)ок

Интервалът от възможните int стойности се определя от операционната с-ма!
int променливата ако се създава по подразбиране като тип long ,то тя ще позволи интервал от стойности +2 147 483 647 до -2 147 483 648

int променливата ако се създава по подразбиране като тип short ,то тогава интервалът ще е от +32 767 до -32 768 (не разбрахте нали:)няма нищо:)в процеса на програмиране ще схванете:)просто запомнете теа двата типа long , short дълго , късо:)

Тези ключови думи се поставят преди ключовата дума int
Определителят short int се използва за спестяване на памет и треа да сте сигурни,че диапазона от стойности няма да е превишен
Когато бъде декларирана променлива от тип int ,по подразбиране тя може да съдържа както положителни ,така и отрицателни цели числа или така наречените стойности със знакове.
Примерно: променливата винаги да съдържа положителни цели числа
то тогава ние ще декларираме една неотрицателна unsigned int променлива
(unsigned е ключова дума от общо 46-те ключови думи в програмен език C++ ,те имат специално значение в програмите и не могат да се използват за друго)
Една unsigned short int променлива има допустим диапазон от ст-ст 0 до 65 535 ,а една unsigned long int от 0 до 4 294 967 295

примерче:

```
# include <iostream>
using namespace std;
int main()
{
cout « “short int: “ « sizeof (short int) « “bytes\n“
cout « “long int: “ « sizeof (long int) « “bytes\n“
cout <<“default int: “ « sizeof (int) « “bytes\n“;
return 0;
}
```

ползвам оператора sizeof ,за да проверя количеството памет , което се заделя за типовете данни int при всяка с-ма е различно:)

Операторът sizeof връща желочислена ст-ст,която представлява броя байтове
Броят байтове,заделени за типовете данни зависи от реализацията-за всяка с-ма е различно

Сега ще разгледам що е то Масиви (Arrays)
(задължително трябва да се знаят понятията как са съответно на английски затова ще има доста найменования на английски)
Все още съм на тема 2. Съхраняване на данни:)

Много е важно в програмирането да се разберат масивите и ще се опитам да Ви ги обясня сега:)

Масив:
Масивът е променлива, която може да съхранява множество елементи с данни.
Обикновената променлива можеше да съхранява само един елемент нали помните!
Обектите представляват променливи и имат същите св-ва както всеки тип променливи.
Можем да създаваме масиви от обекти.
Синтаксисът за декларирането на масив от обекти е същият като този,който се използва за деклариране на масив от всеки друг тип променливи.

Номерирането на елементите на масива започва от нула,а не от едно!
Отделните данни се съхраняват последователно в елементи на масива и те са номерирани...
т.е. първата ст-ст се съхранява в елемент нула,втората стойност в елемент 1 и т.н.
Трябва да знаете,че масива се декларира по същия начин,както другите променливи!
Само се добавя в декларацията размера на масива и то се прави като се пишат квадратни скоби[] след името му
arr[0] //arr такова име съм му задал идва от arrays(масиви) затова е така:)
arr[1]
arr[2] и т.н.

Масивът може да бъде инициализиран при декларирането му чрез присвояване на ст-сти на всички елементи във вид на списък с разделител запетайка,но се поставя във фигурни скобки { } и реда завършва естествено с точка и запетайка ;

Нека сега разгледаме това примерче:
програмката създава масив с 3 елемента и присвоява стойности на всеки от тях

```
#include <iostream>
using namespace std;
int main()
{
int arr[3] = {10, 20, 30};     //  присвоени стойности
cout « “arr[0] : “ « arr[0] « endl;
cout « “arr[1] : “ « arr[1] « endl;
cout « “arr[2] : “ « arr[2] « endl;
return 0;
}
```

и ето го резултата: arr[0] : 10
arr[1] : 20
arr[2] : 30
а сега нека да направим промянка във втория елемент,преди да се визуализират ст-стите в изхода от програмката...кодът ни става следния:

```
#include
using namespace std;
int main()
{
int arr[3] = {10, 20, 30};     //  присвоени стойности

arr[1] = 40;                    //виждате променяме втория елемент:)
cout « “arr[0] : “ « arr[0] « endl;
cout « “arr[1] : “ « arr[1] « endl;
cout « “arr[2] : “ « arr[2] « endl;
return 0;
}
```

ето и резултата: arr[0] : 10
arr[1] : 40
arr[2] : 30

Масиви от знакове:
Масивите не са ограничени до типа данни int, а могат да бъдат създадени за всеки тип данни : int,float,double,bool,char

Всички елементи в даден масив трябва да бъдат от един и същ тип!
Масивът от знакове ,при който всеки елемент съхранява един знак може да бъде използван за съхраняване на текстов низ,но последния елемент трябва да съдържа специалния нулев знак \0
ето и примерче:

```
#include <iostream>
using namespace std;
int main()
{
bool flags[3] = { true, true, false };
double nums[3] = {1.83, 4.56, 7.89 };
char fname[5] = { 't','e','s','t', '\0' };
cout « flags[2] « endl;
cout « nums[0] « endl;
cout « fname[0] « endl;      //извежда знак
cout « fname « endl;         //извежда низа
return 0;
}
```

ето резултата:
0
1.83
t
test

извлича се един знак от масива от типа char и после цялото му низово съдържание...

Многомерни масиви:
Синтаксисът е следния:
type name[1] [2]...[n]; //типа и името му
В скобките се задава дължината
При многомерния можете да имате 2,3,4...до n измерения,където n е произволно голямо число.
Масивите с много голям брой измерения обикновено поглъщат цялата налична памет:)
При многомерните масиви общият брой елементи е равен на произведението от обявените дължини на всяко от измеренията.
Двумерните масиви се използват за съхраняване на инфо във формата на мрежа,както е например координатната мрежа.
Може да се създават многомерни масиви от обекти.

Ето примерче за създаване на двумерен масив
всеки индекс има 3 елемента:

0	**1**	**2**
1	2	3
4	5	6

два реда и три колони
първият ред,или индекс съдържа числата от 1 до 3,а втория ред или индекс-числата от 4 до 6

```
#include <iostream>
using namespace std;
int main()
{
int arr[3] [3] = { {1,2,3} , {4,5,6} };

cout « “arr[0] [0] sadarja “ « arr[0] [0] « endl;
cout « “arr[0] [1] sadarja “ « arr[0] [1] « endl;
cout « “arr[0] [2] sadarja “ « arr[0] [2] « endl;
cout « “arr[1] [0] sadarja “ « arr[1] [0] « endl;
cout « “arr[1] [1] sadarja “ « arr[1] [1] « endl;
cout « “arr[0] [2] sadarja “ « arr[1] [2] « endl;
return 0;
}
```

добре разгледайте подреждането в квадратните скобки:) присвояваме първоначални ст-сти на всеки елемент от индексите в декларацията на масива
ще се изведе в команд промпта следното:

```
arr[0] [0] sadarja 1
arr[0] [1] sadarja 2
arr[0] [2] sadarja 3
arr[1] [0] sadarja 4
arr[1] [1] sadarja 5
arr[1] [2] sadarja 6
```

Всеки един пример,който съм дал тук си го пишете вие в текстовия редактор или програмна среда,а не го копирайте и пействайте...свиквайте да изписвате кода!

Нека сега създадем един <u>многомерен масив от обекти:</u>
Например следната програмка създава двумерен масив от обекти и ги инициализира(присвоява стойност) :

```
#include <iostream>
using namespace std;

class samp {
int a;
public:
samp (int n)  { a = n; }
int get_a ()    { return a; }
};
int main ()
{
samp ob[4] [2] = {
1, 2,
3, 4,
5, 6,
7, 8
};
int i;

for (i=0; i<4; i++)  {
cout « ob[i] [0] .get_a () « '  ';
cout « ob[i] [1] .get_a () « “\n“;
}
cout « “\n“;

return 0;
}
```

Разбира се този пример в момента не е точно за начинаещи:)понеже още не съм обяснил работата с класове(дефинирането) и ключовата дума public навярно сега е непонятна за Вас,както и др.думички в кода:)споко! надолу ще говорим за тях:)
...та...програмката ще изведе:
1 2
3 4
5 6
7 8

Ето сега едно друго примерче, използвайки масивите:

```
#include <iostream>
#include <iomanip>   //виждате включва се този хедър,който служи за форматиране на входните и изходни
                     //потоци        манипулаторите за потоците за вход-изход иначе казано
using namespace std;

int main()
{
  int value[5] = { 1, 2, 3 }; //ст-ст е value
  int Junk [5];   //можете и друго име да му дадете на втория
  cout << endl;

  for(int i=0; i<5; i++)
    cout << setw(12) << value[i];

  cout << endl;

  for(i=0; i<5; i++)
    cout << setw(12) << Junk[i];

  cout << endl;
  return 0;
}
```

ето резултата:

```
           1           2           3           0           0
 -858993460  -858993460  -858993460  -858993460  -858993460
Press any key to continue
```

виждате програмката ни генерира 2 реда,като при всеки компютър ще е различно:)досетете се защо:)

Сега да наблегна на вектора:) Vector
Вектори:

Той е алтернатива на обикновения масив,а предимство е това,че неговият размер може да го променяме според нуждите на програмата.Както масивите,векторите могат да бъдат създавани за всеки тип данни, а елементите им се номерират пак започвайки от нулата...
Запомнете,че за да използвате вектори в една програма трябва да добавите C++ библиотеката <vector> хедъра е това <vector.h> все в гъз:)
тва го правим чрез директивата #include нали се сещате:)
#include <iostream> //включи хедър
#include <vector> //включи хедър файл вектор4е:) така го 4етете всичко това:)
накордваме си ги тез,който ще ни треат за програмката:)
Тази библиотека <vector> съдържа предварително дефинираните методи,които се използват за работа с вектори.Всъщност трябва да познавате библиотечката на c++,но ще я разгледам по-надолце,а и в процес на работа ще я изучите:)и ще знаете какво трябва за дадена програма...Библиотеките,използвани от работеща програма на c++ се състоят от стотици редове код на c++,c,асемблер.Щеше да е мнооо бавно,ако се налагаше целия код да бъде компилиран наново всеки път,затова библиотеките се разпространяват като обектен код,който се свързва с програмата.Това се прави от специална програма наречена линкер,който изследва кода и включва функциите,към които има обръщение...надолу ще обясня обстойно за библиотеките:)

сега да обясня дефинираните методи относно хедъра вектор <vector>

at (номера на елемент)	**извлича ст-та на указания елемент**
back()	**извлича стойността на последния елемент**
clear()	**изтрива вектора**
pop_back()	**премахва последен елемент**
push_back(ст-ст)	**добавя елемент в края на вектора,със зададената ст-ст**
size()	**получава броя на елементите**
empty()	**връща истина(1) при празен вектор обаче или връща неистина(0) в противен случай**
front()	**получава ст-ста на първия елемент**

Декларацията за създаване на вектор има ето този синтаксис:
vector < тип данни > името на вектора (размера) ;

```
vector<int> ime ; // създава целочислен вектор с дължина нула
vector<char> cv(5);  // създава символен вектор с 5 елемента
vector<char> cv(6, 'x');  // инициализира символен вектор с 6 елемента
vector<char> iv4 (iv);     // създава целочислен вектор от друг такъв
```

По подразбиране всички елементи на вектор от тип int ще се инициализират със ст-ст нула!!!
В декларацията можете да задавате първоначална ст-ст след размера ползвайки този синтаксис:
vector <тип данни > името на вектора (размер , стойност) ;

За vector са дефинирани тези оператори за сравнение: (надолу ще обясня обстойно операторите)
==,<,<=,!=,>,>=
тук набързо ги давам: оператора за сравнение == означава равно на, < е по-малко, <= по-малко или равно на, != е различно от, > е по-голямо от, >= е по-голямо или равно на
Индексният оператор [ ] също е дефиниран за vector

Методите за работа с вектори просто се добавят към името на вектора с оператора точка .
име на вектора . метод()

ето примерчета сега:

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
vector<int> v;   // създава вектор с нулева дължина
int i;

cout <<“razmer = “ « v.size () « endl;    //показва първоначалния размер на  v
for (i=0; i<10; i++) v.push_back (i);//слага ст-сти в края на вектора при необходимост вектора ще нараства
//показва текущия размер на v
cout « “teku6to sadarjanie: \n“;
cout « “razmera sega = “ « v.size () « endl;
//показва съдържанието на вектора
for (i=0; i<v.size() ; i++) cout « v[i] « “  “;
cout « endl;
//слага още ст-ти в края на вектора като при необходимост вектора може да нараства
for (i=0; i<10; i++) v.push_back(i+10);
//показва текущия размер на v
cout « “razmera sega = “ « v.size() « endl;

// показва съдържанието на вектора
cout « “teku6to sadarjanie: \n“;
for (i=0; i<v.size(); i++) cout « v[i] « “  “;  // v.size    v като вектор , а size() го обясних по-горе:)
cout « endl;

// променя съдържанието на вектора
for (i=0; i<v.size(); i++) v[i] = v[i] + v[i];

cout « “udvoen razmer: \n“;
for (i=0; i<v.size(); i++) cout « v[i] « “  “;
cout « endl;

return 0;
}
```

изходът от програмата е:

```
razmer = 0
razmera sega = 10
teku6to sadarjanie:
0 1 2 3 4 5 6 7 8 9
razmera sega = 20
teku6to sadarjanie:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
udvoen razmer:
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38
```

В main() се създава целочислен вектор с име v
Не използваме инициализация тук! Вектора е празен с капацитет нула т.е. Това е вектор с нулева дължина...програмата потвърждава това като извиква функцията size()
Към края на v се добавят десет елемента с член функцията push_back()
От там пък v нараства и побира новите елементи...след това се добавят още 10 елемента и v автоматично увеличава размера си.

Ето сега да демонстрирам как се ползват методите за обработка на векторните елементи:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
vector <int> vec(3,10);
cout « “razmer na vectora e: “ « vec.size() « endl;
vec.push_back(7); vec.push_back(8); vec.push_back(9);
cout « “\t3 elementa dobaveni“ « endl;
cout « “razmera na vectora sega e: “ « vec.size() « endl;
cout « “\tparvia element e: “ « vec.front() « endl;
cout « “\tvtoriat element e: “ « vec.at(1) « endl;
cout « “\tposledniat element e: “ « vec.back() « endl;
vec.pop_back();
cout « “\tposledniat element premahnat“<< endl;
cout « “\tposledniat element sega e: “ « vec.back() « endl;
cout « “razmera na vectora sega e: “ « vec.size() « endl;
cout « “prazen li e vectora?:  “<< vec.empty() « endl;
vec.clear();
cout « “\tvectora e iz4isten“ « endl;
cout « “razmera na vectora sega e: “ « vec.size() « endl;
cout « “prazen li e vectora?: “ << vec.empty() « endl;
return 0;
}
```

извежда се следното:

```
razmer na vectora e: 3
            3 elementa dobaveni
razmera na vectora sega e: 6
parvia element e: 10
vtoriat element e: 10
posledniat element e: 9
posledniat element premahnat
posledniat element sega e: 8
razmera na vectora sega e: 5
prazen li e vectora?: 0
vectora e iz4isten
razmera na vectora sega e: 0
prazen li e vectora?: 1
```

Константите:
дефиниция: Както програмите имат променливи,то те така могат да имат и константи...
те представляват фиксирани стойности в програмата...почти всяка програма ползва някъде константи
Данните,които няма да се променят по време на изпълнението на програмата трябва да се съхраняват като константна величина...т.е. те няма да са в променлива.При опит на програмата да промени стойността,то тогава компилатора отчита грешка...За имената на константите винаги ползвайте главни букви...така ще ги разграничавате от имената на променливите.Константата може да бъде създадена за всеки тип данни чрез поставяне на ключова дума const пред декларацията и после има интервал.
Константите трябва винаги да ги инициализирате в декларацията...при променливите спомняте си,че не е така:)
класическият пример е с константата Пи (Pi=3.14159)

```
#include <iostream>
using namespace std;
int main()
{
const double PI=3.14159;
double rad;       //radius:)
cout « “Vavedete radiusa: “;
cin » rad;
cout « “Diametar: “ « (PI*(rad*rad)) « endl;
cout « “Plo6t: “ « (PI*(rad*rad)) « endl;
cout « “Okrajnost: “ « (PI*(rad*rad)) « endl;
return 0;
}
```

изхода от програмката е:

```
Vavedete radiusa: 20
Diametar: 40
Plo6t: 1256.64
Okrajnost: 125.664
```

Константите може да ги разделим на три типа:литерали,символни константи и константни променливи
Символните константи се транслират от предпроцесора в литерали.Ето примерчета за константи с плаваща запетая:3.1415 12000.5 63e-2 85.66789e12
Символните константи се транслират в кодове на ASCII („аски“ американски стандартен код за обмен на информация-това е приетия стандарт за обикновен текст...знаковете се представят чрез числов ASCII код в интервал от 0-127 .... буквата А е числовия код 65)
пример:СОТ е друг начин за представянето на числото 48
Дали една ст-ст е число или буква се определя от начина,по който се използва,а не от числовото съдържание.
Ключовата дума enum се ползва за създаването на серия от целочислени константи по сбит начин.
Може да ги именувате след ключова дума enum ,ако искате при задължителното деклариране...
Имената на константите се записват като списък с разделителна запетая и във фигурни скобки
Всяка от константите има стойност с единица по-голяма от предшестващата я константа в списъка.
На константите може да се присвояват произволни различни стойности като стойността на следващите константи винаги нараства с единица!
Ако една програма ползва променливи от един и същ тип и определител е удобно да се създава константа за представяне на този тип...това е един вид съкратен запис...
често срещаният пример се дава: многократно да се изписва unsigned short int
и затова да се създаде константа чрез ключовата дума typedef която ще представлява този тип...
Декларацията обаче на тази константа трябва да бъде в началото на програмата след предпроцесорните директиви...ето и синтаксиса: typedef тип данни ИМЕТО ;

Дефинирането на константите: предпроцесорната директива #define може да се ползва за задаване на константни текстови стойности...

```
#define ИМЕ НА КОНСТАНТАТА “текста тук(низа) “
```

тази директива #define трябва да я поставяте в началото на програмния код
както поставяме директивата #include

```
#include <iostream>
using namespace std;
#define LINIA "_____"
#define SAZDAL "Brigante"
#define PRIMER4E "Test4e"
#define ETI "\t"
int main ()
{
cout « ETI « LINIA <<endl;
cout « ETI « SAZDAL « endl;
cout « ETI « "si pravi" « PRIMER4E « endl;
cout « ETI « LINIA « endl;
return 0;
}
```

изходът от кода е:

```
_____
Brigante
si pravi Test4e
_____
```

Всеки елемент с данни и ако е съхранен в променлива може да бъде конвертиран в променлива от различен тип чрез процеса предефиниране (casting)... предефинирането указва типа данни,в който трябва да бъде конвертирана стойността преди името на променливата:

име на променливата = (типа данни) име на променливата ;

Директивата #define се ползва,за да предостави лесен и ефективен начин за създаване на четливи константи(символни константи),за да позволи да се пишат макроси(които са подобни на функциите,но се реализират чрез замяна на текст по времето на компилирането);
директивата може да се ползва и за дефиниране на символ с цел контролиране на условие по време на компилирането ...
Повечето програми работят с числа,а те се помнят трудно и затова дефинираме...
примерно да дефинираме константата Пи
#define PI 3.1459265
просто като ползваме тази директива и при всяко срещане на PI в програмата компилатора ще замени PI с числото 3.14...
Символните константи се използват най-често за представяне на произволни числа като максимален размер на низ или размер на масив.Обикновено всички такива числа се дефинират като символни константи в началото на програмата казахме вече:)

Константите в C++ могат да се разделят и на два типа именовани константи и неименовани константи. Неименованите константи са такива, които притежават тип и стойност, но нямат име, докато именованите константи имат както тип и стойност, така и име.

Неименовани константи

Неименовите константи са четири вида: целочислени, дробно-десетични, символни, и низови. Вместо понятието низови може да се срещне в литературата и стрингови константи.

Целочислени

Целочислените константи се състоят от последователност от осмични, десетични или шестнадесетични цифри. Пре десетичните константи като база за представяне на стойностите се използва десетичната бойна система дснова 10 и цифрите 0 - 9.

Примери:

1 // Десетичните константи

23 // не притежават 0 в началото си

345

67899

Първата цифра на десетичните константи не може да бъде 0.

01 // Осмичните

056 // константи

075 // започват винаги с 0

Това от какъв тип ще бъде една целочислена константа зависи от нейната стойност.Десетична константа притежава тип int при положение, че нейната стойност се представя чрез този тип. Ако обаче нейната стойност е твърде голяма , за да се представи чрез int, то тя получава тип long или unsigned long. За осмичните и шестнадесетичните константи се отнася същото. Освен това съществува възможността програмистът сам да извърши явно преобразуването на константите чрез един суфикс след тях.
L се използва са преобразуване към тип long

U се използва за пребразуване към тип unsigned int.

Възможна е също и комбинация от двата суфикса - UL или LU.

За типовете със плаваща точка ако сложите след числото *F* то ще се третира като *float.*

тип данни	Примери за константи
int	1 123 21000 -234
long int	35000*L* -34*L*
unsigned int	100000*U* 987*U*
float	123.23*F* 4.34e-3*F*
double	123.23 -0.987324
long double	1001.2*L*

<u>Символни константи</u>

Една символна константа се сътои от един или няколко последователни символа затворени в горни апострофи.Символните константи притежават тип char и са подобни на целочислените константи.

Низови константи

C++ поддържа един друг тип константа в добавка към тези предефинирани типове данни : низ. Низа е множество от символи затворено в двойни кавички(" "). Например "*this is a test*" е низ. Вие не трябва да бъркате низовете със символите . Единична символна константа е затворена между единични кавички такава като *'a'*. Обаче *"a"* е низ съдържащ само една буква.

C++ определя и две булеви константи : *true* и *false*.

3.Извършване на операции

-аритметични операции
-логически оператори
-оператори за присвояване на стойности
-сравнения
-условен оператор

Да разгледаме сега аритметичните оператори:

оператор	оператор
+	събиране
-	изваждане
*	умножение
/	деление
%	деление на модул или с остатък
++	нарастване(инкремент) increment
--	декремент decrement

Оператора % връща остатъка от целочислено деление.Той е полезенпри определянето на това,дали едно число е четно или нечетно.
Операторът ++ и – променят зададената ст-ст с единица и връщат резултатната нова ст-ст.
Операторите за инкрементиране и декрементиране могат също така да бъдат поставени преди или след съответната им ст-ст и ефекта е различен...ако бъдат поставени преди операнда неговата ст-ст ще се промени веднага,а ако се поставят след операнда ,то неговата ст-ст първо се изчислява и след това се променя стойността.

Логическите оператори са:

Оператор	Значение
&&	AND логическо И
||	OR логическо ИЛИ
!	NOT логическо НЕ

Логическите оператори се използват да свържат две стойности или в случая на ! да обърнат стойността.Логическите оператори се използват с операнди,които имат булеви стойности истина(true) неистина (false)
Логическият оператор && (И) оценява двата операнда и връща истина само ако и двата операнда са истина.В противен случай && връща неистина...
Операторът || (ИЛИ) оценя двата си операнда и връща истина,ако единия от тях е истина...ако обаче и двата операнда са неистина ,оператора || (ИЛИ) ще върне неистина.
Логическият оператор ! (НЕ) е унарен оператор ползва се пред един операнд...връща обратната ст-ст на зададения операнд...ако променливата е имала стойност истина ,тогава тази променлива ще върне стойността неистина.Операторът ! се ползва в програми на C++ за превключване на стойността на променлива в последователни итерации през цикъл чрез конструкция : примерно a=!a
В програми на C++ нулата представлява логическата ст-ст на неистина,а всяка ненулева ст-ст като 1 е булевата ст-ст истина.

```
#include <iostream>
using namespace std;
int main()
{
int a = 1, b = 0 ;
cout « “a =  “ « a « “\tb =  “ « b « endl;  //хоризонтална табулация с  \t
cout « “I (AND) primer4e: “ « endl;
cout « “\t a && a = “  «  ( a && a ) « “ ( true ) \n“;
cout « “\t a && b = “  «  ( a && b ) « “ ( false ) \n“;
cout « “\t b && b = “  «  ( b && b ) « “ ( false ) \n“;
cout « “ILI (OR) primer4e: “ « endl;
cout « “\t  a  || a = “  «  ( a  ||  a ) « “ ( true ) \n“;
cout « “\t  a  || b = “  «  ( a  ||  b ) « “ ( true ) \n“;
```

```
cout « "\t  b  ||  b = "  «  ( b  ||  b ) « " ( false ) \n";
cout « "NE (NOT) primer4e: " « endl;
cout « "\t  a = "  «  a  « " !a = " « !a  « endl;
cout « "\t  b = "  «  b  « " !b = " « !b « endl;
return 0;
}
```

изходът от кода е:

```
a = 1     b = 0
I (AND) primer4e:
                a && a = 1 ( true )
                a && b = 0 ( false )
                b && b = 0 ( false )
ILI (OR) primer4e:
                a | | a = 1 ( true )
                a | | b = 1 ( true )
                b | | b = 0 ( false )
NE (NOT) primer4e:
                a = 1  !a = 0
                b = 0  !b = 1
```

разгледайте добре кода и го разберете:)
тук тестваме булевите стойности като са показани всички логически оператори
операторът ! обръща булевата ст-ст ,оператора && връща 1 ,ако двата операнда имат ненулеви ст-сти
операторът || връща 1 ,ако единия операнд има ненулева ст-ст...

Оператори за присвояване на стойности:

оператор	примерче	еквивалентността му
=	a = b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a – b
*=	a = b	a = a*b
/=	a /= b	a = a/b
%=	a % = b	a = a % b

Операторът = е оператор за присвояване (на стойност на променливи)
Когато присвоявате обща стойност на няколко променливи , вие можете да ги "вържете заедно "
Например: *a = b = c = 10 ;*
присвоява на a , b и c стойност 10

на оператора = не забравяйте да гледате като присвояване,а не да го бъркате с равенството:)
оператор за равенство е ==

в горния пример за еквивалентност дето съм дал
a = b (на променливата a е присвоена ст-стта,която се съдържа в променливата b и това е вече новата ст-ст съхранена в променливата a)

Операторът += се използва за добавяне на стойност към съществуваща ст-ст ,която е съхранена в променлива.
В примера от табличката += първо добавя ст-стта на променливата a към стойността на променлива b ,след това присвоява резултата на променлива a

Другите оператори работят по същия начин...първо се извършват аритметичните операции м/у двете стойности и после се присвоява резултата към първата променлива

При оператора %= първия операнд a се разделя на втория b и после остатъка от операцията се присвоява на променливата a

Оператори за сравнение(релационни):

Те се ползват за сравняване на две числови стойности...
Релационните и логическите оператори се използват за получаване на *true / false* резултати и са често използвани заедно.

Оператор	Значение
>	По–голямо от
>=	По–голямо или равно на
<	По–малко от
<=	По–малко или равно на
= =	Равно на
!=	Неравно,различно от

Операторът за равенство == сравнява двата операнда и връща 1 (истина) при равни стойности разбира се...иначе връща 0 (неистина)
ако операндите са знакови се сравняват техните ASCII кодове...

Операторът за неравенство != връща 1 ,ако двата операнда не са равни ,иначе връща 0

Операторът > сравнява двата операнда и връща 1 , ако първия операнд е по-голям от втория...
а връща 0 ,ако стойността на първия операнд е по-малка или равна на ст-стта на втория.Операторът > се ползва често при проверяване на ст-ст за обратно преброяване в цикъл.

Операторът < прави същото сравнение,но връща 1 ,ако първия операнд е по-малък от втория
иначе връща 0

... при >= и <= ,ако двата операнда имат равни стойности се връща истина

Условен оператор:

За проверка на условия се използва оператора ?:
Той първо оценява дали някакъв израз е истина,или неистина и после в зависимост от резултата от оценяването изпълнява едната от две зададени конструкции...
ето и синтаксиса на условния оператор:

(условен израз) ? ако е истина направи това : ако е неистина направи това ;

пример:

```
#include <iostream>
using namespace std;
int main()
{
int nomer1 = 21325, nomer2 = 14340;
char bukvi;     //променлива bukvi от типа char
cout « nomer1 « “ e  “;
(nomer1 %2 != 0)  ?  cout « “ne4etno“ : cout « “4etno“;
cout « endl « nomer2 « “ e “;
(nomer2 %2 != 0)  ?  cout « “ne4etno“ : cout « “4etno“;
letter = (nomer2 %2 != 0)  ?  'Y' : 'N';   // YES  NO
cout « “\nDali e “ « nomer2 « “ ne4etno?: “ « bukvi;
return 0;
}
```

резултата е:

```
21325 e ne4etno
14340 e 4etno
Dali e 14340 ne4etno?: N
```

Тук в този пример оценяваме стойностите на две целочислени променливи,за да се определи дали те са четни или нечетни числа.

Операторът sizeof

sizeof **е също и ключова дума, той е оператор по време на компилация, използван за определяне размера в байтове на променлива или типове данни включително класове , структури и обединения. Ако се използва с тип , името на типа трябва да бъде затворено с кръгли скоби .**
За повечето 32 – битови компилатори следващия пример отпечатва 4 :

```
int x ;
cout << sizeof x ;
```

Операторът *sizeof* връща целочислена ст-ст,която представлява броя байтове! Броят байтове,заделени за типовете данни зависи от реализацията- затова изходът от програмата,която съставяте ще е различен това зависи от системата Ви:)

Операторите си имат приоритет: знаете,че в математиката оператора за умножение * има по-висок приоритет(статут) от оператора за събиране +
И в програмния език C++ е така... примерно в израза: a=9+c*6
приоритета на операторите определя дали първо да бъде извършено събирането на числата или умножението им:) ето сега ще изброя приоритетите на операторите,като ще започна от тези с най-високият приоритет:

<u>оператор</u>

( ) извикване на функция [ ] индекс на масива
-> указател за клас
! логическото НЕ sizeof размер
++ нарастване(инкремент) - - декремент
+ положителен знак - отрицателен знак
*** указател & адрес**
*** умножение / деление % деление по модул**
+ събиране - изваждане
< по-малко от <= по-малко или равно на
> по-голямо от >= по-голямо или равно
= = равенство != неравенство
&& логическо И
|| логическо ИЛИ (OR)
?: условен оператор
= += -+ *= /= %= операторите за присвояване
, запетайката е най-ниския:)

спазвайте ги!

Има и побитови оператори: те се използват за извършване на двоична аритметика!
C и C++ доставят оператори,които действат върху действителните битове, които съдържа стойността . Побитовите оператори могат да бъдат използвани само върху целочислени типове . Побитовите оператори са :

Оператор	Значение
&	AND
|	OR
^	XOR
~	Единствено допълнение
>>	Преместване надясно
<<	Преместване наляво

Таблицата за истинност на & , | и ^ е :

p	q	p&q	p|q	p^q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Тези правила се прилагат към всеки бит във всеки операнд, когато побитовите AND , OR или XOR се изпълнят . Пример за побитова AND операция е показан тук :

```
  0 1 0 0 1 1 0 1
& 0 0 1 1 1 0 1 1
  0 0 0 0 1 0 0 1
```

Побитовата OR операция изглежда така :

```
  0 1 0 0 1 1 0 1
| 0 0 1 1 1 0 1 1
  0 1 1 1 1 1 1 1
```

Побитова XOR операция е показана тук :

```
  0 1 0 0 1 1 0 1
^ 0 0 1 1 1 0 1 1
  0 1 1 1 0 1 1 0
```

Операторът за единствено допълнение е: ~

Операторът ~ ще инвертира всички битове в неговия операнд . Например ако символната променлива *ch* има следната схема :

0 0 1 1 1 0 0 1

тогава

ch = ~ch ;

поставя битовия шаблон

1 1 0 0 0 1 1 0

в *ch* .

Преместващите оператори >> и <<

Десния (>>) и левия (<<) преместващи оператори преместват всички битове във целочислена стойност със зададени позиции . Когато битовете се преместват с 1 назад , се прибавя 0 в другия край (ако стойността, която се премества е отрицателна и се изпълни дясно преместване , тогава се поставят 1 отпред, за да се запази знзка) . Числото от дясната страна на преместващия оператор определя броя на преместващите позиции . Общата форма на всеки преместващ оператор е :

```
стойност >> число
стойност << число
```

Тук число определя броя на позициите на които се премества стойността . Подавайки битовия шаблон (и приемайки неозначена стойност) :

0 0 1 1 1 1 0 1

при преместване надясно се получава :

0 0 0 1 1 1 1 0

а при преместване наляво :

0 1 1 1 1 0 1 0

Преместването надясно е в действителност деление на 2 , а наляво е умножение по 2 . За много компютри, преместването е по – бързо от умножението или делението . Ако ви трябва бърз начин да умножите или разделите на 2 , разгледайте употребата на преместващите оператори . Например следващия фрагмент от код първо ще умножи и след това ще раздели стойността в *x* на 2 :

```
int x ;
x = x << 1 ;
x = x >> 1;
```

Разбира се, когато използвате преместващите оператори,за да изпълните умножение , то трябва да внимавате да не преместите битовете извън края:)

Приоритета на побитовите оператори е:

Приоритет	Оператор
Най-висок	~
	>> <<
	&
	^
Най-нисък	|

Указателни оператори

Двата указателни оператора са * и &. Указател е променлива,която съдържа адреса на друг обект . Или представено различно-променлива,която съдържа адреса на друг обект се казва че сочи към този обект.

Указателен оператор &

Оператора & връща адреса на обекта,който предхожда . Например: ако цялото число *x* е разположено в паметта на адрес 1000 ,тогава

p = & x ;

поставя стойността 1000 в *p.* & може да бъде означен като „адреса на... “
Например предишния израз може да бъде прочетен като “постави адреса на *x* във *p* “

Оператора ?

Оператора ? е тернарен оператор (работи със три операнда) .Има следната обща форма :

израз1 ? израз 2 : израз 3;

Ако израз1 е *true* , тогава резултата от операцията е израз2 , в противен случай е стойността на израз3 .

? се използва често да замества *if - else* условието от този общ вид:

***if* (израз1) променлива = израз2 ;**
***else* променлива = израз2 ;**

Например поредицата

```
if ( y < 10 )  x = 20 ;
else  x = 40 ;
```

Може да бъде написана така :

```
x = ( y < 10 ) ? 20 : 40 ;
```

Тук *x* присвоява стойност 20 ,ако *y* е по – малко от 10 и 40 ,ако това не е така . Една причина ,че оператора ? съществува освен спестяването на писане е ,че компилатора може да произведе много бърз код за това условие - много по-бързо отколкото за подобни *if – else* изрази.

Оператори за членове

Оператора . (точка) и оператора -> (стрелка) се използват за обръщане към отделните членове на класове , структури и обединения. Операторът точка е приложен към действителния обект . Операторът стрелка се използва с указател към обект . Например подавайки следната структура :

```
struct  date_time  {
        char  date[ 16 ] ;
        int  time ;
} tm ;
```

за присвояване на стойноста *“3 / 12 / 99 “* на *date* член от обект *tm* вие трябва да напишете :

```
strcpy ( tm.date , “3 / 12 / 99 “ ) ;
```

Обаче,ако *p_tm* е указател към обект от тип *date_time* се използва следния израз:

```
strcpy ( p_tm -> date , “3 / 12 / 99 “ ) ;
```

да не Ви отегчавам и плаша сега:) по-надолу ще разгледаме работата с класове:)

Оператора запетайка

Оператора запетайка предизвиква да бъде изпълнена поредица от операции . Стойността на целия израз със запетайки е стойността на последния израз от списъка . Например след изпълнението на следния фрагмент :

```
y = 15 ;
```

x = (y = y – 5 , 50 / y) ;

x ще има стойност 5 понеже първоначалната стойност на *y* от 15 се намалява със 5 и тогава се разделя на 50 получавайки като резултат 5. Можете да смятате оператора запетайка като "направи това и това ".Оператора се използва често във изразите *for* . Например :

for (z = 10 , b = 20 ; z < b ; z++ , b --) { // . . .

Тук *z* и *b* се инициализират и изменят чрез използване на разделени със запетайки изрази .

Операторът за определяне област на видимост е ::

Оператора за резрешаване на интервал :: определя интервала към който принадлежат членовете. Той има следната обща форма :

име :: име_на_член

Тук име е името на класа или именованото пространство,което съдържа члена определен от име_на_член . Казано различно: име определя интервала,в който може да бъде открит идентификатора определен от име_на_член . За обръщане към глобален интервал не трябва да определяте име на интервал . Например за обръщане към глобална променлива наречена *count* , която ще бъде скрита от локална променлива count т рябва да използвате този израз :

:: count

Оператора за разрешаване на интервал не се поддържа от C .

В C++ операторите могат да бъдат предефинирани чрез използване на ключовата дума *operator*

Ключовата дума *operator* се използва за създаване на предефинирана функция оператор . Операторните функции имат две разновидности : член и нечлен . Общата форма на операторна член функция е :

```
връщан- тип    име_на_клас :: operator# (списък- с – параметри )
 {
  // . . .
 }
```

Тук връщан – тип е връщания от функцията тип , име_на_клас е името на класа , за който се предефинира оператора и # е оператора който се предефинира. Когато предефинирате унарен оператор , списък – с – параметри е празен (оператора е подаден косвено в *this*)
Когато предефинирате бинарен оператор списък – с – параметри определя операнда от дясната страна на оператора (оператора от лявата страна е подаден косвено в *this*) За нечлен функции , операторната функция има тази обща форма :

```
връщан – тип  operator# ( списък – с – параметри )
  {
  // . . .
}
```

4.Създаване на конструкции:

Конструкциите в програмирането на C++ се използват за описване на хода на изпълнението на програма Те дефинират различните цикли в кода...

Сега ще разгледаме условната конструкция if
Ключовата дума if се използва за извършване на основната условна проверка, която оценява даден израз и връща булева ст-ст истина или неистина...конструкциите след израза за оценяване ще бъдат изпълнени само ако израза е истина.

Синтаксисът на конструкцията if изглежда по следния начин:

```
if (условен израз)
  {  и вече тук ни е кода за изпълнението ако е разбира се истина
   }
```

Кодът,който трябва да бъде изпълнен може да бъде съставен от множество конструкции,поставени в скобките (фигурни скобки ги наричахме нали не сте забравили:)
обаче всяка една от тях ще трябва да завършва с точка и запетайка

```
{
  cout « “dara bara\n“;        //ето завършва с точка и запетая:)
  cout « “dara bara\n“;
  cout « “dara bara\n“;
 }
```

и сега едно примерче,което обикновено се дава за условен израз:

```
#include <iostream>
using namespace std;

int main()
    {
       if ( 8 > 5 )
       {
          cout « “dam, 8 e po-goliamo ot 5 taka si e mamka mu\n“;
          cout « “no ti si go znae6e nali? \n“;
        }
        return 0;
    }
```

вече знаете какво ще изведе кода:)така,че е излишно да се дават картинки от терминала=команд промпта:)
какво направихме? Ами написахме програмен код,в който условния израз оценява дали едно число е по-голямо от друго:) И ако първото число е по-голямо от второто,то израза ще бъде истина и => трябва да бъдат изпълнени конструкциите във фигурните скобки... Ако израза е неистина , то тогава конструкциите след условния израз няма да бъдат изпълнени и програмата преминава към следващата условна проверка в кода ни...

Условните конструкции if могат да бъдат влагани в други блокове от конструкции if за проверка на множество условия...когато оценявате множество условни изрази трябва да поставяте всеки израз в скоби!
Ето примерче:

```
#include <iostream>
using namespace std;

int main()
 {
       if ( 8 > 5 )
       {
```

```
            if ( 'E' == 'E' )           //  == равно нали помните знака за равенство:)
            {
              if ( 4 != 7 )              //   !=   различно от
                {
                    cout « “mdam, 8 e po-goliamo ot 5 \n“;
                    cout « “ E si e ravno na E\n“;
                    cout « “a 4 si e napravo razli4no ot 7 \n“;
                 }
            }
        }
         return 0;
 }
```

проследете затварянето на конструкциите(скобките как са:)
ясно е какво ще се изведе при истинност на 3-те израза:)

Друг метод за оценяване на множество изрази в една конструкция if става посредством логическия оператор && (И)
Условните изрази,които имахме в примерчето стават ето така:

```
        if ( ( 8 > 5 )  && ( 'E'=='E' )  &&  ( 4 != 7 )
          {
           cout..... и квото беше:)
          }
```

С конструкцията if може да се използва и ключовата дума else
това се прави,когато се задава алтернативен код,който да бъде изпълнен,когато израза обаче е <u>неистина.</u> Наричат го условно разклоняване:)
програмката ни ще поеме по друг път според резултата от оценката на условния израз

Синтаксисът на блоковата конструкция if else е следният:

if (условен израз) { направи това ако е истина } else { направи това }

примерче:

```
#include <iostream.h>
using namespace std;
     int main()
{
   int x = 3;
   int y = 4;

  if (x > y) {
    cout << "x is bigger than y" << endl;    // x е по-голямо от y   (свиквайте и на английски да изписвате!)
  }
  else {
    cout << "x is smaller than y" << endl;  //x е по-малко от y
  }
  return 0;
}
```

резултата е:
x is smaller than y

Условното разклоняване(както го нарекохме) с дълги конструкции if else може да бъде извършено по-ефективно с помощта на конструкцията switch
при оценяване на някакво цяло число например

switch взема зададената целочислена ст-ст и търси съвпадаща ст-ст сред няколко конструкции case и ако има съвпадаща конструкция case ,то тогава ще бъде изпълнен асоциирания с нея код...
обаче ако няма съвпадение,то тогава ще бъде изпълнен кодът на конструкция default

Всяка конструкция case трябва да завършва с конструкцията break , която от своя страна пречи на програмата да продължи да търси в блока switch

Общата форма на израза е :

```
switch ( израз )
    {
      case константа 1 : последователност – от – изрази 1 ;
           break ;
       case константа 2 : последователност – от – изрази 2 ;
           break ;
       .
       .
       .
       case константа N : последователност – от – изрази N ;
           break ;
       default : изрази- по – подразбиране ;
      }
```

Всяка последователност от изрази може да бъде от един или няколко израза . Частта *default* е опционна . И двете израз и *case* константи трябва да са целочислени типове.
s*witch* работи чрез проверяване на израз с константи . Ако е открито съвпадение се изпълнява тази последователност . Ако последователността от изрази свързани със съвпадащото *case* не съдържа *break* , изпълнението ще продължи в следващия *case* . Казано различно: от точката на съвпадение , изпълнението ще продължи докато или не се срещне израз *break* или *switch* не завърши . Ако няма съвпадение и случая *default* съществува , се изпълнява неговата последователност от изрази. В противен случай не се изпълнява нищо.

Пример:

```
#include <iostream>
using namespace std;

int main ()
  {
char bukva;
cout « “vavedete koqto i da e bukva ot klaviaturata : “;
cin » bukva;

switch (bukva)
{
    case 'a' : cout « “bukvata \'a\' e namerena\n“; break;
    case 'b' : cout « “bukvata \'b\' e namerena\n“; break;
    case 'c' : cout « “bukvata \'c\' e namerena\n“; break;
     default   :  cout « “bukvata ne e  a, b ili c\n“;
}
return 0;
}
```

изходът от кода е:

```
vavedete koqto i da e bukva ot klaviaturata : c
bukvata 'c'  e namerena
```

Цикъл:

Цикълът е код в програмата,който автоматично се повтаря!
Пълното изпълнение на всичките конструкции в цикъла се нарича преминаване през цикъла!
Идеята на циклите е: да можем да кажем „върни се към стъпка 1 или 2 ,или 3.....“

Дължината на цикъла се контролира от условен израз като стойността му се проверява в цикъл...
и докато условният израз е истина ,то цикъла ще продължава да се изпълнява , а когато условния израз стане неистина ,то тогава цикъла ще завърши!

В програмен език C++ има три основни типа цикли: това са
for , while и do-while
най-често се ползва цикъла for

синтаксисът му е следния:

for (инициализатор ; условен израз ; инкремент) { конструкциите }

Инициализаторът се ползва за задаване на началната ст-ст за брояча на итерациите(повторения), извършени от цикъла. Използва се целочислена ст-ст. След всяка итерация на цикъла условния израз се оценява и итерирането продължава докато този израз е истина! И когато стане неистина цикъла веднага ще приключи... при всяка итерация броячът се увеличава с единица...после се изпълняват конструкциите в цикъла...

Примерът,който винаги се дава на начинаещите е:
Да се изведат всички числа между 1 и 1000.
И тук сега,ако трябва да правим така,че да извеждаме всяко едно число поотделно,то по-добре е да не се захващаме с програмиране:) ние ще направим следното,което прави всеки уважаващ себе си програмист:)
Ще зададем общ модел за действие
почваме да пишем цикъл:) вместо 1000 действия ще направим 5:)

1. **на i ще дадем ст-ст 1**
2. **i по-голямо ли е от 1000? ако е така,то тогава излез**
3. **изведи i**
4. **прибави 1 към i**
5. **върни се на стъпка 2**

пример:

```
#include <iostream>
using namespace std;

int main ()
  {
 int i;
for ( i = 0; i < 4; i++ )
     {
cout « “iteracia “ « i « endl;
}
return 0;
}
```

тук демонстрирам цикъл for ,който извежда номера на текущата итерация при всяко преминаване и като стигне брояча до 4 условният ни израз става неистина,и цикъла прекратява своето испълнение...

изхода на програмката е:

```
iteracia 0
iteracia 1
iteracia 2
iteracia 3
```

Друг тип цикли използват ключова дума while
като израза трябва да бъде оценен дали е истина или неистина...и ако е истина се изпълняват конструкциите във фигурните скобки след него...а след като бъдат изпълнени отново се оценява условния израз и цикъла се изпълнява пак,докато условния израз не стане неистина...

ВАЖНО: БЛОКЪТ ОТ КОНСТРУКЦИИ НА ЦИКЪЛА ТРЯБВА ДА СЪДЪРЖА КОД,КОЙТО ДА ПРОМЕНИ УСЛОВНИЯ ИЗРАЗ,ТАКА ЧЕ ДА БЪДЕ ОЦЕНЕН КАТО НЕИСТИНА! ИНАЧЕ СЕ СЪЗДАВА ТАКА НАРЕЧЕНИЯ БЕЗКРАЕН ЦИКЪЛ,КОЙТО БЛОКИРА ОБИКНОВЕНО СИСТЕМАТА

Цикълът do-while е вариант на цикъла while
Но често пъти цикъла while е по-подходящ от do-while понеже неговите конструкции не се изпълняват автоматично при първата итерация.
Ако израза е истина,цикъла продължава нататък след ключовата дума do ,докато условният израз бъде оценен като неистина и тогава цикъла завършва...

За разлика от цикъла while , конструкциите в цикъла do-while винаги се изпълняват поне веднъж,понеже условния израз се оценява чак в края на цикъла...

Конструкции <u>break</u> и <u>continue</u>

Конструкцията break моментално прекратява изпълнението на цикъла- не се извършват повече итерации...доста честа програмистка грешка е забравяне слагането на конструкция break:))))))
по-скоро неправилното и поставяне...
ползва се за излизане от произволен цикъл!
break може да бъде включена в блока от конструкции на произволен цикъл,предшестван от условна проверка...когато тази проверка върне ст-ст истина ,то break моментално прекратява изпълнението на цикъла и повече итерации няма да се извършват

Ключовата дума continue в C++ се ползва за прекъсване на изпълнението на цикъл,обаче има разлика спрямо ключовата дума break и това е,че continue спира изпълнението само на текущата итерация на цикъла!
continue е предшествана от условен израз в блока от конструкции на цикъла...
когато условния израз е истина,текущата итерация се прекратява веднага,а после започва и следващата
Броячът на цикъла трябва да бъде променен <u>преди</u> достигане на условния израз за конструкция continue ...така се избягва създаване на безкраен цикъл!!!

Конструкцията *goto*

Навсякъде в книгите ще я срещате под наименованието „злощастна конструкция“ ха-ха:)
няма нищо злощастно в нея-има тъпи,злощастни програмисти!!!
Ключовата дума си съществува откакто го има програмирането=информатиката...
Тя е много мощна запазена дума! Стига да знае програмиста как да използва конструкцията!
Просто се злоупотребява с нея и се създаваше нечетлив объркващ код...програмите прескачат от място на място по непонятен начин:)
Ключовата дума *goto* предизвиква изпълнението на програмата да прескочи до етикет определен във израза *goto* . Формата на *goto* е :

```
goto етикет ;
.
.
.
етикет :
```

Всички етикети трябва да завършват с двоеточие и не трябва да влизат в конфликт с ключови думи или имена на функции ! Освен това *goto* може само да прескача в текущата функция , а не от една функция към друга...

Добре де,не я използвайте:)))))))

Макар *goto* да е изхвърлен от употреба като метод за програмен контрол , понякога има употреба Едно от приложенията му е като начин за изход от дълбоко вложени цикли.
Например разгледайте този фрагмент :

```
int  i , j , k ;
int  stop = 0 ;
for ( i = 0 ; i < 100  &&  !stop ; i++ )  {
    for ( j = 0 ; j < 10  &&  !stop ; j++ )  {
        for ( k = 0 ; k < 20 ;  k++ )  {
           //. . .
         if ( something ( ) )  {
              stop = 1 ;
              break ;
          }
        }
    }
 }
```

Както можете да видите променливата *stop* се използва за отказ от два външни цикъла,ако се случи някакво програмно събитие. Обаче по–добър начин да изпълните това е показан тук чрез използване на *goto* :

```
int  i , j , k ;
for ( I = 0 ; I < 100 ; I++ )  {
    for ( j = 0 ; j < 10  ; j++ )  {
        for ( k = 0 ; k < 20 ;  k++ )  {
           //. . .
         if ( something ( ) )  {
            goto  done ;
          }
        }
    }
 }
done :   // . . .
```

Както можете да видите, употребата на *goto* елиминира свръхпретрупването,което се получава чрез добавянето на повтарящи се проверки на *stop* (предишната версия). Докато употребата на *goto* като форма с общо предназначение за контрол на цикъл трябва да бъде избягвано , понякога може да бъде ангажирана с голям успех!

Или още по-простичко казано ключовата дума goto позволява на програмата да прескача на етикети на други места в програмата,точно както една хипервръзка(links) прави това в уеб страницата...

5.Низове(Strings):

Променливи от тип *string*

Знаковите низове трябва винаги да бъдат поставяни в кавички!

В C++ класът <string> предоставя методи за обработка на текстови низове...и тези методи се правят достъпни като се добави този клас в началото на кода чрез директивата #include
и вече можем да си декларираме променлива от тип string
и също така да си ги съхраняваме тези текстови низове...
Програмен език C++ осигурява два начина за работа с низове.

Първият е да се ползва завършващ с null масив от символи, а вторият метод е като се ползват обекти на класа string. Класът string представлява версия на един по-общ шаблонен клас,наречен basic_string. Класът basic_string има две версии: string , който поддържа 8 битови символни низове и wstring, който поддържа широки символни низове. 8 битовите са най-разпространените символи в програмирането. Класът string е доста обемен, с много конструктори и член-функции...цял преглед на класа няма да правя:) накратко ще обясня:
накои от конструкторите на класа string са:

```
string();
string(const char *str);
string(const string &str);
```

първата форма създава празен низ обект,втората създава стринг обект с помощта на завършен с null низ указан е от str...третата форма създава стринг обект от друг такъв обект.

Прилагат се за string обекти операторите за присвояване(=),за кокатенация(+), равно(==),различно(!=), по-малко(<),по-малко или равно(<=),по-голямо(>),индексиране([ ]), изход(<<) , вход (>>)

Тези оператори позволяват използването на обекти на string в обикновени изрази и премахват нуждата от извикването на функции като strcpy() , strcat()

Операторът + се ползва за добавяне на символ към края на string обект.
За да се присвои част от един низ на друг се ползва функцията assign()

```
string &assign(const char *низа , size_type брой);
```

Може да се вмъкнат или заменят символи в даден низ с помощта на insert() или replace()

```
string&insert (size_type начало,const string &обект);
string&replace(size_type начало, const string&обект);
```

с инсърт вмъкваме обект в извикващия низ от позицията,указана с начало
с риплейс заменяме брой символа от извикващия обект с обект

Може да се изтрие символ/и от един низ,като се ползва функцията erase()
ето формата й:

```
string &erase(size_type начало = 0, size_type брой  = npos);
```

функцията изтрива брой символи от извикващия низ,започвайки от начало.Връща се псевдоним на извикващия низ.
Има няколко член-функции,които служат за претърсване на низ.Това са find() и rfind()

```
size_type find(const string &обект ,size_type начало = 0) const;
size_type rfind(const string &обект , size_type начало = npos) const;
```

Започвайки от начало find() претърсва извикващия низ до първото срещане на низа,който се съдържа с обект.Ако търсения низ е намерен find() връща позицията,в която в открито съвпадението.
При rfind() е обратното...намира последното срещане на обект в извикващия низ.

За сравняване на част от един низ с друг се ползва член-функцията compare()

```
int compare(size_type начало ,size_type брой ,const string &обект) const;
```

В нея брой символи от обект,започващи от начало се сравняват с извикващия низ.

Пример на класа string:

```
#include <iostream>
#include <string>  //ето включваме си го:)
using namespace std;

int main()
{
   string str1 (“demonstrirane na nizove“);
   string str2 (“niz 2“);
   string str3;

// присвояваме низ
   str3 = str1;
   cout « str1 « “\n“ « str3 « “\n“;

// конкатенация на два низа
   str3 = str1 + str2;
 cout « str3 « “\n“;

//сравняваме низовете
   if (str3 > str1) cout « “str3 > str1\n“;
   if (str3 == str1+str2)
 cout « “str3 == str1+str2\n“;

// присвояване на обикновен низ
 str1 = “obiknoven niz \n“;
cout « str1;

//създава се стринг обект с помощта на друг обект
string str4(str1);
cout « str4;

// приемане на стринг обект като вход
cout « “vavedete string: “;
cin » str4;
cout « str4;

   return 0;
}
```

Изходът от програмката ни е:

```
demonstrirane na nizove
demonstrirane na nizove
demonstrirane na nizove niz 2
str3 > str1
str3 == str1+str2
obiknoven niz
obiknoven niz
vavedete string: test
test
```

няма нужда да указваме размерите на низовете...string обекта автоматично си променя размера,за да поберат размера на присвоения низ

друг пример:

```
#include <string>   //дали ще го включим преди или след iostream.h значение няма:)
#include <iostream>
using namespace std;

int main()
{
  string str1(“dobre doshli“);
  string str2 = “v sveta na“;
  string str3;

  str3 = “ programiraneto “;

cout « “str1: « str1 « endl;
cout « “str2: « str2« endl;
cout « “str3: « str3 « endl;

return 0;
}
```

изходът е:

```
str1: dobre doshli
str2: v sveta na
str3: programiraneto
```

друг пример:

```
#include <string>
#include <iostream>
using namespace std;

int main()
{
//ще декларираме 3 низови променливи
string str1;
string str2;
string str3;
cout « “vavedete si imeto: “;
cin » str1;  // присвоява се низа с името на str1
cout « “blagodaria“ « str1 « endl;
cout « “vavedete familnoto si ime: “ « str2;
cin » str2;        // присвояване  фамилното име на str1
str3 = str1;       // присвояване на името на str3
cout « “dobre doshal gospodariu“ « str3 « “ “ « str2 « endl;
return 0;
}
```

изходът смятам е ясен:)

има една особеност и тя е,че по този начин с функцията in ние няма да можем да въведем цяло изречение понеже при въвеждане на интервал ,то низът ще свърши...просто спира да чете въведените данни при достигането на интервал:)

затова с функцията getline ще четем потока от входни данни...
тя чете докато попадне на знака за нов ред \n
тя вкючва интервалите и затова може да бъде използвана за присвояване на низове с интервали

изглежда ето така:
getline (cin, str1) ;
просто стандартния вход cin се поставя в скобките,а вторият аргумент е името на променливата от тип string и там ще бъде съхранен входния низ...
може да се сложи и трети аргумент вътре в скобките той да бъде разделител и при достигането му функцията getline ще спре...
getline (cin, str1, '\t'); // като разделитет съм сложил знака за табулация \t (Tab)

метода за проверката на празни низове се извършва с empty, който връща истина (1) или неистина (0)
този метод е полезен при искане на входни данни,като се въвежда поне един знак,преди програмката да продължи...
пример:

```
#include <string>
#include <iostream>
using namespace std;

int main()
{
string ime;
while (ime.empty() )
{
cout « “vavedete ime: “;
getline (cin, ime) ;
}
cout « “oki“ « ime « endl;
return 0;
}
```

изходът от програмката е:

```
vavedete ime:
vavedete ime:
vavedete ime: brigante
oki brigante
```

Операторът за събиране + или += се ползват за конкатенация на два низа в един дълъг низ...
може да се ползва и метода append

Два низа могат да се сравняват с помощта на оператора за равенство = = или за неравенство != и с метода compare

Съдържанието на една низова променлива може да бъде копирано в друга низова променлива чрез обикновено присвояване посредством оператора за присвояване =
Класът <string> предоставя метода assign, който може да се ползва за тази цел.

Класът <string> предоставя метода swap, който може да бъде използван за размяна на съдържанието на две низови променливи.

С помощта на метода find на класа <string> може да се претърси даден низ,за да се установи дали той съдържа конкретен подниз.За да се претърси цял низ,то трябва да се зададе нулев индексен елемент като начална точка.Ако подниза бъде намерен,то тогава метода find връща индексния номер на първото срещане на първия знак на подниза в претърсвания низ.Ако търсенето не намери такъв подниз,метода връща ст-ст string::npos
Индексите на знаковете в низ започват от нула,а не от едно!
Има и други методи на класа <string> свързани с find. Те са: find_first_of и find_first_not_of

find намира първото срещане на конкретен низ
find_first_of намира първото срещане на някой от знаковете в зададен низ
find_first_not_of намира първото срещане на знак,който не принадлежи на зададения низ

А методите find_last_of и find_last_not_of работят по същия начин,но те започват да търсят от края на низа и преминават напред!

А с помощта на метода insert на класа <string> един низ може да бъде вмъкнат в друг низ.

Противоположният метод на метода insert е метода erase,който се ползва за премахване на конкретни части от низ.Първият му аргумент задава индексната позиция,от която да започне изтриването.
А втория аргумент задава общия брой знакове,които да бъдат премахнати след началната точка.
Методът replace комбинира методите erase и insert в една единствена операция.Той задава началната точка и броя знакове,които да бъдат премахнати,но има и трети аргумент,който бива вмъкнат след премахването на подниза.

С помощта на метода at на класа <string> могат да бъдат извличани отделни знакове от низ.
Последният знак в низ винаги има индексен номер,с единица по-малък от размера на низа понеже индексното номериране започва от нула,както вече споменах.

Следващата програма демонстрира функциите insert(), erase( ) и replace( )

```
#include <iostream>
#include <string>
using namespace std;

int main( )
{
string str1("This is a test");
string str2("ABCDEFG");
cout << "Initial strings:\n";
cout << "str1: " << str1 << endl;
cout << "str2: " << str2 << "\n\n";   // слагаме две \n\n за прилежност на програмката:)през един ред ще е

// демонстрира insert()
cout << "Insert str2 into strl:\n";
strl.insert(5, str2);
cout << strl << "\n\n";

// демонстрира erase()
cout << "Remove 7 characters from strl: \n";
str1.erase(5, 7) ;
cout << strl <<"\n\n";

// демонстрира replace
cout << "Replace 2 characters in strl with str2: \n";
str1. replace (5, 2, str2) ;
cout << str1 << endl ;

return 0;
}
```

Изходът от програмата е :

```
Initial strings:
strl: This is a test
str2: ABCDEFG

Insert str2 into strl:
This ABCDEFGis a test

Remove 7 characters from strl:
This is a test

Replace 2 characters in str1 with str2:
This ABCDEFG a test
```

6. Четене и запис на файлове

Създаване и променяне съдържанието на текстови файлове.

Програмите могат да записват във файл и да четат информацията във файла от твърдия диск(хард диск)...
ползва се класът <fstream> file stream
този клас съдържа методи за работа с файловете...
добавяме си го към нашата си програмка с директивата #include
и става примерно:

```
#include <iostream>
#include <string>
#include <fstream>    //ето добавихме си го:)
using namespace std;
.....................................
```

За всеки файл,който ще бъде отворен,трябва предварително да се създаде файлов обект.
Той представлява ofstream обект за запис на данни във файла или ifstream обект за четене на информация от него.Всеки ofstream обект се използва по същия начин както функцията cout апък ifstream обектите те се използват подобно на функцията cin (тя чете от стандартния вход)

Синтаксисът е: ofstream obekt(“fail4eto mi.txt“);

-синтаксисът за създаване на файлов обект за запис на данни включва ключовата дума ofstream после следва интервал и името на файловия обект и след това в кръгли скоби се поставя наименованието на текстовия файл.Имената на файловете се поставят в кавички!

Правилният синтаксис за създаването на изходен файлов обект с име obekt , който записва данни във файла fail4eto mi.txt е горния пример:)

Може да си зададем и целия път до файла: “D:\NewFolder\fail4eto mi.txt“
надявам се знаете,че NewFolder не означава папка:) запомнете,че това са си ДИРЕКТОРИИ !!!

Ако не сме въвели пътя до файла програмата ще си го търси в директорията,в която тя се намира!!!

Преди да запише данни във файл програмата винаги първо проверява дали файловият обект е създаден като се ползва конструкция if
и ако проверката си даде положителен резултат,то програмата записва данни в зададения файл...
ако съществува същият такъв файл,то той ще бъде заместен от новия...засега без предупреждение:)

Процесът след всичко това ще си завърши и програмата ще си затвори файла като ползва метода close

Сега примерче:

```
#include <fstream>
#include <string>
#include <iostream>
using namespace std;

int main()
{
   string str = “\n\Programiraneto na c++ ne e lesno,“;
   str.append(“\n\tno s tarpenie i uporitost,i mnogo pisane na kod“);
   str.append(“\n\tsled vreme se izrastva,i na 4ovek mu se izqsnqvat ne6tata.“);
   str.append(“\n\tTo e qsno,4e programisti nqma da vi napravq,no pone 6te vi vaveda v sveta na c++ :)“);

   ofstream File(“moiatfile.txt“);
   if (! File)        // сега проверяваме дали файла е отворен
    {
      cout « “Gre6ka pri otvarqne na fail“ « endl;
```

```
        return -1;
      }
    File « str « endl;        // запис на данни във файла
    File.close ( );           //  затваряне на файла

return 0;
}
```

тук демонстрирам създаването на изходен файлов обект с име File,който създава текстов файл нарекли сме го moiatfile.txt .... не сме указали точният път....значи ще го търси в нейната си директория сложихме табулации,които ще се запазят в създадения текстови файл,който ще се отвори с текстови редактор (Notepad)

Сега да добавим данни към файл:
-при задаването на файлов обект в кръглите скобки след името му по избор могат да се добавят допълнителни аргументи определящи файлови режими конктролиращи поведението на съответния файлов обект...

режим	операция
ios::out	отваряне на файл за запис на данни
ios::in	отваряне на файл за четене на данни
ios::app	отваряне на файл за добавяне на данни в края на съдържанието му
ios::trunc	запис на новите данни на мястото на досегашните във файла (по подразбиране е)
ios::ate	отваряне на файл без да се записват данни на мястото на старите(новата инфо е на произволно място)
ios::binary	третиране на файла като двоичен,а не като текстов-данните могат да се съхраняват в различен от текстовия формат

задаване на няколко режима се прави с разделяне с вертикална черта |

синтаксисът за отваряне на файл за запис на данни в двоичен вид е:

```
ofstream файловият обект  ( "името на обекта ") ,
        ios::out | ios::binary) ;
```

ако не сме задали режима,то по подразбиране файла става текстов!
Често се ползва режимът iso::app при него има гаранция,че съдържанието,което вече си съществува няма да бъде заменено или изтрито автоматично с нови данни...

Четене на знакове от файл:

Ifstream файловият обект притежава метода get, който се използва в цикъл за прочитане на данни от файла. При всяко преминаване през цикъла се присвоява по един знак като стойност на променлива от тип char, която е аргумент на get.

Четене на редове от файл:

По-ефективен начин за прочитане на съдържанието на ifstream файлов обект е да се използва методът getline за прочитане на информацията от текстовия файл ред по ред.

Пример:

```
#include <fstream>
#include <string>
#include <iostream>
using namespace std;

int main()

{
```

```
    string str;
    ifstream myFile ("file.txt") ;
    if (! myFile)          // проверяваме си дали файлът е отворен
    {
       cout << "Error/Грешка" << endl;
       return -1;

    while (! myFile.eof())
    {
       getline(myFile, str);
       cout « str « endl;
    }
myFile.close ();
return 0;
}
```

В този пример ifstream обектът е зададен като първи аргумент на метода, а променливата string, на която се присвоява като стойност всеки отделен ред, е зададена като втори аргумент. След това съдържанието на всеки ред се визуализира в конзолен прозорец като стандартен изход.

Форматиране с помощта на getline

Методът getline може също така да зададе разделител, при който да спира четенето на реда. Това дава възможност за отделяне на текста от подреден списък с данни във файл.

Входно-изходни манипулатори

Дотук използвахме подразбиращите се настройки за форматиране на библиотеката iostream. Освен тях обаче могат да се зададат и други настройки с помощта на входно-изходните манипулатори на C++
ето табличка:

Манипулатор	Операция
boolalpha	Извежда булевите променливи като true или false
noboolalpha	Извежда булевите променливи като 0 или 1
dec (подразбиращ се)	Визуализира целите числа в десетичен вид
hex	Визуализира целите числа в шестнадесетичен вид
oct	Визуализира целите числа в осмичен вид
left	Визуализира текста подравнен вляво
right	Визуализира текста подравнен вдясно
internal	Визуализира знака пред цифрата подравнен вляво, а стойността – вдясно.
noshowbase	Скрива префикса, който указва бройната система
showbase	Визуализира префикса, който указва бройната система
noshowpoint (подразбиращ се)	Визуализира десетична точка само ако числото има дробна част
showpoint	Винаги визуализира десетичната точка
noshowpos (подразбиращ се)	Скрива знака + пред положителните числа
showpos	Визуализира знака + пред положителните числа
sklpws (подразбиращ се)	Позволява празните места (празни места,табулации, нови редове) да бъдат пропуснати от оператора за вход »
noskipws	Не позволява на » да пропуска празните места.
fixed (подразбиращ се)	Визуализира числата с плаваща запетайка във фиксирана нотация
scientific	Визуализира числата с плаваща запетая като 10, повдигнато на степен
nouppercase (подразбиращ се)	Визуализира абревиатурата Ox за шестнадесетичните числа и e за числата,представени като 10 на степен.
uppercase	Визуализира абревиатурата OX за шестнадесетичнитечисла и E за тези представени като 10 на степен

Манипулаторите от таблицата променят състоянието на iostream обекта. Това означава, че веднъж приложени върху него те ще повлияят всички по-нататъшни входни и изходни операции, които се извършват върху него!!!

Манипулаторите от следващата таблица обаче се използват за форматиране на определени изходни данни, но не променят състоянието на обекта:

Манипулатор	**Операция**
setw(w)	**Задава изходната ширина със стойност w (необходимо е да бъде включен класът<iomanip>)**
width(w)	**Задава изходната ширина със стойност w,като се използва метод на класа <iostream>**
setfill(ch)	**Запълва празните места в изходната информация с ch (необходимо е да бъде включен класът <iomanip>)**
fill(ch)	**Запълва празните места в изходната информация отново с ch, но се използва метод на класа <iostream>**
setprecision(n)	**Числата с плаваща десетична запетая се визуализират с точност до n Този формат за визуализиране не повлиява изчисленията.**

Манипулаторът setw е алтернативен на width, а setfill – на fill

---

За достъп до манипулаторите, които приемат параметри такива като *setw ( )* , ще трябва да включите < *iomanip* > в програмата

7. Използване на функции

Основи на функциите...

Функциите съдържат код, който дава специфични възможности на програмата. Когато бъде извикана от програма,съответната функция изпълнява конструкциите си и след това може да върне стойност на програмата...функциите имат предимства:

• те правят програмата по-лесна за разбиране и поддръжка.

• изпробваните функции, за които е сигурно, че работят надеждно, могат да бъдат използвани и в други програми!

Деклариране на функции...

Всяка функция се декларира в началната част на програмния код, като задължително се включва типът на данните, които тя връща като стойност. След името й се поставят кръгли скоби, в които се записват аргументите... синтаксиса изглежда така:

тип на върнатите данни име на функцията (списък с аргументи) ;

Следващият код декларира функция, наречена „name", която няма аргументи и не връща никакии стойности:

```
void name ();
```

Дефиниране на функции...

Дефиницията на функцията се включва по-нататък в кода на програмата, като съдържа повторение на декларацията заедно с тялото на функцията...

Дефиницията на декларираната по-горе функция може да изглежда по следния начин:

```
void  name()
{
cout « "name1" « endl;
}
```

<u>Област на видимост на променливите:</u>

Променливите, които са декларирани в дадена функция, са валидни само за нея и не са достъпни за използване от други функции. Това явление е известно като „област на видимост" на променливите. Целта е в главната функция main да има предимно извиквания на функции...

За да могат да бъдат използвани в програма, функциите първо трябва да бъдат декларирани. Най-често декларациите им се поставят преди главната програма, а дефинициите след нея.

Често функциите получават стойности под формата на аргументи от извикващата програма. Те могат да са от различен тип и с произволен брой, но винаги трябва да съвпадат със зададените в прототипната декларация на функцията. По подобен начин функцията може да връща стойност от произволен тип, стига това да е типът, зададен в прототипа на функцията.

При предаване на аргументи на функция, тя получава само стойностите, а не самите променливи. С други думи, получава копие на оригинала - това се нарича „предаване по стойност".
На аргументите на функции могат да бъдат зададени подразбиращи се стойности при декларирането им. Те ще бъдат използвани стига да не се присвояват други стойности на съответния аргумент в програмата. Такива стойности могат да бъдат зададени на голям брой аргументи, но не бива да се забравя, че те трябва да се поставят в края на списъка след тези, които са без подразбиращи се стойности.Функциите могат да бъдат извиквани не само от главната част на програма, но и от други функции.

Рекурсивни функции:

Функция, чието тяло съдържа извикване към самата нея се нарича рекурсивна. Както при циклите, така и тук трябва да има някаква условна проверка, която да позволява излизане от непрекъснатото изпълнение на функцията.Обикновено рекурсивните функции се изпълняват по-бавно от еквивалентните им цикли, което в повечето случаи ги прави по-малко предпочитани.

Понякога обаче (например при алгоритми) е по-подходящо да се използват именно те, за да не се утежнява твърде много програмният код.

Пример:

```
#include <iostream>
using namespace std;
void recur(int num);
int main()
{
recur(0);
return 0;
}
void recur(int num)
{
cout « "Red4e " << num « endl;
num++;
if (num > 10) return;
else recur (num);
}
```

изходът е:

```
Red4e 0
Red4e 1
Red4e 2
Red4e 3
Red4e 4
Red4e 5
Red4e 6
Red4e 7
Red4e 8
Red4e 9
Red4e 10
```

Рекурсивните функции натоварват повече системните ресурси!

Предефиниране на функции:

Предефинирането предлага начин за получаване на различни функции с едно и също име. Аргументите на всяка от тях трябва да се различават по брой, тип, или и по двете...
Компилаторът ще стартира правилната функция в зависимост от аргументите, зададени при извикването.Този процес е известен като разделяне на функциите.

Инлайн функции:

При всяко извикване на функция, програмата веднага се премества към местоположението й, изпълнява нейните конструкции и след това се връща обратно на предишното място, за да продължи хода си. Това постоянно прескачане забавя програмата и може да бъде избегнато чрез добавяне на ключовата дума inline при декларирането на функцията. По този начин компилаторът директно поставя кода на функцията на мястото, където е извикана.
Например в следващата програма всяко присвояване на стойност на променливата num ще се компилира като
num =num* num;

```
#include <iostream>
using namespace std;
inline int square(int n);
int main().
{
int num;
cout « "vavedi 4islo: “;
cin » num;
num = square(num);
cout « "Resultat: " « num « endl;
num = square(num);
```

```
cout « "Resultat: " « num « endl;
num = square(num);
cout « "Resultat: " « num « endl;
return 0;
}
int square(int n)
{
return n * n;
}
```

резултата:

```
vavedi 4islo: 5
Resultat: 25
Resultat: 625
Resultat: 390625
```

Използването на инлайн функции е подходящо само в случаите, когато кодът в тялото на функцията е не по-дълъг от 1-2 реда. Многобройните извиквания на по-обемисти инлайн функции ще увеличат значително големината на програмата, тъй като цялото им съдържание заменя всяко извикване на съответната функция.

Стандартни iostream функции

Най – често употребяваните функции в нов стил ще ги опиша тук:

bad ( )

```
# include < iostream >
bool  bad ( )  const ;
```

Функцията *bad ( )* е член на *ios* . Функцията *bad ( )* връща *true* ,ако е станала фатална I/O грешка в свързания поток , в противен случай се връща *false* . Свързана функция е *good ( )*

clear ( )

```
# include < iostream >
void  clear ( iostate  flags = goodbit ) ;
```

Функцията *clear ( )* е член на *ios* . Функцията *clear ( )* изчиства флаговете за състоянието свързани с поток . Ако *flags* е *goodbit* (както е по – подразбиране) тогава всички флагове за грешки се изчистват (установяват се на 0) . В противен случай флаговете ще бъдат установени на стойността на *flags* . Свързана функция е *rdstate ( )*

eof ( )

```
# include < iostream >
bool  eof ( )  const ;
```

Функцията *eof ( )* е член на *ios* . Функцията *eof ( )* връща *true* ,когато е достигнат края на свързания входен файл , в противен случай се връща *false* . Свързани функции са *bad ( ) , fail ( ) , good ( ) , rdstate ( )* и *clear ( )*

fail ( )

```
# include < iostream >
bool  fail ( )  const ;
```

Функцията *fail ( )* е член на *ios* Функцията *fail ( )* връща *true* ако стане I/O грешка в свързания поток , в противен случай връща *false* . Свързани функции са *good ( ) , eof ( ) , bad ( ) , clear ( )* и *rdstate ( )*

fill ( )

```
# include < iostream >
char  fill ( )  const ;
char  fill ( char  ch ) ;
```

Функцията *fill ( )* е член на *ios* . По подразбиране ,когато трябва да бъде запълнено поле , то се запълва със интервали . Обаче можете да промените запълващия символ чрез използване на функцията *fill ( )* и определяйки новия запълващ символ в *ch* . Функцията ще върне стария запълващ символ . За да получите текущия запълващ символ , използвайте първата форма на *fill ( )* , която връща текущия запълващ символ . Свързани функции са *precision ( )* и *width ( )*

flags ( )

```
# include < iostream >
fmtflags  flags ( )  const ;
fmtflags  flags ( fmtflags  f ) ;
```

Функцията *flags ( )* е член на *ios* . Първата форма на *flags ( )* просто връща текущите форматни настройки на флаговете в свързания поток . Втората форма на *flags ( )* установява всички форматни флагове свързани с поток чрез *f* и когато използвате тази версия битовия шаблон открит в *f* се копира във форматните флагове свързани с потока . Тази версия също връща предишните настройки . Свързани функции са *unsetf ( )* и *setf ( )*

flush ( )

```
# include < iostream >
ostream  &flush ( ) ;
```

Функцията *flush ( )* е член на *ostream* . Функцията *flush ( )* предизвиква буфера свързан към определения изходен поток да бъде физически записан на диска . Функцията връща псевдоним към свързания с нея поток . Свързани функции са *put ( )* и *write ( )*

fstream ( ) , ifstream ( ) и ofstream ( )

```
# include < fstream >
fstream ( ) ;
fstream ( const  char  *filename ,
        openmode  mode = ios : : in | ios : : out ) ;
ifstream ( ) ;
ifstream ( const  char  *filename ,
        openmode  mode = ios : : in ) ;
ofstream ( ) ;
ofstream ( const  char  *filename ,
        openmode  mode = ios : : out | ios : : trunc ) ;
```

***fstream ( ) , ifstream ( )* и *ofstream ( )* са конструкторите съответно на класовете *fstream , ifstream* и *ofstream* . Версиите без параметри на *fstream ( ) , ifstream ( )* и *ofstream ( )* създават поток ,който не е свързан със никакъв файл . Този поток може да бъде свързан към файл чрез използване на *open ( )* Версиите на *fstream ( ) , ifstream ( )* и *ofstream ( )* ,които приемат име на файл като първи параметър са най – често използвани в приложните програми . Въпреки ,че те са изцяло предназначени да отварят файл чрез използване на функцията *open ( )* , в повечето случай няма да го правите понеже тези *fstream ( ) , ifstream ( )* и *ofstream ( )* функции – конструктори автоматично отварят файла ,когато потока е създаден . Функциите конструктори имат еднакви параметри и действие като функцията *open ( )* Например това е най – общия начин за отваряне на файл :**

```
ifstream  mystream ( " myfile " ) ;
```

Ако по някаква причина файла на може да бъде отворен , стойността на променливата на свързания поток ще бъде 0 . По тази причина и при използване на функция конструктор и при изрично определяне извикване на *open ( )* , ще трябва да се уверите че файла действително е бил отворен чрез проверка стойността на потока . Типа *openmode* е дефиниран в класа *ios_base* Свързани функции са *close ( )* и *open ( )*

gcount ( )

```
# include < iostream >
streamsize  gcount ( )  const ;
```

Функцията *gcount ( )* е член на *istream* . Функцията *gcount ( )* връща броя на прочетените символи при последната входна операция . Свързани функции са *get ( ) , getline ( )* и *read ( )*

get ( )

```
# include < iostream >
int  get ( ) ;
istream  &get ( char  &ch ) ;
istream  &get ( char  *buf , streamsize  num ) ;
 istream  &get ( char  *buf , streamsize  num , char  delim ) ;
 istream  &get ( streambuf  &buf ) ;
 istream  &get ( streambuf  &buf , char  delim ) ;
```

Функцията *get ( )* е член на *istream* . Най – общо *get ( )* чете символи от входния поток . Формата без параметри на *get ( )* чете единствен символ от свързания поток и връща неговата стойност . *get (char &ch)* прочита символ от свързания поток и слага стойността му в *ch*

Тя връща псевдоним за потока .
get (char *buf , streamsize num) **прочита символите в масив указван от *buf* докато или *num – 1* символа бъдат прочетени , достигнат е нов ред или е срещнат края на файла . Масива указван от *buf* ще бъде нулево – терминиран от *get ( )* Ако е срещнат символа нов ред във входния поток , той не се извлича . Вместо това остава в потока до следващата входна операция . Тази функция връща псевдоним за потока .**
get (char *buf , streamsize num , char delim) **прочита символите в масива указван от *buf* докато или *num – 1* символа бъдат прочетени , символа определен от *delim* бъде открит или е срещнат края на файла . Масива указван от *buf* ще бъде нулево – терминиран от *get ( )* . Ако се срещне разграничителния символ във входния поток , той не се извлича . Вместо това той остава в потока до следващата водна операция . Тази функция връща псевдоним за потока .**
get (streambuf &buf) **чете символи от входния поток във *streambuf* обект . Символите се четят докато се достигне нов ред или се срещне края на файла**
Тази функция връща псевдоним за потока .
get (streambuf &buf , char delim) **чете символи от входния поток във *streambuf* обект . Символите се четат докато се открие символа определен от delim или се срещне края на файла . Тази функция връща псевдоним за потока .**
Тя връща псевдоним за потока . Ако се срещне разграничителния символ във входния поток , той не се извлича .Свързани функции са *put ( ) , read ( )* и *getline ( )*

getline ( )

```
# include < iostream >
istream  &getline ( char  *buf , streamsize  num ) ;
istream  &getline ( char  *buf , streamsize  num , char  delim ) ;
```

Функцията *getline ( )* е член на istream
getline (char *buf , streamsize num) **чете символите в масив указван от *buf* докато или *num – 1* символа бъдат прочетени , достигнат е нов ред или е срещнат края на файла . Масива указван от *buf* ще бъде нулево – терминиран от *getline ( )* . Ако се срещне символ за нов ред във входния поток , той се извлича , но не се слага във *buf* . Тази функция връща псевдоним за потока**
getline (char *buf , streamsize num , char delim) **чете символи в масив указван от *buf* докато или *num – 1* символа бъдат прочетени , символа определен от *delim* бъде открит или е срещнат края на файла . Масива указван от *buf* ще бъде нулево – терминиран от *getline ( )* . Ако се срещне разграничителния символ във входния поток , той се извлича но не се слага във *buf* . Тази функция връща псевдоним за потока . Свързани функции са *get ( )* и *read ( )***

good ( )

```
# include < iostream >
bool  good ( )  const ;
```

Функцията *good ( )* е член на *ios* . Функцията *good ( )* връща *true* ,ако не се срещнат I/O грешки в свързания поток , в противен случай връща *false* . Свързани функции са *bad ( ) , fail ( ) , eof ( ) , clear ( )* и *rdstate ( )*

ignore ( )

```
# include < iostream >
istream  &ignore ( streamsize  num = 1 , int  delim = EOF ) ;
```

Функцията *ignore ( )* е член на *istream* . Вие може да използвате член функцията *ignore ( )* да чете и отхвърля символи от входния поток . Тя чете и отхвърля символи докато или бъдат отхвърлени *num* символа (1 по – подразбиране) или се срещне символа определен от *delim*
Ако се срещне разграничителния символ , той се премахва от входния поток . Функцията връща псевдоним за потока . Свързани функции са *get ( )* и *getline ( )*

open ( )

```
# include < fstream >
void  fstream : : open ( const  char  *filename ,
                openmode  mode = ios : : in | ios : : out ) ;
void  ifstream : : open ( const  char  *filename ,
                openmode  mode = ios : : in ) ;
void  ofstream : : open ( const  char  *filename ,
                openmode  mode = ios : : out | ios : : trunc ) ;
```

Функцията *open ( )* е член на *fstream , ifstream* и *ofstream* . Файл се свързва с поток чрез използване на функцията *open ( )*

Тук *filename* е името на файла ,което може да включва и пътя . Стойността на *mode* определя как да се отвори файла . Тя трябва да бъде една (или повече) от тези стойности :

ios : : app	***ios : : in***
ios : : ate	***ios : : out***
ios : : binary	***ios : : trunc***

Вие можете да комбинирате две или повече от тези стойности като ги оградите заедно в кръгли скоби . Включването на *ios : : app* предизвиква целия изход към файла да бъде добавен в края му . Тази стойност може да бъде използвана само с файлове с възможност за изход . Включването на *ios : : ate* предизвиква търсене към края на файла до съвпадение когато файла е отворен . Въпреки че *ios : : ate* предизвиква отместване към края на файла I/O операциите могат да се извършат навсякъде във файла . Стойността *ios : : binary* предизвиква файла да бъде отворен за двоични I/O операции . По подразбиране файл се отваря в текстов режим . Стойността *ios : : in* определя ,че файла има възможност за вход . Стойността *ios : : out* определя че файла има възможност за изход . Обаче създаването на поток с използване на *ifstream* подразбира вход , а създаването на файл чрез използване на *ofstream* е по подразбиране изход и отварянето на файл с *fstream* означава вход / изход . Във всички случай ,ако *open ( )* пропадне потока ще бъде 0 . По тази причина преди да използвате файл ще трябва да проверите да се уверите че отварящата операция е успешна . Свързани функции са *close ( ) , fstream ( ) , ifstream ( )* и *ofstream ( )*

В библиотеката в стар стил , конструктора *fstream* не съдържа подразбираща се стойност за параметъра *mode* . Така той не отваря автоматично поток за входни и изходни операции . Така когато използвате старата библиотека ,за да отваряте поток за вход / изход и двете стойности *ios : : in* и *ios : : out* трябва да бъдат изрично определени

peek ( )

```
# include < iostream >
int peek ( ) ;
```

Функцията *peek ( )* е член на *istream* . Функцията *peek ( )* връща следващия символ в потока, ако е достигнат края на файла . Ако не е така при всички случай тя премахва символа от потока . Свързана функция е *get ( )*

precision ( )

```
# include < iostream >
streamsize precision ( ) const ;
streamsize precision ( streamsize p ) ;
```

Функцията *precision ( )* е член на *ios* . По подразбиране се показват шест цифри точност ,когато се извежда стойност с плаваща точка . Обаче с използването на втората форма на *precision ( )* ще можете да установите това число на стойността определена в *p* . Връща се първоначалната стойност . Първата версия на *precision ( )* връща текущата стойност . Свързани функции са *width ( )* и *fill ( )*

put ( )

```
# include < iostream >
ostream &put ( char ch ) ;
```

Функцията *put ( )* е член на *ostream* . Функцията *put ( )* записва *ch* в свързания изходен поток . Тя връща псевдоним за потока . Свързани функции са *write ( )* и *get ( )*

putback ( )

```
# include < iostream >
istream &putback ( char ch ) ;
```

Функцията *putback ( )* е член на *istream* . Функцията *putback ( )* връща *ch* в свързания входен поток Свързана функция е *peek ( )*

rdstate ( )

```
# include < iostream >
iostate rdstate ( ) const ;
```

Функцията *rdstate ( )* е член на *ios* . Функцията *rdstate ( )* връща състоянието на свързания поток. Системата за I/O на C++ поддържа статус информация за резултата от всяка I/O операция отнасяща се до всеки активен поток . Текущото състояние на I/O системата се съдържа в обект от тип iostate, в който са дефинирани следните флагове :

Име	**Значение**
goodbit	**Не са станали грешки**

eofbit	Срещнат е края на файла
failbit	Станала е нефатална I/O грешка
badbit	Станала е фатална I/O грешка

Тези флагове са изброени във *ios . rdstate ( )* връща *goodbit* , когато не е станала грешка , в противен случай се установява бита за грешка .
Свързани функции са *eof ( ) , good ( ) , bad ( ) , clear ( )* и *fail ( )*

read ( )

```
# include < iostream >
istream  &read ( char  *buf , streamsize  num ) ;
```

Функцията *read ( )* е член на *istream* . Функцията *read ( )* чете num байта от свързания входен поток и ги слага в буфер указван от *buf* . Ако е достигнат края на файла преди да са прочетени num символа , *read ( )* просто спира и буфера ще съдържа колкото символа е имало
read ( ) връща псевдоним за потока . Свързани функции са *gcount ( ) , get ( ) , getline ( )* и *write ( )*

seekg ( ) и seekp ( )

```
# include < iostream >
istream  &seekg ( off_type  offset , ios : : seekdir  origin ) ;
istream  &seekg ( pos_type  position ) ;
ostream  &seekp ( off_type  offset , ios : : seekdir  origin ) ;
ostream  &seekp ( pos_type  position ) ;
```

Функцията *seekg ( )* е член на *istream* , а функцията *seekp ( )* е член на *ostream* . В системата за I/O на C++ изпълнявате произволен достъп чрез използване на функциите *seekg ( )* и *seekp ( )* . Към този момент системата на C++ за I/O управлява два указателя свързани файл . Единия е get pointer ,който определя къде във файла ще стане следващата входна операция . Другия е put pointer ,който определя къде във файла ще стане следващата изходна операция . Всеки път, когато се извърши въвеждане или извеждане сътветния указател се увеличава автоматично в последователност . Обаче чрез използване на *seekg ( )* и *seekp ( )* е възможно да достигнете файла по непоследователен начин . Двупараметровата версия на *seekg ( )* премества get pointer на *offset* брой байтове от мястото определено от *origin* . Двупараметровата версия на *seekp ( )* премества put pointer на *offset* брой байтове от мястото определено от *origin* . Параметъра *offset* е от тип *off_type* който е способен да съдържа най – голямата валидна стойност която *offset* може да има .
Параметъра *origin* е от тип *seekdir* и е изброяване което има тези стойности :

ios : : beg	отместване от началото
ios : : cur	отместване от текущата позиция
ios : : end	отместване от края

Еднопараметровите версии на *seekg ( )* и *seekp ( )* преместват файловите указатели на положението определено от *position* . Тази стойност трябва да бъде получена преди това чрез извикване на *tellg ( )* или *tellp ( )* съответно *pos_type* е тип, който е способен да съдържа най – голямата валидна стойност, която може да има *position* . Тези функции връщат псевдоним за свързания поток .
Свързани функции са *tellg ( )* и *tellp ( )*

setf ( )

```
# include < iostream >
fmtflags  setf ( fmtflags  flags ) ;
fmtflags  setf ( fmtflags  flags1 , fmtflags  flags2 ) ;
```

Функцията *setf ( )* е член на *ios* . *setf ( )* вдига форматните флагове свързани с поток
Първата версия на *setf ( )* вдига форматните флагове определени от *flags*
(всички други флагове са непроменени) . Например за вдигане на флага *showpos* за *cout* може да използвате този израз :

```
cout . setf ( ios : : showpos ) ;
```

Когато искате да установите повече от един флаг можете да оградите заедно в кръгли скоби стойностите на флаговете, които искате да установите . Важно е да разберете, че извиквзнето на *setf ( )* се отнася само за определения поток .Няма концепция за самостоятелно извикване на *setf ( )* . Казано различно няма концепция в C++ за глобално форматно състояние . Всеки поток поддържа собствената си форматна статус информация индивидуално . Втората версия на *setf ( )* въздейства само на флаговете, които сте установили във *flags2* . Съответните флагове първо са пренастроени и след това вдигнати съгласно флаговете определени от *flags1* . Дори, ако *flags1* съдържа друго множество флагове , само онези определени от *flags2* ще бъдат променени . И двете версии на *setf ( )* връщат предишните настройки на форматните флагове свързани с потока .

Свързани функции са *unsetf ( )* и *flags ( )*

sync_with_stdio ( )

```
# include < iostream >
static bool sync_with_stdio ( bool sync = true ) ;
```

Функцията *sync_with_stdio ( )* е член на *ios* . Извикването на *sync_with_stdio ( )* позволява стандартната C – базирана система за I/O да бъде безопасно използвана едновременно с класово базираната система на C++ за I/O . За премахване на синхронизацията със *stdio* , подайте false на *sync_with_stdio ( )* . Връща се предишната настройка – *true* за синхронизиране и *false* ,ако няма синхронизация .

tellg ( ) и tellp ( )

```
# include < iostream >
pos_type tellg ( ) ;
pos_type tellp ( ) ;
```

Функцията *tellg ( )* е член на *istream* , а функцията *tellp ( )* е член на *ostream* . Системата за I/O на C++ управлява два указателя свързани с файл . Единия е get pointer ,който определя къде във файла ще стане следващата входна операция . Другия е put pointer , който определя къде във файла ще стане следващата изходна операция . Всеки път, когато се извърши въвеждане или извеждане сътветния указател се увеличава автоматично и последователно . Вие можете да определите текущата позиция на get pointer използвайки *tellg ( )* и на put pointer използвайки *tellp ( )* . *pos_type* е тип ,който е способен да съдържа най – голямата стойност , която всяка от функциите може да върне . Стойностите върнати от *tellg ( )* и *tellp ( )* могат да бъдат използвани като параметри съответно на *seekg ( )* и *seekp ( )* . Свързани функции са *seekg ( )* и *seekp ( )*

unsetf ( )

```
# include < iostream >
void unsetf ( fmtflags flags ) ;
```

Функцията *unsetf ( )* е член на *ios* . Функцията *unsetf ( )* се използва да свали един или повече форматни флага . Флаговете определени от *flags* се свалят (всички други флагове остават непроменени) . Свързани функции са *setf ( )* и *flags ( )* .

width ( )

```
# include < iostream >
streamsize width ( ) const ;
streamsize width ( streamsize w ) ;
```

Функцията *width ( )* е член на *ios* . За да получите текущата ширина на полето използвайте първата форма на *width ( )* . Тази версия връща текущата ширина на полето . За да установите ширината на полето използвайте втората форма . Тук , *w* става новата ширина на полето , а се връща предишната стойност . Свързани функции са *precision ( )* и *fill ( )*

write ( )

```
# include < iostream >
ostream &write ( const char *buf , streamsize num ) ;
```

Функцията *write ( )* е член на *ostream* . Функцията *write ( )* записва *num* байта в определения изходен поток от буфера указван чрез *buf* . Тя връща псевдоним за потока . Свързани функции са *read ( )* и *put ( )*

8. Създаване на класове и обекти

Същност на класовете:

Класът е основната част в капсулирането на C++. Класът е дефиниран чрез използването на ключовата дума *class*

Променливите и функциите които оформят класа са наречени членове .Общата форма на class е показана тук:

```
class име-на-клас : списък с наследявания {
  // членове private по подразбиране
protected:
  // членове private които могат да се наследяват
public:
  // членове public
} списък от обекти;
```

Тук име_на_клас е името на класовия тип. Веднъж когато декларацията на класа бъде съставена, име-на-клас става нов тип име за данни което може да се използва за деклариране на обекти от класа .Списъка с обекти е разделен със запетайки списък от обекти от тип име-на-клас.Този списък е по избор. Обектите класове могат да бъдат декларирани по-късно във вашата програма просто чрез използване на името на класа.Списъка с наследявания е също по избор.Когато е представен той определя базовия клас или класоветекоито наследяваповия клас.Класа може да включва функция конструктор и функция деструктор (и двете са по избор).Конструктора се извиква, когато се създава първи обект от класа.Деструктора се извиква ,когато се разрушава обект. Конструктора има същото име като класа.Деструктора има същото име като класа но предшествано със ~ (тилда).Нито конструктора нито деструктора имат връщани типове. В класовата йерархия, конструкторите се изпълняват по ред на наследяването, а деструкторите се изпълняват в обратен ред. По подразбиране всички елементи на класа са *private* за този клас и могат да бъдат достъпни само за другите членове на този клас.За да позволите на елемент от класа да бъде достъпен за функции ,които не са членове на класа вие трябва да го декларирате след ключовата дума *public*

Класът представлява дефиниран от програмиста тип данни, който може да емулира,наподобява реален обект. Класовете притежават атрибути и свойства, с които симулират тези на реалния обект. Атрибутите са известни като „членове", а действията - като „методи" на класа.

Не трябва да се забравя, че дефиницията на класа само създава тип данни, който капсулира атрибутите и действията.За създаването на обект е необходимо да се създаде инстанция на този тип данни. Това се извършва по същия начин, както създаването на инстанции на стандартните типове данни в C++.

Дефиниране на клас:
Общоприето е името на всеки клас да започва с главна буква.
Дефиницията на клас в C++ започва с ключовата дума class последвана от интервал и зададено от програмиста име:) След това във фигурни скоби { и } се поставят членовете и методите му.
Дефиницията завършва с точка и запетая след затварящата скоба.

За да се осигури външен достъп до членовете и методите на класа, пред списъка, в който са изброени, трябва да се постави ключовата дума public с двоеточие.

пример:
дефинираме клас Cat (котка)

```
class Cat    // с голяма буква да започват
{
public:
void bark();
int vazrast;
int teglo;
```

string cviat;
} macinka, mirelka; // ето тук винаги се поставят точка и запетая...и си добавихме и име на обекта:)

Създаване на обект:
Обектът представлява инстанция на зададен от програмиста клас, като на неговите членове могат да бъдат присвоени данни, а методите му могат да бъдат извиквани от програмата.

Дефиниране на методи на обект:
дефиницията на метод на клас, когато е извън тази на самия клас, трябва да идентифицира класа, за когото се отнася...необходимо е да се въведе името на класа и двойно двоеточие :: преди името на метода в дефиницията му.
Знакът :: е оператор за разделяне на областта на видимост, който показва, че дадената дефиниция се отнася за метода на указания клас!!!

Съхраняване на частни данни:
По-добре от гледна точка на сигурността е, когато членовете на клас не са достъпни директно извън него. Така се осигурява по-добро предпазване на данните. Тази техника е известна като скриване на данните и се изразява в поставянето на private: пред списъка с членове, вместо public:

За присвояване и извличане на стойности от тези членове се използват специални методи за достъп, които се добавят към частта public: на класа. За присвояване на данни се използват т.нар. set методи, а за извличане на данни - get методи. Методите за достъп се именуват като променливата, до която осъществяват достъп, но първата буква трябва да бъде главна и пред името да има префикс съответно set или get

Методите за достъп трябва да бъдат дефинирани, както винаги. Всеки обект има специален указател, наречен this , който реферира самия обект. По този начин членовете на обекта могат да се записват например във вида this -> age.
Това е полезно при дефинициите на методите за достъп, където често аргументът и членът на класа имат едно и също име. Чрез указателя this става разграничаването между двата.

Конструктори и деструктори:
Членовете на класа могат да бъдат инициализирани с помощта на специален метод, наречен конструктор. Той изисква аргументи, които ще бъдат началните стойности на членовете на класа. Този метод обаче не трябва да връща никаква стойност, дори void . Името на конструктора винаги трябва да съвпада с това на класа.Освен конструктор, трябва да се декларира и съответстващ му деструктор, който изчиства данните от тази част от паметта, която е била заделена от конструктора. Името на деструктора е същото като това на класа, но пред него се поставя знакът тилда (~) -казах го вече:) Този метод не притежава аргументи и не връща стойности...

Константни обектни методи:
Методите, които никога не променят стойността на член на клас, трябва да бъдат декларирани като константни посредством ключовата дума const, която се поставя както в декларацията, така и в дефиницията им. Тя се въвежда непосредствено след затварящата скоба на аргументите и е полезна за предотвратяване на грешки. За методи, които съдържат не повече от 1-2 реда код, може да се използва ключовата дума inline по същия начин, както при функциите. Един метод може да включи кода от тялото си в декларацията на класа, така че автоматично да стане инлайн.

Използване на класове в други класове:
След като веднъж е създаден даден клас, той може да се използва в декларациите на други класове чрез добавяне на комлилаторната директива #include към файла на класа. Тогава методите му стават достъпни за използване в следващата декларация на клас.

9. Указатели към данни

Операторът „адрес на“ :

Важно е да сте наясно с начина, по който се осъществява съхранението на данни в компютъра. При деклариране на някаква променлива в програма се запазва място в паметта на компютъра за съхранение на стойностите (данните), които са й присвоени. Това място се разделя на последователно номерирани запаметяващи регистри, като всеки от тях може да съдържа 1 байт данни.
Броят резервирани байтове в паметта зависи от типа на променливата. Заделената памет се реферира чрез уникалното име на променливата. Паметта на компютъра би могла да се сравни с дълга редица от клетки. Всяка от тях притежава уникален адрес, представен в шестнадесетичен формат. Може да се направи аналогия и с дълга редица от къщи, всяка от които има уникален номер (в десетичен формат), и в нея живеят хора. В програмите на C++ къщите са клетките, а хората - променливите.
Операторът & (адрес на) връща адреса в паметта на произволна променлива в шестнадесетичен формат.
След като чрез декларацията на променлива е запазено място в паметта, в него могат да се съхраняват данни от подходящ тип, като се използва операторът за присвояване =
Например при въвеждане на num = 100 стойността от дясната страна (100) се съхранява в тази част от паметта, указана от променливата num
Стойността от лявата страна на оператора = се нарича лява стойност (L-стойност), а тази от дясната страна - дясна или R-стойност.
„L" означава местоположение(location), а „R" - прочитане (read)

Има си правило при програмирането на C++ и то гласи, че дясната стойност не може да бъде от лявата страна на оператора =
За разлика от нея, лявата стойност може да се намира както от едната, така и от другата му страна...

Указателите са неразделна част от програмирането на C++. Те представляват променливи, съхраняващи адреса в паметта на други променливи. Указателите се декларират по същия начин, както и останалите променливи, като единствената разлика е, че пред името им се поставя *
Този знак представлява оператора за дереференция и просто показва, че декларираната променлива е указател. Типът на указателя трябва да съвпада с този на променливата, към която сочи.
След като бъде декларирана, на променливата указател може да бъде присвоен адресът на променлива, като се използва операторът &. Пред името на променливата в конструкцията за присвояване не трябва да се поставя операторът за дереференция *, освен ако указателят не се инициализира веднага в самата декларация на променливата. Името на променливата указател, когато се използва самостоятелно, реферира адрес в паметта, представен в шестнадесетичен вид.
Когато операторът за дереференция * се използва в декларация на променлива, той просто показва, че тя е указател...когато обаче същият оператор се намира пред указател на друго място в програмата, той реферира данните, съхранени в присвоения му адрес от паметта.
Когато пред името на променлива указател е поставен операторът за дереференция, той реферира данните, съхранени в адреса в паметта, присвоен на този указател.Това означава, че дадена програма може да получи адреса , присвоен на променлива указател, като използва само името му. Данните, които се съхраняват в този адрес, могат да бъдат получени като пред името на указателя се постави операторът за дереференция *

След като е създадена променлива указател с присвоен адрес, на нея може да се присвои друг адрес или пък да бъде преместена, като се използва аритметична операция.Операторът за инкрементиране ++ и операторът за декрементиране -- преместват указателя към следващия или предишния адрес за същия тип данни - колкото по-голям е типът, толкова по-значително е преместването.
По-големи премествания се постигат с помощта на операторите += и -=
Аритметиката с указатели е особено полезна при масив, тъй като елементите им заемат последователно разположени местоположения в паметта. При присвояване на името на масив на указател автоматично се присвоява адресът на първия негов елемент.Инкрементирането на указателя с единица го премества към следващия елемент.Освен че осигуряват достъп до стойността на променлива,указателите могат да се използват и за промяна на тази стойност. За целта пред името на променливата указател се поставя операторът за дереференция * като по този начин се извършва присвояването на нова стойност от подходящ тип.В програмите на C++ аргументите на функциите предават данните си по стойност на локална променлива в извиканата функция.

Това означава, че последната работи не с оригиналната стойност, а с нейно копие. Това се преодолява с предаване на указател към оригиналната стойност,което позволява функцията директно да работи с нея
Всяка програма, създадена със C++, може да съдържа масиви от указатели, в които всеки елемент съдържа адреса на друга променлива.
Указателите могат да сочат към функции, въпреки че тази възможност се използва по-рядко, отколкото указателите към стойности.Указателят към функция по същество не се различава от този към данни, но винаги трябва да е заграден в кръгли скоби при използване на оператора за дереференция * ,за да се избегне грешка при компилирането. След него също в кръгли скоби се поставят всички аргументи, които се предават на функцията при използването на оператора за дереференция.

Стандартната библиотека с шаблони на C++

Едно от главните постижения, които станаха през стандартизационния процес на C++ беше включването на стандартната библиотека с шаблони или Standard Template Library(STL)
STL предоставя общо предназначени , шаблонни класове и функции, които изпълняват множество популярни и общо използвани алгоритми и структури за данни . Например тя включва поддръжка на вектори , списъци , опашки и стекове . Тя също дефинира и разнообразни начини за достъп до тях . Понеже STL е изградена от шаблонни класове , алгоритмите и структурите за данни могат да бъдат приложени към почти всеки тип данни . STL е голяма библиотека и не всички от нейните качества могат да бъдат пълно описани в тази книжка:) Също версията на STL описана тук е тази определена от стандартизационния комитет ANSI / ISO . Вашият компилатор може да предоставя по – различна версия на STL , така че трябва да проверите неговата документация и работа с него...

Преглед на контейнери , алгоритми и итератори

STL е съставена от три компонента : контейнери , алгоритми и итератори. Те работят заедно, за да доставят основни решения за разнообразни програмни проблеми . Всеки е описан кратко тук .

Контейнери

Контейнерите са обекти , които съдържат други обекти . Има няколко различни типа контейнери Например класа *vector* дефинира динамичен масив , *queue* създава опашка и *list* изгражда линеен списък . В добавка към основните контейнери STL също дефинира и асоциативни контейнери, които позволяват ефикасно получаване на стойности базирани на ключове . Например *map* доставя достъп до стойности с уникални ключове . Всеки контейнерен клас дефинира множество от функции, които могат да бъдат приложени към контейнера ...
list **контейнера включва функции , които вмъкват , изтриват и сливат елементи...**

Алгоритми

Алгоритмите действат върху контейнерите . Те включват възможности за инициализиране , сортиране , претърсване и преобразуване на съдържанието на контейнерите . Много алгоритми действат в последователност, което е линеен списък от елементи в контейнера

Итератори

Итераторите са обекти, които са повече или по – малко указатели . Те дават възможност да имаме достъп до съдържанието на контейнер по много подобен начин, при който използваме указател за достъп до масив . Има 5 типа итератори :

Итератор	Разрешен достъп
Произволен достъп	Съхранява и извлича стойности; елементите могат да бъдат достъпвани произволно
Двупосочен	Съхранява и извлича стойности; движение напред и назад
Прав	Съхранява и извлича стойности; движение само напред
Входен	Извлича , но не съхранява стойности ; движение само напред
Изходен	Съхранява , но не извлича стойности ; движение само напред

Итератор, който има най – големи възможности за достъп може да бъде използван на мястото на такъв с по – малки възможности . Например правия итератор може да бъде използван на мястото на входния . Итераторите се манипулират като указатели . Може да се увеличават и намаляват . Може да се прилага оператора * към тях . Итераторите са декларирани чрез използване на типа *iterator* дефиниран за различни контейнери .

STL също поддържа и обратни итератори . Обратните итератори са или двупосочни или с произволен достъп, които се преместват в последователност в обратна посока . Така ако обратния итератор сочи края на поредицата , увеличавайки го ще предизвикаме той да сочи към един елемент преди края

Алокатори

Всеки контейнер дефинира за себе си алокатор , който е обект от клас *allocator* .
Алокаторите управляват заделянето на памет, когато се създава нов контейнер

Предикати и сравнителни функции

Някои от алгоритмите и контейнерите използват специален тип функции наречени предикати .
Има две вариации на предикатите - унарни и бинарни . Унарния предикат приема един аргумент . Бинарния приема два аргумента . Тези функции връщат резултат *true / false* . Но точното условие, което ги кара да върнат *true* или *false* се дефинира от вас
В бинарния предикат , аргументите са винаги в ред първи , втори отнасящи се за функцията, която извиква предиката . И за двата предиката , аргументите ще съдържат стойности от типа на обектите, които се съхраняват в контейнер . Някои алгоритми и класове използват специален тип бинарен предикат, който сравнява два елемента . Сравняващите функции връщат *true* ,ако техния първи аргумент е по – малък от втория

Инструменти и функционални хедъри

В добавка към хедърите изисквни от различните класове на STL стандартната библиотека на C++ включва < *utility* > и < *functional* > хедъри , които доставят поддръжка за STL . Например в < *utility* > е дефиниран шаблонния клас *pair* , който може да съдържа двойка от стойности .
Шаблонната функция *less ( )* декларирана във < *functional* > определя кога един обект е по – малък от друг
Шаблоните в < *functional* > ви позволяват да изградите обекти, които дефинират *operator( )* . Те са наречени функционални обекти и могат да бъдат използвани на мястото на указателите към функции на много места

Контейнерите , алгоритмите и итераторите работят заедно . Най – добрия начин да разберете това е да видите пример . Следващата програма демонстрира контейнера *vector*
той има предимство, че автоматично манипулира свойте размери за съхраняване , нараствайки , ако е необходимо
***vector* предоставя методи, за да можете да определите размера му и да добавяте или премахвате елементи . Следващата програма демонстрира използването на класа vector :**

```
// кьс пример демонстриращ vector
# include < iostream >
#include < vector >
using namespace std ;

int main ( )
{
  vector < int > v ; // създава vector с нулева дължина
 int i ;
  // показва първоначалната дължина на v
  cout << " size = " << v . size ( ) << endl ;
  /* поставя стойности към края на v –
     vector ще нарасне ако е необходимо */
  for ( i = 0 ; i < 10 ; i ++ ) v . push_back ( i ) ;
  // показва текущата дължина на v
  cout << " size now = " << v . size ( ) << endl ;
  // можеме да достъпваме съдържанието на vector
```

```
    // чрез индекс
    for ( i = 0 ; i < 10 ; i ++ )  cout << v [ i ]  <<  " " ;
    cout << endl ;
    // можеме да достъпваме първия и последния елемент
    cout << " front  = " << v . front ( ) << endl ;
    cout << " back  = " << v . back ( ) << endl ;
    // достъп чрез итератор
    vector < int > : : iterator  p = v . begin ( ) ;
    while ( p != v . end ( ) ) {
          cout  <<  * p  <<  " " ;
          p++ ;
    }

    return 0 ;
}
```

Изхода от тази програма е :

```
size = 0
size now = 10
0 1 2 3 4 5 6 7 8 9
front = 0
back = 9
0 1 2 3 4 5 6 7 8 9
```

В тази програма вектора е първоначално създаден с нулева дължина . Член функцията *push_back ( )* слага стойности в края на вектора , увеличавайки размера му ако е необходимо . Функцията *size ( )* показва размера на вектора . Вектора може да бъде индексиран като нормален масив . Също може да бъде достъпван чрез итератор . Функцията *begin ( )* връща итератор към началото на вектора . Функцията *end ( )* връща итератор към края на вектора . Забележете как е деклариран итератора *p* . Типа *iterator* е дефиниран за няколко контейнерни класа .

Контейнерни класове

Контейнерите дефинирани от STL са показани тук :

Контейнер	Описание	Изискван хедър
bitset	Множество от битове	< *bitset* >
deque	Опашка с два края	< *deque* >
list	Линеен списък	< *list* >
map	Съхранява двойки ключ / стойност в които на един ключ могат да отговарят две или повече стойности	< *map* >
multimap	Съхранява двойки ключ / стойност в които на един ключ могат да отговарят две или повече стойности	< *map* >
multiset	Множество в което всеки елемент не е необходимо да е уникален	< *set* >
priority_queue	Опашка с приоритети	< *queue* >
queue	Опашка	< *queue* >
set	Множество в което всеки елемент е уникален	< *set* >
stack	Стек	< *stack* >
vector	Динамичен масив	< *list* >

Класа *string* , който управлява символните низове е също контейнер

Понеже контейнерите са изпълнени чрез шаблонни класове се използват различни типове съхранители на данни . Понеже имената на съхранителните типове в шаблонен клас са произволни , контейнерините класове декларират *typedef* версии на тези типове . Това прави имената на типовете конкретни . Някой от най – общите *typedef* имена са показани тук :

size_type	**Някакъв целочислен тип еквивалентен на *size_t***
reference	**Псевдоним на елемент**
const_reference	**const псевдоним на елемент**
iterator	**Итератор**
const_iterator	**const итератор**
reverse_iterator	**Обратен итератор**
const_reverse_iterator	***const* обратен итератор**

value_type Тип на стойността съхранявана в контейнер

allocator_type	**Типа на алокатора**
key_type	**Типа на ключа**
key_compare	**Типа на функцията която сравнява два ключа**
value_compare	**Типа на функцията която сравнява две стойности**

bitset

Класа *bitset* поддържа операции върху множество от битове . Неговата шаблонна спецификация е :

template < size_t N > class bitset

Тук , *N* определя дължината на *bitset* в битове . Той има следните конструктори :

bitset ( ) ;
bitset (unsigned long bits) ;
explicit bitset (const string &s , size_t i = 0 , size_t num = -1);

Първата форма конструира празен *bitset* . Втората форма конструира *bitset* ,който има собствено множество битове съгласно тези определени в *bits* . Третата форма конструира *bitset* чрез използване на низа *s* , започвайки от *i* . Низа трябва да съдържа само единици или нули . Само num или *s.size ( ) – i* стойности се използват , което е по – малко . Извеждащите оператори << и >> се дефинират за *bitset bitset* съдържа следните член – функции :

Член	**Описание**
bool any ( ) const ;	**Връща *true* ако всеки бит в извиквания *bitset* е 1 ; в противен случай връща 0**
size_type count ( ) const ;	**Връща броя на битовете с 1**
bitset < N > &flip ( ) ;	**Инвертира състоянието на всички битове в извиквания *bitset* и връща **this***
bitset < N > &flip (size_t i) ;	**Инвертира бита на позиция *i* в извикания *bitset* и връща **this***
bool none ( ) const ;	**Връща *true* ако няма установени битове в извикания *bitset***
bool operator != (const bitset <N> &op2) const ;	**Връща *true* ако извикания *bitset* се различава от този определен в дясната страна на оператора *op2***
bool operator == (const bitset <N> &op2) const ;	**Връща *true* ако извикания *bitset* е еднакъв със този определен в дясната страна на оператора *op2***
bitset <N> &operator &= (const bitset <N> &op2);	**Операция AND върху всеки бит на извикания *bitset* със съответния бит в *op2* и поставя резултата в извикания *bitset* ; връща **this***
bitset <N> &operator ^ = (const bitset <N> &op2);	**Операция XOR върху всеки бит на извикания *bitset* със**

	съответния бит в *op2* и поставя резултата в извикания *bitset* ; връща **this*
bitset <N> &operator | = (const bitset <N> &op2);	Операция OR върху всеки бит на извикания *bitset* със съответния бит в *op2* и поставя резултата в извикания *bitset* ; връща **this*
bitset <N> &operator ~ = ( ) const ;	Инвертира състоянието на всички битове в извикания *bitset* и връща **this*
bitset <N> &operator << = (size_t num) ;	Премества наляво всеки бит в извикания *bitset* на *num* позиции и поставя резултата в извикания *bitset* ; връща **this*
bitset <N> &operator >> = (size_t num) ;	Премества надясно всеки бит в извикания *bitset* на *num* позиции и поставя резултата в извикания *bitset* ; връща **this*
reference operator [ ] (size_type i) ;	Връща псевдоним за бит *i* в извикания *bitset*
bitset <N> &reset ( ) ;	Изчиства всички битове в извикания *bitset* и връща **this*
bitset <N> &reset (size_t i) ;	Изчиства бита на позиция *i* в извикания *bitset* и връща **this*
bitset <N> &set ( ) ;	Установява всички битове в извикания *bitset* и връща **this*
bitset <N> &set (size_t i , int val = 1) ;	Установява бита на позиция *i* на стойността определена от *val* в извикания *bitset* и връща **this* Всяка ненулева стойност за *val* е приета да бъде *1*
size_t size ( ) const ;	Връща броя на битовете който може да съдържа дадения *bitset*
bool test (size_t i) ;	Връща състоянието на бита на позиция *i*
string to_string ( ) const ;	Връща низ който съдържа представяне на битовия шаблон в извикания *bitset*
unsigned long to_ulong ( ) const ;	Конвертира извикания *bitset* в unsigned long integer

deque

Класа *deque* поддържа опашка със два края . Неговата шаблонна спецификация е :

template < class T , class Allocator = allocator < T >>
class deque

Тук *T* е типа данни съхранявани в *deque* . Той има следните конструктори :

explicit deque (const Allocator &a = Allocator ( )) ;
explicit deque (size_type num , const T &val = T ( ) ,
const Allocator &a = Allocator ( )) ;
deque (const deque < T , Allocator > &ob) ;
template < class InIter > deque (InIter start , InIter end ,
const Allocator &a = Allocator ( )) ;

Първата форма конструира празна *deque* . Втората форма конструира *deque* ,която има *num* елемента със стойност *val* . Третата форма изгражда *deque* която има еднакви елементи с *ob* . Четвъртата форма изгражда опашка, която съдържа елементите в интервала определен от *start* и *end* . Следващите сравнителни оператори са дефинирани за *deque* = = , < , < = , ! = , > и > = *deque* съдържа следните член функции:

Член	Описание
template < class InIter > void assign (InIter start , InIter end) ;	Присвоява на *deque* поредицата дефинирана от *start* и *end*
template < class Size , class T >	Присвоява на *deque num*

void assign (Size num , const T &val = T ( )) ;	елемента със стойност *val*
reference at (size_type i) ;	Връща псевдоним за елемента
const_reference at (size_type i) const ;	определен чрез *i*
reference back ( ) ;	Връща псевдоним за последния
const_reference back ( ) const ;	елемент в *deque*
iterator begin ( ) ;	Връща итератор за първия
const_iterator begin ( ) const ;	елемент в *deque*
void clear ( ) ;	Премахва всички елементи от *deque*
bool empty ( ) const ;	Връща true ако извиканата deque е празна и *false* в противен случай
iterator end ( ) ;	Връща итератор към края
const_iterator end ( ) const ;	на *deque*
iterator erase (iterator i) ;	Премахва елемента указван от *i* връща итератор към елемента след премахнатия
iterator erase (iterator start , iterator end) ;	Премахва елементите в интервала *start – end* ,връща итератор към елемента след последния премахнат елемент
reference front ( ) ;	Връща псевдоним към първия
const_reference front ( ) const ;	елемент на *deque*
allocator_type get_allocator ( ) const ;	Връща алокатор за *deque*
iterator insert (iterator i , const T &val = T ( )) ;	Вмъква *val* непосредствено преди елемента определен от *i* Връща се итератор към елемента
void insert (iterator i , size_type num , const T &val) ;	Вмъква *num* копия от *val* непосредствено преди елемента определен от *i*
template < class InIter > void insert (iterator i , InIter start , InIter end) ;	Вмъква поредицата определена от *start – end* непосредствено преди елемента определен от *i*
size_type max_size ( ) const ;	Връща максималния брой елементи които може да съдържа *deque*
reference operator [ ] (size_type i) const ;	Връща псевдоним за *i* - тия
const_reference operator [ ] (size_type i) const ;	елемент
void pop_back ( ) ;	Премахва последния елемент в *deque*
void pop_front ( ) ;	Премахва първия елемент в *deque*
void push_back (const T &val) ;	Добавя елемент със стойност *val* към края на *deque*
void push_front (const T &val) ;	Добавя елемент със стойност *val* към началото на *deque*
reverse_iterator rbegin ( ) ;	Връща обратен итератор към
const_reverse_iterator rbegin ( ) const ;	края на *deque*
reverse_iterator rend ( ) ;	Връща обратен итератор към
const_reverse_iterator rend ( ) const ;	началото на *deque*
void resize (size_type num , T val = T ( )) ;	Променя размера на *deque* към този определен от *num* . Ако *deque* трябва да бъде удължена , тогава елементите със стойността определена от *val* се добавят към края
size_type size ( ) const ;	Връща текущия брой елементи в *deque*

void swap (deque < T , Allocator > &ob) ; Разменя елементите съхранявани в извикваната *deque* със тези в *ob*

list

Класа *list* поддържа списък . Неговата шаблонна спецификация е :

template < class T , class Allocator = allocator < T >>
class list

Тук *T* е типа данни съхранявани в списъка . Той има следните конструктори :

explicit list (const Allocator &a = Allocator ( )) ;
explicit list (size_type num , const T &val = T ( ) ,
const Allocator &a = Allocator ( )) ;
list (const list < T , Allocator > &ob) ;
template < class InIter > list (InIter start , InIter end ,
const Allocator &a = Allocator ( )) ;

Първата форма конструира празен списък . Втората форма конструира списък , който има *num* елемента със стойност *val* . Третата форма изгражда списък, който има еднакви елементи с *ob* . Четвъртата форма изгражда списък , който съдържа елементите в интервала определен от *start* и *end* . Следващите сравнителни оператори са дефинирани за *list* = = , < , < = , ! = , > и > = *list* съдържа следните член функции :

Член	Описание
template < class InIter > void assign (InIter start , InIter end) ;	Присвоява на списъка поредицата дефинирана от *start* и *end*
template < class Size , class T > void assign (Size num , const T &val = T ( )) ;	Присвоява на списъка *num* елемента със стойност *val*
reference back ( ) ;	Връща псевдоним за последния
const_reference back ( ) const ;	елемент в списъка
iterator begin ( ) ;	Връща итератор за първия
const_iterator begin ( ) const ;	елемент в списъка
void clear ( ) ;	Премахва всички елементи от списъка
bool empty ( ) const ;	Връща *true* ако извикания списък е празен и *false* в противен случай
iterator end ( ) ;	Връща итератор към края
const_iterator end ( ) const ;	на списъка
iterator erase (iterator i) ;	Премахва елемента указван от *i* връща итератор към елемента след премахнатия
iterator erase (iterator start , iterator end) ;	Премахва елементите в интервала *start – end* ,връща итератор към елемента след последния премахнат елемент
reference front ( ) ;	Връща псевдоним към първия
const_reference front ( ) const ;	елемент на списъка
allocator_type get_allocator ( ) const ;	Връща алокатор за списъка
iterator insert (iterator i , const T &val = T ( )) ;	Вмъква *val* непосредствено преди елемента определен от *i* Връща се итератор към елемента
void insert (iterator i , size_type num , const T &val) ;	Вмъква *num* копия от *val* непосредствено преди елемента определен от *i*
template < class InIter > void insert (iterator i , InIter start , InIter end) ;	Вмъква поредицата определена от *start – end* непосредствено преди елемента определен от *i*
size_type max_size ( ) const ;	Връща максималния брой елементи които може да съдържа списъка
void merge (list < T , Allocator > &ob) ;	Слива подредения списък
template < class Comp >	съдържан в *ob* със подредения

void merge (list < T , Allocator > &ob Comp cmpfn) ;	**извикван списък . Резултатния списък е подреден . След сливането , списъка съдържан от *ob* е празен . Във втората форма сравнителната функция може да бъде определена така че да определя кога един елемент е по – малък от друг**
void pop_back ( ) ;	**Премахва последния елемент в списъка**
void pop_front ( ) ;	**Премахва първия елемент в списъка**
void push_back (const T &val) ;	**Добавя елемент със стойност *val* към края на списъка**
void push_front (const T &val) ;	**Добавя елемент със стойност *val* към началото на списъка**
reverse_iterator rbegin ( ) ; *const_reverse_iterator rbegin ( ) const ;*	**Връща обратен итератор към края на списъка**
void remove (const T &val) ;	**Премахва елементите със стойност *val* от списъка**
template < class UnPred > *void remove_if (UnPred pr) ;* *const T &val = T ( )) ;*	**Премахва елементите за които унарния предикат *pr* е *true***
reverse_iterator rend ( ) ; *const_reverse_iterator rend ( ) const ;*	**Връща обратен итератор към началото на списъка**
void resize (size_type num , *T val = T ( )) ;*	**Променя размера на списъка към този определен от *num* . Ако списъка трябва да бъде удължен, тогава елементите със стойността определена от *val* се добавят към края**
void reverse ( ) ;	**Обръща извиквания списък**
size_type size ( ) const ;	**Връща текущия брой елементи в списъка**
void sort ( ) ; *template < class Comp >* *void sort (Comp cmpfn) ;*	**Сортира списъка . Втората форма сортира списъка използвайки сравнителната функция *fn* за определяне кога един елемент е по – малък от друг**
void splice (iterator i , *list < T , Allocator > &ob) ;*	**Съдържанието на *ob* се вмъква в извиквания списък на позицията определена от *i* . След операцията *ob* е празен**
void splice (iterator i , *list < T , Allocator > &ob ,* *iterator el) ;*	**Елемента указван от *el* е премахнат от списъка *ob* и съхранен в извиквания списък на позицията определена от *i* .**
void splice (iterator i , *list < T , Allocator > &ob ,* *iterator start , iterator end) ;*	**Интервала определен от *start* и *end* е премахнат от *ob* и съхранен в извиквания списък започвайки от позицията определена от *i***
void swap (list < T , Allocator > &ob) ;	**Разменя елементите съхранявани в извиквания списък със тези в *ob***
void unique ( ) ; *template < class BinPred >* *void unique (BinPred pr) ;*	**Премахва дублиращите се елементи от извиквания списък . Втората форма използва *pr* за да определи уникалността**

map

Класа *map* поддържа асоциативен контейнер, в който уникални ключове се съпоставят на стойности . Неговата шаблонна спецификация е :

template < class Key , class T , class Comp = less < Key > ,
class Allocator = allocator < T >> class map

***Key* е типа данни на ключовете , *T* е типа данни на стойностите, които ще се съхраняват (съпоставят) и *Comp* е функция, която сравнява два ключа . Той има следните конструктори :**

```
explicit map ( const Comp &cmpfn = Comp ( ) ,
               const Allocator &a = Allocator ( ) ) ;
map ( const map < Key , T , Comp , Allocator > &ob ) ;
template < class InIter > map ( InIter start , InIter end ,
                                const Comp &cmpfn = Comp ( ) ,
                                const Allocator &a = Allocator ( ) ) ;
```

Първата форма конструира празна карта. Втората форма изгражда карта, която съдържа еднакви елементи с *ob* . Третата форма изгражда карта, която съдържа елементите в интервала определен от *start* и *end* . Функцията определена чрез *cmpfn* , ако е представена , определя нареждането на картата Следващите сравнителни оператори са дефинирани за *map* = = , < , < = , ! = , > и > =
Тук са показани член – функциите на *map* . В описанията , *key_type* е типа на ключа , а *value_type* представя *pair < Key , T >*

Член	Описание
iterator begin ( ) ;	Връща итератор за първия елемент в картата
const_iterator begin ( ) const ;	
void clear ( ) ;	Премахва всички елементи от картата
size_type count (const key_type &k) const ;	Връща броя на срещанията на к в картата (1 или 0)
bool empty ( ) const ;	Връща *true* ако извиканата карта е празна и *false* в противен случай
iterator end ( ) ;	Връща итератор към края на картата
const_iterator end ( ) const ;	
pair < iterator , iterator > equal_range (const key_type &k) ;	Връща двойка итератори които указват първия и последния елемент в картата която съдържа определен ключ
pair < const_iterator , const_iterator > equal_range (const key_type &k) const ;	
void erase (iterator i) ;	Премахва елемента указван от *i*
void erase (iterator start , iterator end) ;	Премахва елементите в интервала *start – end*
size_type erase (const key_type &k) ;	Премахва от картата елементите които имат ключове със стойност *к*
iterator find (const key_type &k) ;	Връща итератор към определен ключ . Ако ключа не е открит се връща итератор към края на картата
const_iterator find (const key_type &k) const ;	
allocator_type get_allocator ( ) const ;	Връща алокатор за картата
iterator insert (iterator i , const value_type &val) ;	Вмъква *val* на или след елемента определен от *i* . Връща се итератор към елемента
template < class InIter > void insert (InIter start , InIter end) ;	Вмъква интервал от елементи
pair < iterator , bool > insert (const value_type &val) ;	Вмъква *val* в извикваната карта . Връща се итератор към елемента Елемента се вмъква ако още не е съществувал . Ако елемента се вмъкне се връща двойка *< iterator , true >* . В противен случай се връща *< iterator , false >*
key_compare key_comp ( ) const ;	Връща функция обект която сравнява ключове
iterator lower_bound (const key_type &k) ;	Връща итератор към първия

const_iterator lower_bound (const key_type &k) const ;	елемент в картата със ключ равен или по – голям от к
size_type max_size ( ) const ;	Връща максималния брой елементи които може да съдържа картата
reference operator[ ] (const key_type &i) ;	Връща псевдоним за елемента определен от i . Ako този елемент не съществува , той се вмъква
reverse_iterator rbegin ( ) ;	Връща обратен итератор към
const_reverse_iterator rbegin ( ) const ;	края на картата
reverse_iterator rend ( ) ;	Връща обратен итератор към
const_reverse_iterator rend ( ) const ;	началото на картата
size_type size ( ) const ;	Връща текущия брой елементи в картата
void swap (map < Key , T , Comp , Allocator > &ob) ;	Разменя елементите съхранявани в извикваната карта със тези в *ob*
iterator upper_bound (const key_type &k) ;	Връща итератор към първия
const_iterator upper_bound (const key_type &k) const ;	елемент в картата със ключ по – голям от *к*
value_compare value_comp ( ) const ;	Връща функция обект, която сравнява стойности

multimap

Класа *multimap* поддържа асоциативен контейнер, в който е възможно еднакви ключове да се съпоставят на стойности . Неговата шаблонна спецификация е :

template < class Key , class T , class Comp = less < Key > ,
class Allocator = allocator < T >> class multimap

***Key* е типа данни на ключовете , *T* е типа данни на стойностите, които ще се съхраняват (съпоставят) и *Comp* е функция която сравнява два ключа . Той има следните конструктори :**

explicit multimap (const Comp &cmpfn = Comp ( ) ,
const Allocator &a = Allocator ( )) ;
multimap (const multimap < Key , T , Comp , Allocator > &ob) ;
template < class InIter > multimap (InIter start , InIter end ,
const Comp &cmpfn = Comp ( ) ,
const Allocator &a = Allocator ( )) ;

Първата форма конструира празен *multimap* . Втората форма изгражда *multimap* , която съдържа еднакви елементи с *ob* . Третата форма изгражда *multimap* , която съдържа елементите в интервала определен от *start* и *end* . Функцията определена чрез *cmpfn* , ako е представена , определя нареждането на *multimap* Следващите сравнителни оператори са дефинирани за *multimap*
= = , < , < = , ! = , > и > =

Тук са показани член функциите на *multimap* . В описанията *key_type* е типа на ключа , *T* е стойността, а *value_type* представя *pair < Key , T >*

Член	Описание
iterator begin ( ) ;	Връща итератор за първия
const_iterator begin ( ) const ;	елемент в *multimap*
void clear ( ) ;	Премахва всички елементи от *multimap*
size_type count (const key_type &k) const ;	Връща броя на срещанията на *к* в *multimap*
bool empty ( ) const ;	Връща *true* ако извикания *multimap* е празен и *false* в противен случай
iterator end ( ) ;	Връща итератор към края
const_iterator end ( ) const ;	на *multimap*
pair < iterator , iterator > equal_range (const key_type &k) ;	Връща двойка итератори които указват първия и последния
pair < const_iterator , const_iterator >	елемент в *multimap* който

equal_range (const key_type &k) const ;	**съдържа определен ключ**
void erase (iterator i) ;	**Премахва елемента указван от *i***
void erase (iterator start , iterator end) ;	**Премахва елементите в интервала *start – end***
size_type erase (const key_type &k) ;	**Премахва от *multimap* елементите които имат ключове със стойност *к***
iterator find (const key_type &k) ;	**Връща итератор към определен**
const_iterator find (const key_type &k) const ;	**ключ . Ако ключа не е открит се връща итератор към края на *multimap***
allocator_type get_allocator ( ) const ;	**Връща алокатор за *multimap***
iterator insert (iterator i , const value_type &val) ;	**Вмъква *val* на или след елемента определен от *i* . Връща се итератор към елемента**
template < class InIter > void insert (InIter start , InIter end) ;	**Вмъква интервал от елементи**
iterator insert (const value_type &val) ;	**Вмъква *val* в извиквания *multimap***
key_compare key_comp ( ) const ;	**Връща функция обект която сравнява ключове**
iterator lower_bound (const key_type &k) ;	**Връща итератор към първия**
const_iterator lower_bound (const key_type &k) const ;	**елемент в *multimap* със ключ равен или по – голям от *к***
size_type max_size ( ) const ;	**Връща максималния брой елементи които може да съдържа *multimap***
reverse_iterator rbegin ( ) ;	**Връща обратен итератор към**
const_reverse_iterator rbegin ( ) const ;	**края на *multimap***
reverse_iterator rend ( ) ;	**Връща обратен итератор към**
const_reverse_iterator rend ( ) const ;	**началото на *multimap***
size_type size ( ) const ;	**Връща текущия брой елементи в *multimap***
void swap (multimap < Key , T , Comp , Allocator > &ob) ;	**Разменя елементите съхранявани в извиквания *multimap* със тези в *ob***
iterator upper_bound (const key_type &k) ;	**Връща итератор към първия**
const_iterator upper_bound (const key_type &k) const ;	**елемент в *multimap* със ключ по – голям от *к***
value_compare value_comp ( ) const ;	**Връща функция обект, която сравнява стойности**

multiset

Класа *multiset* поддържа множество, в което е възможно еднакви ключове да се съпоставят на стойности . Неговата шаблонна спецификация е :

template < class Key , class Comp = less < Key > ,
class Allocator = allocator < Key >> class multiset

Тук *Key* е типа данни на ключовете , а *Comp* е функция ,която сравнява два ключа . Той има следните конструктори :

```
explicit multiset ( const Comp &cmpfn = Comp ( ) ,
                    const Allocator &a = Allocator ( ) ) ;
multiset ( const multiset < Key , Comp , Allocator > &ob ) ;
template < class InIter > multiset ( InIter start , InIter end ,
                                     const Comp &cmpfn = Comp ( ) ,
                                     const Allocator &a = Allocator ( ) ) ;
```

Първата форма конструира празен *multiset* . Втората форма изгражда *multiset* ,който съдържа еднакви елементи с *ob* . Третата форма изгражда *multiset* ,който съдържа елементите в интервала определен от *start* и *end* . Функцията определена чрез *cmpfn* , ako е представена , определя нареждането на множеството . Следващите сравнителни оператори са дефинирани за *multiset* = = , < , < = , ! = , > и > =

Тук са показани член – функциите на *multiset* . В описанията и двете *key_type* и *value_type* са *typedef* за *Key* :

Член	Описание
iterator begin ( ) ;	Връща итератор за първия елемент в *multiset*
const_iterator begin ( ) const ;	
void clear ( ) ;	Премахва всички елементи от *multiset*
size_type count (const key_type &k) const ;	Връща броя на срещанията на *к* в *multiset*
bool empty ( ) const ;	Връща *true* ако извикания *multiset* е празен и *false* в противен случай
iterator end ( ) ;	Връща итератор към края на *multiset*
const_iterator end ( ) const ;	
pair < iterator , iterator > equal_range (const key_type &k) const ;	Връща двойка итератори които указват първия и последния елемент в *multiset* ,който съдържа определен ключ
void erase (iterator i) ;	Премахва елемента указван от *i*
void erase (iterator start , iterator end) ;	Премахва елементите в интервала *start – end*
size_type erase (const key_type &k) ;	Премахва от *multiset* елементите, които имат ключове със стойност *к*
iterator find (const key_type &k) const ;	Връща итератор към определен ключ . Ако ключа не е открит се връща итератор към края на *multiset*
allocator_type get_allocator ( ) const ;	Връща алокатор за *multiset*
iterator insert (iterator i , const value_type &val) ;	Вмъква *val* на или след елемента определен от *i* . Връща се итератор към елемента
template < class InIter > void insert (InIter start , InIter end) ;	Вмъква интервал от елементи
iterator insert (const value_type &val) ;	Вмъква *val* в извиквания *multiset* Връща се итератор към елемента
key_compare key_comp ( ) const ;	Връща функция обект която сравнява ключове
iterator lower_bound (const key_type &k) const ;	Връща итератор към първия елемент в *multiset* със ключ равен или по – голям от *к*
size_type max_size ( ) const ;	Връща максималния брой елементи които може да съдържа *multiset*
reverse_iterator rbegin ( ) ;	Връща обратен итератор към края на *multiset*
const_reverse_iterator rbegin ( ) const ;	
reverse_iterator rend ( ) ;	Връща обратен итератор към началото на *multiset*
const_reverse_iterator rend ( ) const ;	
size_type size ( ) const ;	Връща текущия брой елементи в *multiset*
void swap (multiset < Key , Comp , Allocator > &ob) ;	Разменя елементите съхранявани в извиквания *multiset* със тези в *ob*
iterator upper_bound (const key_type &k) const ;	Връща итератор към първия елемент в *multiset* със ключ по – голям от *к*
value_compare value_comp ( ) const ;	Връща функция обект която сравнява стойности

queue

Класа *queue* поддържа опашка с един край . Неговата шаблонна спецификация е :

```
template < class T , class Container = deque < T >>
                                    class queue
```

Тук *T* е типа данни, които се съхраняват , а *Container* е типа контейнер използван да съдържа опашката .Той има следния конструктор :

```
explicit queue ( const Container &cnt = Container ( ) ) ;
```

Конструктора *queue ( )* създава празна опашка . По – подразбиране той използва *deque* като контейнер Също можете да използвате *list* като контейнер за опашка . Контейнера се съдържа в защитен обект наречен с от тип *Container* . Следващите сравнителни оператори са дефинирани за *queue*

= = , < , < = , ! = , > и > =

***queue* съдържа следните член – функции :**

Член	Описание
value_type &back ( ) ;	Връща псевдоним за последния
const value_type &back ( ) const ;	елемент в опашката
bool empty ( ) const ;	Връща *true* ако извиканата опашка е празна и *false* в противен случай
value_type &front ( ) ;	Връща псевдоним за първия
const value_type &front ( ) const ;	елемент в опашката
void pop ( ) ;	Премахва първия елемент в опашката
void push (const T &val) ;	Добавя елемент със стойност *val* към края на опашката
size_type size ( ) const ;	Връща текущия брой елементи в опашката

priority_queue

Класа *priority_queue* поддържа приоритетна опашка с един край . Неговата шаблонна спецификация е :

```
template < class T , class Container = vector < T > ,
   class Comp = less < Container : : value_type >>
   class priority_queue
```

Тук *T* е типа на съхраняваните данни . *Container* е типа контейнер използван да съдържа опашката , а *Comp* определя сравнителната функция, която определя кога един член на приоритетната опашка е с по – нисък приоритет от друг . Той има следните конструктори :

```
explicit priority_queue ( const Comp &cmpfn = Comp ( ) ,
                          Container &cnt = Container ( ) ) ;
template < class InIter > priority_queue ( InIter start , InIter
                  end , const Comp &cmpfn = Comp ( ) ,
                          Container &cnt = Container ( ) ) ;
```

Първия *priority_queue ( )* конструктор създава празна приоритетна опашка . Втория изгражда опашка, която съдържа елементите в интервала определен от *start* и *end* . По – подразбиране той използва *vector* като контейнер . Може да използвате *deque* като контейнер за опашка . Контейнера се съдържа в защитен обект наречен *c* от тип *Container* . *priority_queue* съдържа следните член – функции :

Член	Описание
bool empty ( ) const ;	Връща *true* ако извиканата приоритетна опашка е празна и *false* в противен случай
void pop ( ) ;	Премахва първия елемент в приоритетната опашка
void push (const T &val) ;	Добавя елемент към приоритетната опашка
size_type size ( ) const ;	Връща текущия брой елементи в приоритетната опашка
value_type &top ( ) ;	Връща псевдоним за елемента
const_value_type &top ( ) const ;	с най – висок приоритет . Елемента не се премахва

set

Класа *set* поддържа множество, в което е възможно уникални ключове да се съпоставят на

стойности . Неговата шаблонна спецификация е :

```
template < class Key , class Comp = less < Key > ,
class Allocator = allocator < Key >> class set
```

Тук , *Key* е типа данни на ключовете , а *Comp* е функция , която сравнява два ключа . Той има следните конструктори :

```
explicit set ( const Comp &cmpfn = Comp ( ) ,
               const Allocator &a = Allocator ( ) ) ;
set ( const multiset < Key , Comp , Allocator > &ob ) ;
template < class InIter > set ( InIter start , InIter end ,
                                const Comp &cmpfn = Comp ( ) ,
                               const Allocator &a = Allocator ( ) ) ;
```

Първата форма конструира празно множество . Втората форма изгражда множество, което съдържа еднакви елементи с *ob* . Третата форма изгражда множество, което съдържа елементите в интервала определен от *start* и *end* . Функцията определена чрез *cmpfn* , ako е представена , определя нареждането на множеството . Следващите сравнителни оператори са дефинирани за *set* = = , < , < = , ! = , > и > =

Тук са показани член – функциите на *set* . В описанията , и двете *key_type* и *value_type* са *typedef* за *Key* :

Член	Описание
`iterator begin ( ) ;`	Връща итератор за първия елемент в множеството
`const_iterator begin ( ) const ;`	
`void clear ( ) ;`	Премахва всички елементи от множеството
`size_type count ( const key_type &k ) const ;`	Връща броя на срещанията на *к* в множеството
`bool empty ( ) const ;`	Връща *true* ако извиканото множеството е празно и *false* в противен случай
`iterator end ( ) ;`	Връща итератор към края на списъка
`const_iterator end ( ) const ;`	
`pair < iterator , iterator > equal_range ( const key_type &k ) const ;`	Връща двойка итератори които указват първия и последния елемент в множеството което съдържа определен ключ
`void erase ( iterator i ) ;`	Премахва елемента указван от *i*
`void erase ( iterator start , iterator end ) ;`	Премахва елементите в интервала *start – end*
`size_type erase ( const key_type &k ) ;`	Премахва от множеството елементите които имат ключове със стойност *к* . Връща се броя на премахнатите елементи
`iterator find ( const key_type &k ) const ;`	Връща итератор към определен ключ . Ако ключа не е открит се връща итератор към края на множеството
`allocator_type get_allocator ( ) const ;`	Връща алокатор за множеството
`iterator insert ( iterator i , const value_type &val ) ;`	Вмъква *val* на или след елемента определен от *i* . Дублиращите се елементи не се вмъкват . Връща се итератор към елемента
`template < class InIter > void insert ( InIter start , InIter end ) ;`	Вмъква интервал от елементи . Дублиращите се елементи не се вмъкват
`pair < iterator , bool > insert ( const value_type &val ) ;`	Вмъква *val* в извикваното множество . Връща се итератор към елемента . Елемента се вмъква ако все още не е съществувал . Ако елемента се вмъкне се връща двойка < *iterator* , *true* > . В противен случай се връща < *iterator* , *false* >

key_compare key_comp ( ) const ;	Връща функция обект която сравнява ключове
iterator lower_bound (const key_type &k) const ;	Връща итератор към първия елемент в множеството със ключ равен или по – голям от *к*
size_type max_size ( ) const ;	Връща максималния брой елементи които може да съдържа множеството
reverse_iterator rbegin ( ) ;	Връща обратен итератор към
const_reverse_iterator rbegin ( ) const ;	края на множеството
reverse_iterator rend ( ) ;	Връща обратен итератор към
const_reverse_iterator rend ( ) const ;	началото на множеството
size_type size ( ) const ;	Връща текущия брой елементи в множеството
void swap (multiset < Key , Comp , Allocator > &ob) ;	Разменя елементите съхранявани в извикваното множество със тези в *ob*
iterator upper_bound (const key_type &k) const ;	Връща итератор към първия елемент в множеството със ключ по – голям от *к*
value_compare value_comp ( ) const ;	Връща функция обект, която сравнява стойности

stack

Класа *stack* поддържа стек . Неговата шаблонна спецификация е :

template < class T , class Container = deque < T >>
class stack

Тук *T* е типа на съхраняваните данни , а *Container* е типа контейнер използван да съдържа опашката . Той има следния конструктор :

explicit stack (const Container &cnt = Container ( )) ;

Конструктора *stack ( )* създава празен стек . По – подразбиране той използва *deque* като контейнер Контейнера се съдържа в защитен обект наречен с от тип *Container* . Следващите сравнителни оператори са дефинирани за *stack* = = , < , < = , ! = , > и > =

***stack* съдържа следните член – функции :**

Член	Описание
bool empty ( ) const ;	Връща *true* ако извикания стек е празен и *false* в противен случай
void pop ( ) ;	Премахва върха на стека който технически е последния елемент в контейнера
void push (const T &val) ;	Слага елемент към края на стека Последния елемент в контейнера представлява върха на стека
size_type size ( ) const ;	Връща текущия брой елементи в стека
value_type &top ( ) ;	Връща псевдоним за върха на
const_value_type &top ( ) const ;	стека , който е последния елемент в контейнера . Елемента не се премахва

vector

Класа *vector* поддържа динамичен масив . Неговата шаблонна спецификация е :

template < class T , class Allocator = allocator < T >>
class vector

Тук *T* е типа на съхраняваните данни , а *Allocator* определя алокатора . Той има следните конструктори :

explicit vector (const Allocator &a = Allocator ( )) ;
explicit vector (size_type num , const T &val = T ( ) ,
const Allocator &a = Allocator ( )) ;

vector (const vector < T , Allocator > &ob) ;
template < class InIter > vector (InIter start , InIter end ,
const Allocator &a = Allocator ( )) ;

Първата форма конструира празен вектор . Втората форма конструира вектор , който има *num* елемента със стойност *val* . Третата форма изгражда вектор, който има еднакви елементи с *ob* . Четвъртата форма изгражда вектор, който съдържа елементите в интервала определен от *start* и *end* . Следващите сравнителни оператори са дефинирани за *vector* = = , < , < = , ! = , > и > = *vector* съдържа следните член функции :

Член	Описание
template < class InIter > void assign (InIter start , InIter end) ;	Присвоява на вектора поредицата дефинирана от *start* и *end*
template < class Size , class T > void assign (Size num , const T &val = T ( )) ;	Присвоява на вектора *num* елемента със стойност *val*
reference at (size_type i) ;	Връща псевдоним за елемента
const_reference at (size_type i) const ;	определен чрез *i*
reference back ( ) ;	Връща псевдоним за последния
const_reference back ( ) const ;	елемент във вектора
iterator begin ( ) ;	Връща итератор за първия
const_iterator begin ( ) const ;	елемент във вектора
size_type capacity ( ) const ;	Връща текущия капацитет на вектора . Това е броя елементи които може да съдържа преди да трябва да заделя повече памет
void clear ( ) ;	Премахва всички елементи от вектора
bool empty ( ) const ;	Връща *true* ако извикания вектор е празен и *false* в противен случай
iterator end ( ) ;	Връща итератор към края
const_iterator end ( ) const ;	на вектора
iterator erase (iterator i) ;	Премахва елемента указван от *i* връща итератор към елемента след премахнатия
iterator erase (iterator start , iterator end) ;	Премахва елементите в интервала *start – end* ,връща итератор към елемента след последния премахнат елемент
reference front ( ) ;	Връща псевдоним към първия
const_reference front ( ) const ;	елемент във вектора
allocator_type get_allocator ( ) const ;	Връща алокатор за вектора
iterator insert (iterator i , const T &val = T ( )) ;	Вмъква *val* непосредствено преди елемента определен от *i* Връща се итератор към елемента
void insert (iterator i , size_type num , const T &val) ;	Вмъква *num* копия от *val* непосредствено преди елемента определен от *i*
template < class InIter > void insert (iterator i , InIter start , InIter end) ;	Вмъква поредицата определена от *start – end* непосредствено преди елемента определен от *i*
size_type max_size ( ) const ;	Връща максималния брой елементи които може да съдържа вектора
reference operator [ ] (size_type i) const ;	Връща псевдоним за *i* - тия
const_reference operator [ ] (size_type i) const ;	елемент
void pop_back ( ) ;	Премахва последния елемент във вектора
void push_back (const T &val) ;	Добавя елемент със стойност *val* към края на вектора

reverse_iterator rbegin ( ) ; Връща обратен итератор към
const_reverse_iterator rbegin ( ) const ; края на вектора
reverse_iterator rend ( ) ; Връща обратен итератор към
const_reverse_iterator rend ( ) const ; началото на вектора
void reserve (size_type num) ; Установява капацитета на вектора така че да е равен най – малко на *num*
void resize (size_type num , T val = T ( )) ; Променя размера на вектора към този определен от *num* . Ако вектора трябва да бъде удължен , тогава елементите със стойността определена от *val* се добавят към края
size_type size ( ) const ; Връща текущия брой елементи във вектора
void swap (vector < T , Allocator > &ob) ; Разменя елементите съхранявани във извиквания вектор със тези в *ob*

STL също съдържа специализиран вектор за Булеви стойности . Той включва цялата функционалност на vector и добавя тези два члена :

void flip ( ) ; Инвертира всички битове във вектора
static void swap (reference i , reference j) ; Разменя битовете определени от i и j

Алгоритми

Стандартните алгоритми са описани тук . Всички алгоритми са шаблонни функции . За опростяване на описанията няма да бъде представяна шаблонната спецификация . Вместо това описанията ще използват навсякъде следните общи типови имена :

Общо име	Представя
BiIter	Двупосочен итератор
ForIter	Прав итератор
InIter	Входен итератор
OutIter	Изходен итератор
RandIter	Итератор с произволен достъп
T	Някакъв тип данни
Size	Някакъв тип integer
Func	Някакъв тип функция
Generator	Функция която генерира обекти
BinPred	Бинарен предикат
UnPred	Унарен предикат
Comp	Сравнителна функция която връща резултата от *arg1 < arg2*

adjacent_find

```
ForIter adjacent_find ( ForIter start , ForIter end ) ;
ForIter adjacent_find ( ForIter start , ForIter end ,
                        BinPred pfn ) ;
```

Алгоритъма *adjacent_find ( )* търси съседни съответстващи елементи в поредица определена от *start* и *end* и връща итератор за първия елемент . Ако не е открита съседна двойка се връща *end* . Първата версия търси за еквивалентни елементи . Втората версия позволява да определите метод за определяне на съвпадащите елементи

binary_search

```
bool binary_search ( ForIter start , ForIter end ,
                     const T &val ) ;
bool binary_search ( ForIter start , ForIter end ,
                     const T &val , Comp cmpfn ) ;
```

Алгоритъма *binary_search ( )* извършва двоично търсене в подредената редица започваща от *start* и завършваща с *end* за стойността определена от *val* . Тя връща *true* ,ако открие *val* и *false* в

противен случай . Първата версия сравнява елементите в определената редица за еднаквост . Втората версия позволява да определите ваша сравнителна функция .

copy

OutIter copy (InIter start , InIter end , OutIter result) ;

Алгоритъма *copy ( )* копира последователноста започваща от *start* и завършваща с *end* слагайки резултата в редицата указвана от *result* . Тя връща указател за края на резултатната последователност . Интервала, който ще бъде копиран не трябва да превишава *result* .

copy_backward

BiIter2 copy_backward (BiIter start , BiIter end ,
BiIter2 result) ;

Алгоритъма *copy_backward ( )* е същия като *copy ( )* с изключение на това, че премества елементите от края в началото на поредицата .

count

size_t count (InIter start , InIter end , const T &val) ;

Алгоритъма *count ( )* връща броя елементи в поредицата започваща от *start* и завършваща с *end* които съвпадат с *val*

count_if

size_t count_if (InIter start , InIter end , UnPred pfn) ;

Алгоритъма *count_if ( )* връща броя елементи в поредицата започваща от *start* и завършваща с *end* за които унарния предикат *pfn* връща *true*

equal

bool equal (InIter1 start1 , InIter1 end1 , InIter2 start2) ;

Алгоритъма *equal ( )* определя дали два интервала са еднакви . Интервала определен от *start1* и *end1* се сравнява с поредицата указвана от *start2* . Ако са еднакви се връща *true* , в противен случай се
връща *false*

equal_range

pair < ForIter , ForIter > equal_range (ForIter start ,
ForIter end , const T &val) ;
pair < ForIter , ForIter > equal_range (ForIter start ,
ForIter end , const T &val , Comp cmpfn) ;

Алгоритъма *equal_range ( )* връща интервал, в който може да бъде вмъкнат елемент в последователността без да се наруши подреждането на редицата . Региона , в който се търси за такъв интервал е определен от *start* и *end* . Стойността се подава във *val* . За да определите ваш търсещ критерии , определете сравнителната функция *cmpfn* . Шаблонния клас *pair* е помощен, който може да съдържа двойка обекти с техни *first* и *second* членове

fill и fill_n

void fill (ForIter start , ForIter end , const T &val) ;
void fill_n (ForIter start , Size num , const T &val) ;

Алогоритмите *fill ( )* и *fill_n ( )* попълват интервал със стойността определена от *val* . За *fill ( )* интервала е определен от *start* и *end* . За *fill_n ( )* интервала започва от *start* и продължава *num* елемента .

find

InIter find (InIter start , InIter end , const T &val) ;

Алгоритъма *find ( )* претърсва интервала от *start* до *end* за стойността определена от *val* . Той връща итератор към първото съвпадение на елемент или *end* ,ако стойността не е в редицата

find_end

FwdIter1 find_end (ForIter1 start1 , ForIter1 end1 ,
ForIter2 start2 , ForIter2 end2) ;
FwdIter1 find_end (ForIter1 start1 , ForIter1 end1 ,
ForIter2 start2 , ForIter2 end2
BinPred pfn) ;

Алгоритъма *find_end ( )* търси последния итератор от подниза определен от *start2* и *end2* в интервала *start1 - end1* . Ако редицата е открита се връща итератор към последния елемент в редицата . В противен случай се връща итератора *end1* . Втората форма ви позволява да

определите бинарен предикат, който определя кога елементите съвпадат .

find_first_of

```
FwdIter1 find_first_of ( ForIter1 start1 , ForIter1 end1 ,
                         ForIter2 start2 , ForIter2 end2 ) ;
FwdIter1 find_first_of ( ForIter1 start1 , ForIter1 end1 ,
                         ForIter2 start2 , ForIter2 end2
                         BinPred pfn ) ;
```

Алгоритъма *find_first_of ( )* открива първия елемент в поредицата определена от *start1* и *end1* , който съвпада с елемент от интервала *start2* и *end2* . Ако няма съвпадащ елемнт се връща итератора *end1* . Втората форма ви позволява да определите бинарен предикат, който определя кога елементите съвпадат .

find_if

```
InIter find_if ( InIter start , InIter end , UnPred pfn ) ;
```

Алгоритъма *find_if ( )* претърсва интервала *start – end* за елемент, за който унарния предикат *pfn* връща *true* . Той връща итератор към първото срещане или *end* ,ако стойността не е в поредицата .

for_each

```
Func for_each ( InIter start , InIter end , Func fn ) ;
```

Алгоритъма *for_each ( )* добавя функцията *fn* към интервала от елементи определен от *start – end* . Тя връща *fn* .

generate и generate_n

```
void generate ( ForIter start , ForIter end , Generator fngen ) ;
void generate_n ( OutIter start , Size num , Generator fngen ) ;
```

Алогоритмите *generate ( )* и *generate_n ( )* присвояват елементи в интервала от стойности върнати от генераторната функция . За *generate ( )* интервала, който е бил присвоен е определен от *start* и *end* .

За *generate_n ( )* интервала започва от *start* и продължава *num* елемента . Генераторната функция се подава във *fngen* . Тя няма параметри .

includes

```
bool includes ( InIter1 start1 , InIter1 end1 , InIter2 start2 ,
                InIter2 end2 ) ;
bool includes ( InIter1 start1 , InIter1 end1 , InIter2 start2 ,
                InIter2 end2 , Comp cmpfn ) ;
```

Алгоритъма *includes ( )* определя дали поредицата *start1 – end1* включва всички елементи от поредицата *start2 – end2* . Тя връща *true* ,ако са открити всички елементи и *false* в противен случай . Втората форма ви позволява да определите сравнителна функция която определя кога един елемент е по – малък от друг

inplace_merge

```
void inplace_merge ( BiIter start , BiIter mid , BiIter end ) ;
void inplace_merge ( BiIter start , BiIter mid , BiIter end ,
                     Comp cmpfn ) ;
```

Алгоритъма *inplace_merge ( )* слива в единствена редица интервала определен от *start* и *mid* с интервала определен от *mid* и *end* . И двата интервала трябва да бъдат сортирани във нарастващ ред . След изпълнението , резултатната поредица е сортирана в нарастващ ред . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг .

iter_swap

```
void iter_swap ( ForIter1 i , ForIter2 j ) ;
```

Алгоритъма *iter_swap ( )* разменя стойностите указвани чрез неговите два итератора аргументи .

lexicographical_compare

```
bool lexicographical_compare ( InIter1 start1 , InIter1 end1 ,
                               InIter2 start2 , InIter2 end2 ) ;
bool lexicographical_compare ( InIter1 start1 , InIter1 end1 ,
                               InIter2 start2 , InIter2 end2
```

Comp cmpfn) ;

Алгоритъма lexicographical_compare ( ) **сравнява по азбучен ред редицата определена от** ***start1*** **и** ***end1*** **със редицата определена от** ***start2*** **и** ***end2*** **. Тя връща** ***true*** **ako първата редица е лексикографски по – малка от втората**

Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг .

lower_bound

ForIter lower_bound (ForIter start , ForIter end ,
const T &val) ;
ForIter lower_bound (ForIter start , ForIter end ,
const T &val Comp cmpfn) ;

Алгоритъма ***lower_bound ( )*** **открива първата точка в редицата определена от** ***start*** **и** ***end*** **,която не е**
по – малка от ***val*** **. Тя връща итератор за тази точка . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг .**

make_heap

void make_heap (RandIter start , RandIter end) ;
void make_heap (RandIter start , RandIter end ,
Comp cmpfn) ;

Алгоритъма ***make_heap ( )*** **изгражда хийп от редицата определена от** ***start*** **и** ***end*** **. Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг .**

max

const T &max (const T &i , const T &j) ;
const T &max (const T &i , const T &j , Comp cmpfn) ;

Алгоритъма ***max ( )*** **връща максималната от две стойности . Втората форма ви позволява да определите сравнителна функция , която определя кога един елемент е по – малък от друг .**

max_element

ForIter max_element (ForIter start , ForIter last) ;
ForIter max_element (ForIter start , ForIter last ,
Comp cmpfn) ;

Алгоритъма ***max_element ( )*** **връща итератор към максималния елемент в интервала** ***start*** **и** ***last*** **. Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е**
по – малък от друг

merge

OutIter merge (InIter1 start1 , InIter1 end1 , InIter2 start2 ,
InIter2 end2 , OutIter result) ;
OutIter merge (InIter1 start1 , InIter1 end1 , InIter2 start2 ,
InIter2 end2 , OutIter result , Comp cmpfn) ;

Алгоритъма ***merge ( )*** **слива две подредени редици , поставяйки резултата в трета поредица . Редиците, които се сливат се дефинират от** ***start1*** **,** ***end1*** **и** ***start2*** **,** ***end2*** **. Резултата се слага в редица указвана от** ***result*** **. Връща се итератор за края на резултатната редица . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг**

min

const T &min (const T &i , const T &j) ;
const T &min (const T &i , const T &j , Comp cmpfn) ;

Алгоритъма ***min ( )*** **връща минималната от две стойности . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг .**

min_element

ForIter min_element (ForIter start , ForIter last) ;
ForIter min_element (ForIter start , ForIter last ,
Comp cmpfn) ;

Алгоритъма ***min_element ( )*** **връща итератор към минималния елемент в интервала** ***start*** **и** ***last*** **. Втората форма ви позволява да определите сравнителна функция, която определя кога един**

елемент е по – малък от друг .

mismatch

pair < InIter1 , InIter2 > mismatch (InIter1 start1 ,
InIter1 end1 ,InIter2 start2) ;
pair < InIter1 , InIter2 > mismatch (InIter1 start1 ,
InIter1 end1 ,InIter2 start2 , BinPred pfn) ;

Алгоритъма *mismatch ( )* открива първото несъвпадение между елементите в две редици . Връща се итератор за двата елемента . Ако не е открито несъвпадение се връщат итератори за двата последни елемента във всяка поредица . Втората форма ви позволява да определите бинарен предикат, който определя кога един елемент е равен на друг . Шаблонния клас *pair* съдържа два члена данни наречени *first* и *second* ,които съдържат двойка стойности .

next_permutation

bool next_permutation (BiIter start , BiIter end) ;
bool next_permutation (BiIter start , BiIter end ,
Comp cmpfn) ;

Алгоритъма *next_permutation ( )* изгражда следваща пермутация на редица . Пермутациите се генерират присвоявайки сортирана редица от ниско към високо представяне на първата пермутация . Ако не съществува следваща пермутация *next_permutation ( )* сортира поредицата на нейната първа пермутация и връща *false* . В противен случай се връща *true* . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг

nth_element

void nth_element (RandIter start , RandIter element ,
RandIter end) ;
void nth_element (RandIter start , RandIter element ,
RandIter end , Comp cmpfn) ;

Алгоритъма *nth_element ( )* подрежда редицата определена от *start* и *end* така че всички елементи по – малки от *element* идват преди елемента и всички елементи по – големи от *element* идват след него . Втората форма ви позволява да определите сравнителна функция , която определя кога един елемент е по – малък от друг

partial_sort

void partial_sort (RandIter start , RandIter mid ,
RandIter end) ;
void partial_sort (RandIter start , RandIter mid ,
RandIter end , Comp cmpfn) ;

Алгоритъма *partial_sort ( )* сортира редицата *start – end* . Обаче след изпълнението само елементите в поредицата *start – mid* ще бъдат сортирани в ред . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг

partial_sort_copy

RandIter partial_sort_copy (InIter start , InIter end ,
RandIter res_start , RandIter res_end) ;
RandIter partial_sort_copy (InIter start , InIter end ,
RandIter res_start , RandIter res_end ,
Comp cmpfn) ;

Алгоритъма *partial_sort_copy ( )* сортира интервала *start – end* и след това копира елементите, които ще бъдат сложени в резултатната редица определена от *res_start* до *res_end* . Връща се итератор за последния елемент копиран в резултатната поредица . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг .

partition

BiIter partition (BiIter start , BiIter end , UnPred pfn) ;

Алгоритъма *partition ( )* подрежда редицата определена от *start* до *end* така че всички елементи за които предиката *pfn* връща *true* идват преди онези, за които предиката връща *false* . Тя връща итератор за началото на елементите за които предиката е *false* .

pop_heap

void pop_heap (RandIter start , RandIter end) ;
void pop_heap (RandIter start , RandIter end , Comp cmpfn) ;

Алгоритъма *pop_heap ( )* разменя *first* и *last – 1* елементити и тогава преизгражда хийпа . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг

prev_permutation

```
bool prev_permutation ( BiIter start , BiIter end ) ;
bool prev_permutation ( BiIter start , BiIter end ,
                        Comp cmpfn ) ;
```

Алгоритъма *prev_permutation ( )* конструира предишната пермутация на редицата . Пермутациите се генерират приемайки сортирана редица от ниско към високо представяне за първа пермутация . Ако не съществува предишна пермутация , *prev_permutation ( )* сортира редицата като нейна последна пермутация и връща *false* . В противен случай връща *true* . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг .

push_heap

```
void push_heap ( RandIter start , RandIter end ) ;
void push_heap ( RandIter start , RandIter end ,
                 Comp cmpfn ) ;
```

Алгоритъма *push_heap ( )* слага елемент в края на хийпа . Интервала определен от *start* до *end* е приет да представлява валиден хийп . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг .

random_shuffle

```
void random_shuffle ( RandIter start , RandIter end ) ;
void random_shuffle ( RandIter start , RandIter end ,
                      Generator rand_gen ) ;
```

Алгоритъма *random_shuffle ( )* разбърква редицата определена от *start – end* . Втората форма ви позволява да определя потребителски генератор на случайни числа . Тази функция трябва да има следната обща форма :

```
rand_gen ( num ) ;
```

Тя трябва да връща случайно число между 0 и *num* .

remove , remove_if , remove_copy и

remove_copy_If

```
ForIter remove ( ForIter start , ForIter end , const T &val ) ;
ForIter remove_if ( ForIter start , ForIter end , UnPred pfn ) ;
OutIter remove_copy ( InIter start , InIter end , OutIter result ,
                      const T &val ) ;
OutIter remove_copy_if ( InIter start , InIter end ,
                         OutIter result , const T &val ,
                         UnPred pfn ) ;
```

Алгоритъма *remove ( )* премахва елементи от определения интервал, които са равни на *val* . Връща итератор за края на останалите елементи .

Алгоритъма *remove_if ( )* премахва елементите от определения интервал, за които предиката *pfn* е *true* . Връща итератор за края на останалите елементи .

Алгоритъма *remove_copy ( )* копира елементите от определения интервал, които са равни на *val* и слага резултата в поредицата определена от *result* . Връща итератор за края на *result* .

Алгоритъма *remove_copy_if ( )* копира елементите от определения интервал, за които предиката *pfn* е *true* и слага резултата в поредицата определена от *result* . Връща итератор за края на *result*

replace , replace_copy , replace_if и

replace_copy_If

```
void replace ( ForIter start , ForIter end , const T &old ,
               Const T &new ) ;
void replace_if ( ForIter start , ForIter end , UnPred pfn ,
               Const T &new ) ;
OutIter replace_copy ( InIter start , InIter end ,
                       OutIter result ,
                       const T &old , Const T &new ) ;
OutIter replace_copy_if ( InIter start , InIter end ,
                          OutIter result ,
```

UnPred pfn , Const T &new) ;

В определения интервал алгоритъма *replace ( )* разменя елементите , които имат стойност *old* с елементите, които имат стойност *new* .
В определения интервал алгоритъма *replace_if ( )* разменя елементите, за които предиката *pfn* е *true* с елементите които имат стойност *new* .
В определения интервал алгоритъма *replace_copy ( )* копира елементи в *result* . В този процес разменя елементите със стойност *old* с елементи със стойност *new* . Първоначалния интервал е непроменен . Връща итератор за края на *result* .
В определения интервал алгоритъма *replace_copy_if ( )* копира елементи в *result* В този процес разменя елементите, за които предиката *pfn* връща *true* със елементи със стойност *new* . Първоначалния интервал е непроменен . Връща итератор за края на *result*

reverse и reverse_copy

void reverse (BiIter start , BiIter end) ;
OutIter reverse_copy (BiIter first , BiIter last ,
OutIter result) ;

Алгоритъма *reverse ( )* обръща реда на интервала определен от *start* и *end*
Алгоритъма *reverse_copy ( )* копира в обърнат ред интервала определен от *start* и *end* и съхранява ресултата в *result* . Връща итератор за края на *result* .

rotate и rotate_copy

void rotate (ForIter start , ForIter mid , ForIter end) ;
OutIter rotate_copy (ForIter start , ForIter mid , ForIter end OutIter result) ;

Алгоритъма *rotate ( )* ротира наляво елементите в интервала от *start* до *end* ,така че елемента определен от *mid* става новия първи елемент .
Алгоритъма *rotate_copy ( )* копира интервала от *start* до *end* съхранявайки резултата във *result* . По време на процеса ротира наляво елементите, така че елемента определен от *mid* става новия първи елемент . Връща итератор за края на *result*

search

ForIter1 search (ForIter1 start1 , ForIter1 end1 ,
ForIter2 start2 , ForIter2 end2) ;
ForIter1 search (ForIter1 start1 , ForIter1 end1 ,
ForIter2 start2 , ForIter2 end2 ,BinPred pfn);

Алгоритъма *search ( )* търси за поредица във редица . Редицата, която се претърсва е определена от *start1* и *end1* . Поредицата, която се търси е определена от *start2* и *end2* . Ако поредицата е открита се връща итератор за нейното начало . В противен случай се връща *end1* . Втората форма ви позволява да определите бинарен предикат, който определя кога един елемент е равен на друг

search_n

ForIter search_n (ForIter start , ForIter end , Size num ,
Const T &val) ;
ForIter search_n (ForIter start , ForIter end , Size num ,
Const T &val , BinPred pfn) ;

Алгоритъма *search_n ( )* търси за редица от подобни на *num* елементи в редица . Редицата, която се претърсва е определена от *start* и *end* . Ако поредицата е открита се връща итератор за нейното начало .В противен случай се връща *end* . Втората форма ви позволява да определите бинарен предикат, който определя кога един елемент е равен на друг

set_difference

OutIter set_difference (InIter1 start1 , InIter1 end1 ,
InIter2 start2 , InIter2 end2 ,
OutIter result) ;
OutIter set_difference (InIter1 start1 , InIter1 end1 ,
InIter2 start2 , InIter2 end2 ,
OutIter result , Comp cmpfn) ;

Алгоритъма *set_difference ( )* произвежда редица която съдържа разликата между две подредени множества определени от *start1* , *end1* и *start2* , *end2* . Така множеството определено от *start2* , *end2* се изважда от множеството *start1* , *end1* . Резултата е подреден и сложен във *result* . Връща итератор за края на *result* . Втората форма ви позволява да определите сравнителна функция , която определя кога един елемент е по – малък от друг

set_intersection

OutIter set_intersection (InIter1 start1 , InIter1 end1 ,
InIter2 start2 , InIter2 end2 ,
OutIter result) ;
OutIter set_intersection (InIter1 start1 , InIter1 end1 ,
InIter2 start2 , InIter2 end2 ,
OutIter result , Comp cmpfn) ;

Алгоритъма *set_intersection ( )* произвежда редица, която съдържа сечението на две подредени множества определени от *start1 , end1* и *start2 , end2* . Това са елементите открити и в двете множества . Резултата е подреден и сложен във *result* . Връща итератор за края на *result* . Втората форма ви позволява да определите сравнителна функция която определя кога един елемент е по – малък от друг .

set_symetric_difference

OutIter set_symetric_difference (InIter1 start1 , InIter1 end1 ,
InIter2 start2 , InIter2 end2 ,
OutIter result) ;
OutIter set_symetric_difference (InIter1 start1 , InIter1 end1 ,
InIter2 start2 , InIter2 end2 ,
OutIter result , Comp cmpfn) ;

Алгоритъма *set_symetric_difference ( )* произвежда редица, която съдържа симетричната разлика между две подредени множества определени от *start1 , end1* и *start2 , end2* . Това е резултатно множество съдържа само онези елементи които не са общи за двете множества . Резултата е подреден и сложен в *result* . Връща итератор за края на *result* .Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг .

set_union

OutIter set_union (InIter1 start1 , InIter1 end1 ,
InIter2 start2 , InIter2 end2 ,OutIter result);
OutIter set_union (InIter1 start1 , InIter1 end1 ,
InIter2 start2 , InIter2 end2 , OutIter result
Comp cmpfn) ;

Алгоритъма *set_union ( )* произвежда редица , която съдържа обединението на две подредени множества определени от *start1 , end1* и *start2 , end2* . Така резултатното множество съдържа тези елементи, които са и в двете множества . Резултата е подреден и сложен във *result* . Връща итератор за края на *result* . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг

sort

void sort (RandIter start , RandIter end) ;
void sort (RandIter start , RandIter end , Comp cmpfn) ;

Алгоритъма *sort ( )* сортира интервала определен от *start* и *end* . Втората форма ви позволява да определите сравнителна функция , която определя кога един елемент е по – малък от друг .

sort_heap

void sort_heap (RandIter start , RandIter end) ;
void sort_heap (RandIter start , RandIter end , Comp cmpfn) ;

Алгоритъма *sort_heap ( )* сортира хийп определен от *start* и *end* . Втората форма ви позволява да определите сравнителна функция , която определя кога един елемент е по – малък от друг

stable_partition

BiIter stable_partition (BiIter start , BiIter end ,
BinPred pfn) ;

Алгоритъма *stable_partition ( )* подрежда редицата определена от *start* и *end* ,така че всички елементи за които предиката определен от *pfn* връща *true* идват преди тези, за които предиката връща *false* . Разделянето е постоянно . Това означава, че съответното подреждане се запазва . Връща итератор към началото на елементите за които предиката е *false*

stable_sort

void stable_sort (RandIter start , RandIter end) ;
void stable_sort (RandIter start , RandIter end , Comp cmpfn);

Алгоритъма *stable_sort ( )* подрежда интервала *start – end* . Сортирането е постоянно . Това означава,

че еднаквите елементи не се пренареждат . Втората форма ви позволява да определите сравнителна функция, която определя кога един елемент е по – малък от друг

swap

void swap (T &i , T &j) ;

Алгоритъма *swap ( )* разменя стойностите отнасящи се за *i* и *j* .

swap_ranges

ForIter2 swap_ranges (ForIter1 start1 , ForIter1 end1 ,
ForIter2 start2) ;

Алгоритъма *swap_ranges ( )* разменя елементите в интервала определен от *start1* и *end1* с елементите в редицата започваща със *start2* . Връща указател към края на редицата определена от *start2* .

transform

OutIter transform (InIter start , InIter end , OutIter result ,
Func unaryfunc) ;
OutIter transform (InIter1 start1 , InIter1 end1 , InIter2
start2 , OutIter result , Func binaryfunc) ;

Алгоритъма *transform ( )* прибавя функция към интервала от елементи и съхранява резултата във *result* . В първата форма интервала е определен от *start* и *end* . Функцията , която ще бъде добавена е определена от *unaryfunc* . Тази функция получава стойност на елемент като нейн параметър и трябва да връща неговата трансформация . Във втората форма , трансформацията е добавена чрез използване на бинарна операторна функция, която получава елемент от редицата и го преобразува във нейн първи параметър и елемент от втората редица за найн втори параметър . И двете версии връщат итератор за края на резултатната редица

Един от най – интересните алгоритми е *transform ( )* понеже той изменя всеки елемент в интервала съгласно предоставената функция . Например , следващата програма използва проста преобразуваща функция наречена *xform ( )* да вдигне на квадрат съдържанието на списък . Забележете че резултатната редица се съхранява в същия списък, който доставя първоначалната редица :

```
// пример за трансформиращ алгоритъм
# include < iostream >
#include < list >
#include < algorithm >
using namespace std ;
// проста преобразуваща функция
int xform ( int i ) {
  return i * i ; // първоначалната стойност на квадрат
}
int main ( )
{
  list < int > xl ;
  int i ;
  // слага стойности във списъка
  for ( i = 0 ; i < 10 ; i ++ )  xl . push_back ( i ) ;
  cout << " Original contents of xl : " ;
  list < int > : : iterator p = xl . begin ( ) ;
  while ( p != xl . end ( ) ) {
     cout << *p << "  " ;
     p++ ;
  }
  cout << endl ;
  // трансформира xl
  p = transform ( xl . begin ( ) , xl . end ( ) , xl . begin ( ) ,
                xform ) ;
  cout << " Transformed contents of xl : " ;
  p = xl . begin ( ) ;
  while ( p != xl . end ( ) ) {
     cout << *p << "  " ;
```

```
        p++ ;
      }
      return 0 ;
    }
```

Изхода от тази програма е :

```
Original contents of xl : 0 1 2 3 4 5 6 7 8 9
Transformed contents of xl : 0 1 4 9 16 25 36 49 64 81
```

Както можете да видите всеки елемент в *xl* е бил повдигнат на квадрат .

unique и unique_copy

```
ForIter unique ( ForIter start , ForIter end ) ;
ForIter unique ( ForIter start , ForIter end , BinPred pfn ) ;
OutIter unique_copy ( ForIter start , ForIter end ,
                      OutIter result ) ;
OutIter unique_copy ( ForIter start , ForIter end ,
                      OutIter result , BinPred pfn ) ;
```

Алгоритъма *unique ( )* елиминира дублиращите се елементи от определения интервал . Втората форма Ви позволява да определите бинарен предикат, който определя кога един елемент е равен на друг *unique ( )* връща итератор за края на интервала . Алгоритъма *unique_copy ( )* копира интервала определен от *start* и *end* , елиминирайки дублиращите се елементи по време на процеса . Резултата е сложен в *result* Втората форма позволява да определите бинарен предикат, който определя кога един елемент е равен на друг . *unique_copy ( )* връща итератор за края на интервала .

upper_bound

```
ForIter upper_bound ( ForIter start , ForIter end ,
                      const T &val ) ;
ForIter upper_bound ( ForIter start , ForIter end ,
                      const T &val , Comp cmpfn ) ;
```

Алгоритъма *upper_bound ( )* открива последната точка в редицата определена от *start* и *end* ,която не е по – голяма от *val* . Връща итератор за тази точка . Втората форма ви позволява да определите сравнителна функция която определя кога един елемент е по – малък от друг

10. Наследяване

В C++ един клас може да наследи характеристиките на друг.Наследеният клас обикновено го наричат базов клас.Наследяващия клас е представен като производен клас. Когато един клас наследява друг се формира класова йерархия .Общата форма за наследяване на клас е:

```
class име_на_клас : достъп име_на_базов_клас {

 // ….

};
```

Тук достъп определя как се наследява базовия клас и той трябва да бъде или *private, public* или *protected* .Той също може да бъде пропуснат и в този случай се приема,че е *public* , ако базовия клас е *struct* , или *private* ,ако базовия клас е *class*. За да наследите повече от един клас ,използвайте разделен със запетайки списък.Ако достъп е *public* , всички *public* и *private* членове от базовия клас стават *public* и *protected* членове на производния клас. Ако достъп е *private* ,всички *public* и *protected* членове на базовия клас стават *protected* членове на производния клас.В следващата класова йерархия *derived* наследява *base* като *private* . Това значи че *i* става *private* член на *derived:*

```
class base{

public:

int  i ;

};

class  derived : private base{

int j;

public:

derived (int a)  {j =i = a;}

int getj ( )  { return j ; }

int  geti ( )  { return i ; }   // ОК derived има достъп до i

};

derived ob(9);      //създава обект  derived

cout  <<   ob.geti ( )  << ob .getj ( ) ;

// ob.i = 10 ;  //  грешка   i  е private за derived
```

<u>Обединение</u> е тип клас,в които всички членове - данни споделят едно и също място в паметта.В C++ обединението може да включва и член-функции и данни. Всички членове на обединението са *public* по подразбиране . За да създадете *private* елементи, вие трябва да използвате ключовата дума *private*. Общата форма за деклариране на обединение е:

```
union  име_на_клас{

// public  членове  по  подразбиране

private:

// private  членове
```

} списък с обекти;

Има няколко ограничения,които се прилагат за обединенията.Първо обединение не може да наследи никакъв друг клас от никакъв тип. Обединение не може да бъде базов клас. Обединение не може да има виртуални член-функции.
Никакви членове не могат да бъдат декларирани като *static*.Обединение не може да има като член обект , които предефинира оператора =
Накрая никакъв обект не може да бъде член на обединение , ако класа на обекта точно определя коструктор или деструктор функции (обектите ,които имат само подразбиращи се конструктори и деструктори са разрешени). Има специален тип обединение в C++ наречено анонимно обединение. Декларацията на анонимното обединение не съдържа име на клас и никакви обекти от това обединение не са декларирани .Вместо това анонимното обединение просто казва на компилатора че неговите член-променливи споделят едно и също място в паметта .Обаче обръщането към променливите става директно , без използване на нормалния синтаксис със операторите точка и стрелка.Променливите, които правят анонимното обединение са на същото ниво на видимост като всяка друга променлива декларирана в същия блок. Това предполага ,че имената на променливите в обединението не трябва да влизат в конфликт с никакви други имена ,които са валидни в техния обсег.
Пример за анонимно обединение:

```
union { // анонимно обединение

int a;

float f;

};

a =10;

cout << f ;
```

Тук *a* и *f* споделят едно и също място в паметта. Както можете да видите имената на променливите в обединението се пишат директно без използване на операторите точка или стрелка . Всички ограничения, които се отнасят за обединенията се напълно отнасят и за анонимните обединения. В добавка анонимните обединения трябва да съдържат само данни никакви член-функции не са позволени. Анонимните обединения не могат да съдържат ключовите думи *private* или *protected*. Накрая анонимно обединение в обхвата на именовано пространство трябва да бъде декларирано като *static*.

Структура се създава чрез използване на ключовата дума struct.В C++ структура също дефинира клас .Единствената разлика между class и struct е че по подразбиране всички членове на структурата са public.За да направите член private , вие трябва да използвате ключовата дума private .Общата форма за деклариране на структура е тази:

```
struct име_на_структура : списък с наследяване {

// public членове по подразбиране

protected:

// private членове които се наследяват

private:

// private членове

} списък- със- обекти;
```

За правилно разбиране на принципа на наследяването е необходимо да знаете, че всички неща произлизат от някаква основна група. В C++ могат да се използват класове за представяне на йерархичните взаимоотношения, като даден клас е производен на друг, който е от по-високо ниво. Класът, от който производният клас наследява свойствата си, се нарича негов базов клас (base class). В декларацията на производния клас може да се посочи базовия клас,от който произлиза чрез добавяне на двоеточие, производен тип (derivation type) и наименованието на базовия клас след това на производния...
синтаксисът за деклариране на клас Kotka, производен на Bozainik е следният:
class Kotka : public Bozainik
Базовият клас винаги трябва да бъде деклариран преди производния, за да се избегне грешка при компилирането.

<u>Защита на данните на класа:</u>
Членовете на базовия клас, зададени като private (частни) не могат да се използват в производния клас. Това може да бъде коригирано чрез задаването им с public(публичен), въпреки че не е желателно те да бъдат достъпни във всички класове. По-добре е те да бъдат зададени като защитени елементи чрез използване на спецификатора protected (защитено). Така ще бъдат достъпни само за производните класове.
Спецификаторите за достъп са: public, private, protected

<u>Декларирането</u> на множество предефинирани (overloaded) методи-конструктори позволява при създаването на обект да бъдат зададени различен брой аргументи или пък аргументи от различен тип.

<u>Предефиниране на методи на базовия клас:</u>
В производния клас може да се декларира метод, който да предефинира (override) друг метод в съответстващия му базов клас. Декларацията на метода в производния клас трябва точно да съвпада с тази в базовия, за да бъде успешно предефинирането. Това означава, че типът на връщаните данни, името и аргументите трябва да бъдат еднакви. Ако в метода на базовия клас е използвана ключовата дума const, същата трябва да се използва и *n* производния.
Предефинирането на методи на базовия клас трябва да се извършва внимателно, за да се избегне неволното скриване на предефинирани методи. Един предефиниран метод в производен клас ще скрие всички предефинирани методи с това име в базовия клас.
Предефинирането на методи в производен клас може да доведе до скриване на предефинирани методи в съответстващия му базов клас.

<u>Извикване на предефинирани методи на базовия клас:</u>
Всеки базов метод може да бъде извикан чрез изрично въвеждане на класа, към който принадлежи, и името му, разделени от <u>оператора за област на видимост</u> (::)
Това може да се използва за извикване на методи на базовия клас, които са били предефинирани от методи на производен клас. Особено важно е да се спази правилната последователност на синтаксиса: името на базовия клас и операторът за област на видимост :: винаги трябва да стоят преди името на обекта. Синтаксисът е следният:

име-на-обект . име-на-базовия-клас::име-на-метода ( );

11. Полиморфизъм

Трите основополагащи стълба на обектно-ориентираното програмиране (ООП) са капсулирането, наследяването и полиморфизмът. Досега демонстрирах *капсулирането* на данни в клас на C++, както и наследяването на характеристиките на базовия клас от производните му класове. Сега ще обърнем внимание на последният от трите фундаментални принципи - *полиморфизмът.*
Терминът „полиморфизъм" (от гръцки език - „множество форми") описва възможността за придаване на различно значение или предназначение на дадена единица, в зависимост от контекста й. Полиморфизмът представлява процес,при който единен интерфейс се прилага за две или повече сходни (но технически различни) ситуации, като по този начин се реализира философията "един интерфейс, множество методи". Полиморфизмът е важен, защото позволява значително да се опростят сложните системи. Един добре дефиниран интерфейс може да се използва за достъп до няколко различни, но свързани помежду си действия, като при това се премахва изкуствено напластената сложност. Накратко, полиморфизмът позволява логическата връзка между подобни свързани действия да стане очевидна; така програмата става по-лесна за разбиране и поддръжка. Когато достъпът до сходни действия се извършва през общ интерфейс, трябва да помните по-малко неща...
Съществуват два термина:Това са *ранното свързване* (early binding) и *късното свързване* (late binding)
Ранното свързване се отнася към тези събития, които са известни по време на компилиране. В частност то се отнася към онези извиквания на функции, които могат да се определят при компилирането. Общностите, които се включват в ранното свързване, са "нормалните" функции, предефинираните (overloaded) функции, не-виртуалните член-функции и приятелските функции. Цялата адресна информация, която е необходима за извикването на подобни функции, е известна по време на компилиране. Главното предимство на ранното свързване (и причината поради, която то е толкова широко използвано) е, че то е много ефективно. Извикванията на функции, свързани по време на компилация, са най-бързия тип извиквания на функции.
Най-големият им недостатък е липсата на гъвкавост.
Късното свързване се отнася за събития, които се очаква да се случат по време на изпълнение. Извикването на късно свързана функция представлява извикване, при което адресът на функцията е неизвестен преди стартирането на програмата. В C++, виртуалната функция е обект за късно свързване. Когато се извършва достъп до една виртуална функция посредством указател към базов клас, програмата трябва да разбере по време на изпълнение, какъв тип обект е укачан и тогава да избере коя версия на предефинираната виртуална функция да изпълни.
Главното предимство на късното свързване е гъвкавостта по време на изпълнение.
Главният недостатък е,че е необходимо повече време за извикване на функцията...те са бавни функции
Предефинираните оператори в C++ могат да се разглеждат като полиморфни. Например знакът * може да означава както оператора за умножение, така и оператора за дереференция, в зависимост от контекста, в който е използван. Още по-голямо значение има фактът, че C++ предоставя възможността специфични обекти на производен клас да бъдат асоциирани (свързани) с указатели на базов клас с цел създаване на полиморфни методи.Ключът към създаването на полиморфен метод е първоначално да се декларира „виртуален" метод на базовия клас (наречен още виртуален базов метод). Това е стандартна декларация, която се различава от останалите само по поставената отпред ключова дума virtual

Декларацията на виртуален метод указва, че съответният клас ще бъде използван като базов и от него ще се създаде друг, който е производен. Последният ще съдържа метод, който да предефинира виртуалния метод на базовия клас.На указател към базовия клас може да бъде присвоен обект от производния клас. Този указател може да се използва за получаване на достъп до стандартни методи на базовия клас и предефиниращи методи на производния клас.

-със следващия код се осъществява присвояване на нов обект от класа Galab (гълъб), производен на базовия клас Ptica (птица), на указател към базовия клас Ptica

```
Ptica *pPtica =  new Galab;
```

Класове за предоставяне на възможности:
Класовете, чието единствено предназначение е създаването на производни класове от тях, се наричат класове за предоставяне на възможности (capability classes). Те осигуряват възможности за производните им класове. Обикновено тези класове не съдържат данни, а само задават определен брой виртуални методи, които могат да бъдат предефинирани в производните им класове.

Абстрактни типове данни:
Достатъчен е само един чист виртуален метод в класа, за да бъде превърнат в абстрактен тип данни.

Абстрактният ТИП данни (ADT) представлява по-скоро теоретична концепция, отколкото реален обект и винаги е базов за други класове. Стандартният базов клас може да бъде превърнат в абстрактен тип данни чрез задаване на начална стойност 0 на един или повече от виртуалните му методи. В този случай те се наричат чисти виртуални методи (pure virtual methods) и винаги трябва да бъдат предефинирани в производните класове.Не е допустимо създаването на обекти от абстракстни класове...при опит ще има съобщение от компилатора за грешка

Ето програма, която илюстрира "един интерфейс, много методи". Тя дефинира абстрактен клас за списък от целочислени стойности. Интерфейсът към списъка се дефинира от чисто виртуалните функции store( ) и retrieve( ) За да съхраните стойност, извикайте функцията store( ). За да възстановите стойност от списъка, извикайте retrieve( )
Базовият клас list не дефинира подразбиращи се методи за тези действия. Вместо това всеки производен клас дефинира какъв точно да е типът на поддържаният списък. В програмата са реализирани два типа списъци: опашка и стек. Въпреки че двата списъка работят по напълно различен начин, достъпът до всеки от тях се осъществява през един и същ интерфейс. Разгледайте внимателно програмата:

```
// Демонстрация на виртуални функции
#include <iostream>
#include <cstdlib>
tinclude <cctype>
using namespace std;
class list {
public:
list *head;     // указател към началото на списъка
list *tail;       // указател към края на списъка
list *next;       // указател към следващия елемент от списъка

int num;   //стойност, която ще се съхранява

list()  { head = tail = next = NULL; }
virtual void store(int i) = 0;
virtual int retrieve( ) = 0;
};

  // Създава списък тип опашка
class queue : publ ic list {
public:
void store (int i);
int retrieve( );
};

void queue::store(int i)
{
list *item;
item = new queue;
if( !item) {
    cout << "Allocation error.\n";
     exit (1) ;
}
item->num = i;

// поставя елемента в края на списъка
if(tail) tail->next = item;
tail = item;
item->next = NULL;
if (!head) head = tail;
}

int queue::retrieve( )
```

```
{
int i ;
list *p;
if('head)
{
cout << "List empty \n";
return 0;
}

// премахва елемента от началото на списъка
i = head->num;
p = head;
head = head->next;
delete p;

return i ;
}

// Създава списък от стеков тип
class stack : public list
{
public:
void store(int i) ;
int retrieve();
};
void stack::store(int i)
{
list *item;

item = new stack;
if (!item) {
cout << "Allocation error \n";
exit (1) ;
}
item->num = i;
// поставя елемента в началото на списъка
// готов за стекова операция

if (head) item->next = head;
head = item;
if (!tail) tail = head;
}
int stack::retrieve( )
{
   int i ;
   list *p;
   if(!head) {
          cout << "List empty \n";
           return 0;
}

// премахва елемента от началото на списъка
i = head->num;
p = head;
head = head->next;
delete p;

return i;
}
```

```
int main( )
{
list *p;

// демонстрира queue
queue q_ob;
p = &q_ob;  // сочи към queue

p->store(1);
p->store(2);
p->store (3);

cout << "Queue: ";
cout << p->retrieve( );
cout << p->retrieve( );
cout << p->retrieve( );
cout '\n';

// демонстрира stack
stack s_ob;
p = &s_ob; // сочи към stack

p->store (1) ;
p->store (2) ;
p->store(3) ;

cout <<"Stack: ";
cout << p->retrieve( );
cout << p->retrieve( );
cout << p->retrieve( );
cout << '\n';

return 0;
}
```

Функцията main( ) от представената програма просто илюстрира, че класовете за списъци наистина работят. Все пак, за да видите защо полиморфизмът по време на изпълнение е толкова мощен, опитайте да използвате следната функция main( )

```
int main()
{
list *p;
stack s_ob;
queue q_ob;
char ch;
int i;

for(i=0; i<10; i++) {
cout <<"Stack or Queue? (S/Q) ";
cin >> ch;
ch = tolower(ch);
if(ch=='q') p = &q_ob;
else p = &s_ob;
p->store(i);
}
cout << "Enter T to terminate\n";
for(;;) {
cout << "Remove from Stack or Queue? (S/Q):
cin >> ch;
```

```
ch = tolower(ch);
if(ch=='t') break;
if(ch=='q') p = &q_ob;
else p = &s_ob;
cout << p->retrieve( ) << '\n';
}
cout << ' \n' ;
return 0;
}
```

Тази main() функция илюстрира колко лесно могат да се обработват случайните събития, които възникват по време на изпълнение, като се използват виртуални функции и полиморфизъм по време на изпълнение. Програмата изпълнява един for цикъл от 0 до 9. При всяка итерация на цикъла трябва да изберете в какъв тип списък - стек или опашка желаете да поставите стойността. В съответствие с вашия отговор базовия указател p се насочва към правилният обект и текущата стойност на i се записва в него...

След края на този цикъл се стартира друг, който иска от вас да укажете от кой списък да се извади стойност. Относно вашият отговор определя избора на списъка.

12. Макроси

Работа с макроси:
Като правило за имената на макроси се използват главни букви.

Всеки път, когато C++ компилаторът бъде стартиран, той първо извиква своя предпроцесор, за да потърси компилаторни директиви, които могат да бъдат включени в сорс кода. Всички директиви започват със знака диез (#) и ще бъдат имплементирани за модифициране на сорс кода, преди той да бъде компилиран в действителност. Промените, направени от компилаторните директиви в предпроцесора, създават нов временен файл, който обикновено не се вижда. Именно този временен файл се използва за компилиране на програмата, а не оригиналният файл със сорс кода.
Временните файлове, създадени от GNU C++ компилатора, могат да бъдат записани за преглед чрез опцията -save-temps в командния ред...създават се два временни файла test.ii и test.s, чието съдържание може да бъде видяно с помощта на произволен текстов редактор. Файлът test.s съдържа асемблерни инструкции от ниско ниво, а файлът test.ii пресъздава оригиналния файл със сорс кода, но заменя директивата #include с подробни инструкции, които добавят класа <iostream>
Компилаторната директива #define се използва за задване на идентификатор и низ, с който предпроцесорът да замени всяко срещане на идентификатора в сорс кода.

Директивата #ifdef позволява извършването на условна проверка, за да се установи дали конкретен низ вече е дефиниран. След нея може да бъде декларирано, във вид на блок от конструкции, действието, което да бъде извършено в случай че проверката върне стойност истина. За задаване на край на блока се използва директивата#endif
По подобен начин директивата #ifndef проверява дали даден низ не е дефиниран преди това.
Когато резултатът е истина, ще бъде изпълнен нейният блок от конструкции.
Тя също трябва да завършва с директивата #endif
Директивата #ifndef е логическата противоположност на директивата #ifdef
Дефинициите могат да бъдат премахнати по всяко време чрез директивата #undef
Съществува и директива #else
Тя може да бъде използвана за задаване на алтернативен код, който да бъде изпълнен в блок #ifdef или в блок #ifndef
Всеки блок трябва да завършва, както обикновено с директива #endif

Програмите на C++ обикновено се състоят от много .hpp хедър файлове и един .cpp файл с реализацията съдържащ главната програма. Много от хедър файловете могат да съдържат класове, които са производни на даден базов клас, и затова трябва да могат да осъществяват достъп до базовия клас, за да бъдат компилирани. Включването (#include) на файла на един и същ базов клас в хедър файловете на няколко производни класа е лесно - но това не е позволено. Популярното решение на този проблем използва компилаторни директиви за проверка на това, дали хедър файлът на базовия клас вече е включен. Ако не е, може да бъде добавен безпроблемно чрез директивата #include

Директивата #define може да бъде използвана за създаване на функции-макроси, които да бъдат заменени в сорс кода преди той да бъде компилиран. Декларацията на функция представлява име на идентификатора, следвано непосредствено от двойка скоби, съдържащи аргументите на функцията ... важно е да не оставяте интервал между името на идентификатора и скобите. След това има интервал и дефиницията на функцията в друга двойка скоби.
Например синтаксисът за деклариране и дефиниране на функция, разделяща стойността на аргумента на 2 изглежда по следния начин:

```
#define HALF(n) (n / 2)
```

Следващият пример декларира и дефинира четири функции-макроси, които са заменени еднократно в кода на главната програма.

```
#define CUBE(n) (n * n * n)
#define SQUARE(n) (n * n)
#define MIN(nl, n2) (nl < n2 ? nl : n2)
#define MAX(nl, n2) (nl > n2 ? nl : n2)
#include <iostream>
usin g namespace std ;
```

```
int main( )
int x = 2, У = 5;
cout « x « " cubed:\t";
cout « COBE(x) « endl;
cout « x « " squared:\t";
cout « SQUARE(x) « endl;
cout « x « "," « y « " max:\t";
cout « MAX(x, y) « endl;
cout « x << "," << y << " min:\t";
cout << MIN(x, y) << e n d l ;
return 0;
```

изходът е:

```
2 cubed :        8
2 squared :      4
2,5 max :        5
2,5 min :        2
```

Трябва да сте много внимателни при използването на функции-макроси, тъй като, за разлика от нормалните функции в C++, те не извършват никаква проверка на типовете - следователно лесно можете да създадете макрос, пораждащ грешки. От друга страна, тъй като макросите заменят функцията инлайн, те спестяват обработката на допълнителната информация за извикването на функция - следователно програмата се изпълнява по-бързо.

II.

Дотук Ви запознах с част от програмирането на C++
Да се надявам,че сте понаучили основата:)
Като цяло се учихме на конзолно програмиране:) и въвеждане на кода в обикновен текстов редактор и компилиране от команд промпт...
Сега започваме с графичното(визуално) програмиране на C++ или още както се нарича Visual C++
Има няколко мощни среди за програмиране или така наречените IDE integrated development environment – програмна среда за разработка на приложения:)
Всяко едно съвременно приложение(application)(програма,софт) се изработва с помоща на програмни среди. Най-общо казано една такава IDE се състои от сорс код редактор(там където въвеждаме нашият код) ,компилатор(интерпретатор), инструменти за изграждане на автоматизацията(линкер), дебъгер(проверяваща програма за грешки и насочваща към тяхното отстраняване)
Много от модерните среди имат вградени инструменти(tools) ,class browser (показват се класовете), object inspector(показват се обектите)
С помощта на тези програмни среди се увеличава скоростта на изграждане на програмите!
Грешно е да мислите,че добрите програмисти използват един обикновен текстов редактор и компилатор:) да, това е така,но ако пишат кратък код и не бързат за никъде:)
Всички добри програмисти професионалисти ползват визуални програмни среди,когато програмират за windows system:) в линукс е подобно,но се ползва Qt4 (qute) визуална среда,която няма вграден компилатор:(няма да разглеждам тази среда и програмирането под линукс и юникс...
http://anjuta.org
http://trolltech.com/products
това са едни от средите за линукс:)
тук ще наблегна на уиндоус програмирането:)
Не бива да мислите,че програмната среда ще прави всичко вместо Вас:) но и доста ще улесни нашето програмиране...
Тук с инфото може да не съм много точен за годините...някои данни ги карам по спомен,а други гледкам в гугълчо:)
та предшественикът на Visual C++ е носел името Microsoft C/C++
Visual C++ 1.0, който съдържал MFC 2.0, бил първата версия на Visual C++, пусната през 1992, в наличност били както 16-bit, така и 32-bit версии. MFC означава Microsoft Foundation Class Library това е огромна библиотека от класове за създаване на графично богати приложения под уиндоус и за уиндоус:)
следва Visual C++ 1.5
Visual C++ 2.0, който съдържа MFC 3.0, бил е първата изцяло 32-bit версия:)
Visual C++ 4.0, който съдържа MFC 4.0, бил е проектиран за Windows 95 и Windows NT
Visual C++ 5.0 с версия MFC4.21 нарича се Developer Studio 97 и има интеграция с др. прогр. езици
Visual C++ 6.0 (известен е като VC6), който съдържа MFC 6.0, бил е пуснат през 1998.
Visual C++ .NET 2002 (известен като Visual C++ 7.0)
Visual C++ .NET 2003 (известен като Visual C++ 7.1), който съдържал MFC 7.1, бил е пуснат през 2003 заедно с .NET 1.1
Visual C++ 2005 (известен като Visual C++ 8.0), който съдържа MFC 8.0, бил е пуснат през ноември 2005 Версията поддържа .NET 2.0
Visual C++ 2008 (известен като Visual C++ 9.0 с кодовото си име Orcas) пуснат през ноември 2007 Версията поддържа .NET 3.5
бъдещата версия Visual C++ 2010 (Visual C++ 10.0) ще е към 2009-2010 :)

Съветвам да програмирате с някоя нова среда на Microsoft като Visual Studio 2008 ,а не на Borland:)
С новите програмни среди си се изграждат приложения и за 64 разрядните процесори, и си избирате коя .NET Framework да бъде:)

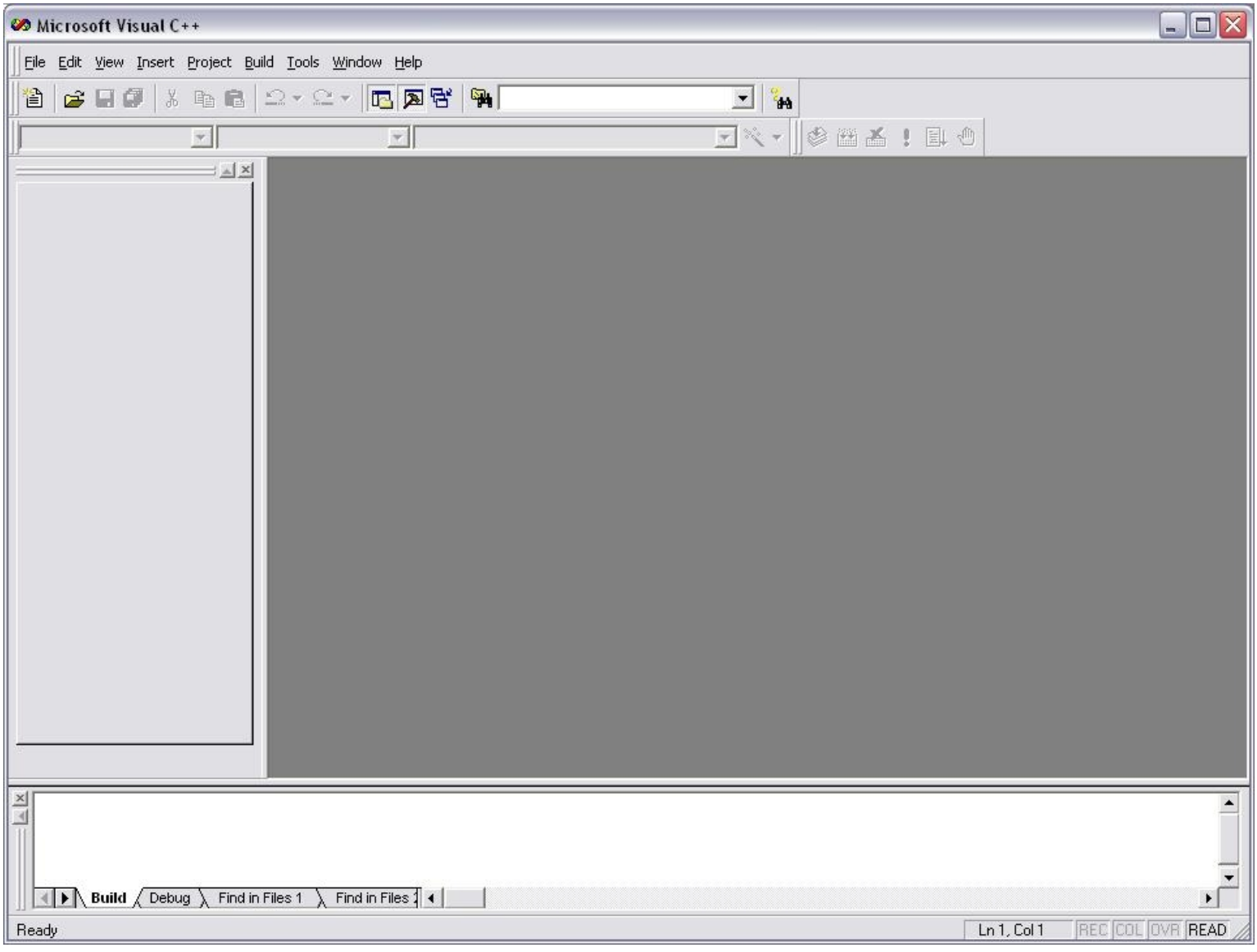
Първо ще започна с кратки обяснения за програмната среда Microsoft Visual C++ 6.0
много програмисти си я ползват все още:) може да си инсталирате след това последния сървиз пак:)
и MSDN(библиотеката е отделен компонент от пакета на средата ... тя представлява справочник за програмистите...обикновено съдържа над 1гб инфо в областта на специфичния програмен език...има примерни сорс кодове...помощтна литература...има го в сайта им...средата може да си я потърсите в торентите:) иначе тя е платена и я няма в сайта на производителя:) няма и да разяснявам с картинки как се инсталира самата програмна среда...чисто и просто не съм свикнал да подценявам хората ☺

1. Програмиране с Microsoft Visual C++ версия 6.0 (Visual WorkBench)

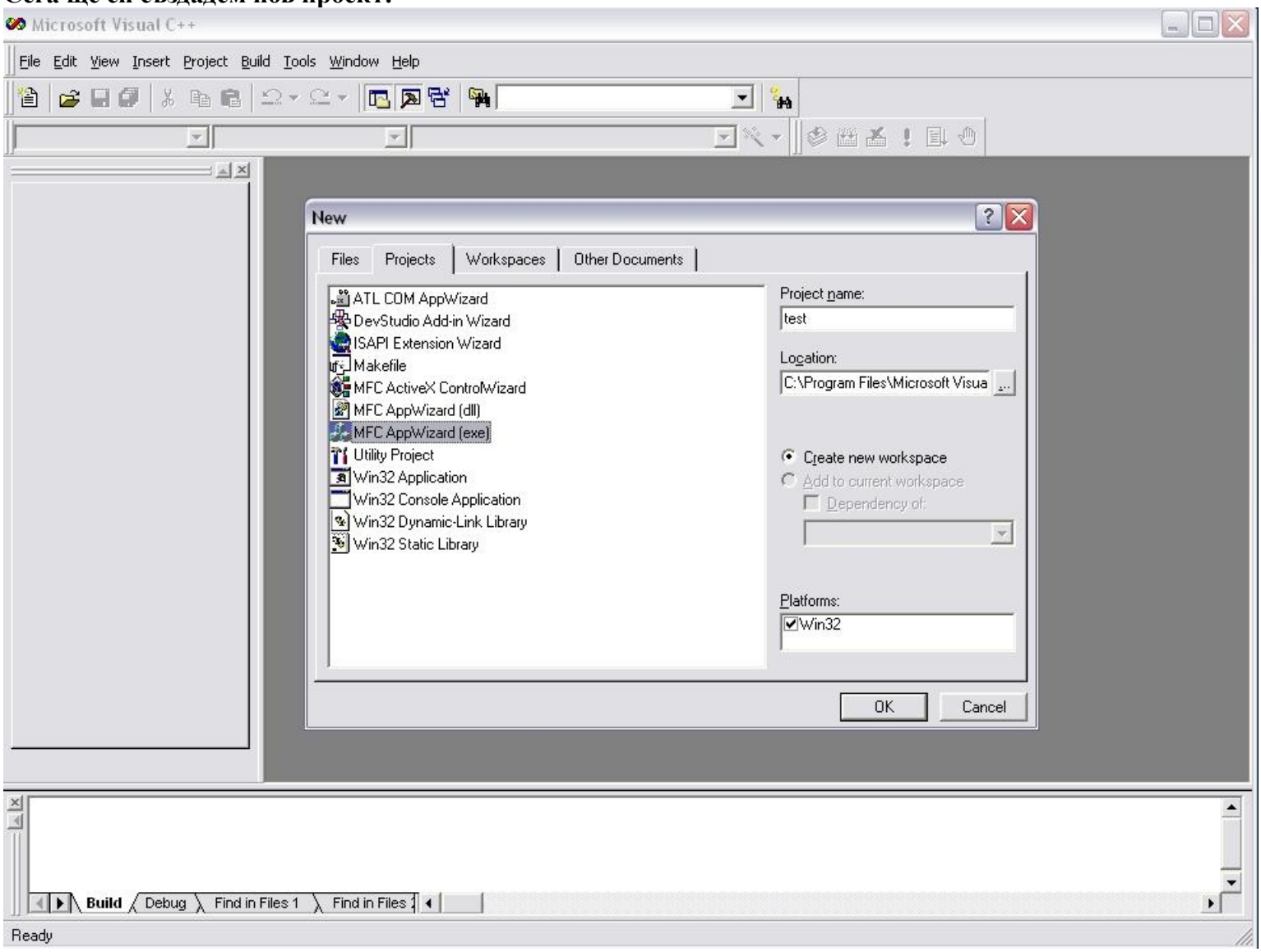
За да разберете дали средата може да работи на вашия компютър трябва да посетите следния сайт и да прочетете системните изисквания:
http://msdn.microsoft.com/en-us/visualc/aa336439.aspx
ами дам на английски е:) все пак ще се занимавате с програмиране:) всичко е на английски:)

http://www.microsoft.com/downloads/details.aspx?FamilyID=A8494EDB-2E89-4676-A16A-5C5477CB9713&displaylang=en
инсталирайте си сървис пака:)

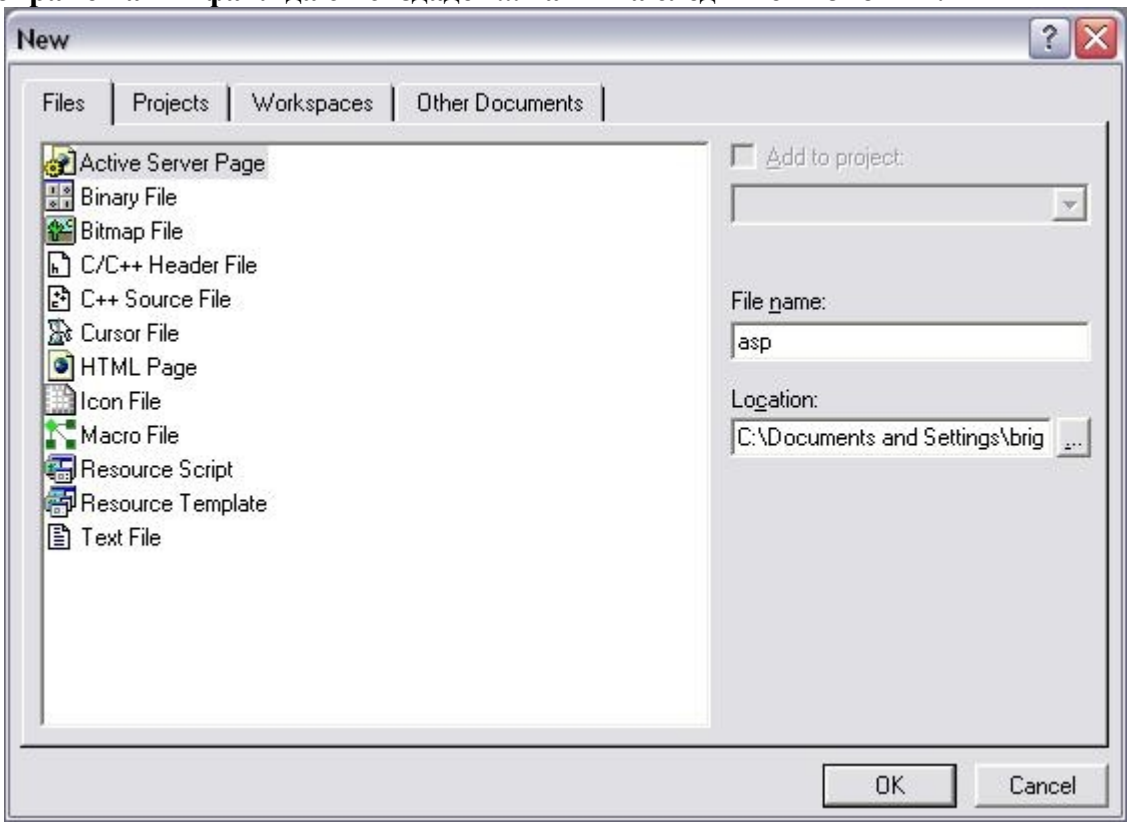
При стартирането на нов проект(New Project) във Visual C++ първо трябва да се избере тип на проекта и да му се сложи именце като всичко това става в прозореца New
Има си съветник (нещо подобно,когато инсталирате дадена програма на компа си:)
та този съветник се нарича MFC AppWizard
той автоматично си генерира част от програмния код за прозоречното ни приложение...
обаче трябва да се научите да разчитате всичкия генериран код...
и веднага можете да си компилирате този визуален прозорец и ще създадете първото си визуално приложение:)))от Build и после натискате Compile
Ето как изглежда програмната среда при първоначалното й стартиране:



Сега ще си създадем нов проект:



Виждате отваря ни се прозорец New с подменюта Files,Projects,Workspace,Other Documents
От Files си избираме какъв файл да си създадем... там има следните възможни:



Active Server Page е за създаване на уеб базирани приложения...създавате си уеб сайтче,но с възможност за комуникиране с бази от данни и управление на сървърната машина...

**Обикновено се ползват скриптови езици като .php , .JavaServer Pages (JSP) , PerlScript и др.
Повечето .asp или .aspx(ново за IIS6 , IIS7) се пишат на VBScript , C# (си шарп)
и се ползват фреймуорковете 1 или 2...**

ASP преминава през три основни издания:

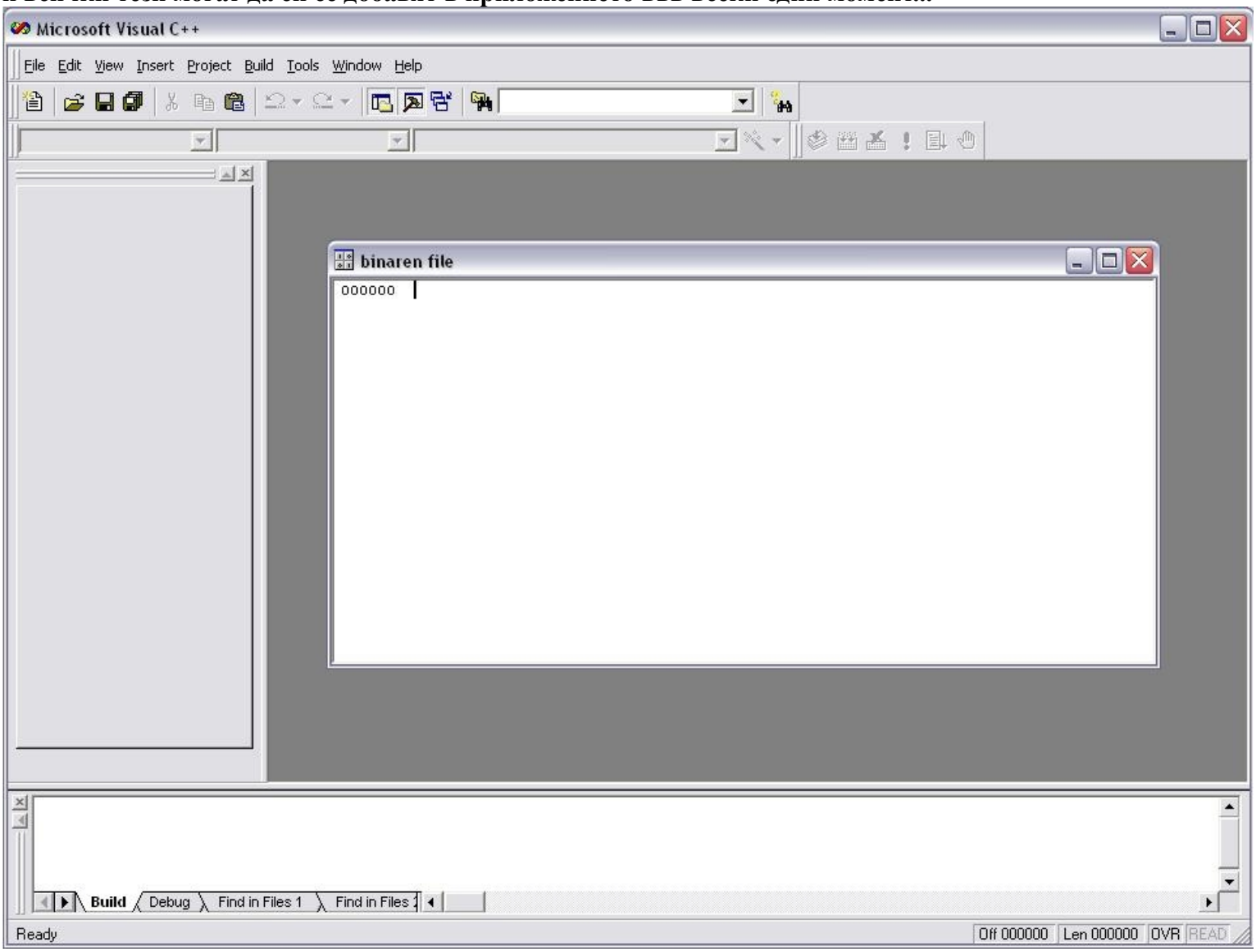
- **ASP version 1.0 (IIS 3.0) 1996 ASP версия 1.0 (разпространяван с IIS 3.0) през декември 1996г**
- **ASP version 2.0 (IIS 4.0) 1997 ASP версия 2.0 (разпространяван с IIS 4.0) през септември 1997 година**
- **ASP version 3.0 (IIS 5.0) 2000 ASP версия 3.0 (разпространяван с IIS 5.0) през ноември 2000 г.**

ASP 3.0 в IIS 6.0 на Windows Server 2003 и IIS 7.0 на Windows Server 2008и VISTA

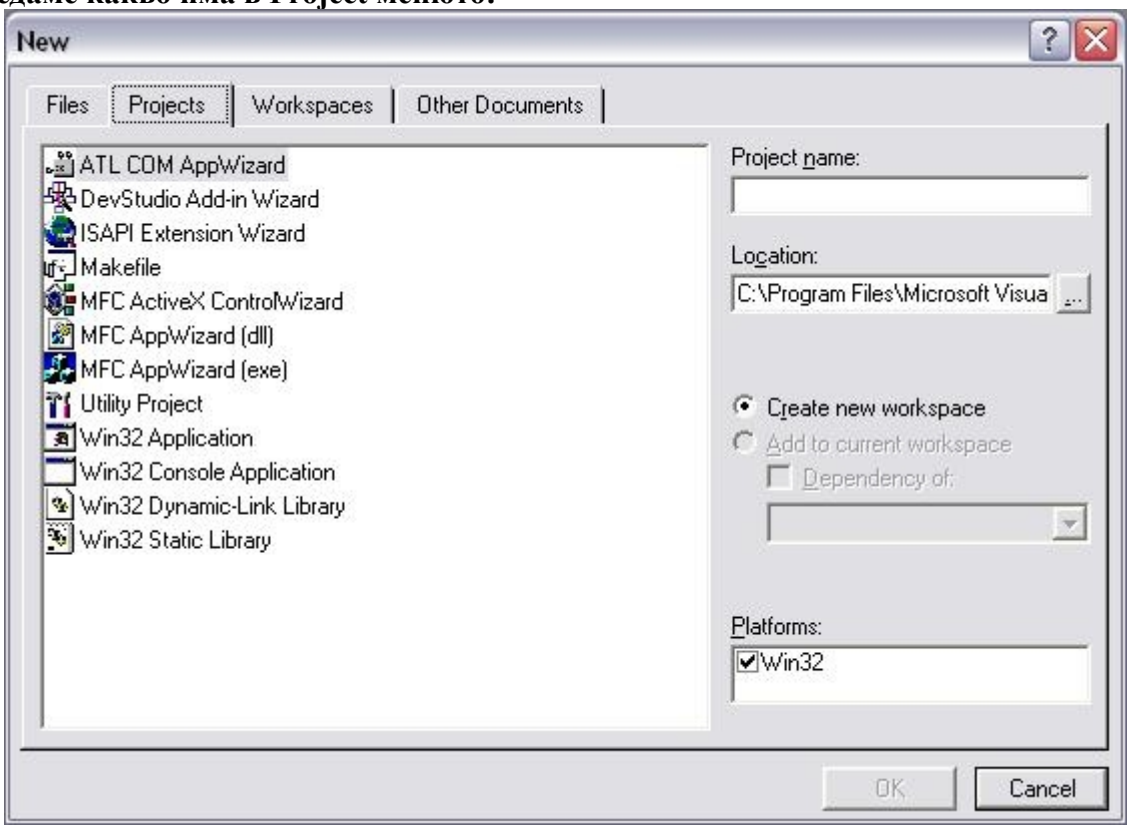
ASP 3,0 в момента е на разположение в IIS 6.0 на Windows Server 2003 и IIS 7,0 на Windows Server 2008.

Всичко това е за обща култура:)

**Следващия е бинарен файл,следван от създаване на някаква картинка:) виждате и възможността на създаване на хедър файлче и на сорс код файла .cpp , курсор файла това си е пак файл картинка като си създавате своя картинка за курсор на мишката:)има си вградено редакторче за растерните изображения...икон файла(Icon File) то си е пак за създаване на картинка .ico file тук в програмирането се ползва за създаване на иконка за самата програма(погледнете сега иконките на десктопа си:) всички те са .ico files които лесно се поместват в програмната среда и тя си ги помества в програмката:)
и текстовия файл си е най-обикновен .txt file ... шаблонен файл(Template) и скриптов
и всички тези могат да си се добавят в приложението във всеки един момент...**



Сега да разгледаме какво има в Project менюто:



ATL COM AppWizard е Active Template Library с COM- обекти и ActiveX контроли
Не може да не сте срещали при браузване из нета от някой сайт като изкочи горе жълта лента и подканяща да се зареди(инсталира) някакъв ActiveX
MFC и ATL са взаимно свързани. Лесно е създаването на ATL базиранки COM обекти,които да ползват MFC и да Ви натресат някой вирус хахахаха:)Програмирането с ATL не е програмиране на C нито MFC Active X контролите не мога да ги обясня понеже трябва да напиша цяла една книга:)Те са си сложни:)

ISAPI помощника се ползва за разработка на приложения и филтри,които работят с програмния интерфейс Internet-Server-API *API* (от англ. Application Programming Interface) Windows API функции Приложно-програмен интерфейс е интерфейсът на изходния код, който операционната с-ма или нейните библиотеки от ниско ниво предлагат за поддръжката на заявките от приложенията или компютърните програми.Образно казано приложно-програмният интерфейс предоставя един по-абстрактен и опростен план за разработчика на приложения, който би му спестил изучаването на няколко различни пласта от операционната или софтуерната система предлагаща интерфейса. По този начин се достига ефективност и бързина в адаптацията на нови софтуерни технологии.

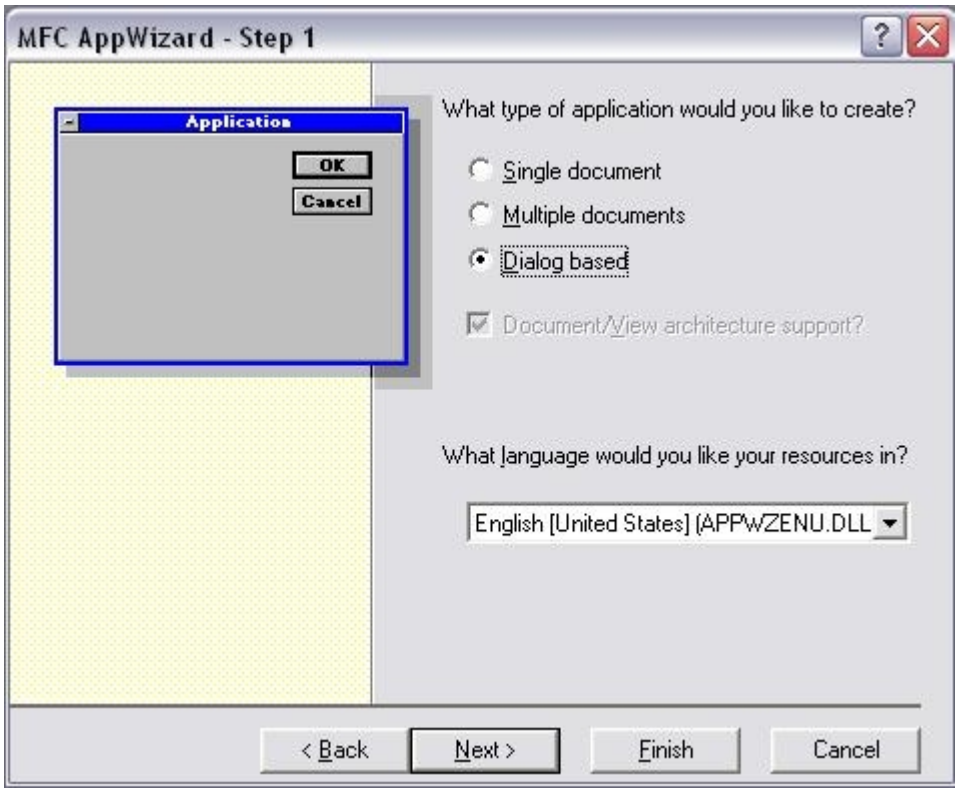
Makefile е специален тип файл,чрез който автоматично ще се изграждат някакви други файлове и ще се управляват в проекта...ползва се в UNIX операционни с-ми...обикновено мейкфаловете се разполагат из различните директории от сорса и служат за събирането на целия проект...те съдържат специални инструкции как и кога да съберат програмата:)по-просто не мога да го обясня:)указват на компилатора кои части да бъдат компилиранки...съдържат shell команди...мейк(make) файлът съхранява опции за компилатора и свързващата програма, и изразява всички взаимовръзки м/у файловете с изходен код мейк програмата чете make файла и после извиква компилатора,асемблиращата програма, ресурс компилатора и линкера(свързваща програма) и така се създава крайния продукт:) т.е изпълнителен файл

MFC ActiveX Control е за създаване на ActiveX контроли – по-надолу ще обясня

MFC AppWizard (dll) е за създаване на библиотечни, допълнителни файлове към основната програмата DLL - Dynamic Link Library=библиотека за динамично свързване...единствено в Windows го има:)

MFC AppWizard (exe) е за създаване на изпълнителен файл с възможност за базирано диалогово прозоречно приложение...генерират се няколко стандартни файла за проекта в графична форма,която си е самият интерфейс на програмката ни:) в тази версия на програмната среда по подразбиране си се създава форма с контроли за два бутона ОК и Cancel...и могат да си се разполагат други контроли...

Ето стъпките на AppWizard (exe)

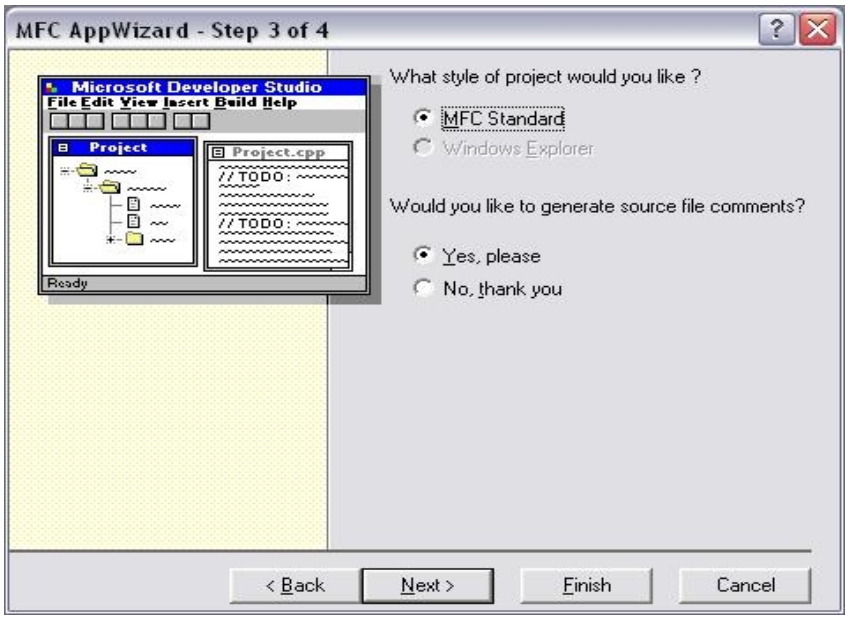


След като сме си озаглавили Project name натискаме бутон ОК и се показва този прозорец със първата стъпка (Step 1) и отмятаме на Dialog based (диалогово,графично базирано приложение) после натискаме бутон Next и се показва стъпка 2

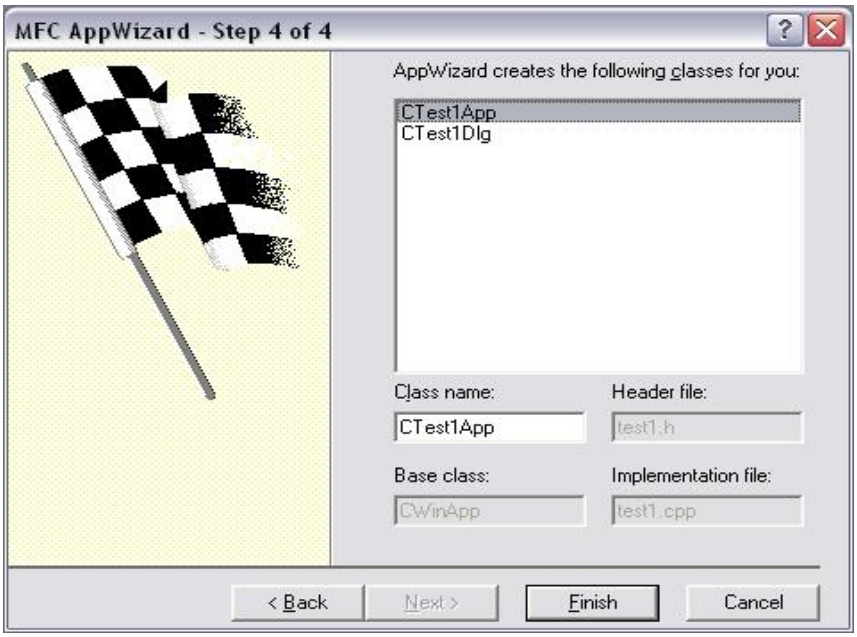
тук по подразбиране си остава така...About box е,че ще имаме прозорче,което ще се генерира

автоматично(това е такова прозорче,когато сте натискали на някоя програма бутон About (Относно)

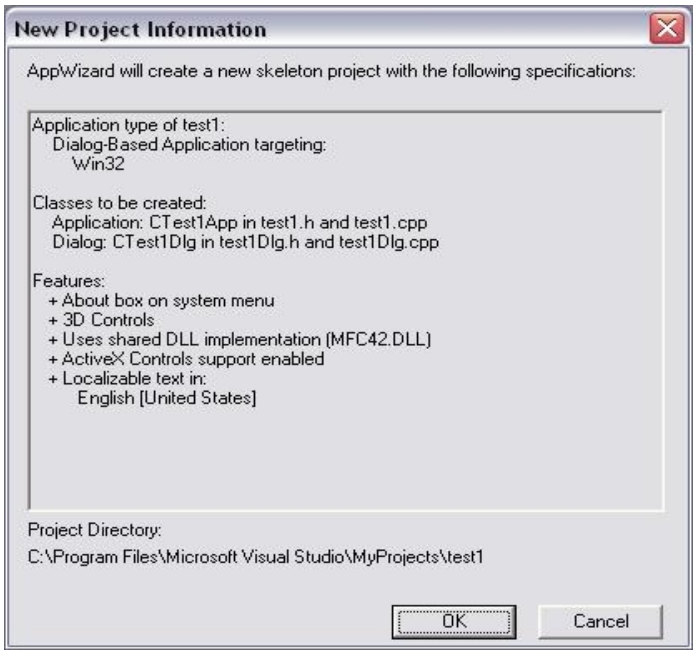
т.е. относно програмата обикновено се пише там кой е създателят,версията на програмката,линк към сайта на създателя...натискате след това бутона Next и се показва следващата стъпка от помощника:



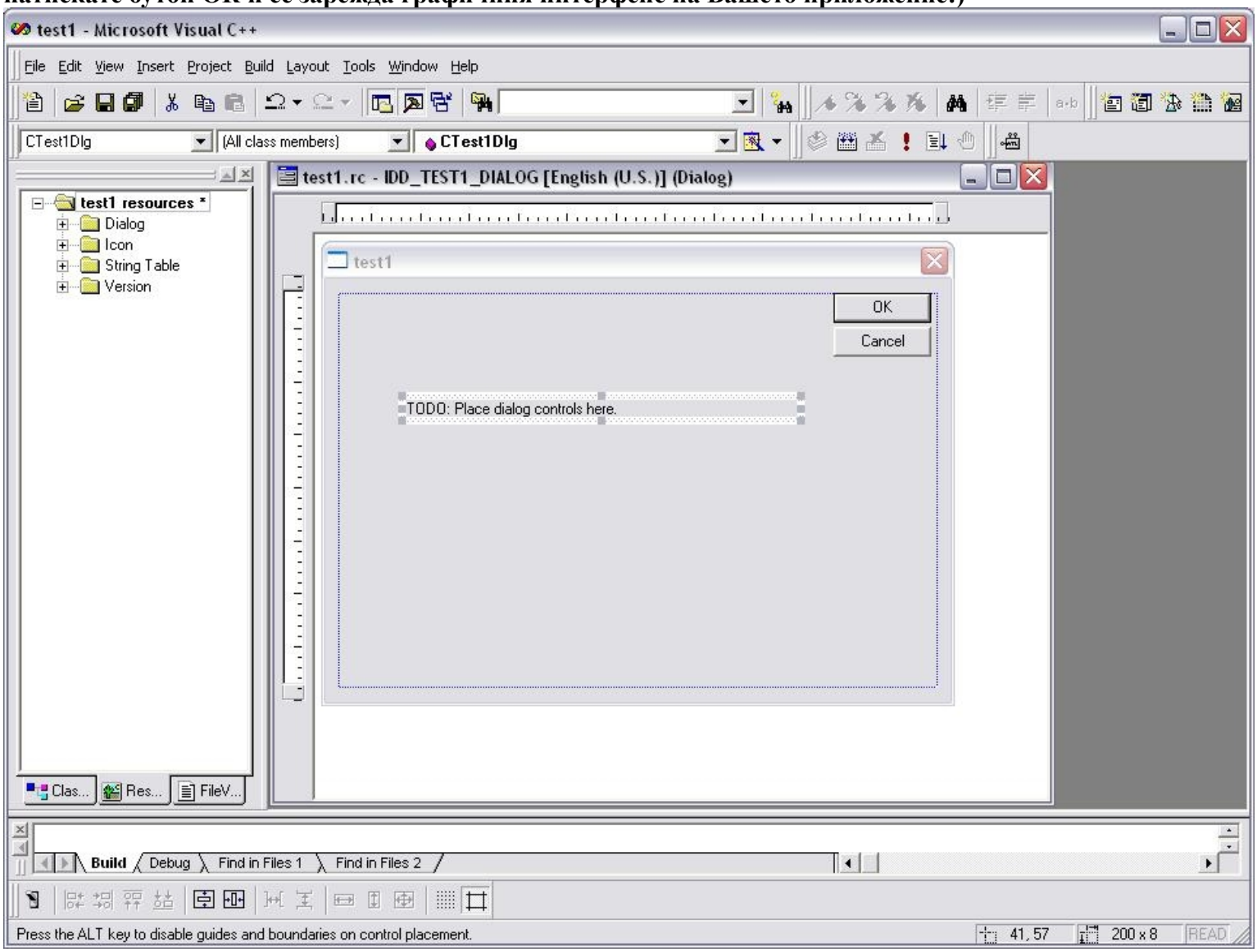
пак бутон Next



и това е последната стъпка на помощника:) виждате там озаглавения файл на проекта като хедър файл и този,в който ще е нашия програмен код...натискате бутон Finish и ще се покаже следното съобщение:



натискате бутон ОК и се зарежда графичния интерфейс на Вашето приложение:)



както споменах тази форма(така се наричат тези визуалните диалогови прозорчета,които се създават) си съдържа генерирани от съветника(помощника) две бутончета ОК и Cancel , и статично текстово поле Веднага след всичко това Вие можете да компилирате целия генериран код.Това става като отидете горе в главното меню на програмната среда: натиснете Build и от падащото меню кликнете в/у Execute (последвано от името на проекта) или с клавишна комбинацийка Ctrl +F5 (то си е написана) забележете,че на Execute буквата x е подчертана(това е показател,че има клавишна комбинация) но това си е създадено от програмистите на самата среда:) и Вие след време във Вашите приложения ще си поставяте подобни неща:)Може да си зададете какъвто си размер искате на диалоговото прозорче просто в края на рамката с мишката го разтегляте и си го орязмерявате:)

Трябва да знаете,че когато програмирате в подобна среда,то тя си създава няколко ресурсни файла...
.dsp Developer Studio Project)това е главният работен файл(проект файла),от който после си отваряте проекта...в по-новите версии е друг формата,но ще стигнем и до там:)
.rc ресурсен файл (ресурс скрипт) - описва ресурсите,менютата,диалогови ресурси,клавиатурни акселератори и т.н.
.ncb
.dsw това е файла на работното пространство(в него има инфо за всеки проект от раб. Пространство) ако искате да работите по вече създаден проект отваряйте този файл или .dsp файла
Visual C++ си създава и някои междинни(intermediate) файлове
.clw
.aps поддържа RESOURCE VIEW
.bsc файл с инфо за браузъра
.clw поддържа class wizard
.txt текстово файлче (обикновено за readme file)
.cpp файл с основния изходен код
.h хедър файловете

.mak външен мейк (MAKE FILE) файл
.plg изгражда лог файл (log file = файл дневник)
няколко директории с файлове в тях,както ресурсни така и изходни изпълнителни
създават се директории(папки) Debug , Release , res

всички данни се съхраняват в ресурсни файлове...всяка програмна среда си създава собствени файлово формати...и после свързващата програма(линкер=linker) свързва(събира) този двоичен ресурсен файл/ове с кода генериран/и от компилера(компилатора) и хопалянкааа създава се изпълнимия файл:)
Тези ресурсни файлове могат да съдържат битмап файлове(изображения=images),икони (.ico files) разни дефиниции по менютата,оформленията на диалоговите прозорци,низовете(стрингове),ваши разни ресурсни формати...и накрая,ако някой файл се прецака:) си има рескю програми
rescue програми се наричат още понеже винаги работните файлове могат да се коруптнат:) (corrupt) (развален,повреден)и по този начин ще може да си възстановите част от ресурсите и сорс кода:)

част от кракерския похват включва точно такива декомпилатори(вид рескю програми)

Най-лесно се декомпилират Java и .NET Microsoft Intermediate Language (MSIL)

Windows позволява динамично свързване, което означава, че специално създадени библиотеки могат да бъдат зареждани и свързвани по време на изпълнение (Dynamic-Link Library или DLL)
Въведена е в уиндоус за намаляване на употребяваната RAM памет и на твърдия диск.
Ако има програмен код,който се употребява повече от 1 път,той се обобщава в един файл(библиотека) и се зарежда само един път в оперативната памет. Няколко приложения могат да си споделят една библиотека за динамично свързване, което спестява както казах памет и дисково пространство:)
Динамичното свързване повишава възможността за модифициране на програмата,защото dll могат да бъдат компилирани и тествани самостоятелно...можете да си създавате собствени dll разширения като надстройка над стандартните dll на MFC

Свързващата програма(Linker) чете .obj и .res файловете създадени от компилатора и ресурсния такъв и отваря .lib файловете, за да включи MFC кода – това е код от библиотеките на уиндоуса и хопалянкааа създава се .exe file :) ха честито:) но хубаво е да знаете,че заглавните файлове си съдържат директивата #pragma ,която специфицира необходимите библиотечни файлчета и не се налага изрично да се указва на свързващата програма кои файлове да чете...

В програмната среда Visual C++ има и трасираща програма наречена Debuger
ами,ако тръгне програмката от първия път, то няма да ползвате тоз дебъгер:)
...в по-новите версии дебъгера е доста усъвършенстван,ама тук в тази 6 версия си е зле:) сам не може да коригира логическите грешки...
определени бутони от панела с инструменти вмъкват и премахват точки на прекъсване и контролират постъпковото изпълнение...прозорците Variables и Watch разширяват визуално един указател към обект и се показват всички полета на производния клас и неговите базови класове...ако позиционирате курсора в/у обикновена променлива, дебъгера показва нейната ст-ст в малко прозорче...за да трасирате програма трябва да се компилира и да се свърже като опциите на компилатора и свързващата програма са установени в състояние за генериране на дебъг инфо...разгледайте си в средата дебъг прозорчето за изхода...обозначените watch променливи...променливите от текущия и от предходните изрази...зтека на извикванията...абе поогледайте и цъкайте смело из програмната среда:)))

AppWizard е генератор на код, който създава работещ скелет на едно Windows приложение с характеристики, имена на класове и имена на файлове с изходен код, които указвате посредством диалогови прозорци... Кодът, генериран от AppWizard, е минималният необходим код, а функционалността е вътре в базовите класове на приложната среда. AppWizard ви помага много бързо да започнете едно ново приложение. Напредналите разработчици могат да създават свои собствени генератори от типа на AppWizard :)

ClassWizard е програма (реализирана като DLL), която е достъпна от менюто View на програмната среда Visual C++. ClassWizard спестява рутинната работа по писането на кода на класовете във Visual C++. Ако ви трябва нов клас, нова виртуална функция или нова функция за обработка на съобщения, ClassWizard ще напише прототипите и телата на функциите и ще генерира кода за свързване на Windows съобщението към съответната функция.
ClassWizard може да променя и кода на дефинирани от вас класове, така че избягвате проблемите по

поддръжката, присъщи за обикновените генератори на код. Някои възможности на ClassWizard са достъпни директно от Wizard панела на Visual C++

Ако започвате ваше собствено приложение от самото начало, вероятно имате добра представа за структурата и разположението на файловете,класовете и член функциите,но ако сте се заели с чуждо приложение(чужд,готов сорс код) ще Ви е необходима помощ:)
Source Browser позволява да разглеждате и редактирате дадено приложение от гледна точка на класовете и функциите...браузъра има следните режими на разглеждане :
• Definitions and References (дефиниции и връзки) — Можете да изберете която и да е функция, променлива, тип, макрос или клас и след това да видите къде са дефинирани и използвани във вашия проект.
• Call Graph/Callers Graph (диаграма на извикваните/диаграма на извикващите) За дадена функция, можете да видите диаграмно представяне на функциите, които тя извиква и на тези, които я извикват.
• Derived Classes and Members/Base Classes and Members (производни класове и членове и базови класове и членове) — Това са диаграми, показващи йерархията на класовете. За даден клас, можете да видите неговите производни или базови класове, заедно с техните членове (полета и функции). Можете да контролирате границите,до които се разпростира йерархията, посредством мишката.
• File Outline (схематично изложение на файловете) За определен файл се появяват класовете, функциите и полетата, заедно с местата, където те са дефинирани и използвани във Вашия проект.
Освен браузъра, Visual C++ има възможност ClassView, която не зависи от базата данни на браузъра. С нея получавате дървовиден изглед на всички класове на проекта, показващ също така и член функциите и член променливите
При двукратно щракване с мишката върху някой елемент ще видите изходния код. За разлика от браузъра, ClassView не показва инфо за йерархията.
Панела на работното пространство (Workspace panel) има ClassView,Resource View,File View
Class View - за преглеждане и обработване на изходящия код на ниво класове от C++
Resource View - позволява намиране и редактиране на всеки от различните ресурси в приложната програма, вклкючително оформлението на диалоговите рамки, икони и менюта.
File View - осигурява преглед на всички файлове, които изграждат създаваната от вас приложна програма

Помощната с-ма във Cisual C++ 6.0 си е MSDN
От менюто Help избирате си Contents ще си го разгледате:) или доброто старо F1 :)))))))) ПОМОООЩ:) позициониране на курсора в/у някоя функция,макрос или клас и после F1 и помощта си иде:)

Диагностицирането става с помощта на SPY++
Дава си дървовиден изглед на системните процеси,нишки прозорците...наблюдават се съобщенията и се изследват прозорците на работещите приложения... а със Source Safe се контролира изходния код

Друго на което първоначално трябва да обърнете внимание в програмната среда е панела за изходяща информация (Output panel) Възможно е този панел да е невидим при началното стартиране на Visual C++ ,но след като компилирате своята първа програма, ще забележите появата му в долната част от екрана на програмната среда... и той ще остане отворен, докато не решите, че е нужно да го затворите. В този панел се извежда всякаква информация, която трябва да получите...тук ще видите данни относно работата на компилатора, както и предупреждения и съобщения за грешки.

Може да си настройвате програмната среда според вашия вкус...може да си премествате където си поискате лентата с инструментите,wokrspace,свойствата(properties)...спокойно си внасяйте промените в програмната среда:) кликайки с десния бутон на мишката и отмятайки опциите...това се нарича индивидуализиране на работната среда:)Възможно е да промените местоположението на произволна лента с инструменти и дори да я оформите като плаваща...

MFC библиотеката е обектно-ориентираният C++ приложен програмен интерфейс за Microsoft Windows...езикът C++ сега е стандартът за разработка на сериозни приложения, а естествено за Windows трябва да има C++ програмен интерфейс,та какъв по-добър интерфейс за Windows може да има от този, произведен от Microsoft - създателят на Windows? Такъв интерфейс е MFC библиотеката!!! Забравете Борланд(Borland),ако сте решили да се занимавате с уиндоус!!! Ползва се всичко,което е създадено от Майкрософт и удобрените от тях неща:)
Приложенията, създадени с приложна среда, използват стандартна структура. Всеки програмист, който

започва голям проект, развива определен вид структура за своя код. Проблемът е в това, че различните програмисти използват различна структура и е трудно за другия програмист да научи тази структура и да се придържа към нея. Приложната среда на MFC библиотеката включва своя собствена структура на приложението - структура, която се е доказала в много софтуерни обкръжения и в много проекти.
С MFC библиотеката вашата програма може да извиква Win32 функции по всяко време, така че можете максимално да се възползвате от възможностите на Windows.
Приложенията, създадени с приложна среда, са малки и бързи, но 32-битовият код е по-обемен,а 64 разредния още:) компилирана с large memory model, една Winl6 програма използва 16-битов адрес за стековите променливи и за голяма част от глобалните. Win32 програмите използват 32-битов адрес за всичко, и дори често използват 32-битови цели числа, защото те са по-ефикасни от 16-битовите...
новия C++ код за обработка на изключения консумира много памет...ама к'во ни пука:)
има си големи помощни динамични библиотеки на заден план:)
скоростта при c++ си е възможно най-добрата все пак работим с машинен код,създаден от оптимизиран най-добър компилатор:)изпълнението е страхотно бързо,ако ги нямаше .dll :)
AppWizard,ClassWizard и ресурсните редактори на Visual C++6.0 значително си намаляват времето,необходимо за написването на кода!
Редакторът на ресурси създава заглавен (header) файл,който съдържа присвоени стойности за #define константи,а AppWizard си генерира скелетния код за цялото приложение,а ClassWizard генерира прототипи и тела на функциите за обработка на съобщения (message handlers)
Самата библиотека MFC се учи много дълго време:)
НИКОГА ДАДЕН ПРОГРАМЕН ЕЗИК НЕ СЕ УЧИ САМО ОТ ЕДИН ИЛИ ДВА УЧЕБНИКА!!!
Програмен език се учи самостоятелно(в университета няма да стане),изчитайки мнооого най-различни учебници от различни автори,сравнявайки си ги и с много писане на мнооого код Вие ще навлезете един ден в материята след доста време:) има си израстване:) ще Ви просветне споко:)
Задължителен е английският език и четенето на такива e-books:)

Трябва също да знаете,че ползвайки програмната среда на Майкрософт и включения компилатор нямате право да използвате получените изпълними програми с търговска цел. Трябва да се заплаща лицензна такса:)

При създаване на работно пространство за нов проект е нужно да следвате стъпките по-долу:

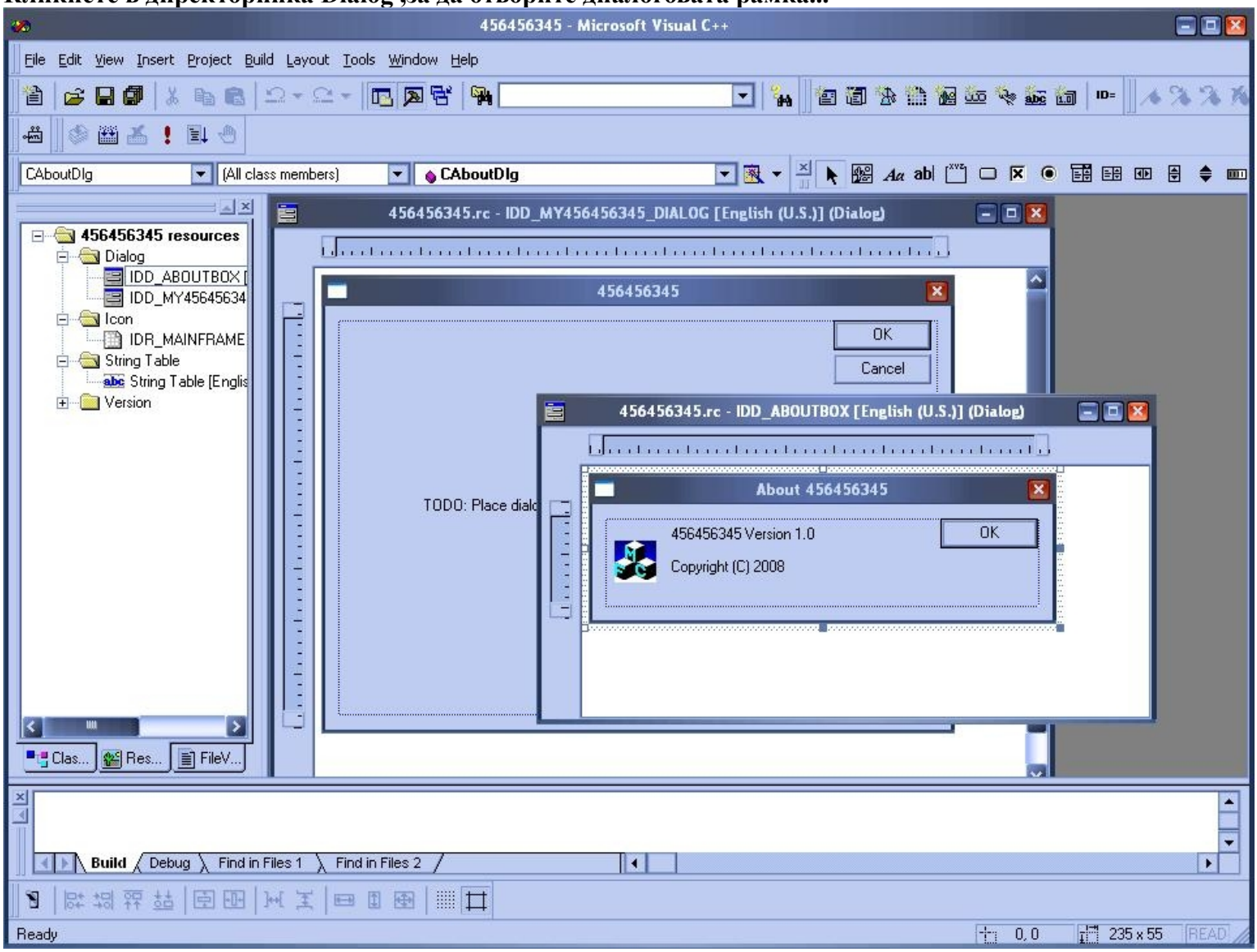
1. **Изберете FileNew. Ще последва отваряне на помощника NewWizard**
2. **Активирайте панела Projects и изберете MFC AppWizard (exe)**
3. **В полето Project Name въведете наименованието на проекта например Hello**
4. **Щракнете Върху ОК. С това посочвате на New Wizard, че трябва да извърши две неща: да създаде директория за проекта (наименованието й ще въведете в полето Location) и да стартира помощника App Wizard**
   По време на своята работа помощникът App Wizard ще Ви зададе серия въпроси относно типа на създаваната приложна програма и нейните особености и функционално предназначение
   Посочената от вас информация ще бъде използвана за създаване скелета на програмата, като веднага след това последната може да бъде компилирана и стартирана,но това вече го обясних:)
   При последната стъпка на помощника ще получите информация за класовете на C++, които AppWizard ще създаде за програмата. Щракнете върху бутона Finish, за да позволите на помощника да генерира скелета на приложната програма.

Ако изберете веднага да компилирате,то докато компилаторът на Visual C++ изгражда програмата, върху екрана ще видите лента за изпълнението на процеса, а в панела Output ще преминават различни съобщения на компилатора. Накрая в този панел трябва да бъде изведена информация, че не са открити грешки и няма никакв и предупредителни съобщения.
При изпълнението на програмата ще бъде представена диалогова рамка със съобщение TODO (да се направи). Там ще се съдържат и бутоните ОК и Cancel.
За да затворите приложението си натиснете един от двата бутона.

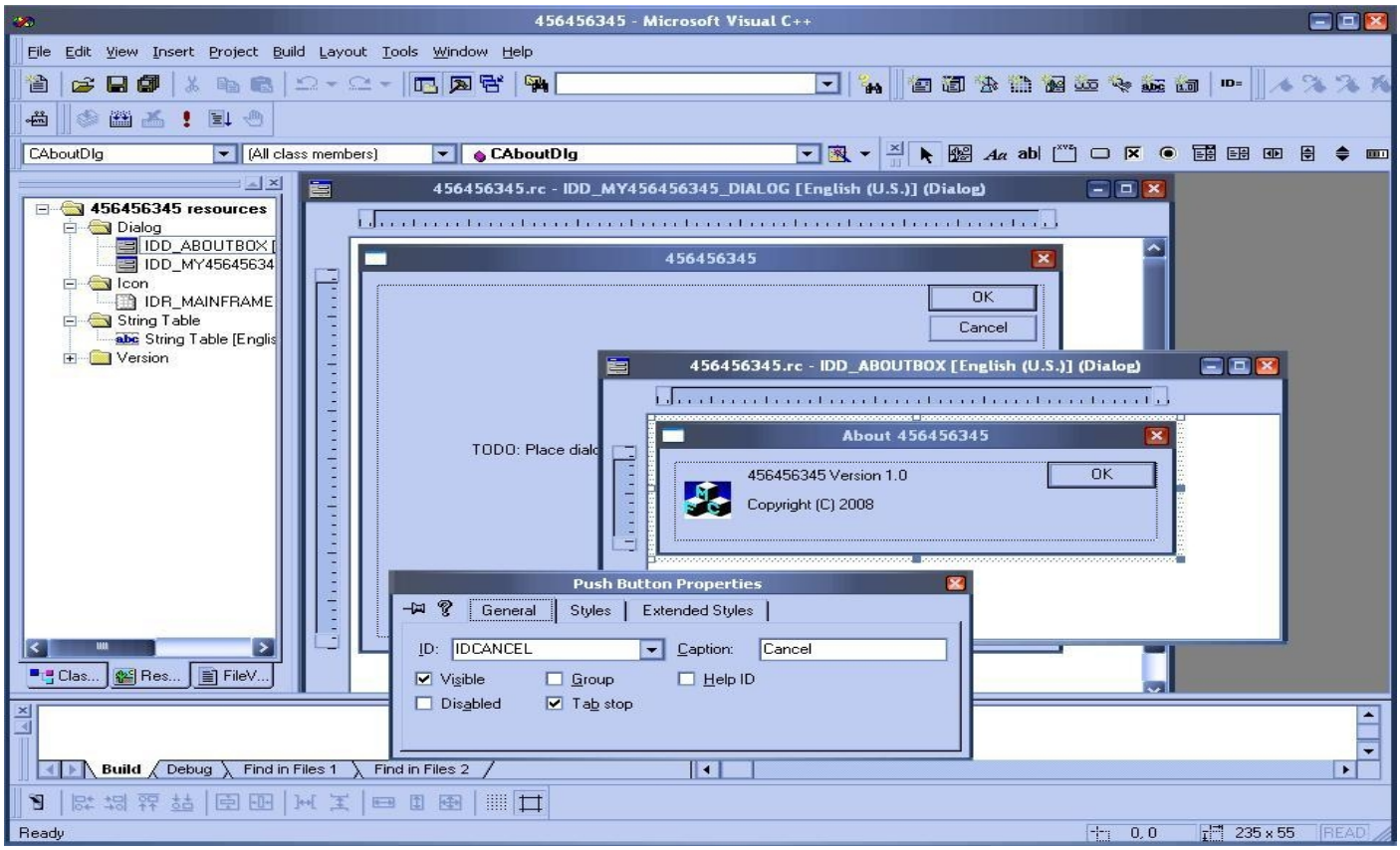
Проектиране прозорец за приложната програма

За целта се придвижваме в работното пространство, за да изберем оттам онзи елемент, който се нуждае от допълнително обработване...за да промените дизайна на диалоговата рамка от приложната програма е нужно да изпълните следващите стъпки:
-Щракнете върху етикета Resource View от работното пространство(Workspace)
-Щракнете(кликнете) върху знака плюс преди иконата, за да видите пълната дървовидна структура на съществуващите диалогови рамки.

Кликнете в директорийка Dialog ,за да отворите диалоговата рамка...



при мен проекта ми е с числа понеже бързам:) озаглавете си го както искате:) виждате и темичката на уиндоуса ми е друга,ама аз често си ги променям:)както и да е...пак гледам да разсейвам положението:) сега може да отидете на бутона Cancel и да кликнете в/у него с десен бутон на мишока и да отворите елемента Properties и ще се отвори диалоговата рамка Properties

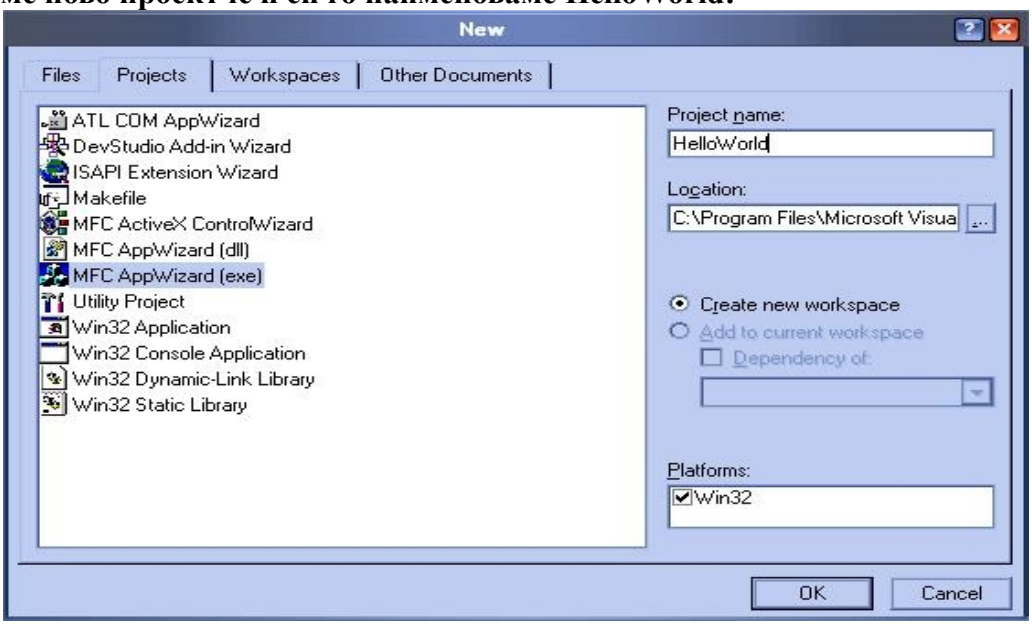


Променете съдържанието в полето Caption (заглавие) на &Close. Затворете диалоговата рамка с характеристиките посредством щракване върху иконата Close, разположена в горния десен ъгъл. Преместете бутона ОК сурнейки го в централната част на диалоговата рамка или където си поискате(избирате си облика на програмата както решите,но трябва да знаете,че си има възприети правила:)
Отворете диалоговата рамка с характеристиките на бутона ОК и променете съдържанието на полето ID (идентификатор) в IDHELLO, а на Caption (заглавие) - в &He11o.
Мисля,че схванахте как да си промените надписчето на бутончетата:)

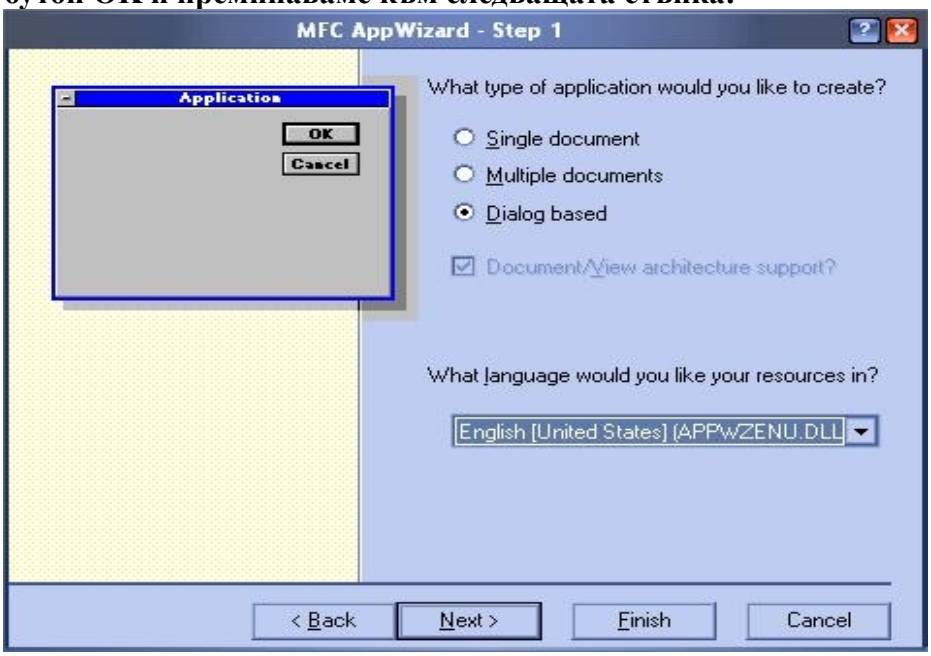
Приложните програми на MFC съдържат в своя изходен код серии макроси, определящи коя функция трябва да бъде активирана в зависимост от съдържанието на полето ID и надписа върху всеки от бутоните за управление.Поради промяната на ID за бутона Hello, макросите не са в състояние да определят функцията и поради това, задействането на бутона остава без резултат.

Не е сложно да добавите допълнителен програмен код към създадената диалогова рамка посредством Class Wizard на Visual C++

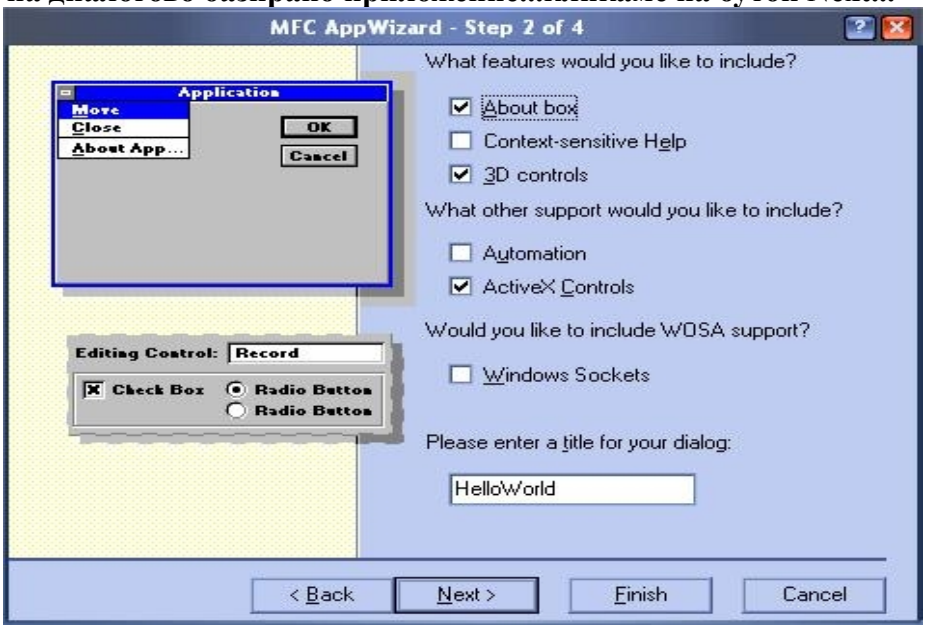
ето сега и програмката hello world,но визуално:)
първо си създаваме ново проектче и си го наименоваме HelloWorld:



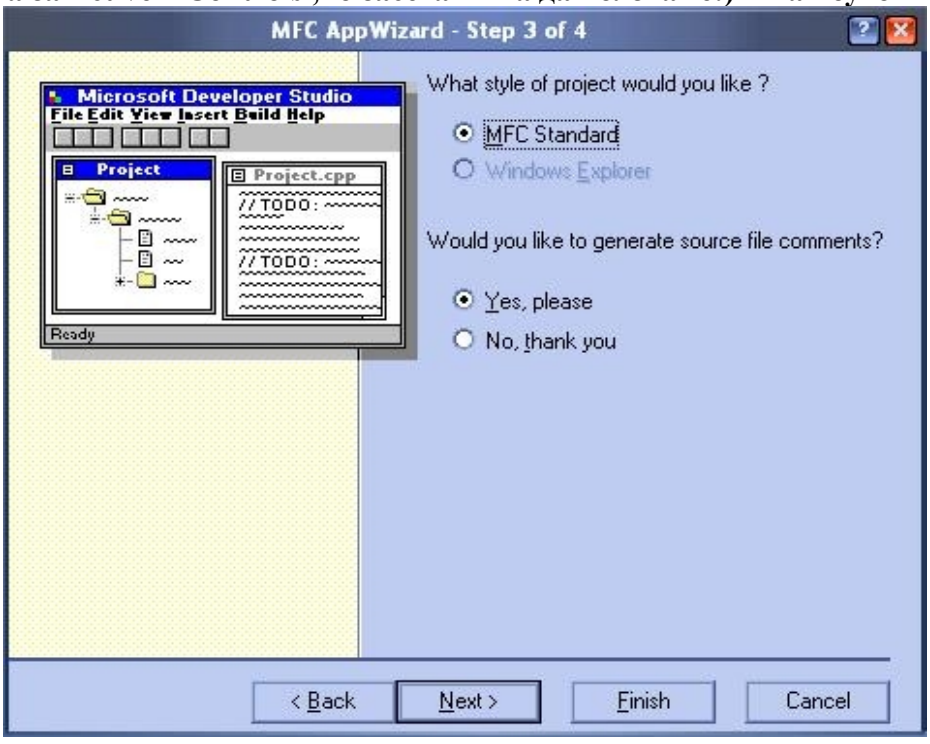
след това натискаме бутон OK и преминаваме към следващата стъпка:



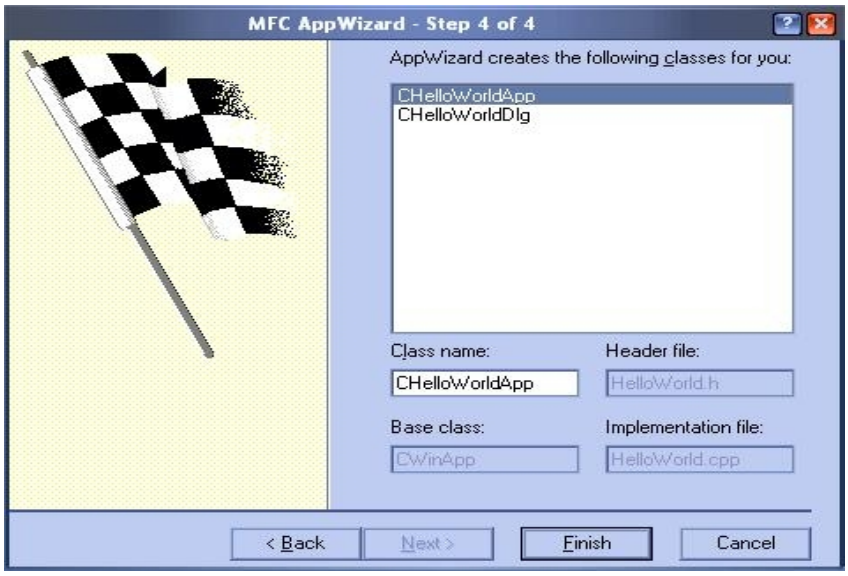
виждате отметнато е на диалогово базирано приложение...кликаме на бутон Next...



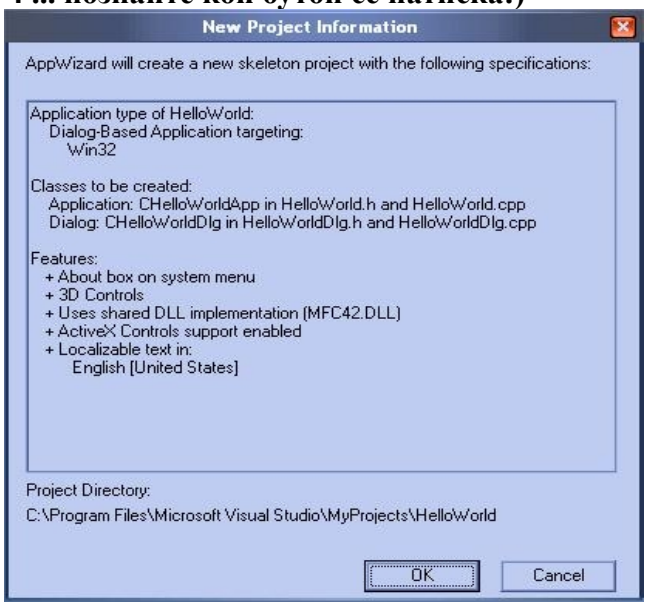
включена е и опцията за ActiveX Controls ,но засега няма да ползваме:) и пак бутон Next...



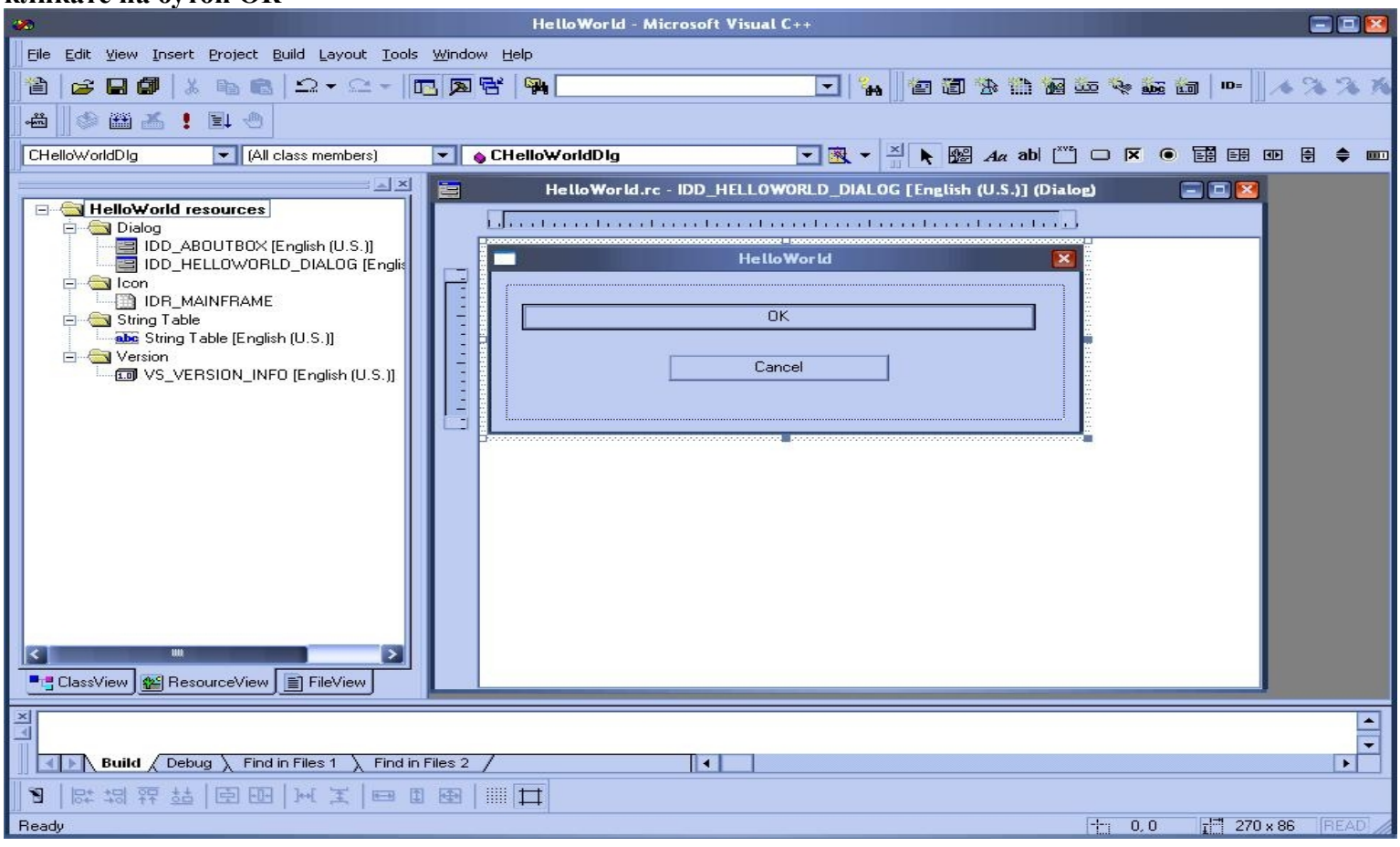
и ето ни на стъпка 3 и пак бутончето Next ...



и ето ни на последната стъпка 4 ... познайте кой бутон се натиска:)



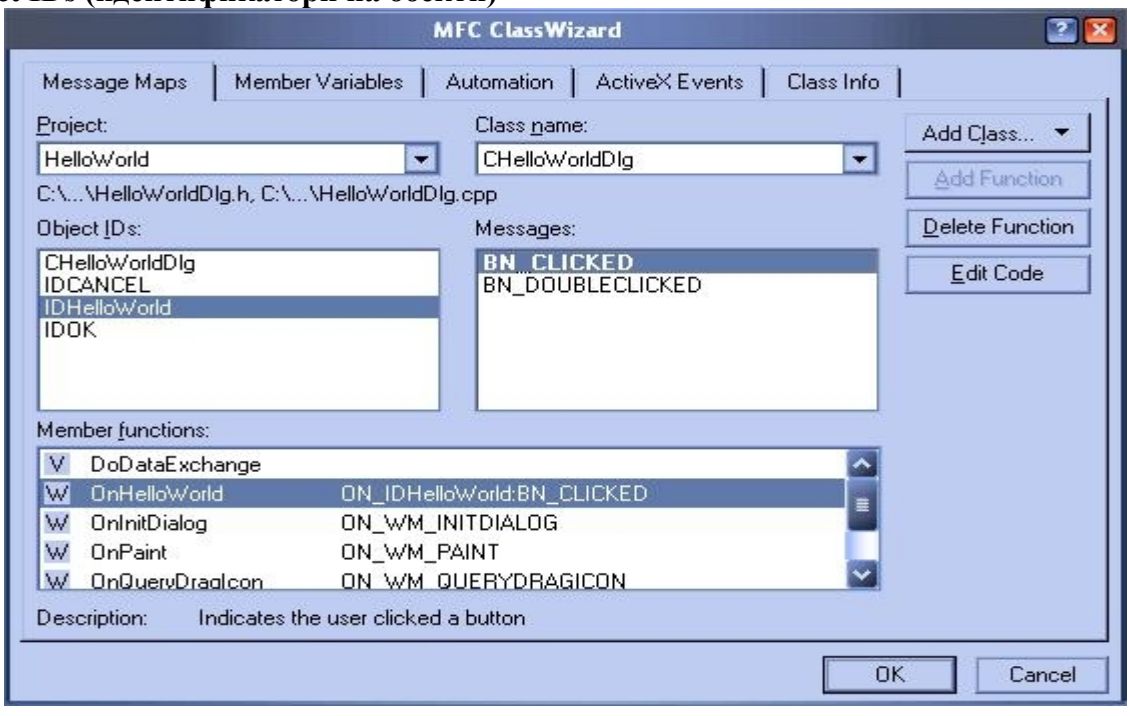
и тук в този диалогов прозорец си виждате къде ще е записан проекта (пътя до проекта...директориите) кликате на бутон ОК



и сега тази диалогова рамка си я привеждате в този вид или в какъвто искате:)
просто разтягате с мишката самите бутончета и рамката...
виждате,че текста TODO и т.н. го няма...изтрива се с Delete (клавиша на клавиатурата)
оразмерявате бутоните и след това в/у самия бутон Cancel давате десен бутон на мишката и отивате на Properties(св-ва) отваря се диалоговата рамка

IDCANCEL си е команда и си служи да затвори приложението,а на Caption: си озаглавяваме бутончето за излизане от Push Button Properties натискате горе в дясно червеното кръсче:)
Отворете сега диалоговата рамка с характеристиките на бутона ОК и променете съдържанието на полето ID (идентификатор) в IDHELLO, именоваме си бутончето Hello World :)

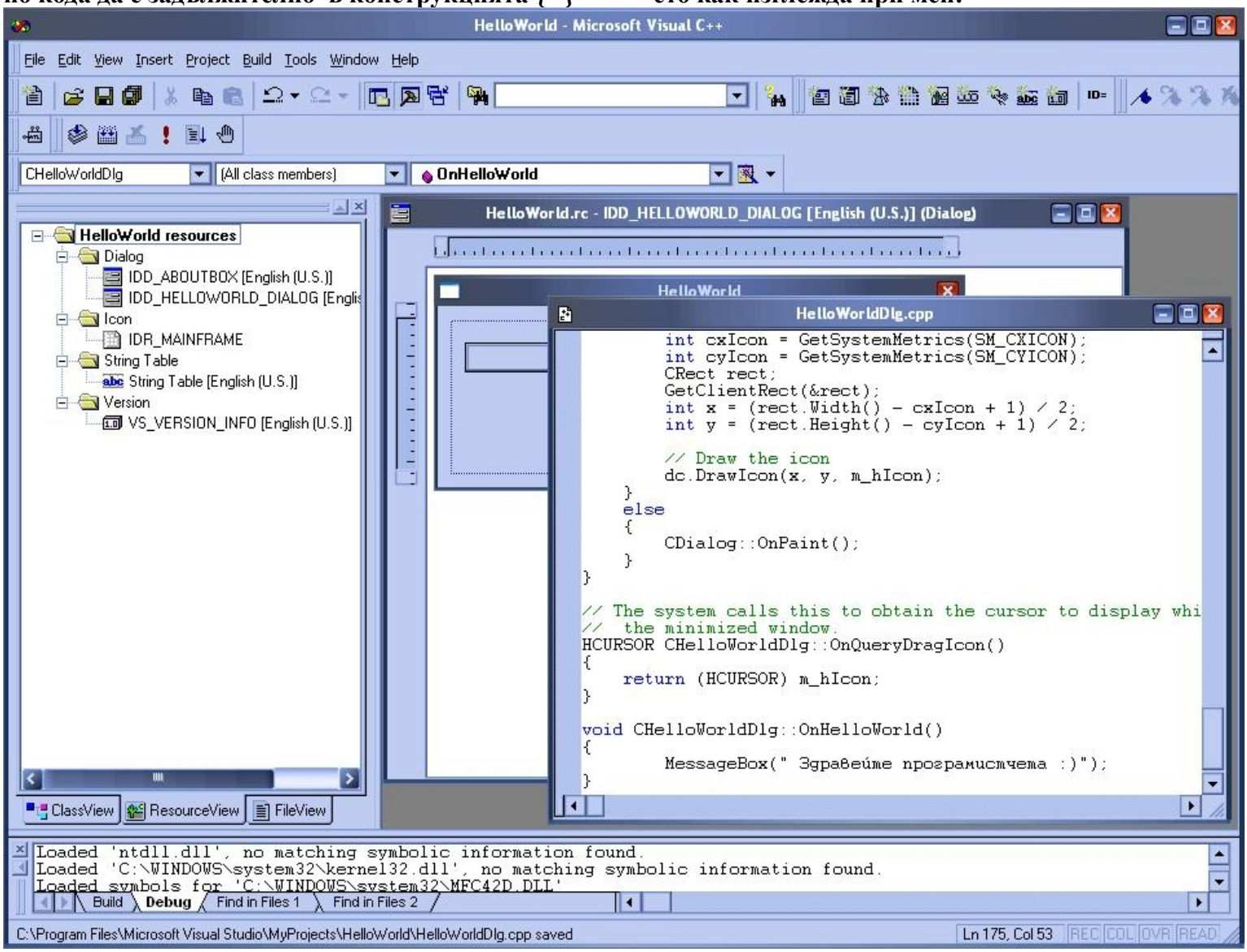
ами сега компилирайте: от Build (горе е в главното меню на програмната среда) -> Ctrl+F5 (компилира се проекта и се стартира,ако е верен кода:) или просто тествайте приложението с натискане на F5 (дебъгване)
ще забележите,че действието на бутона Close се е запазило и щракването върху него води до завършване изпълнението на програмата...щракването върху Hello обаче остава без последствия, защото променихте съдържанието на полето ID за този бутон...
сега следва да се добави кода,за да има функционалност едно от бутончетата:
ще добавите допълнителен програмен код към създадената диалогова рамка посредством Class Wizard на Visual C++
За придаване известна функционалност на бутона Hello World :) трябва да щракнете с десния бутон на мишката и да изберете ClassWizard... от предложеното меню
Ако бутонът Hello World :) е бил маркиран при отваряне на ClassWizard, то той ще бъде избран от списъка Object IDs (идентификатори на обекти)



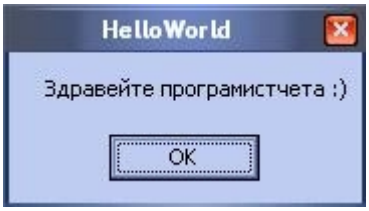
След като маркирате елемента IDHelloWorld от списъка Object Ids , изберете елемента BN_CLICKED от списъка на съобщенията и щракнете върху бутона Add Function...
Ще последва отваряне на диалоговата рамка Add Member Function (добавяне на член-функция)В нея се съдържа предположение за наименованието на функцията...щракнете върху бутона ОК, за да създадете функцията и да я добавите в картата на съобщенията (message map)...
Щракнете върху бутона Edit Code и курсорът ще се позиционира върху изходния код на функцията и то точно на мястото, където бихте могли да започнете въвеждането на новите команди...
Въведете кода като започнете непосредствено след реда с коментара // TODO: Add your control notification handler code here

```
    MessageBox(" Здравейте програмистчета :) ");
}
```

виждате,че //TODO е като коментар и Ви подканя да си въведете програмния код:)може да го оставите или да го изтриете,или на негово място да си напишете Ваш коментар, което си е добра практика, но кода да е задължително в конструкцията { } ето как изглежда при мен:



MessageBox служи за извеждане на оповестителни съобщения в уиндоус ... като низа е м/у скобките и там може да си напишете съобщението,което да се изведе при натискане на бутона...в случая сега бутона е озаглавен Hello World :)
windows автоматично ще си оразмери прозорчето на съобщение,според това колко е дълъг низа и ще си добави бутон ОК



Хубава практика е постоянно да си сейвате,съхранявате проекта...кликайки в/у малката иконка нарисувана като дискетка и най-вече в/у иконката с многото дискетки (да се съхрани всичко)
-това го знаете понеже сте работили и с други програми,в които си е същото:)
примерно в Word или Exel и мн. др.

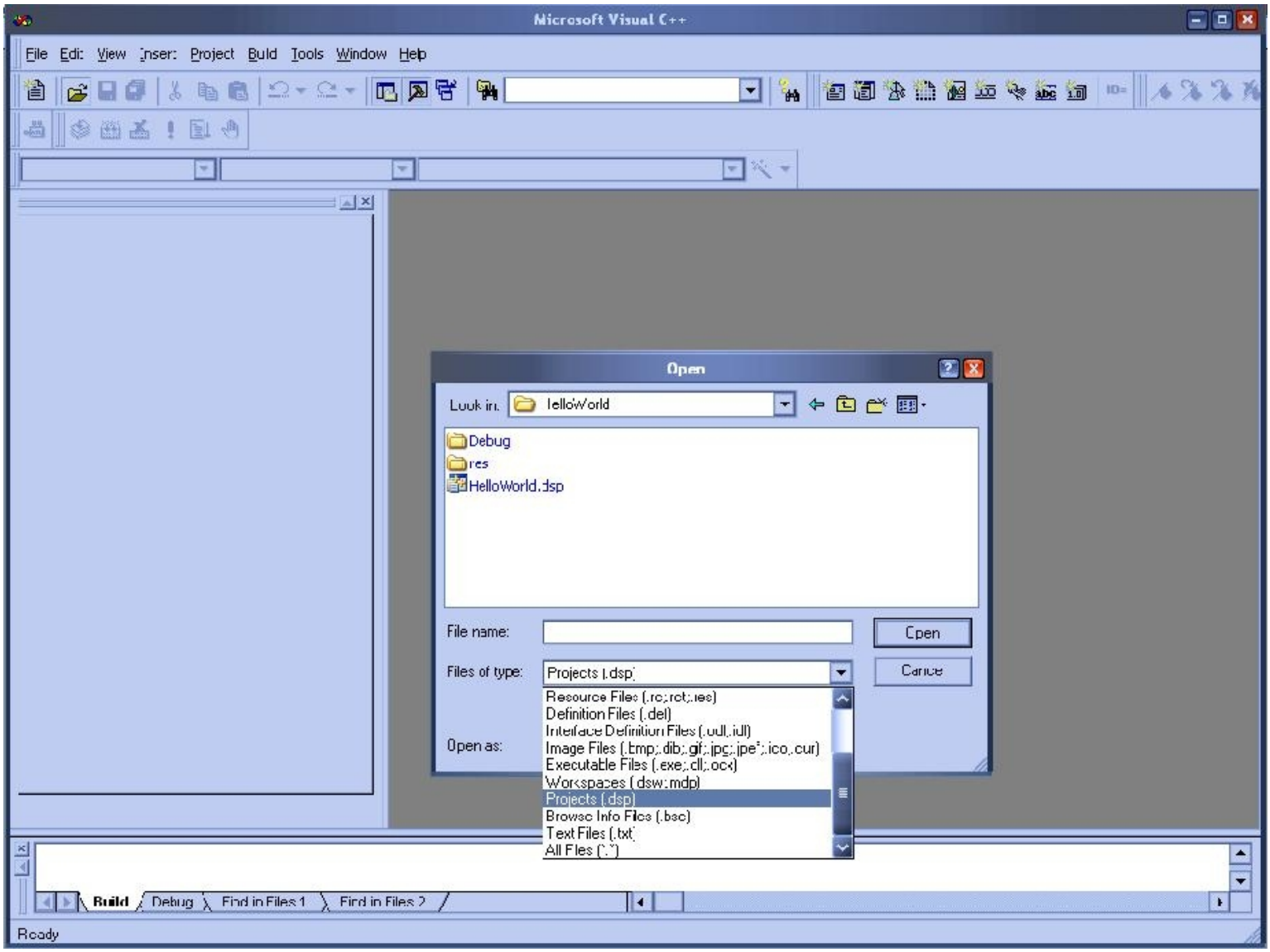
-когато правите промени в кода или оразмеряването на прозорците може след това при повторно компилиране да срещнете съобщение като това:

просто сте уведомени при дебъгването,че са настъпили промени и трябва един вид те да бъдат съхранени в целия проект това е,ако сте забравили да си сейвате проекта:) натискате бутон Yes и компилирането ще бъде извършенко... в случая всички тези файлове при нас си съществуват,има си ги не обръщайте внимание,че пише,че ги няма:)
и накрая ето я нашата „програмка“

при натискането на бутон Hello World :) ще се изведе диалоговия прозорец със съобщението,което сте написали в кода:)

Сега,ако затворите програмната среда и искате да си отворите проекта направете следното:
-трябва да знаете,че по подразбиране проекта е в C:\Program Files\Microsoft Visual Studio\MyProjects

или отивате до там:) или си го извиквате от програмната среда като кликвате на иконката с отворена папчица (Open) или с клавишна комбинация Ctrl+O и ще видите нещо такова:



търсете файл .dsp и си го отворете...

...бях говорил за иконките .ico files ,които се ползват в програмните проекти (всички иконки,които ги имате на десктопа(работен плот) са иконки .ico разширение на файл...

вижте си в C:\Program Files\Microsoft Visual Studio\MyProjects ...в папка е името на проекта отворете... и търсете папка(директория) Debug ...там е изпълнителния файл,който създадохте...

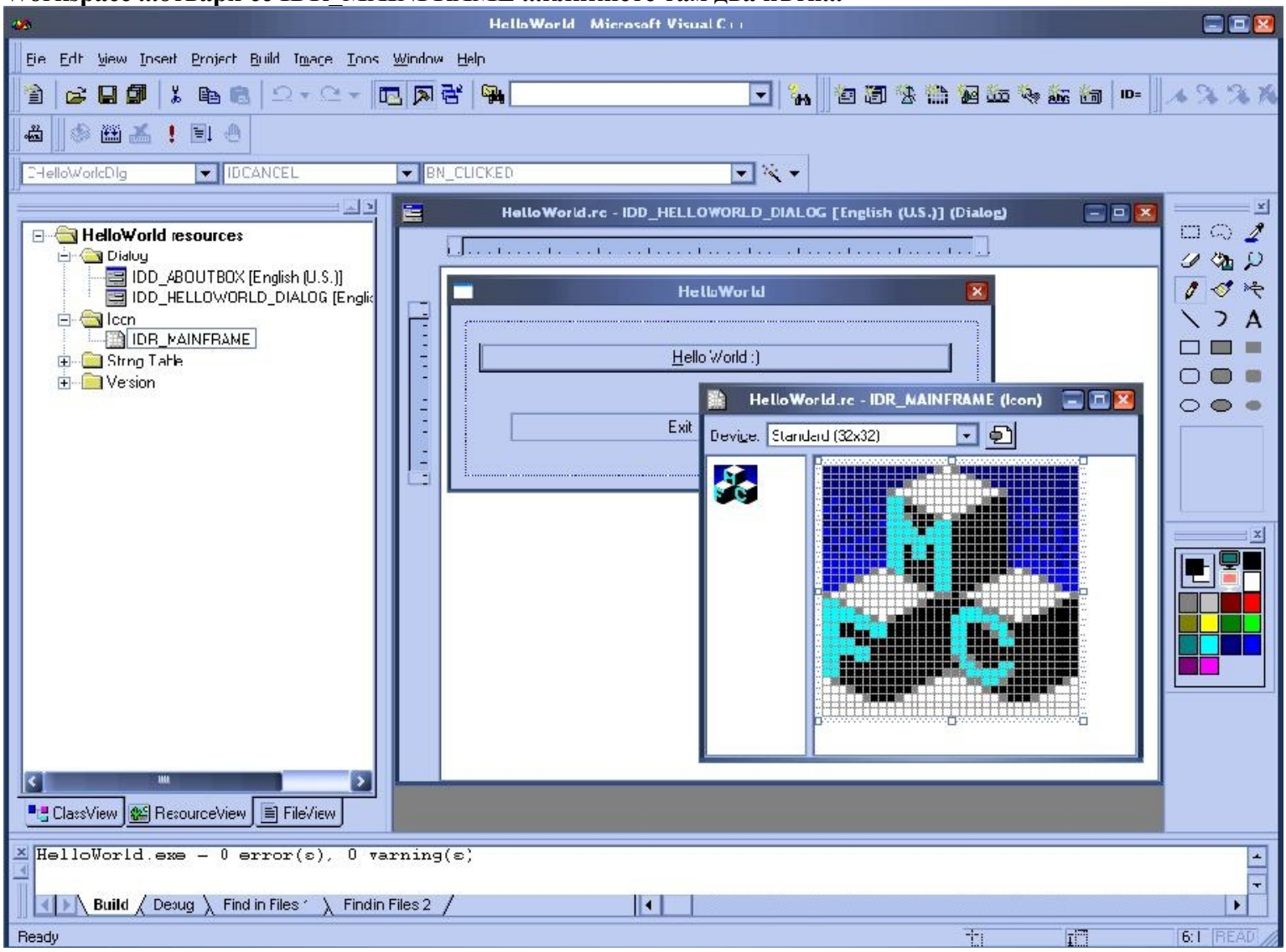
(по-късно ще разберете,че има и директория Release, която се създава като вече искаме да изградиме завършен проект...и изпълнителните файлове там са по-малки като размер,когато е в Release режим...)

и така виждате,че иконката е FMC – тя си е по подразбиране такава,но ние може да си сложиме наша си (има много програмни редактори за създаване на професионално изработени иконки,но засега ще си ползваме редактора на програмната ни среда:) има и разни конвертиращи програми,които обръщат .jpg .gif в .ico files като се задава размера да е стандартно 32x32 с 16 битов цвят или 4-6-24-32bit иконката може да е 1x1 до 255x255 пикселчета... а за Windows Vista е вече 256x256 32 bit абе ползват се,за да се обогати интефейса на програмките:)

-може да си изтеглите пакет с разни иконки от сайта ми: http://brigante.sytes.net/ico.aspx

MFC е съкращение от Microsoft Foundation Classes (библиотека основни класове на Microsoft)...
...извинявайте,че Ви повтарям и пиша по различни начини едно и също,ама така съм сметнал за по-добре,че ще запомните ха-ха:)
Технически погледнато,това е наименование на библиотеката с класове на C++, с която е изградена програма ни и другите,които ще изграждате.....

...та, за да си променим иконката и да си направим някаква наша си правим следното:
най-напред щракнете върху плюсчето на Icon от дървото на ресурсите в приложната програма...на Workspace ...отваря се IDR_MAINFRAME ...кликнете там два пъти...



и сега посредством предлаганите инструменти за рисуване е възможно да промените изображението и да създадете свое:) тук в тази версия на програмната среда си е мнооого бедно:) но почакайте да видите как е в новата версия...Visual C++ 2008 Express Edition там е страхотно:) направо си няма такъв редактор:) ...поорязана е доста:)
Ами хайде поиграйте си и си нарисувайте нещо:) сейвайте после и компилирайте,и би трябвало картинката на проекта да се е променила:) тя е горе в ляво
и кликайки в/у нея ще падне едно малко меню(то е по принцип по подразбиране в уиндоус) там има обикновено команди Restore,Move,Size,Minimize,Maximize, xClose знаете го това,но ето как е при нас:

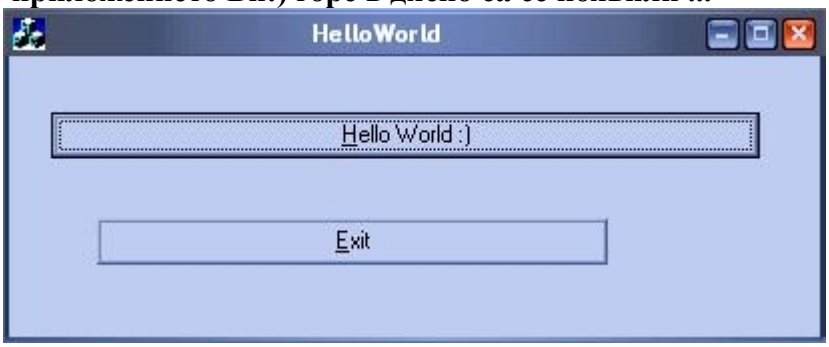
то може да се променя!
А сега забелязвате,че имаме в менюто долу About HelloWorld... така беше озаглавен нашият проект:)

кликайки там ще се появи(изведе) нов прозорец:

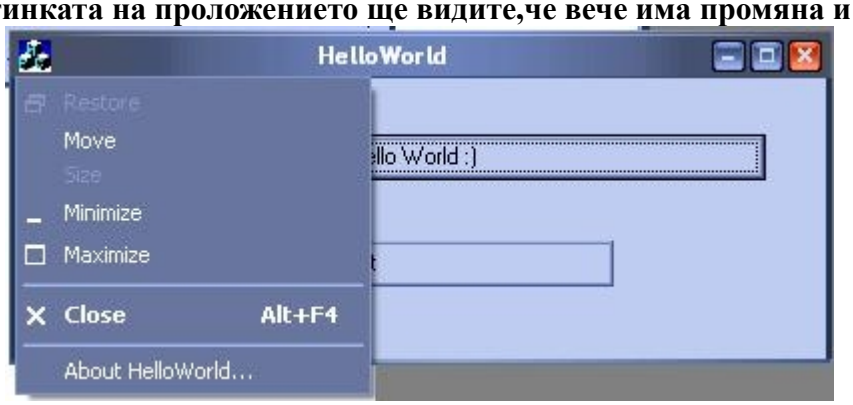
ето го и него:) там обикновено се помества инфо относно програмата...версията й,автора й и т.н.
и може да се поставят картинки,текст,разни ефекти...
и сега,ако искате да промените там нещо,то става от Workspace и кликайки на Dialog
и после два пъти на IDD_ABOUTBOX , и ще се появи диалоговото прозорче(рамка) за редакция на програмката...

хайде сега да поставим бутони за максимално увеличаване(maximize) и намаляване размерите(minimize на диалоговата рамка...това става като маркирате диалоговата рамка все едно ще променяте размера й (когато я издърпвахте,за да си я оразмерите)... понеже сега виждате,че имаме само едно хиксче в червено отдясно горе на нашата програмка:) т.е. искаме вече да имаме и възможност да се минимизира,максимизира...за целта с десен бутон отидете на св-ва(Properties) и оттам на етикета Styles

и просто отметнете („чавките“ някои така се изразяват по форумите:)... всъщност така се разрешава употребата на тези два бутона за минимизиране и максимизиране... след това компилирайте и ще забележите промяната в приложението Ви:) горе в дясно са се появили ...



и ако кликнете в/у картинката на проложението ще видите,че вече има промяна и в падащото меню:



ами това е!

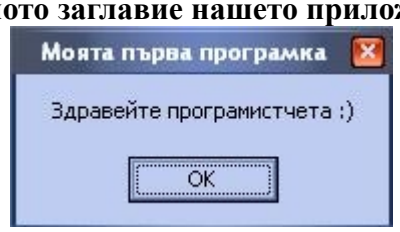
Също така е добре да се знае,че по подразбиране, прозорецът на рамката за съобщения използва за наименование това на приложената програма. Но разчупването на шаблона не е трудно, достатъчно е само да добавите втори символен низ в командата, осъществяваща обръщение към функцията MessageBox. Във всички случаи първият символен низ съдържа текста на извежданото съобщение, а вторият се използва за наименование на прозореца. Ако приложим това правило към

функцията OnHelloWorld, тя би се променила по следния начин:

```
.....................................
// TODO: Add your control notification handler code here
// Поздравление към потребителя
{
      MessageBox( " Здравейте програмистчета :) " , " Моята първа програмка ");
}
```

внимавайте за кавичките,иначе ще се получи грешка в кода!!! винаги трябва да са горни кавички... аз си ги слагам тук малко неправилно в книгата:)но това е от самия текстов редактор,който съм решил да си ползвам:)

ето и как ще изглежда вече в промененото заглавие нашето приложение:

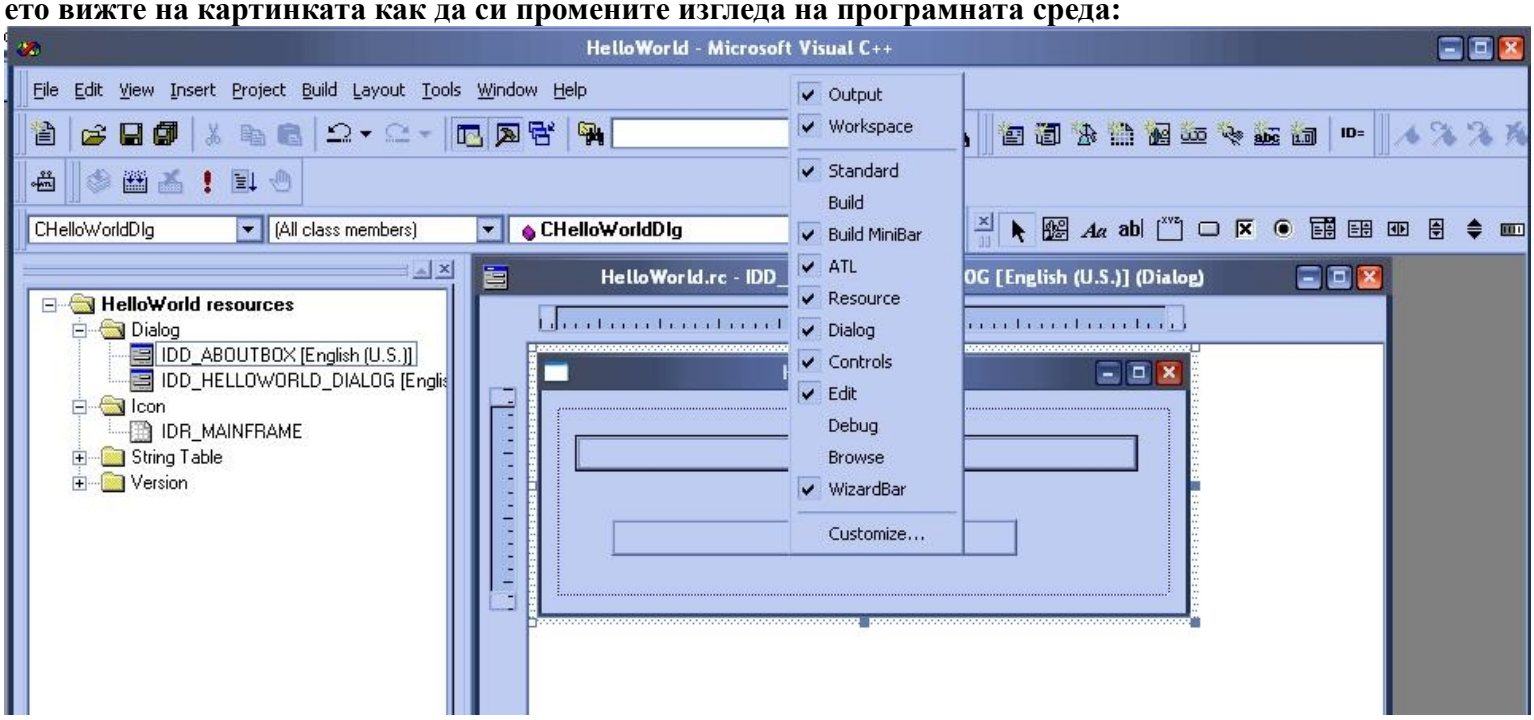


при въведените промени пак може да срещнете това съобщение:

нормално е:)

може да се опитате да си постявате допълнителни бутончета в приложението си...това става от Controls или още наречено Tools (в др.версии Toolbox) там са всички контроли:бутони,статичен текст,вмъкване на изображение(картинка) и мн.др.

ето вижте на картинката как да си промените изгледа на програмната среда:



с десен бутон на мишката горе отстрани на главното меню и ще се покажат,и си отмятате това,което искате да бъде изведено в Visual C++ 6.0

Общите елементи, които се срещат в почти всички програми за Windows са бутони, полета за отметки, текстови полета,рамки с каскадни списъци и др. Всички те се обединяват под наименованието контроли, като повечето от тях са вградени в самата операционна система.Използването им в програмната среда Visual C++ е лесно: достатъчно да приложите технологията хващане-пускане, за *да* да ги поставите в желаната диалогова рамка.В операционната система Windows са вградени няколко стандартни контроли. Към тях се числят плъзгачи (slider), ленти за отчитане изпълнението на определен процес (прогрес индикатори - progress bar), контроли за изобразяване на дървовидни

структури и списъци, и други подобни.
Ще Ви запозная сега с:

Статичен текст (static text)
Поле за въвеждане на текст или текстово поле (edit box)
Бутон за управление (command button)
Поле за отметка (check box)
Радиобутон (radio button)
Текстово поле с каскаден списък (drop-down list box)
познато още и като комбинирано поле (combo box)

Статичен текст (Static Text)
Тази контрола се използва при необходимост от извеждане на текст.Потребителят не е в състояние да промени съдържанието на текста и не е възможно да взаимодейства с тази контрола. Статичният текст е предназначен единствено за четене. Въпреки това не съществуват ограничения за промяна на текста по желания от Вас начин в процеса на изпълнение на приложната програма - всичко зависи от въведения код при създаването й...

Поле за въвеждане на текст (Edit Box)
Това поле се използва в случаи, когато е необходимо да се въведе или да се промени вече въведен текст. Полето за въвеждане на текст (познато още под наименованието текстово поле), е един от основните инструменти, позволяващи на приложната програма да получава специфична информация от потребителя, необходима за по-нататъшната й работа. Въведеният текст е възможно да се използва за различни цели. В повечето случаи се работи с обикновени знаци, като не се допуска форматиране от страна на потребителя.

Бутон за управление (Command Button)
Щракването върху тази контрола от страна на потребителя активира определено действие. Обикновено бутоните притежават текстов надпис, описващ предназначението им: Възможно е да се използва и пиктограма (икона), която се поставя самостоятелно или в комбинация с текст.

Поле за отметка (Check Box)
Полето за отметка представлява малко квадратче, върху което потребителят може да щракне, за да постави знак за отметка... щракването върху поставена отметка предизвиква отстраняването й. Тези полета се използват за активиране или деактивиране действието на дадена функция. Полетата за отметка са предназначени за дискретно управление на променливи.

Радиобутон (Radio Button)
Радиобутонът се представя чрез окръжност, в която след щракване с мишката се появява голяма черна точка. Действието на тази контрола напомня полето за отметка. Разликата е там, че радиобутончетата обикновено се използват групово - по два или повече, като е възможно само един от тях да бъде активен в определен момент...тези елементи са обединени по три, като около тях се поставя групова рамка. Тя позволява визуалното обособяване на отделните групи и намалява вероятността за грешки на потребителя - например опит за едновременно активиране на два или повече радиобутона от една и съща група.

Каскаден списък (Drop-Down List Box)
Тази контрола представлява текстово поле, придружено от списък с възможности за избор. Използва се, когато е нужно да се изведат множество елементи, от които потребителят може да избира.
Понякога при списък от този вид се допуска потребителят да въведе нова стойност, ако не намери подходяща измежду предложените.

Това са си стандартните контроли (инструменти=tools)
Избор {Select)
Статичен текст (Static Text)

Групова рамка (Group Box)
Поле за отметка (Check Box)
Текстово поле с каскаден списък (Drop-Down List Box • Combo Box)
Хоризонтална лента с плъзгач (Horizontal Scrollbar)
Въртене (Spin)
Плъзгач със скала (Slider)
Списък (List Control)
Етикет (Tab Control)
Поле за въвеЖдане на текст с възможност за форматиране (Rich Text Edit)
Месечен календар (Month Calendar)
Потребителски елемент за управление (Custom Control)
Картинка (Picture)
Поле за въвеждане на текст (Edit Box)
Бутон за управление (Command Button)
Радиобутон {Radio Button)
Рамка със списък (List Box)
Вертикална лента с плъзгач (Vertical Scrollbar)
Лента за отчитане изпълнението на процес – прогрес индикатор (Progress Bar)
Горещ клавиш (Hot Key)
Дървовидна структура (Tree Control)
Анимация (Animate)
Дата/час (Date/Time Picker)
IP-адрес (IP Address)
Разширен каскаден списък (Extended Combo Box)

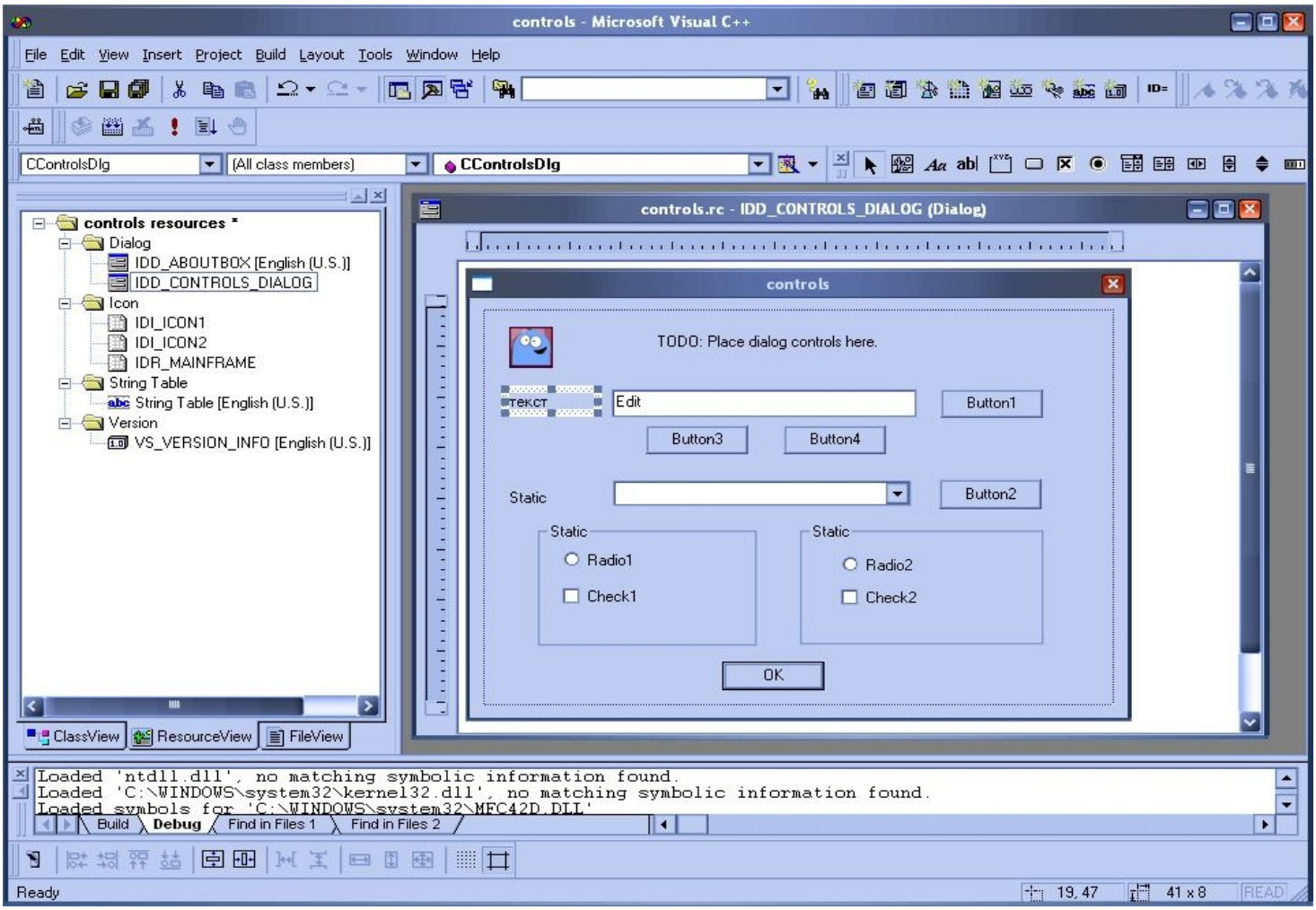
...ами разгледайте си ги кой къде е:)
трябва да знаете,че когато се вмъкват много контроли в приложението,след това се прави задължителна мнемонична проверка...и тогава се започва въвеждането на кода...проверете дали не възникват конфликти при използване на мнемоничните символи от различните контроли, използвани в диалоговата рамка като кликнете с десния бутон на мишката и отидете на Check Mnemonics ,а най-отдолу е Properties (Св-ва) ...ориентирахте ли се? Ок :)))

Ако между мнемоничните символи не възникнат конфликти, Visual C++ ще изведе рамка със съобщение, на което ще узнаете, че всичко е наред...

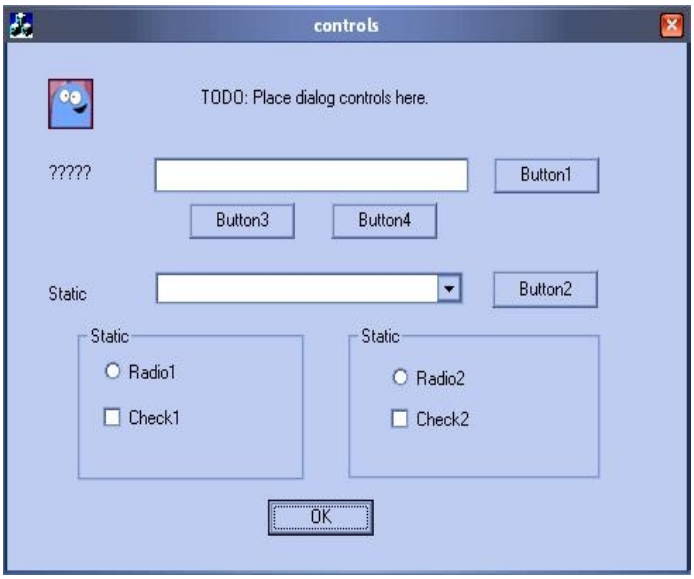
Хайде сега започнете с нов проект...кръстете го примерно test или controls ,или както си искате:)
ще се учим да поставяме контроли:) пак си създавате проекта с помощта на AppWizard и следвате същите стъпки...и започвате да си правите вече скелета на новото проложение...
В горната част на диалоговата рамка ще има текстово поле.Там потребителят ще въведе текст на съобщение, който да бъде изведен при щракване върху бутона,поставен непосредствено до полето.Под тези елементи сложете два бутона...при щракване върху единия от тях в текстовото поле ще се извежда съдържанието на съобщението по подразбиране, а при активиране на другия съдържанието на това поле ще се заличава....отдолу ще има каскаден списък с наименованията на стандартните приложени програми за Windows...и ако потребителят избере някой от изброените елементи, а след това щракне върху бутона за управление вдясно от списъка, ще последва стартиране на избраната програма....
отдолу сложете две групи полета за отметки, които ще влияят върху работата на контролите, разположени в горната половина от диалоговата рамка: контролите за извеждане на съобщението, въведено от потребителя и тези за стартиране на стандартните програми за Windows... левият набор полета за отметки позволява да управлявате действието на всяка от групите контроли,
десният набор е предназначен за разрешаване и забраняване визуализирането на групите с контроли...
в най-долната част на диалоговата рамка е поставен бутон, щракването върху който предизвиква прекратяване изпълнението на програмата

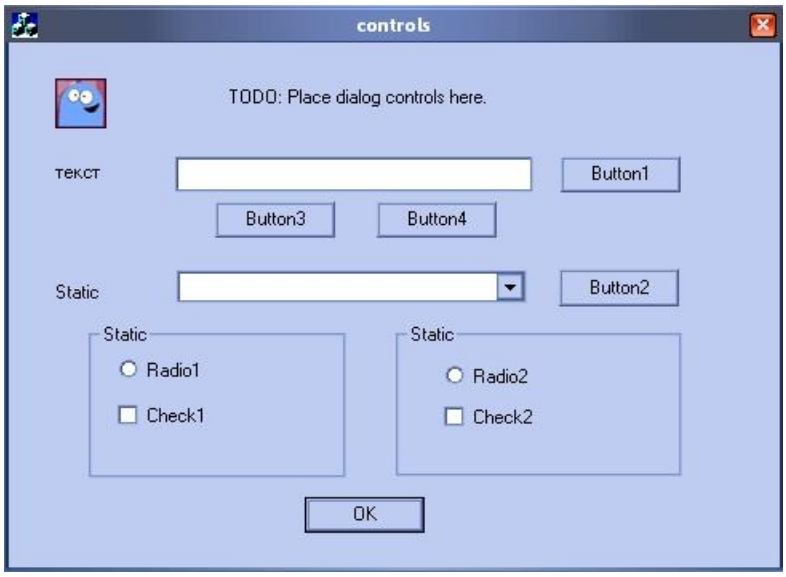
та,това ще представлява новата ни програмка:) хайде сега да й направим скелета (оразмеряване,поставяне на контролите...бутончета и т.н.)

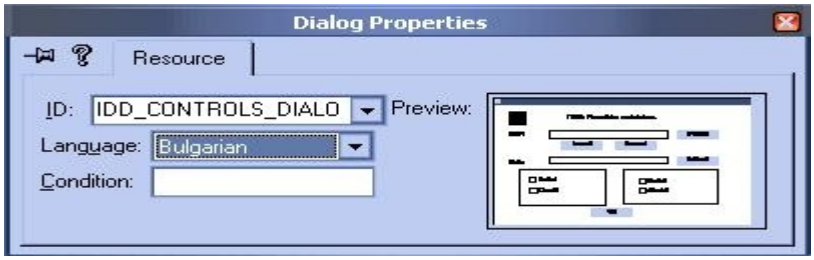
при мен изглежда така:



виждате добавил съм си и картинка:) .ico файлче и съм поставил контролите,и съм си ги сложил където ми кефне:)
сега забележете,че в единия статичен текст съм писал на кирилица...то хубаво,ама по подразбиране програмната среда си е настроена на английски и при компилирането ще имате изведен текст не на кирилица,а със завъртулки, др.буквички:) за да оправите проблема просто идете от Workspace IDD_CONTROLS_DIALOG десен бутон на мишката и Properties...там на Language си отметнете да си е Bulgarian...ето разликите при компилацията:





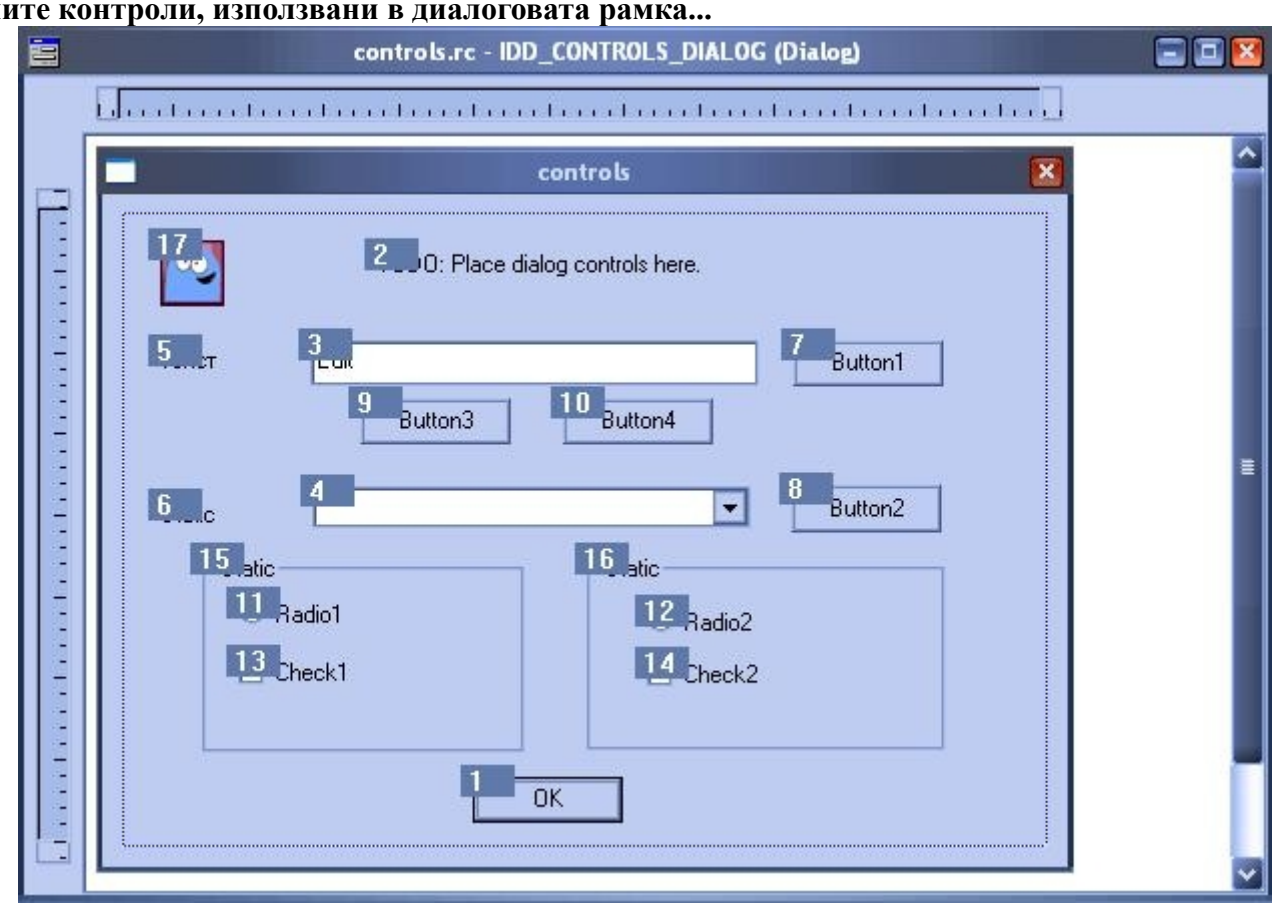


ето какви контроли трябва да имате в скелета на приложението:

3 статични текста (Static text)
1 текстово поле (Edit box)
5 бутона (Buttons)
1 каскаден списък (Combo box)
2 групови рамки (Group box)
4 поленца за отметка (при мен са 2 Radio button и 2 Check box)
1 картинка (Picture)

- ако искате да проверите обхождането (последователността за обхождане на контролите)
след като всички контроли са поставени по местата си в диалоговата рамка, трябва да сте сигурни, че потребителят ще ги обходи в очакваната от Вас последователност, ако за придвижване използва клавиша Tab вместо мишката...нарича се табулиране...

обхождането извършете като си маркирате целия диалогов прозорец(так където слагахте контролите) и от менюто изберете Layout после Tab Order
след активиране на елемента Tab Order ще забележките появата на номера в близост до всяка от контролите...числата посочват последователността, в която ще се извърши обхождането на отделните елементи, ако потребителят използва клабиша Tab
Използвайте мишката и щракнете върху всяко поле с номер,като спазвате последователността, която желаете да следва и потребителят при придвижването си...контролите ще бъдат автоматично преномерирани в съответствие с определената от вас последователност... а за да махнете отметките с цифри пак идете на Layout после Tab Order или натиснете диалоговата си рамка,за да изчезнат...
важно е също да знаете че не се допуска използването на два еднакви мнемонични символа в една диалогова рамка...и последното нещо, което трябва да сторите преди да се съсредоточите в кода на приложната програма е да проверите дали не възникват конфликти при използване на мнемоничните символи от различните контроли, използвани в диалоговата рамка...

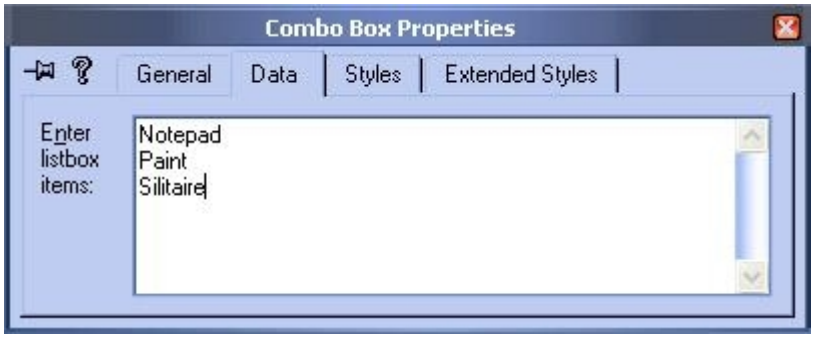


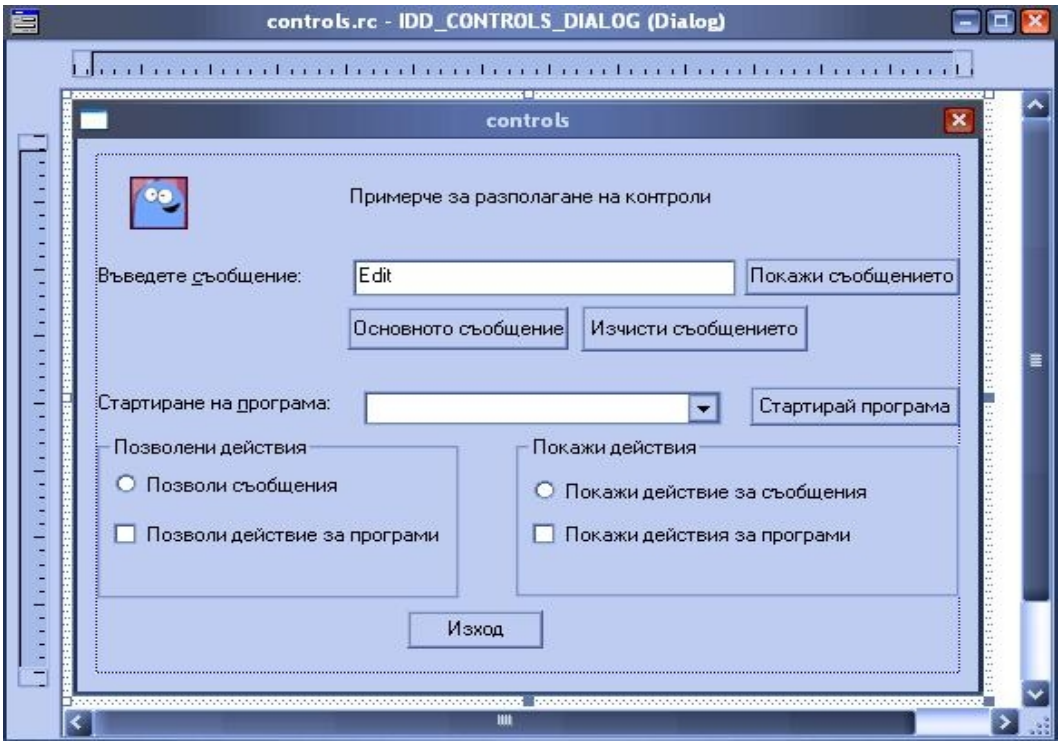
Преди да пристъпите към каквото и да е кодиране, необходимо е да дефинирате променливи за всяка една от контролите, които се нуждаят от задаване на определена стойност – тоест абсолютно всички, без статичния текст и бутоните за управление...

Сега да конфигурираме характеристиките на контролите дето по-горе ги избройх кои съм поставил:
стойности на характеристиките за контролите от диалоговата рамка на програмата

Object (обект)	Property (характеристика)	Setting (стойност)
статичен текст	ID (идентификатор) Caption (заглавие)	IDC_STATIC Примерче за разполагане на контроли
статичен текст	ID Caption	IDC_STATICMSG Въведете съобщение:
статичен текст	ID Caption	IDC_STATICPGM Стартиране на &програма:
текстово поле	ID Caption	IDC_MSG
бутон	ID Caption	IDC_SHWMSG Покажи съобщението
бутон	ID Caption	IDC_DFLTMSG Основното съобщение
бутон	ID Caption	IDC_CLRMSG Изчисти съобщението
бутон	ID Caption	IDC_RUNPGM Стартирай програма
бутон	ID Caption	IDC_EXIT Изход
каскаден списък	ID	IDC_PROGTORUN
групова рамка	ID Caption	IDC_STATIC Позволени действия
групова рамка	ID Caption	IDC_STATIC Покажи действия
поле за отметка	ID Caption	IDC_CKENBLMSG Позволи съобщения
поле за отметка	ID Caption	IDC_CKENBLPGM Позволи действие за програми
поле за отметка	ID Caption	IDC_CKSHWMSG Покажи действие за съобщения
поле за отметка	ID Caption	IDC_CKSHWPGM Покажи действия за програми

хайде сега да добавим елементите за каскадния списък (Combo box)
това става като от диалоговата рамка маркирате,позиционирате в/у каскадния списък и натискате десен бутон на мишката и отивате на св-ва(Properties) и след това на Data и въведете следните стойности Notepad , Paint, Solitaire, като за добавяне на нов ред е необходимо да използвате клавишна комбинация Control+Eriter





ами би трябвало да изглежда ето така облика на диалоговата рамка...
може да си компилирате дебъгвайки и да видите как ще се покаже:)

сега ще прикрепим към контролите променливи:
променливите ще използвате по-късно, когато започнете да съставяте кода на своята програма
стойностите, въвеждани от потребителя посредством контроли от диалоговата рамка върху
екрана, ще бъдат присвоени на дефинираните променливи...съдържанието на контролите върху екрана
ще бъде актуализирано и следователно човекът пред компютъра ще види веднага резултата
от своята работа...
Изпълнете следващите стъпки:
Отворете Class Wizard...щракнете върху етикета Member Variables, за да отворите диалоговата рамка...
(***Дефинирането на променливи към контроли се извършва чрез етикета Member Variables на Class Wizard)***

Изберете идентификатора (ID) на някоя от контролите, към която желаете да дефинирате променлива, например IDCJVISG
Щракнете върху бутона Add Variable
В диалоговата рамка Add Member Variable въведете наименование на променливата...и щракате върху ОК...
Повторете изпълнението и с всички останали контроли, към които е нужно да дефинирате променливи...трябва да дефинирате променливите от тази табличка:

Control (контрола)	Variable name (наименование на променливата)	Category (Категория)	Type (тип)
IDC_MSG	m_strMessage	Value	CString
IDC_PROGTORUN	m_strProgToRun	Value	CString
IDC_CKENBLMSG	m_bEnableMsg	Value	BOOL
IDC_CKENBLPGM	m_bEnablePgm	Value	BOOL
IDCJXSHWMSG	m_bShowMsg	Value	BOOL
IDC_CKSHWPGM	m_bShowPgm	Value	BOOL

Всички наименования на горните променливи започват с префикс m _ , защото това са член-променливи, а тази условност за наименуване е прието да се използва в MFC
След префикса m _ е използвана форма на унгарската система за обозначаване, при която следващите няколко символа от наименованието описват типа на променливата...в случая b означава Булев тип , а str посочва, че променливата е от типа символен низ...
След като приключите с добавяне на всички необходими променливи е нужно да щракнете в/у бутона ОК, за да затворите Class Wizard

Преди да започнете въвеждане на програмен код към контролите от диалоговата рамка на програмата ви, нужно е да инициализирате променливите, а на повечето от тях ще присвоите и начални стойности...
Изпълнете следващите стъпки:
Щракнете върху етикета Message Maps, за да стартирате
Class Wizard. Изберете функцията OnlnitDialog...сега директно посочете в списъка Member functions

Щракнете Върху бутона Edit Code, за да получите изходния код на функцията OnlnitDialog
Намерете маркера TODO,определящ мястото, откъдето може да започне въвеждането на кода и добавете кода:

```
m_strMessage - „Напишете съобщение тук";
m_ShowMsg = TRUE;
m_bShowPgm = TRUE;
m_bEnab1eMsg = TRUE;
m_bEnablePgm = TRUE;
UpdateData(FALSE);
return TRUE;
 }
```

функцията UpdateData е ключът за работа с променливите, декларирани към контроли във Visual C++ тя присвоява данни на променливите и актуализира върху екрана всички контроли в съответствие с новите значения

трябва да проверите дали е възможно потребителят да приключи успешно изпълнението на програмата...понеже отстранихте бутоните за управление ОК и Cancel и създадохте нов бутон за затваряне прозореца на приложната програма...ето защо е нужно да въведете код за функцията, извиквана при щракване върху бутона Изход
Изпълнете следните стъпки: дефинирайте функция към обекта IDQJEXIT на съобщението BN_CLICKED посредством Class Wizard
Щракнете върху бутона Edit Code, за да изведете съдържанието на създадената функция...
Въведете кода:

```
// Изход от програмата
 OnOK();
```

Извеждането текста на съобщението:
Може да добавите функция към бутона Show Message и да извикате функцията MessageBox

```
MessageBox{m_strMessage);
```

Ако сега компилирате програмата и я стартирате, ще забележите,че тя не работи коректно. Какъвто и текст да въведете, винаги ще извежда символния низ, с който инициализирахте променливата m_strMessage във функцията OnlnitDialog. Програмата не изписва онова, което въвеждате в текстовото поле. Неадекватното й поведение се дължи на факта, че не актуализирате стойността на променливата със съдържанието на контролата. Необходимо е да се обърнете към функцията UpdateData, но този път с аргумент TRUE, за да получите данни от текстовото поле и да опресните променливата, преди да извикате функцията MessageBox.

```
UpdateData(TRUE);

// Извеждане на съобщението
 MessageBox(m__strMessage);
```

Ако потребителят предпочита текстовото поле да бъде чисто преди да пристъпи към въвеждане на нов текст, може да прикрепите подходяща функция към бутона Clear Message. Добавянето на функция се извършва по обичайния начин посредством Class Wizard.
Действието на функцията е тривиално: присвояване празен символен низ на променливата m_strMessage, последвано от актуализиране съдържанието на контролите от диалоговата рамка върху екрана, за да се отразят извършените промени.

```
m_strMessage = " ";

// Опресняване съдържанието на екрана
UpdateData(FALSE);
```

Накрая остана да прикрепите функции към полетата за отметки
Позволени действия u Покажи действия
Първото от тях разрешава или забранява на контролите да обработват съобщението, въведено от потребителя. Ако в полето е поставена отметка, действието на контролите е разрешено. В случай, че липсва такава, контролите са забранени. Аналогично, второто поле за отметка разрешава или забранява визуализирането на набора контроли:

```
UpdateData(TRUE);

 // Поставена ли е отметка в полето Позволени действия?
 if (m_bEnableMsg == TRUE)
{

GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
 GetDlgItem(IDC_SHWMSG)->EnableWindow(TRUE);
GetDlgItem(IDC_DFLTMSG)->EnableWindow(TRUE);
 GetDlgItem(IDC_CLRMSG)->EnableWindow(TRUE);
 GetDlgItem(IDC_STATlCMSG)->EnabieWindow(TRUE);
 }
else

 // Не, забраняваме всички контроли,
 // които имат връзка с извеждане съобщението на потребителя
GetDlgItem(lDC_MSG)->EnableWinciow(FALSE);
GetDlgItem(IDC_SHWMSG)->EnableWindow(FALSE);
GetDlgItem(IDC_DFLTMSG)->Enab!eWindow(FALSE);
GetDlgItem(IDC_CLRMSG)->EnableWindow(FALSE);
GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);



 {
 // TODO: Add your control notification handler code here

 ////////////////////////////////////////////////////

 ////////////////////////////////////////////////////

// Получаване на текущите стойности от екрана
UpdateData(TRUE);


if (m_bShowMsg == (TRUE)
{

GetDlgItem(IDC_MSG)->ShowWindow(TRUE);
GetDlgItem(IDC_SHWMSGb)>ShowWindow{TRUE);
 GetDlgItem(IDC_DFLTMSG)->ShowWindow(TRUE);
```

```
 GetDlgItem(IDC_CLRMSG)->ShowWindow(TRUE);
 GetDlgItem(IDC_STATICMSG)->ShowWindow(TRUE);
 }
 else
 {

GetDlgItem(IDC_MSG)->ShowWindow(FALSE);
GetDlgItem(IDCLSHWMSG)->ShowWindow(FALSE);
GetDlgItem(lDC_DFLTMSG)->ShowWindow(FALSE);
GetDlgItem(IDC_CLRMSG)->ShowWindow(FALSE);
GetDlgItem(IDG_STATICMSG)->ShowWindow(FALSE);
}
```

Последното, което остана да реализирате в програмата, е възможността чрез нея да се стартират други приложни програми. Надявам се помните, че въведохте като елементи на каскадния списък наименованията на три програми от Windows. Когато стартирате програмата си следващия път, може да разгледате съдържанието на този списък. Ако изберете някой елемент, наименованието му ще бъде изведено в текстовото поле над списъка.Но все още тази. част от програмата Ви не функционира и поради това е необходимо да въведете код, свързан с бутона Стартирай програма...При изпълнението на този код трябва първо да се получи наименованието на избраната програма от каскадния списък, а след това да се стартира съответният изпълним файл. Използвайте Class Wizard, за да създадете функция към бутона Стартирай програма

```
CString strPgmName;
strPgmName = m_strProgTo Run;

strPgmName.MakeUpper();

 //Потребителят е избрал програма Paint?
 if (strPgmName == „PAINT")
 // Да стартираме програма Paint
WinExec(„pbrush.exe",.$W_SHOW);
// Потребителят е избрал програма Notepad?
 if {strPgmName == „NOTEPAD")
 // Да стартираме програма Notepad
WinExec(„notepad.exe", SW_SHOW);

//Потребителят е избрал програма Solitaire?
 if (strPgmName == „SOLITAIRE")
// Да стартираме програма Solitaire
 WinExec(„sol.exe", SW_SH0W);
```

първото нещо, което прави тази функция е да се обърне към UpdateData, и да актуализира стойностите на променливите с тези на контролите от диалоговата рамка

сега е възможно след компилиране и стартиране на програмата да изберете някоя от приложните програми в каскадния списък и да я стартирате посредством щракване в/у бутона Стартирай програма

Друга функция API, която бихте могли да използвате за същата цел е ShellExecute. Оригиналното й предназначение е да отваря файлове за печат, но е възможно да се употребява и за стартиране на приложни програми.

Менюта и ленти с инструменти:

Средата за разработка на Visual C++ предоставя пълен набор от менюта, позволяващи да управлявате файловете и работните пространства на проектите, да конфигурирате средата, както и да получавате достъп до помощната система, контрола на сорс кода и други външни инструменти. Повечето от менютата имат и съответните ленти с инструменти, които позволяват да избирате опциите с едно щракване... лентите с инструменти са напълно конфигурируеми...можете да определите кои ленти с инструменти да бъдат показани, както и кои бутони да се включват в тях.

Това означава, че имате възможност да конфигурирате вашата среда, така че най-често използваните от вас опции да бъдат лесно достъпни...

Първоначалната инсталация на Visual C++ показва трите най-често използвани ленти с инструменти. Лентата с инструменти Standard, която съдържа най-често използваните команди за работа с файлове; мини-лентата с инструменти Build, съдържаща най-често използваните команди за изграждането и стартирането на приложения; и WizardBar, която предоставя командите за работа с класове.

Щракването с десния бутон на мишката в/у празното пространство на някоя от лентите с инструменти или в/у рамката на главния прозорец ще покаже списък с наличните ленти с инструменти и ще позволи да превключите тяхното състояние - дали са видими или не са.

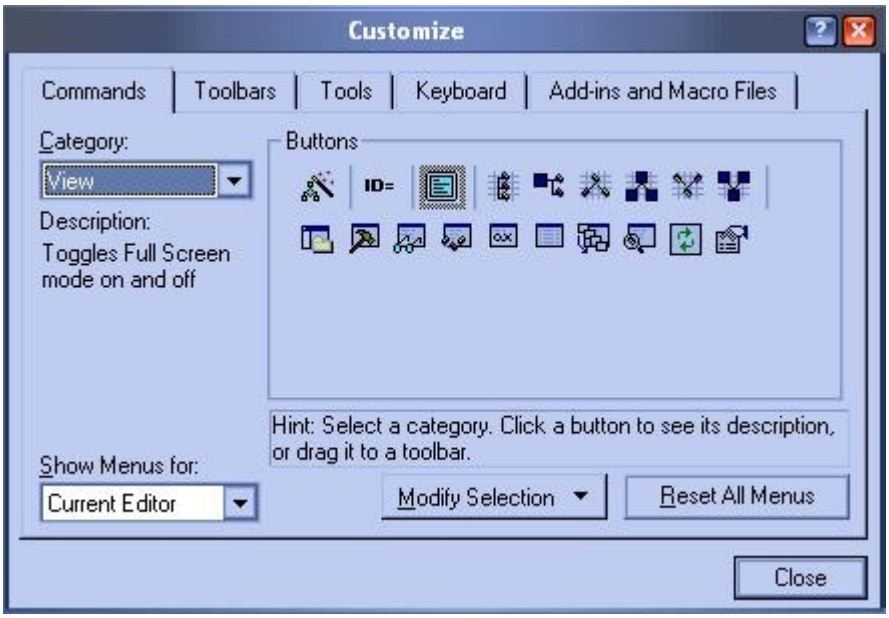
...За да добавите бутон към лента с инструменти:

1. От менюто Tools изберете Customize. Появява се диалоговият прозорец Customize

2. Щракнете в/у страницата Commands

3. В списъка Category щракнете в/у View

4. Щракнете в/у иконата Full Screen (третата от ляво на дясно в най-горния ред на иконките) и забележете, че се появява описание на командата под падащия списък Category



5. Изтеглете иконата Full Screen и я сложете във вашата лента с инструменти Standard

6. Затворете диалоговия прозорец Customize

- докато диалоговият прозорец Customize е отворен,можете да премахвате бутони от лентата с инструменти като щракнете в/у тях с десния бутон на мишката и изберете Delete от контекстното меню...

... За да зададете клавиш за пряк достъп:

В менюто Tools щракнете върху Customize. Появява се диалоговият прозорец Customize.

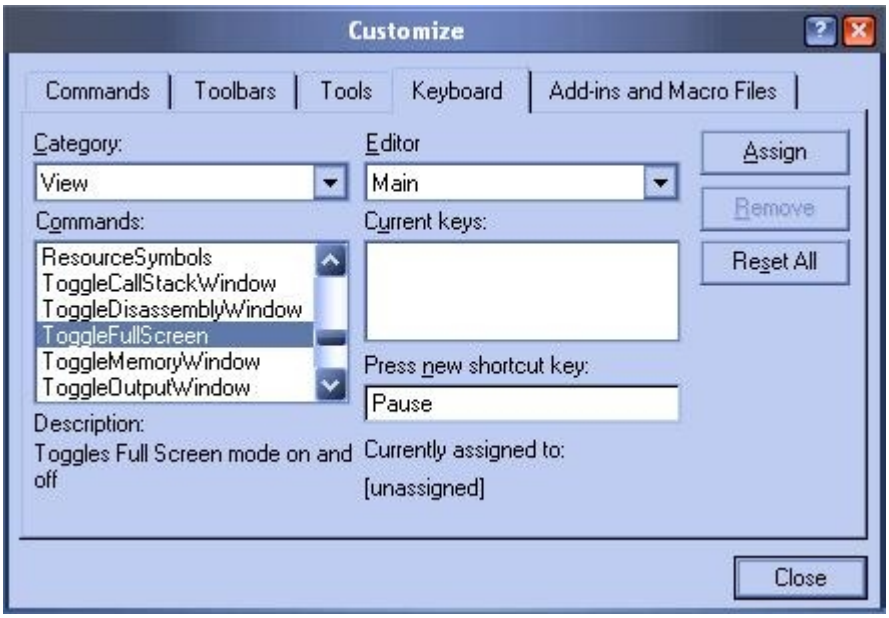
щракнете върху етикета на страницата Keyboard

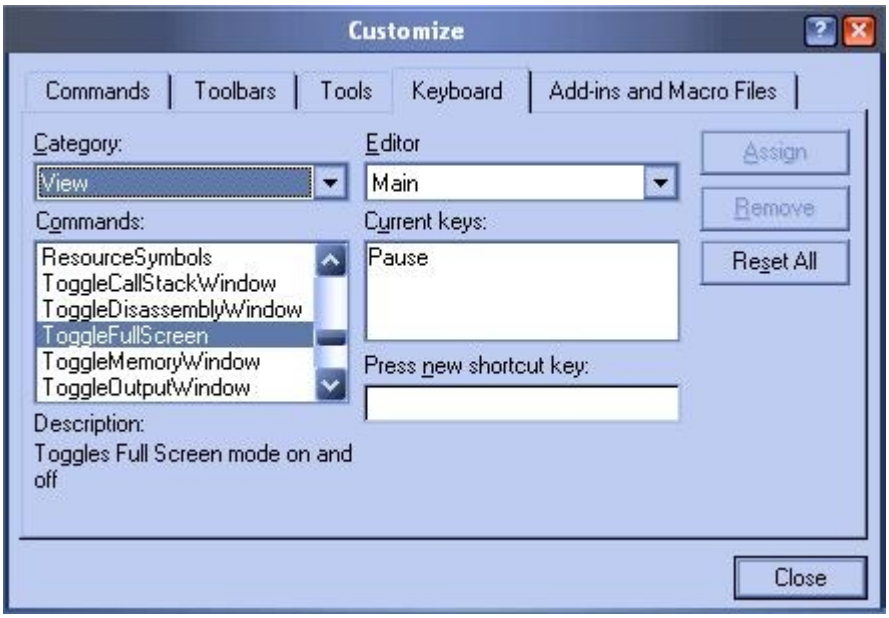
в списъка Category щракнете върху View

В полето Commands щракнете в/у ToggleFullScreen и забележете, че се появява описание на командата под падащия списък Category

щракнете в edit контрола Press new shortcut key и след това натиснете PAUSE (Pause Break) от клавиатурата... щракнете върху Assign...

ето как изглежда:





разбира се всичко това тествайте при отворен някакъв проект:)

... <u>За да промените настройките на проект:</u>
В менюто Project щракнете върху Settings и се появява диалогов прозорец Project Settings
Щракнете върху страницата C/C++
В списъка Category щракнете върху C++ Language
В полето Settings For щракнете върху All Configurations и забележете, че само настройките, които са общи за всички конфигурации, се показват в полето Common Options
Изберете Enable Run-Time Type Information (RTTI) (информация за типа по време на изпълнение), за да може RTTI да бъде достъпна за всички конфигурации на проекта...кликнете на ОК...

някои от опциите, които можете да настроите с помощта на диалоговият прозорец Project Settings са:
General Settings: Тук можете да определите дали да свързвате изпълнимата програма със статични MFC библиотеки...можете също така да установите и директории за изходните файлове

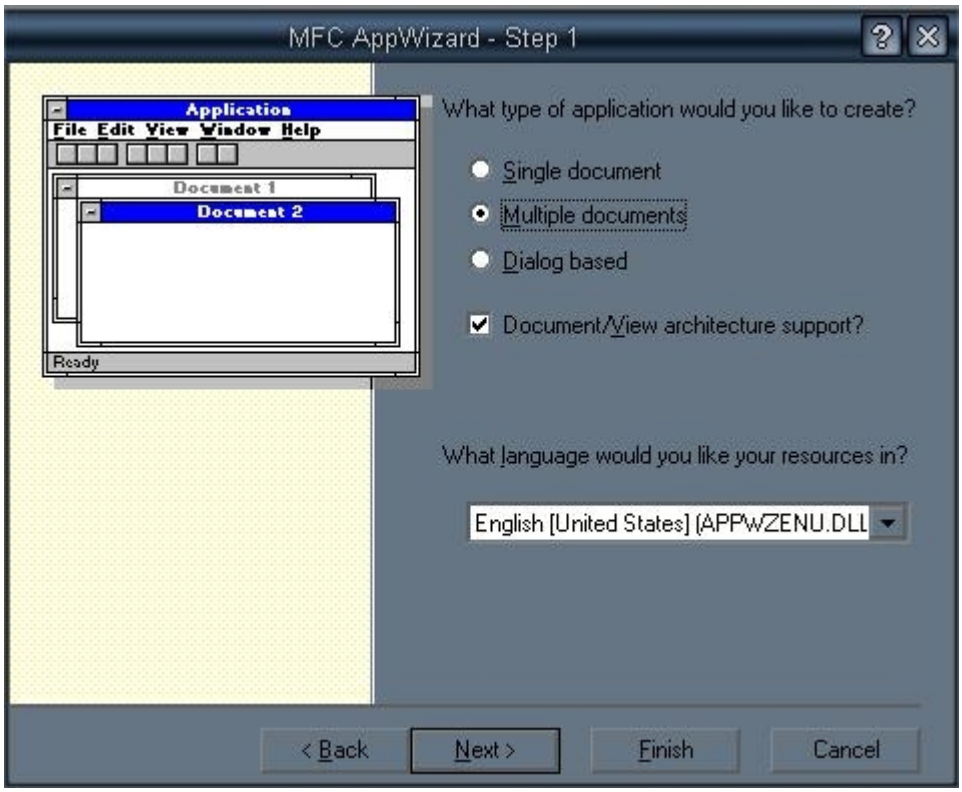
Debug Settings: Тук можете да зададете на вашата програма аргументи от командния ред, когато я стартирате от дебъгера. Можете също да използвате I/O пренасочване, все едно че се намирате на командния ред

C/C++ Settings: Тук можете да определите основни настройки на компилатора , възможности на езика, конвенции на извикване, зависещи от процесора настройки, оптимизации, препроцесорни дефиниции

Linker Settings: Тук можете да определите допълнителни библиотеки за свързване с вашата програма

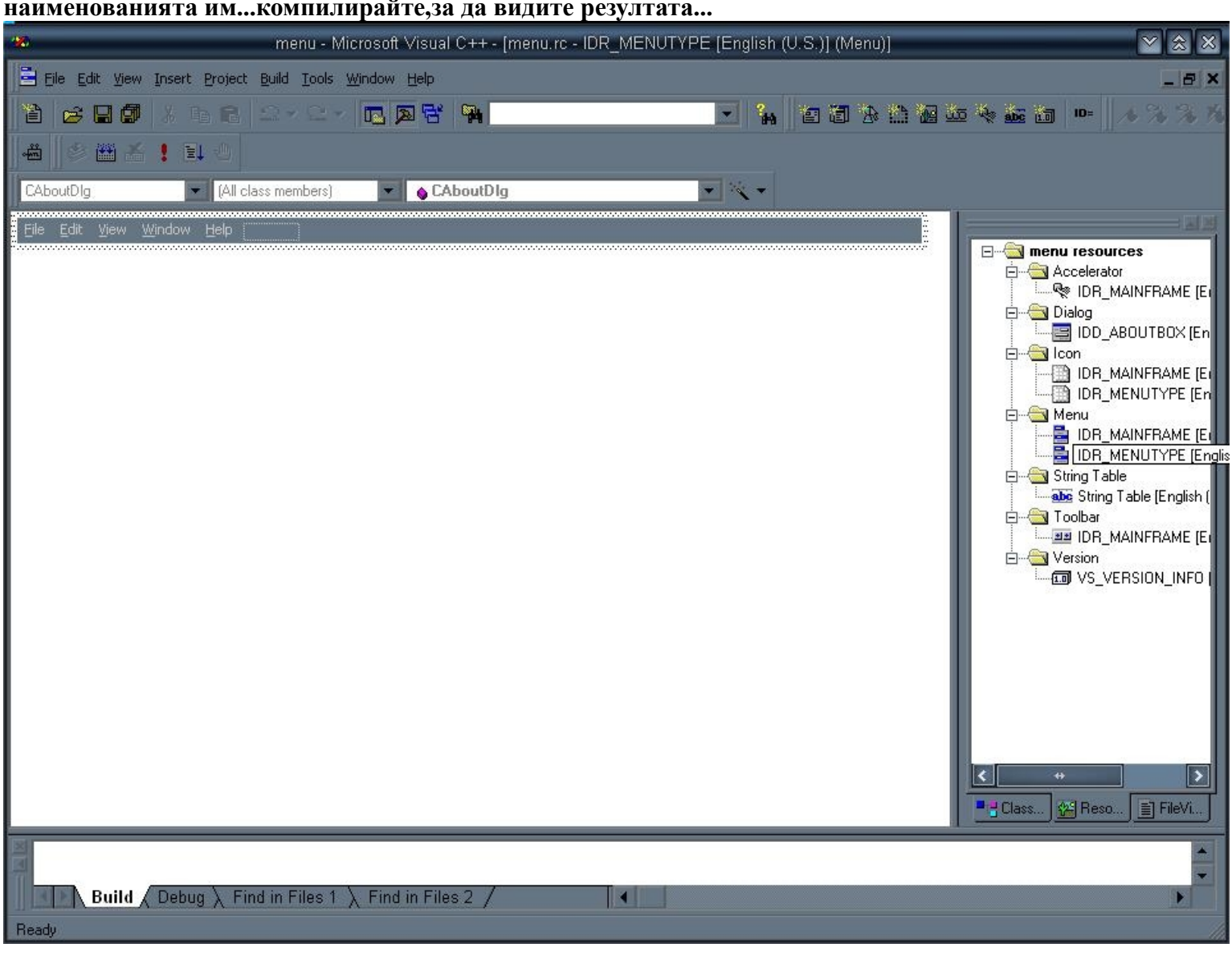
Създаване на менюта, ленти с инструменти, лента за състояние

Създавате си нов проект като този път вместо Dialog based отметнете на Multiple documents както е показано:



Би трябвало проекта да изглежда ето така:

Кликайки в/у менютата File Edit View Window Help те са в работния прозорец ... можете да променяте наименованията им...компилирайте,за да видите резултата...



Изграждане на приложения с библиотеката Microsoft Foundation Classes

досега използвахме MFC AppWizard за създаване на проект,който е базиран на Microsoft Foundation Classes и сега ще се опитам да обясня спецификата на MFC и ролята им (на класовете) в разработката на уиндоус приложения:)

Библиотеката MFC представлява колекция от C ++ класове и глобални функции, които са проектирани за бърза разработка на проложения под Microsoft Windows. MFC предлага много предимства за C++ разработчиците.Тя опростява Windows програмирането,съкращава времето за разработка и прави кода по преносим... MFC предоставя лесен достъп до трудните за програмиране технологии като ActiveX и програмирането за Интернет. Библиотеката улеснява програмирането на такива възможности на потребителския интерфейс като листове за свойства, предпечатен преглед, контекстни менюта , потребителски плаващи ленти с инструменти и ToolTip контроли...
Win32 API е набор от функции, структури, съобщения,макроси и интерфейси, предоставящи еднотипен интерфейс, който позволява да разработвате приложения за произволни Win32 платформи.
Ето някои от услугите,които Win32 API предлага:
Windows Management - Предоставя средствата за създаването и управлението на потребителски интерфейс
Windows Controls - Предоставя набор от общи контроли на потребителския интерфейс. Използването на тези контроли помага да се запази приликата на потребителския интерфейс на приложението с този на шела и другите приложения
Shell Features(възможностите на шела) - Предоставя достъп до обекти и ресурси на системата, като файлове, устройства за съхранение, принтери и мрежовите ресурси
Grafics Device Interface - Предоставя функциите и съответните структури, които се използват за генерирането на графичен изход за екрана, за принтери и за други графични устройства
System Services – предоставя достъп до ресурсите на компютъра посредством възможностите на операционната с-ма

Например Win32 концепцията за прозорец се капсулира от MFC класа CWnd или един клас наречен CWnd капсулира HWND манипулатора(HWND е дефиниран от Win32 тип данни,който представя Win32 прозорец)...капсулирането означава,че класът CWnd съдържа една член-променлива от тип HWND,а неговите член функции капсулират извикванията към Win32 функциите ,приемащи HWND като параметър...член функциите на MFC класовете имат същото име като Win32 функцията,която капсулират...
MFC капсулира повечето,но не всички неща на Win32 API

Йерархия на MFC класовете:
Библиотеката MFC е реализирана като набор от C++ класове...един от основните методи за преизползване на код в C++ е механизма за *наследяване* (inheritance). Един C++ клас може да бъде производен на родителски клас и да наследи неговите характеристики...библиотеката MFC по подобие на много други библиотеки със C++ класове организира своето съдържание в йерархия на наследяване. Тази йерархия съдържа голямо количество класове със специфични функции, които са производни на малък брой базови класове...

Категории класове на MFC:

Категория от класове на MFC	Описание
Архитектура на приложението	Класовете за архитектура на приложението представят основните елементи от архитектурата на едно приложение и включват класа CWinApp, който представя самото приложение
Потребителски интерфейс	Класовете за потребителски интерфейс обикновено включват елементите на Windows приложението, които са видими за потребителя. Това включва прозорци, диалози,менюта и контроли. Класовете за потребителски интерфейс капсулират и устройствения контекст на Windows, както и обектите на интерфейса за графични устройства (GDI)

Колекции	**MFC предоставя няколко лесни за използване класове за колекции,включващи масиви(arrays),списъци(lists),карти(maps) те са с и без шаблони...**
Общо предназначение	**MFC включва няколко класове с общо предназначение, които не капсулират Win32 API функции...тези класове представят както прости типове данни,като точки и правоъгълници,така и сложни типове като низовете**
ActiveX	**MFC предоставя класове, които опростяват процеса на добавяне на ActiveX възможности към Вашите приложения и значително съкращават времето за разработка. ActiveX класовете работят с останалите класове за фреймуърк на приложения, за да предоставят лесен достъп до ActiveX API интерфейса.**
База данни	**Достъпът до данни посредством свързването към база данни е една от най-разпространените задачи на програмирането в Windows среда. MFC предоставя класове, които позволяват операции с бази данни чрез Open Database Connectivity(ODBC) и Data Access Object (DAO)**
Интернет	**Създаването на приложения, работещи през Интернет или Интранет се превърна в основна задача за разработчиците. MFC включва Winlnet API и Internet Server API (ISAPI), които предоставят класове за приложения, работещи съответно от страна на клиента и от страна на сървъра.**
Глобални функции	**MFC предоставя няколко функции, които не са членове на класове. Тези глобални функции обикновено започват с Afx и предоставят на програмиста средства с общо предназначение. Най-често използваният пример за такава функция е AfxMessageBox( )**

DLL библиотеките на MFC

AppWizard дава право да изберете как да използвате MFC – дали като споделена DLL библиотека, или като статична библиотека, която е свързана с изпълнимия файл на вашето приложение. Ако изберете опцията за споделена DLL библиотека, ще трябва да сте сигурни,че библиотеката на MFC - MFCxx.DLL (xx представлява номера на версията,която ползвате), както и стандартната библиотека на Visual C++ MSVCRT.DLL са инсталирани на компютъра на потребителя...
Обикновено,за да се осигури това условие dll библиотеките се разпространяват заедно с приложението. Избирайки свързването към споделена DLL библиотека, можете да намалите значително размера на изпълнимата програма! Тази стратегия има смисъл, когато смятате да инсталирате голям брой създадени с MFC приложения на един комп
По този начин всички приложения ще споделят един набор от DLL библиотеки, вместо да заемат излишно дисково пространство с множество копияна едни и същи DLL библиотеки.

Разширени DLL библиотеки на MFC

По време на софтуерната разработка със C++ често ще създавате потребителски класове, които след това ще преизползвате в др. приложения. Подобно на MFC, тези потребителски класове се опаковат в DLL библиотеки. Класове, които са достъпни за приложенията чрез DLL, се наричат експортирани (exported) техните public член-функции и член-променливи са видими за клиентските приложения. Например DLL библиотеките на MFC експортират класа CString. Това означава, че приложения, които се свързват с DLL библиотеките на MFC, могат да създават и използват CString обекти в своя код.

Можете да щракнете върху MFC AppWizard (dll) в диалоговия прозорец New Projects, за да създадете DLL библиотеки, които да експортират ваши собствени класове. AppWizard се отнася към DLL библиотеки от този тип като към обикновени (regular) DLL библиотеки.
Обикновените DLL библиотеки могат да използват MFC класове в своите имплементации...да разгледаме случая, когато искате да експортирате клас, който е производен на MFC клас... да предположим, че имплементирате клас за потребителски диалогов прозорец CMyDialog , наследяващ функционалността си от MFC класа CDialog и предоставящ набор от public член-функции, които могат да бъдат извикани от дадено клиентско приложение за установяването и извличането на стойности от контролите. Как да осигурите правилното експортиране на базовия клас (CDialog) и достъпа на клиентските приложения до неговите public член-функции и член-променливи?
За да дадете възможност на MFC приложенията да използват CMyDialog като MFC клас,да извикват член функции на базовия клас CDialog и да се обръщат към CMyDialog чрез CDialog указатели, трябва да пакетирате този клас в специален тип DLL библиотека,известна като разширена dll библиотека на MFC (MFC extension dll library)
Една разширена DLL библиотека на MFC имплементира преизползваеми класове, които са производни на съществуващи MFC класове. Разширените DLL библиотеки на MFC позволяват да предоставяте разширена версия на MFC.
Разширените DLL библиотеки на MFC се изграждат със споделената DLL версия на MFC. Само MFC изпълними файлове (или DLL библиотеки), които са изградени със споделената версия на MFC, могат да използват разширена DLL библиотека. Както и клиентското приложение, така и разширената DLL библиотека трябва да използва една и съща версия на DLL библиотеките на MFC.
За да изградите разширената DLL библиотека, щракнете върху
MFC AppWizard (dll) в диалоговия прозорец New Projects и изберете M FC Extension DLL в първата страница на MFC DLL App-Wizard ... ...

Библиотеката Microsoft Foundation Classes (MFC) е реализирана като колекция от C++ класове, които са принципно проектирани за създаването на приложения за Microsoft Windows.
MFC е проектирана, за да капсулира най-често използваните функции на Win32 API.
Win32 API е набор от функции, структури,съобщения, макроси и интерфейси, предоставящи еднотипен интерфейс, който позволява да разработвате приложения за произволни Win32 платформи.
MFC опростява разработката под Windows, като скрива някои от най-сложните възможности на Win32 API.
MFC би трябвало да се използва навсякъде, където това е възможно, защото спестява време и усилия!
Дори да използвате Microsoft Foundation Classes пак можете да се обръщате към Win32 API, когато трябва да извършвате някои задачи от ниско ниво, които не се поддържат от MFC
Въпреки че Win32 API е проектиран като интерфейс за всички 32-битови(разрядни) Windows операционни системи, съществуват известни разлики между платформите. Например Windows NT използва вътрешно Unicode низове и поддържа NTFS, а Windows 95 и Windows98 не.
Въпреки че използването на MFC изолира разработчиците от повечето различия между платформите, все пак трябва да сте наясно с някои въпроси, свързани с различните платформи.
MFC е реализирана като <u>йерархичен набор</u> от C++ класове,които използват механизма за наследяване на езика C++, за да предоставят на Win32 разработчиците преизползваема и разширяема база от код

Win32 архитектура за приложения:

За да разберете как MFC имплементира едно Windows приложение трябва да сте наясно с архитектурата на Win32 платформата и с приложенията, които работят на нея.
Доброто познаване на целевата платформа е от съществено значение за разработката на ефективни приложения.

***<u>Процеси и нишки</u>*:**

Едно написано за Windows приложение се състои от един или повече процеси. Най-просто казано *процес (process)* е една инстанция на изпълнимата програма. Процесът има адресно пространство и определени ресурси, както и една или повече нишки, които работят в контекста на процеса.
***Нишката (thread)* е основната единица, за която операционната система заделя процесорно време и е най-малкото парче код , което може да бъде планирано за изпълнение...една нишка работи в адресното**

пространство на процеса и използва ресурсите,които са заделени за него...
Процесът винаги има поне една нишка за изпълнение известна е като първична (primary) нишка...
Можете да създавате допълнителни вторични (secondary) нишки за изпълнението на фонови задачи или да се възползвате от възможността за множество нишки (многонишковост) на 32 битовите (разрядни) уиндоус операц. с-ми...използването на повече от една нишка в едно проложение е известно като многонишково програмиране (multithreading)

<u>Стартиране на приложения:</u>
Когато дадено приложение се стартира, операционната система създава един процес и започна да изпълнява първичната нишка на този процес. Когато тази нишка приключи, приключва и процесът! Тази първична нишка е предоставена на операционната система в кода за стартиране като адрес на функция. Всички Windows приложения дефинират една функция за входяща точка, наречена WinMain() Адресът на WinMain() се предоставя като първична нишка.
След това приложението преминава към създаването на прозорци, които ще съставляват потребителския интерфейс. Преди прозорците да могат да бъдат показани, е нужно да се регистрират различни типове прозоречни класове (window classes) в операционната система.
Прозоречните класове са шаблони, предоставящи подробностите за начина, по който прозорците трябва да се създадат. По време на процеса на регистрация на прозоречни класове, прозорците се асоциират с прозоречна процедура (window procedure), чрез която можете да определите какво да показва прозорецът и как да отговаря на потребителски вход, като дефинирате начина, по който прозорецът ще отговаря на системните съобщения.
<u>Windows съобщения:</u>
Докато прозорецът се явява основната форма за комуникация между потребител и приложение, то различните типове системни съобщения ръководят вътрешните комуникации между операционната с-ма, приложенията и техните компоненти. Например, когато създавате една инстанция на дадено приложение, операционната
с-ма изпраща серия от съобщения към приложението, което отговаря като се инициализира. Действия с клавиатурата или мишката карат операц. система да генерира съобщения и да ги изпраща на самото приложение. Главната задача на едно Windows приложение може да се разглежда като обработка на получаваните съобщения. Тази обработка включва пренасочването на съобщения към прозореца, за който те са предназначени, както и изпълнението на очакваните отговори в зависимост от типа и параметрите на съобщенията. Задачата на разработчика на приложението е да свърже тези съобщения към функциите,които ще ги обработват и да осигури съответен отговор на тези съобщения.
Всяка нишка за изпълнение,която създава прозорец се асоциира с опашка за съобщения.Опашката за съобщенията представлява структура от данни, в която операц. С-ма сухранява съобщенията за даден прозорец. Всички уиндоус приложения имат главен прозорец на приложение.Всички главни прозорци имат цикъл за съобщения,който пък е програмен код ,който извлича съобщенията от опашката за съобщения и ги насочва към съответната прозоречна процедура.А тя би могла да предостави специфичната за приложението обработка на съобщението или го подава на подразбиращата се прозоречна процедура(default window procedure) ,а тя е дефинирана от операц. с-ма функция,която предоставя подразбиращата се обработка за съобщения...примерно едно съобщение изпратено,за да уведоми приложението,че потребителя е минимизирал главния прозорец на приложението ще се обработи по един и същ начин от абсолютно всички приложения.Съобщенията могат да бъдат генерирани както от функциите PostMessage( ) и SendMessage( ) , така и от хардуерни събития
Може да се ползват тези Win32API функции или техните MFC еквиваленти CWin::PostMessage( ) и CWin::SendMessage( ) ,за да изпращате Уиндоус съобщения към или от приложението си...
Функцията PostMessage( ) поставя съобщението в асоциираната с прозореца опашка за съобщения и връща управлението, без да чака прозореца да обработи съобщението.
SendMessage( ) праща съощение на даден прозорец и не се връща,докато прозореца не е обработил съобщението.

<u>Фреймуърк(фреймуорк) за приложения на MFC: Framework</u>

Освен че капсулира Win32 API, MFC дефинира и малка група класове за представянето на общи обекти за приложения и установява взаимовръзки в тази група, с които се реализират основните поведения на Windows приложенията.
Тези класове за архитектура на приложението заедно с няколко глобални функции съставляват фреймуърк за приложения,които можете да използвате като основа за конструирането на приложения. Можете да използвате MFC AppWizard за генериране на набор от класове, които са производни на

фреймуърк класовете. Можете също така да надграждате тези класове, за да конструирате приложение, което да отговаря на ваши определени изисквания.
MFC класа CWinApp представя приложението като цяло. CWinApp е производен на класа CWinThread който представлява нишка в едно MFC приложение. CWinApp представя първичната нишка на процеса на приложението и капсулира инициализацията,работата и завършването на едно уин приложение.
Ето и член функциите на CWinApp

InitInstance( )	**Инициализира всяка нова инстанция на работещо под Windows приложение.Показва главния прозорец на приложението**
Run( )	**Предоставя имплементация на цикъла за съобщения**
OnIdle( )	**Извиква се от фреймуърка, когато не се обработват никакви Windows съобщения. Можете да предефинирате тази функция, за да изпълнявате фонови задачи.**
ExitInstance( )	**Извиква се при всяко приключване на дадено копие на вашето приложение.**

Приложение, което е изградено с MFC фреймуърк, трябва да имплементира само един производен на CWinApp клас. Също така трябва да осигурите собствена предефинирана версия на член функцията InitInstance(). Функцията InitInstance() се извиква директно от функцията WinMain(), така че това е мястото за специфичната инициализация на вашето приложение.

Фреимуъркът предоставя стандартна WinMain() функция за вашето приложение. WinMain() извиква няколко глобални функции за изпълнението на стандартни инициализации, като например регистриране на прозоречни класове. След това тя извиква член-функция InitInstance() на обекта на приложението за инициализиране на приложението. После WinMain() извиква член-функцията Run() на обекта на приложението, за да стартира цикъла за съобщения. Цикълът за съобщения ще продължи да приема и насочва съобщения докато не получи съобщение WM_QUIT
Тогава той извиква функцията ExitInstance( ) на обекта на приложението и се връща.
След това WinMain() ще извика функции за почистване и ще затвори приложението.

ActiveX контролите и обикновените контроли на Windows

Един ActiveX контрол е софтуерен модул, който се включва във вашата C++ програма по същия начин, по който се включва и един Windows контрол ... така изглежда на пръв поглед:)
да разгледаме сега приликите и разликите между ActiveX контролите и тези контроли, които познавате като: текстовото поле(edit) , списъчното поле(list box) и др. Windows контроли с общо предназначение (common controls) , които работят по почти същия начин...
<u>Обикновените контроли</u>
...текстовото поле(edit) , списъчното поле(list box) и др. всички те са дъщерни прозорци, които вие използвате най-често в диалози и които са представени от MFC класове, като CEdit или CTreeControl
Програмата-клиент винаги е отговорна за създаването на дъщерния прозорец на съответния контрол.
Обикновените контроли изпращат на диалога нотифициращи командни съобщения (стандартни Windows съобщения), като BN_CLICKED и др.
Ако искате да извършите някакво действие върху даден контрол, извиквате *C*++ функция, член на класа на този контрол, която изпраща Windows съобщение към него. Контролите са "пълноправни" прозорци, защото всички техни класове са наследници на CWnd ,така че ако искате да вземете текста от един edit-контрол , извиквате CWnd::GetWindowText тази функция работи чрез изпращане на съобщение към контрола...

Windows контролите са съставна част на Windows, въпреки че контролите с общо предназначение се намират в отделна DLL. Друг вид обикновени контроли, така наречените потребителски контроли (custom controls), се създават от програмиста и се държат като обикновените контроли, като изпращат WM_COMMAND нотификации към родителския прозорец и получават дефинирани от потребителя съобщения...

<u>По какво ActiveX контролите си приличат с обикновените контроли:</u>

Можете да считате един ActiveX контрол за дъщерен прозорец, точно както е и обикновеният контрол. Ако искате да включите даден ActiveX контрол в диалог, използвате диалоговия редактор, за да го поставите там и идентификаторът на контрола се появява в ресурсния шаблон. Ако създавате „летящ“ AcliveX контрол (т.е. не в ресурса, а по време на изпълнение на програмата) трябва да извикате функцията Create на класа, представящ този контрол...обикновено във функцията за обработка на WM_CREATE за родителския прозорец. За да извършвате някакви действия с даден ActiveX контрол извиквате C++ функция, както и при обикновените контроли. Прозорецът, който съдържа контрола, се нарича контейнер...

<u>По какво ActiveX се различават от обикновените контроли:</u>

Най-забележителните особености на ActiveX контролите са техните свойства(properties) и методи (methods) Онези C++ функции,които извиквате, за да работите с една инстанция на даден контрол,са тясно свързани със свойствата и методите. Свойствата (наричани още и характеристики) имат символни имена, на които се съпоставят целочислени индекси. На всяко св-во създателят (дизайнерът) на контрола приписва име, като BackColor или GridCellEffect, и тип (като string, integer, double и др.) Има дори и картинен тип за битмапи и икони. Програмата-клиент може да зададе ст-ст на дадено свойство (характеристика) на един ActiveX контрол като укаже неговия целочислен индекс и съответната стойност. Програмата съответно може да извлече стойността на дадено свойство, като укаже неговия индекс и приеме върнатата стойност. В дадени случаи ClassWizard ви позволява да дефинирате член-променливи в класа на прозореца на вашата програма-клиент, които са асоциирани със свойствата на контролите, съдържани от клиентския клас. Генерираният Dialog Data Exchange (DDX) код извършва обмен на данни между свойствата на контрола и член-променливите на класа и на клиента.

Методите на ActiveX контролите са като функциите. Един метод има символично име, набор от параметри и връщана стойност. Извикването на един метод става чрез извикване на съответната член-функция на класа, представящ контрола. Един разработчик на контроли може да дефинира всякакви методи като например PreviousYear, LowerControlRods и т.н.

Един ActiveX контрол не изпраща WM_ нотифициращи съобщения към своя контейнер по начина, по който го правят обикновените контроли. Вместо това той "предизвиква" събития. Едно събитие (event) има символично име и може да има произволна последователност от параметри - то в действителност е функция на контейнера, която контролът извиква. Подобно на нотифициращите съобщения на обикновените контроли, събитията също не връщат резултат към ActiveX контрола, който ги предизвиква. Примерни събития са Click, KeyDown, NewMonth и т.н. Събитията се "асоциират" с функция във вашия клиентски клас, по същия начин, както и нотифициращите съобщения на контролите...

В MFC , ActiveX контролите се държат точно като дъщерни прозорци,но има значителен слой код между прозореца-контейнер и прозореца на контрола...всъщност, контролът може и въобще да няма прозорец. Когато извикате Create прозорецът на контрола не се създава директно...вместо това кодът на контрола се зарежда и получава команда за активиране на място (in place activation)

След това ActiveX контролът създава свой собствен прозорец, до който MFC ви дава достъп чрез указател от тип CWnd

Обаче не е добра идея клиентът да използва директно манипулатора hWnd на контрола...

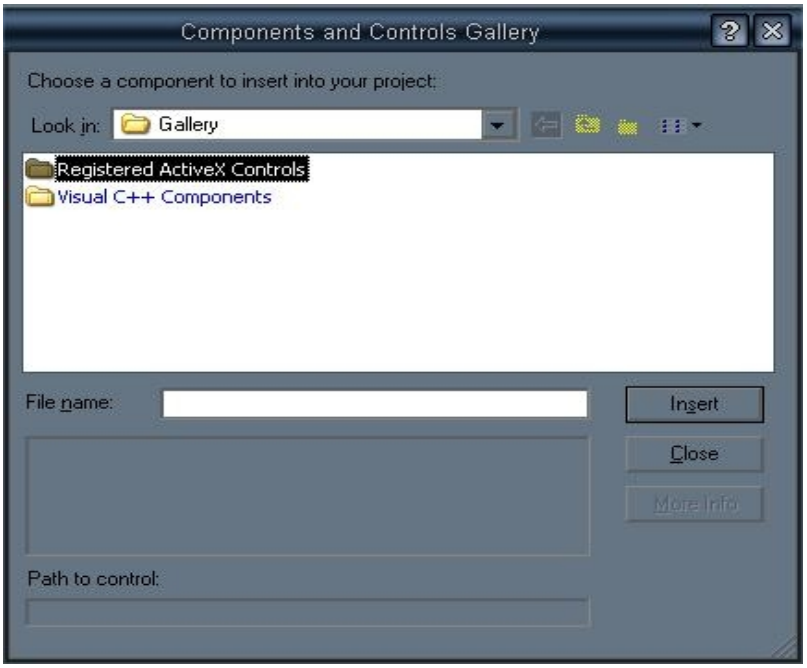
За съхраняване на ActiveX контроли обикновено се използва DLL , но той често има разширение OCX Вашата програма-контейнер зарежда съответните DLL библиотеки , когато има нужда от тях, използвайки COM техники ,зависещи от регистратурата (Registry) на Windows

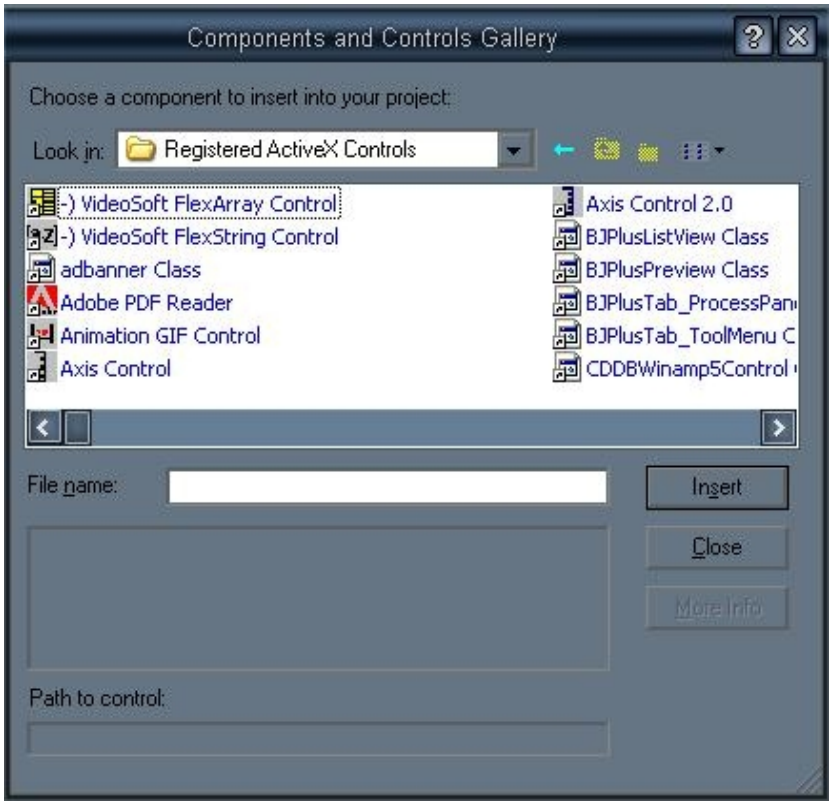
след като веднъж сте указали един ActiveX контрол по време на създаване, той ще бъде зареден по време на изпълнение. Важно е да се знае, че когато разпространявате една програма, изискваща специални ActiveX контроли, ще трябва да включите и OCX файловете и подходяща инсталираща (setup) програма...(може да се ползва уиндоус с-та за целта или winrar програмата и много др.)

<u>Инсталиране на ActiveX контроли:</u>

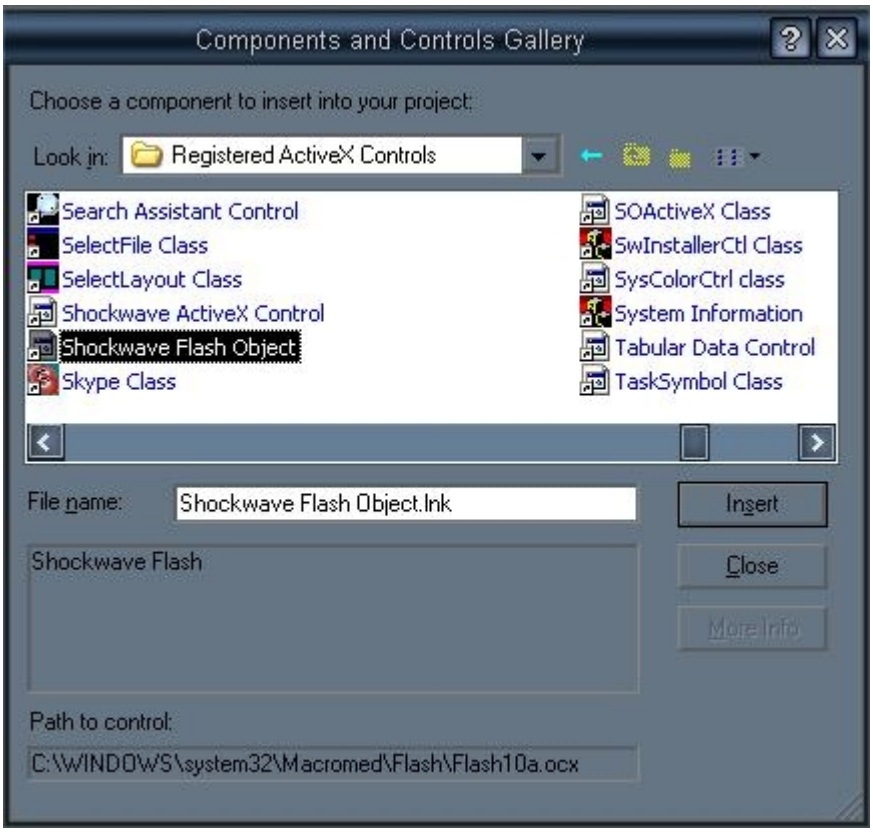
Примерно намерили сте си един хубав ActiveX контрол и искате да го използвате във вашия проект... Първата стъпка, която трябва да предприемете е да копирате DLL-файла на този контрол на вашия диск. Можете да го сложите където желаете, но ще ви бъде по-лесно да следите вашите ActiveX

контроли, ако ги съхранявате на едно място, като например системната директория...
Копирайте съответните допълни файлове, като помощни (HLP) или лицензни (LIC) в същата директория...
Следващата стъпка е да регистрирате контрола в Windows Registry...ActiveX контролът се регистрира сам, когато една програма-клиент извиква специална "експортирана" функция...някои контроли имат лицензни изисквания, което може да доведе до необходимостта от въвеждане на допълнителна информация в Registry
Лицензираните контроли обикновено идват с инсталиращи програми, които се грижат за тези подробности...
След като регистрирате вашия ActiveX контрол, трябва до го инсталирате във всеки проект, който го използва. Това не означава, че OCX файлът се копира. Това значи, че Class Wizard генерира копие на C++ клас, който е специфичен за контрола и че той (контролът) се появява в палитрата с контроли на диалоговия редактор за този проект...
За да инсталирате един ActiveX контрол в даден проект, от менюто Project изберете Add To Project-> след което изберете Components And Controls...
Кликайте в/у Registered ActiveX Controls директорията както е показано:

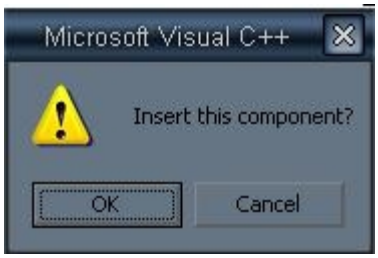




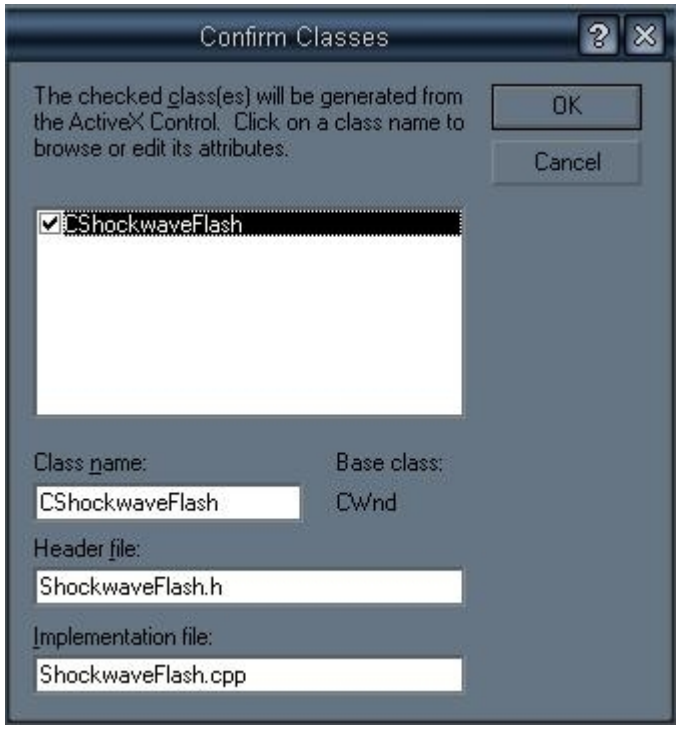
Това дава изглед(списък) на всички ActiveX контроли текущо регнати на системата...
...разгледайте си ги...и аз си избрах Flash(Shockwave:) control ...



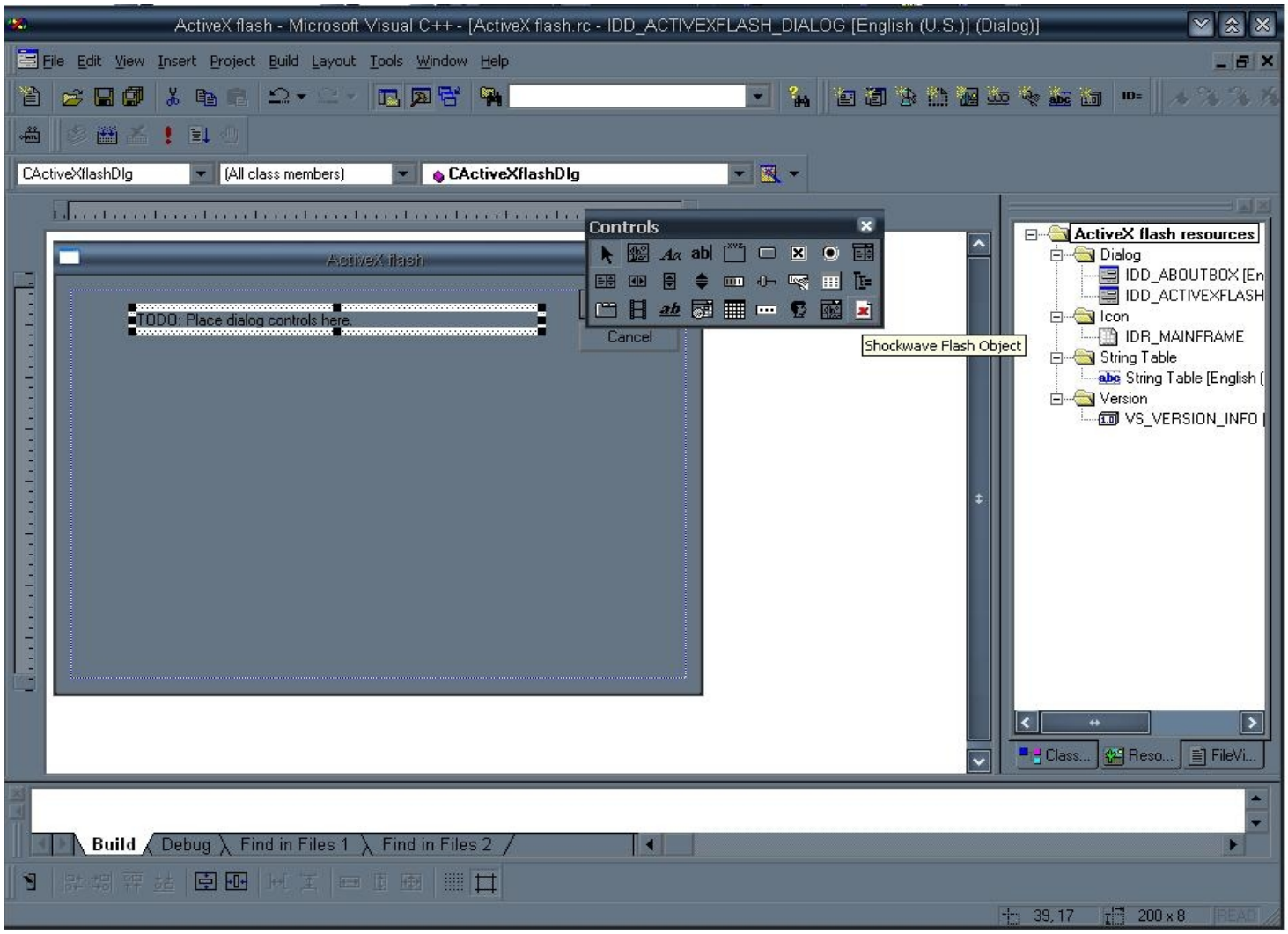
...избирате и натискате бутон Insert(Вмъкни) ...



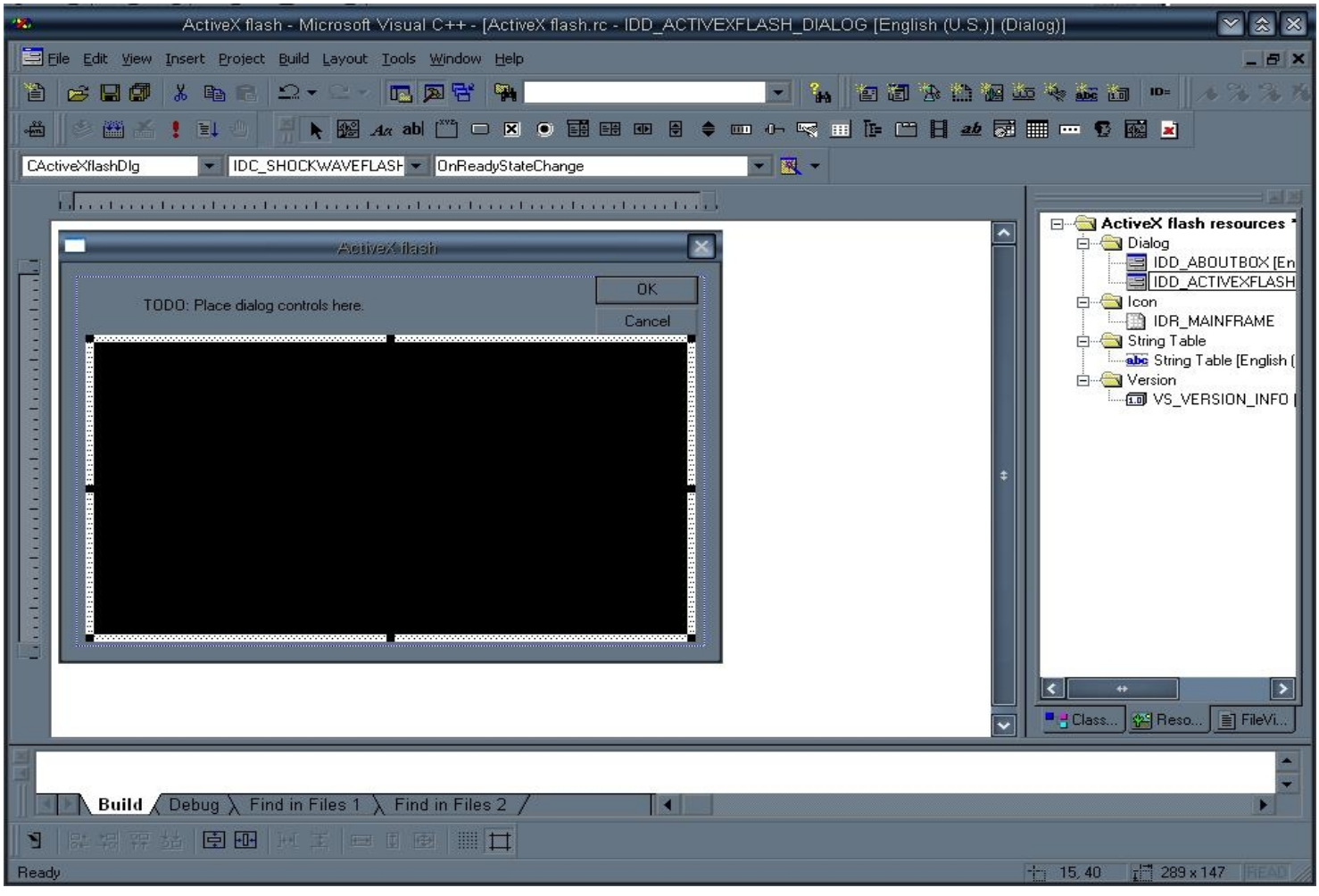
...появява се съощение от вида(Да вмъкна ли този компонент?) ... ОК



... и пак бутон ОК ...



... и както ще забележите контролът(Shockwave Flash Object) е вече в палитрата ... просто натискате в/у него и после го постявате към работното поле на приложението си...(появява се това черното поле)



Може да тествате да си поставите контрола Month Calendar ... после натиснете F5 и вижте резултата...

Може би сте забелязали,че хард диска се пълни с ActiveX контроли и то най-вече като се приемат от разни уеб сайтове...трябва и да се внимава какви точни такива контроли позволяваме да се инсталират...затова,ако сайта е непознат и не вдъхва доверие не трябва да инсталирате този контрол... принципно браузърите по подразбиране ги забраняват...показва се горе в браузъра в жълта(обикновено е така) лента контрола,който да бъде инсталиран...дори и да знаете,че този контрол няма да навреди на системата,то тези контроли са си трудни за ползване и ако нямате тяхната документация ще е трудно да се ползват...
ActiveX контролите са бинарни(двоични) обекти и са неразширяеми...не можете да добавяте свойство,събитие към даден ActiveX контрол...имат доста недостатъци относно новите концепции на MFC ,но няма да се впускам в обяснения...

Библиотека с класове MFC

MFC (Microsoft Foundation Classes) е библиотека с класове,предназначени за програмиране в средата на Windows, в която са включени основните Windows-API-функции. MFC съществено облекчава работата при обектно-ориентираното програмиране. MFC библиотеката систематизира API-функциите, структурите и типовете данни на Windows. Тя определя като обекти различните елементи, от които се състои Windows-програмата, и организира за тях класове, в които са събрани всички необходими данни и методи за тяхната обработка. Вътрешната организация на класовете е такава, че ви освобождава допълнително от цяла редица формалности - като например, предаването на дескриптори. Йерархичната й структура облекчава класификацията и търсенето при подбиране на необходимия MFC клас. Ето и кратки описания на класовете MFC, подредени по азбучен ред...

• В дясната част на първия ред от описанието на всеки от класовете е указано името на базовия му клас.

• В края на описанието на всеки клас е дадено името на заглавния файл(хедър файл), в който е дефиницията на класа. Предварителни декларации за по-голямата част от MFC класовете се съдържат в заглавните файлове <afxwin.h> и <afxext.h>

Можете да ги видите и в сайта на Майкрософт кликайки на линка:
http://msdn.microsoft.com/bg-bg/library/bk77x1wx(en-us,VS.80).aspx

Class	Header file
CAnimateCtrl	afxcmn.h
CArchive	afx.h
CArchiveException	afx.h
CArray	afxtempl.h
CAsyncMonikerFile	afxole.h
CAsyncSocket	afxsock.h
CBitmap	afxwin.h
CBitmapButton	afxext.h
CBrush	afxwin.h
CButton	afxwin.h
CByteArray	afxcoll.h
CCachedDataPathProperty	afxctl.h
CCheckListBox	afxwin.h
CClientDC	afxwin.h
CCmdTarget	afxwin.h
CCmdUI	afxwin.h

CColorDialog	afxdlgs.h
CComboBox	afxwin.h
CComboBoxEx	afxcmn.h
CCommandLineInfo	afxwin.h
CCommonDialog	afxdlgs.h
CConnectionPoint	afxdisp.h
CControlBar	afxext.h
CCriticalSection	afxmt.h
CCtrlView	afxwin.h
CDaoDatabase	afxdao.h
CDaoException	afxdao.h
CDaoFieldExchange	afxdao.h
CDaoQueryDef	afxdao.h
CDaoRecordset	afxdao.h
CDaoRecordView	afxdao.h
CDaoTableDef	afxdao.h
CDaoWorkspace	afxdao.h
CDatabase	afxdb.h
CDataExchange	afxwin.h
CDataPathProperty	afxctl.h
CDateTimeCtrl	afxdtctl.h
CDBException	afxdb.h
CDBVariant	afxdb.h
CDC	afxwin.h
CDHtmlDialog	afxdhtml.h
CDialog	afxwin.h
CDialogBar	afxext.h

CDocItem	afxole.h
CDockState	afxadv.h
CDocObjectServer	afxdocob.h
CDocObjectServerItem	afxdocob.h
CDocTemplate	afxwin.h
CDocument	afxwin.h
CDragListBox	afxcmn.h
CDumpContext	afx.h
CDWordArray	afxcoll.h
CEdit	afxwin.h
CEditView	afxext.h
CEvent	afxmt.h
CException	afx.h
CFieldExchange	afxdb.h
CFile	afx.h
CFileDialog	afxdlgs.h
CFileException	afx.h
CFileFind	afx.h
CFindReplaceDialog	afxdlgs.h
CFont	afxwin.h
CFontDialog	afxdlgs.h
CFontHolder	afxctl.h
CFormView	afxext.h
CFrameWnd	afxwin.h
CFtpConnection	afxinet.h

CFtpFileFind	afxinet.h
CGdiObject	afxwin.h
CGopherConnection	afxinet.h
CGopherFile	afxinet.h
CGopherFileFind	afxinet.h
CGopherLocator	afxinet.h
CHeaderCtrl	afxcmn.h
CHotKeyCtrl	afxcmn.h
CHtmlEditCtrl	afxhtml.h
CHtmlEditCtrlBase	afxhtml.h
CHtmlEditDoc	afxhtml.h
CHtmlEditView	afxhtml.h
CHtmlStream	afxisapi.h
CHtmlView	afxhtml.h
CHttpArgList	afxisapi.h
CHttpConnection	afxinet.h
CHttpFile	afxinet.h
CHttpFilter	afxisapi.h
CHttpFilterContext	afxisapi.h
CHttpServer	afxisapi.h
CHttpServerContext	afxisapi.h
CImageList	afxcmn.h
CInternetConnection	afxinet.h
CInternetException	afxinet.h
CInternetFile	afxinet.h
CInternetSession	afxinet.h

CIPAddressCtrl	afxcmn.h
CLinkCtrl	afxcmn.h
CList	afxtempl.h
CListBox	afxwin.h
CListCtrl	afxcmn.h
CListView	afxcview.h
CLongBinary	afxdb_.h
CMap	afxtempl.h
CMapPtrToPtr	afxcoll.h
CMapPtrToWord	afxcoll.h
CMapStringToOb	afxcoll.h
CMapStringToPtr	afxcoll.h
CMapStringToString	afxcoll.h
CMapWordToOb	afxcoll.h
CMapWordToPtr	afxcoll.h
CMDIChildWnd	afxwin.h
CMDIFrameWnd	afxwin.h
CMemFile	afx.h
CMemoryException	afx.h
CMenu	afxwin.h
CMetaFileDC	afxext.h
CMiniFrameWnd	afxwin.h
CMonikerFile	afxole.h
CMonthCalCtrl	afxdtctl.h
CMultiDocTemplate	afxwin.h
CMultiLock	afxmt.h

CMultiPageDHtmlDialog	afxdhtml.h
CMutex	afxmt.h
CNotSupportedException	afx.h
CObArray	afxcoll.h
CObject	afx.h
CObList	afxcoll.h
COccManager	afxocc.h
COleBusyDialog	afxodlgs.h
COleChangeIconDialog	afxodlgs.h
COleChangeSourceDialog	afxodlgs.h
COleClientItem	afxole.h
COleCmdUI	afxdocob.h
COleControl	afxctl.h
COleControlContainer	afxocc.h
COleControlModule	afxctl.h
COleControlSite	afxocc.h
COleConvertDialog	afxodlgs.h
COleCurrency	afxdisp.h
COleDataObject	afxole.h
COleDataSource	afxole.h
COleDBRecordView	afxoledb.h
COleDialog	afxodlgs.h
COleDispatchDriver	afxdisp.h
COleDispatchException	afxdisp.h
COleDocObjectItem	afxole.h
COleDocument	afxole.h

COleDropSource	afxole.h
COleDropTarget	afxole.h
COleException	afxdisp.h
COleInsertDialog	afxodlgs.h
COleIPFrameWnd	afxole.h
COleLinkingDoc	afxole.h
COleLinksDialog	afxodlgs.h
COleMessageFilter	afxole.h
COleObjectFactory	afxdisp.h
COlePasteSpecialDialog	afxodlgs.h
COlePropertiesDialog	afxodlgs.h
COlePropertyPage	afxctl.h
COleResizeBar	afxole.h
COleSafeArray	afxdisp.h
COleServerDoc	afxole.h
COleServerItem	afxole.h
COleStreamFile	afxole.h
COleTemplateServer	afxdisp.h
COleUpdateDialog	afxodlgs.h
COleVariant	afxdisp.h
CPageSetupDialog	afxdlgs.h
CPaintDC	afxwin.h
CPalette	afxwin.h
CPen	afxwin.h
CPictureHolder	afxctl.h
CPoint	atltypes.h

CPrintDialog	afxdlgs.h
CPrintDialogEx	afxdlgs.h
CProgressCtrl	afxcmn.h
CPropertyPage	afxdlgs.h
CPropertySheet	afxdlgs.h
CPropExchange	afxctl.h
CPtrArray	afxcoll.h
CPtrList	afxcoll.h
CReBar	afxext.h
CReBarCtrl	afxcmn.h
CRecentFileList	afxadv.h
CRecordset	afxdb.h
CRecordView	afxdb.h
CRect	atltypes.h
CRectTracker	afxext.h
CResourceException	afxwin.h
CRgn	afxwin.h
CRichEditCntrItem	afxrich.h
CRichEditCtrl	afxcmn.h
CRichEditDoc	afxrich.h
CRichEditView	afxrich.h
CScrollBar	afxwin.h
CScrollView	afxwin.h
CSemaphore	afxmt.h
CSharedFile	afxadv.h
CSingleDocTemplate	afxwin.h
CSingleLock	afxmt.h

CSize	atltypes.h
CSliderCtrl	afxcmn.h
CSocket	afxsock.h
CSocketFile	afxsock.h
CSpinButtonCtrl	afxcmn.h
CSplitterWnd	afxext.h
CStatic	afxwin.h
CStatusBar	afxext.h
CStatusBarCtrl	afxcmn.h
CStdioFile	afx.h
CStringArray	afxcoll.h
CStringList	afxcoll.h
CSyncObject	afxmt.h
CTabCtrl	afxcmn.h
CToolBar	afxext.h
CToolBarCtrl	afxcmn.h
CToolTipCtrl	afxcmn.h
CTreeCtrl	afxcmn.h
CTreeView	afxcview.h
CTypedPtrArray	afxtempl.h
CTypedPtrList	afxtempl.h
CTypedPtrMap	afxtempl.h
CUIntArray	afxcoll.h
CUserException	afxwin.h
CView	afxwin.h
CWaitCursor	afxwin.h

CWinApp	afxwin.h
CWindowDC	afxwin.h
CWinThread	afxwin.h
CWnd	afxwin.h
CWordArray	afxcoll.h

клас / това е дясната част:) с базовия му клас /

CAnimateCtrl CObject/CCmdTarget/CWnd
Този клас реализира функционалните възможности на управляващ елемент на Windows (Win95/98 u WinNT (версии 3.51 и следващи) за гледане на видеоклипове във формат AVI. Определението на класа се намира във файл <afxcmn.h> (а това е хедър файла...заглавния)

CArchive липсва базов клас
Класът CArchive съвместно с класа CFile осигурява запис на сложни структури от обекти във файлове на харддиска и последващото им възстановяване от файловете. Този процес се нарича сериализация. В действителност архивният обект представлява двоичен поток данни, свързан с един от файловете. Интерфейсът за сериализация е реализиран с помощта на презареждаемите оператори за четене (>>) и запис (<<). Той поддържа както основните типове данни, така и обектите от класове, производни от класа CObject, в който е изпълнено необходимото презареждане на операторите. Определението на класа се съдържа във файл <afx.h>
Обектите от класа CArchive могат да работят както с основните типове данни, така и с обектите от класове, производни от CObject, в които е налице метод Serialize() и се използват макроси DECLARE_SERIAL и IMPLEMENT_SERIAL

CArchiveException CObject/CException
Този клас е предназначен за обработка на изключения при работа с архиви (обекти от класа CArchive) Определението на класа се съдържа във файл <afx.h>

CArray CObject
Шаблонът на класа template <class TYPE, class ARG_TYPE>class CArray създава динамичен управляем масив. Аргументът TYPE определя типа данни на съхраняваните елементи. Аргументът ARG_TYPE е тип, предназначен за достъп до обектите, съхранявани в масива, и често се използва за указване на параметъра TYPE. Определението на класа се съдържа във файл <afxtempl.h>

CAsyncMonikerFile CObject/CFile/COIeStreamFile/CMonikerFile
Този клас се използва за въвеждане на асинхронни моникери в управляващи елементи от тип ActiveX (преди са се наричали управляващи елементи от тип OLE). Определението на класа се съдържа във файл <afxole.h>

CAsyncSocket CObject
Този клас служи за създаване на т.нар. *сокети* (гнезда)(сокет програмиране) на Windows API. Определението на класа се съдържа във файл <afxsock.h>

CBitmap CObject/CGdiObject
Този клас формира битовата карта на GDI и вклкзчва методи за нейната обработка. Определението на класа се съдържа във файл <afxwin.h>

CBitmapButton — **CObject/CCmdTarget/CWnd/CButton**

Този клас е предназначен за създаване на бутони, върху които вместо текст ще има двоично изображение. За всеки бутон може да има до четири растерни изображения, които се показват в зависимост от състоянието на бутона: нормално (не е натиснат, но е достъпен); натиснат; фокусът (т.е. текущата позиция за въвеждане) е върху него; или е недостъпен.
Определението на класа се съдържа във файл <afxext.h>

CBrush — **CObject/CGdiObject**

Този клас служи за създаване на GDI-четка, която може да бъде избрана като текуща в контекста на устройството. Определението на класа се съдържа във файл <afxwin.h>

CButton — **CObject/CCmdTarget/CWnd**

Този клас служи за създаване на управляващи елементи от тип „Бутон". Те представляват малки дъщерни прозорци, от които може да се инициират определени действия или да се направи избор. Бутонът изменя външния си вид, след като щракнете върху него с мишката. Класът поддържа създаването на командни бутони, които извършват определено действие, след като ги натиснете, флагове и радиобутони. Определението на класа се съдържа във файл <afxwin.h>

CByteArray — **CObject**

Този клас е предназначен за управление на динамични масиви от байтове. Определението на класа се съдържа във файл <afxcoll.h>

CCachedDataPathProperty — **Cobject/CFile/COIeStreamFile/CMonikerFile/CAsyncMonikerFile/CDataPathProperty**

Този клас изпълнява асинхронно предаване на свойство на OLE-управляващ елемент и кеширане с помощта на разположен в паметта файл. Този файл се съхранява в оперативната памет и се използва по време на предаването, тъй като е необходима висока скорост. Определението на класа се съдържа във файл <afxctl.h>

CCheckListBox — **CObject/CCmdTarget/CWnd/CListBox**

Този клас позволява да се създават управляващи елементи,в които са комбинирани свойствата на списъчните полета и полетата за маркировка. Става дума за списък, в който вляво от всеки от текстовите елементи е добавено поле за маркировка, с което се избира съответния елемент от списъка. Определението на класа се съдържа във файл <afxwin.h>

CClientDC — **CObject/CDC**

Този клас представя контекста на екрана за клиентската област на прозореца. Обектите от този клас се използват,например, при рисуване в отговор на събитията, постъпващи от мишката. Определението на класа се съдържа във файл <afxwin.h>

CCmdTarget — **CObject**

Базов клас за всички обекти, които могат да приемат и да изпращат съобщения. Определението на класа се съдържа във файл <afxwin.h>

CCmdUI — **базовият клас липсва**

Този клас предоставя програмен интерфейс за модификация на вида (активен, неактивен, маркиран, немаркиран) на обектите от потребителския интерфейс (например, команди от менюта или бутони от палитрите с инструменти) с помощта на макроса ONUPDATE_COMMAND_UI. Определението на класа се съдържа във файл <afxwin.h>

CColorDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog**

Стандартен диалогов прозорец за избор на цвят, който съдържа списък на цветовете, определени за видеосистемата. Основно свойство на класа е структурната променлива CHOOSECOLOR m_cc, която позволява да се настройва обекта от диалоговия клас преди неговото използване. Диалоговият прозорец се показва с метода DoModal() След затварянето на диалоговия прозорец може да се получат стойностите на параметрите, избрани от потребителя в диалоговия прозорец. Определението на класа се съдържа във файл <Bafxdlgs.h>

CComboBox **CObject/CCmd/Target/CWnd**

Този клас служи за създаване на управляващ елемент от тип „Комбиниран списък". Този елемент представлява комбинация между прозорец за списък и прозорец на поле за въвеждане. В картата на съобщенията за този прозорец следва да се добавят съответстващите функции за обработка на уведомителните съобщения, изпращани от управляващия елемент на неговия родителски прозорец. Определението на класа се съдържа във файл <afxwin.h>

Класът CComboBoxEx **CObject/CCmd/Target/CWnd/ CComboBox**

разширява функционалните възможности на управляващия елемент „Комбиниран списък", добавяики поддръжка за списъци с изображения. Ако създавате комбиниран списък с CComboBoxEx, не е необходимо да пишете собствен код за рисуване на изображението.
Определението на класа се съдържа във файл <afxcmn.h>

CCommandLineInfo **CObject**

Спомагателен клас за показване и синтактичен разбор на командния ред за стартиране на програмата. Определението на класа се съдържа във файл <afxwin.h>

CCommonDialog **CObject/CCmdTarget/CWnd/CDialog**

Базов клас за стандартни диалогови прозорци в Windows.
Определението на класа се съдържа във файл <afxdlgs.h>

CConnectionPoint **CObject/CCmdTarget**

Този клас определя специален интерфейс, който се използва за свързване с други OLE-обекти и се нарича точка за съединение. Определението на класа се съдържа във файл<afdisp.h>

CControlBar **CObject/CCmdTarget/CWnd**

Базов клас за класовете за управляващи ленти и области (CStatusBar, CToolBar, CDialogBar, CReBar u ColeResizeBar). Това са прозорци, които обикновено се долепват до лявата или дясна страна на рамката на основния прозорец и могат да съдържат дъщерни елементи. Определението на класа се съдържа във файл <afxext.h>

CCreateContext **базовият клас отсъства**

Структурата CCreateContext се използва при създаване на прозорец с рамка и представяне, свързано с документ. При създаване на прозорец, стойностите в тази структура осигуряват връзката с компонентите, съставящи документа,и представянето на данните. При презареждане на фрагмент от процеса на създаване, трябва да се използва CCreateContext
Структурата CCreateContext съдържа указатели към документа, прозореца с рамка, представянето и шаблона на документа, а също и указател към обекта от клас CRuntimeClass, който идентифицира типа на създаваното представяне. Информацията за класа за времето на изпълнение и указателя към текущия документ се използват при динамично създаване на ново представяне. Определението на класа се съдържа във файл <afxext.h>

CCriticalSection **CObject/CSyncObject**

Обектът за клас CCriticalSection представлява обект за синхронизация, който разрешава временен достъп до ресурс или сектор от програмния код само на една нишка в един и същ момент от времето. Критичните области (critical sections) от кода се определят в ситуации, в които само една нишка може да променя данни или да осъществява достъп до контролирания ресурс, например, до централния възел на свързани списъци. Критичните области се използват вместо взаимното изключване, когато при изпълнението на програмата е възникнала критична ситуация и използваните ресурси не превишават граничните норми
Определението на класа се съдържа във файл <afxmt.h>

CCtrlView **CObject/CCmdTarget/CWnd/CView**

Базов клас за CEditView, CListView, CRichEditView u CTreeView. Тези класове служат за настройка на архитектурата документ/представяне при използване на новите стандартни управляващи елементи на Windows 95/98 и WindowsNT. Определението на класа се съдържа във файл <afxwin.h>

CDaoDatabase **CObject**

Този клас предлага интерфейс и достъп до бази от данни,които позволяват да се изпълняват операции над данни. Определението на класа се съдържа във файл <afxdao.h>

CDaoException **CObject/CException**

Този клас е предназначен за обработка на изключенията, които възникват при грешки в операциите за обработка на бази от данни на основата на обектите от DAO-класовете.
Определението на класа се съдържа във файл <afxdao.h>

CDaoFieldExchange

Поддържа процедурите за обмен на полета от записи DAO (DFX), използвани от DAO класовете за бази от данни... класът се използва при обмен на данни от процедурите за запис на потребителски типове данни (обектите от дадения клас не се използват непосредствено). DFX-процедурите изпълняват обмен на данни между елементите на полетата данни на обекта CDaoRecordset и съответните полета от текущия запис на източника на данните (и в двете посоки).
Определението на класа се съдържа във файл <afxdao.h>

CDaoQueryDef **CObject**

Обектът от този клас представлява определение на заявка към база от данни
определението на класа се съдържа във файл <afxdao.h>

CDaoRecordset **CObject**

Този клас представлява множеството от записи, избрани в съответствие с някакъв критерий от източника на данни.
Определението на класа се съдържа във файл <afxdao.h>

CDaoRecordView **CObject/CCmdTarget/CWnd/ CView/CScrollView/CFormView**

Обектът от този клас показва записите от базата данни управляващите елементи и е непосредствено свързан с обекта от клас CDaoRecordset.
Определението на класа се съдържа във файл <afxdao.h>

CDaoTableDef **CObject**

Обектите от този клас служат за създаване и управление на таблици в бази от данни или присъединени към тях таблици. Определението на класа се съдържа във файл <afxdao.h>

CDaoWorkspace **CObject**

Обектите от този клас контролират именуването и защитата на сесията за работа с базата от данни.
В рамките на един такъв обект (работна среда) могат да бъдат активни едновременно няколко обекта от класа CDaoDatabase.
Определението на класа се съдържа във файл <afxdao.h>

CDatabase **CObject**

Този клас реализира интерфейс и достъп до източници на данни, с помощта на които могат да се изпълняват операции с данните. Определението на класа се съдържа във файл <afxdb.h>

CDataExchange **базовият клас отсъства**

Този клас предлага поддръжка на DDX и DDV процедури (DDX -dialog data exchange - обмен на диалогови данни и DDV – dialog data validation - подтвърждение на диалогови данни).
Той е необходим за реализация на собствени процедури за обмен на данни за потребителски управляващи елементи или типове данни.
Определението на класа се съдържа във файл <afxwin.h>

CDataPathProperty **CObject/CFile/COleStreamFile/CMonikerFile/CAsyncMonikerFile**

Гози клас поддържа свойството за асинхронно зареждане на управляващ елемент от тип ActiveX.
Определението на класа се съдържа във файл <afxctl.h>

CDateTimeCtrl **CObject/CCmdTarget/CWnd**

Обектите от класа CDateTimeCtrl енкапсулират функционалните възможности на управляващ елемент за избор на дата и час, който предлага прост потребителски интерфейс за настройка на дата и час. Интерфейсът съдържа полета, всяко от които показва част от информацията за датата и часа, съхранявана в управляващия елемент. Потребителят може да променя информацията, която се съхранява в управляващия елемент, като променя съдържанието в съответното поле. Преминаването между полетата се извършва с помощта на мишката или клавиатурата. Определението на класа се съдържа във файл <afxdtctl.h>

CDBException **CObject/CException**

Обектите от класа CDBException се създават и викат от методите на ODBC-класовете на базите от данни. Определението на класа се съдържа във файл <afxdb.h>

CDBVariant **клас отсъства**

Гози клас представлява разновидност на типа данни за ODBC класове на MFC библиотеката.
С помощта на обектите от дадения клас могат да се съхраняват данните без информации за типовете данни. Определението на класа се съдържа във файл <afxdb.h>

CDC **CObject**

Този клас е базов за класовете от контекста на устройството и съдържа общи методи за работа с контекста на устройството (например, функции за извеждане на текстова и графична информация в прозорец, както и методи за настройка на режима на показване или получаване на контекста на устройството). Определението на класа се съдържа във файл <afxwin.h>

CDHTMLDialog **CObject/CCmdTarget/CWnd/CDialog**

Този клас е предназначен за създаване на диалогови прозорци със средствата на HTML
Показваният HTML-прозорец може да се зарежда както от HTML-ресурси, така и от URL-адреса. Осигурява обмен на данни с HTML управляващи елементи и обработка на събития, свързани с HTML управляващи елементи (например, щракания с мишката). Определението на класа се съдържа във файл <afxdhtml.h>

CDialog **CObject/CCmdTarget/CWnd**

Базов клас за диалогови прозорци. Модален диалогов прозорец, който не позволява да се предаде фокуса (точката за въвеждане) на друг прозорец дотогава, докато той не бъде затворен, се създава с помощта на метода DoModal()
За създаване на немодални диалогови прозорци, които позволяват да се предава фокуса за въвеждане на други прозорци, се използва методът Create()
Определението на класа се съдържа във файл <afxwin.h>

CDialogBar **CObject/CCmdTarget/CWnd/CControlBar**

Този клас служи за създаване на немодален диалогов прозорец на базата на диалогов шаблон. Потребителят преминава между управляващите елементи на такъв диалогов прозорец с клавиша [Tab]
Определението на класа се съдържа във файл <afxext.h>

CDocItem **CObject/CCmdTarget**

Този клас е базов за обектите, които са компоненти на данните в документа. Обектите в клас CDocItem се използват за представяне на OLE-елементи както в документите на сървъра, така и на клиента. Определението на класа се съдържа във файл <afxole.h>

CDockState **CObject**

Този клас е предназначен за съхраняване и зареждане чрез сериализация на информацията за състоянието на използваните палитри (ленти) с инструменти. Определението на класа се съдържа във файл <atxadv.h>

CDocObjectServer **CObject/CCmdTarget**

Гози клас служи за организиране на нов интерфейс на сървър за управляващи елементи от тип ActiveX. CDocObjectServer-документите могат да включват обекти от класа CDocObjectServerItem. Определението на класа се съдържа във файл <afxdocob.h>

CDocObjectServerItem **CObject/CCmdTarget/ CDocItem/COIeServerItem**

Гози клас е предназначен за реализиране на специфично поведение на DocObject сървъри. Определението на класа се съдържа във файл <afxdocobj.h>

CDocTemplate **CObject/CCmdTarget**

Абстрактен базов клас за CSingleDocTemplate u CMultiDocTemplate, който съдържа базов набор от функции за шаблони за документи. Определението на класа се съдържа във файл <afxwin.h>
Шаблонът за документа определя връзките между три типа класове:
класа за документа, производен от клас CDocument;
класа за представянето, което показва данните за класа от документа;
класа за прозореца на приложението, който съдържа представянето
При организиране на интерфейса за единичен документ класът на прозореца за приложения трябва да бъде производен от класа CFrameWnd, а за многодокументен интерфейс - от класа CMDIChildWnd

CDocument **CObject/CCmdTarget**

Клас за съхраняване на данни в съответствие с архитектурата документ/представяне
Определението на класа се съдържа във файл <afxwin.h>

CDragListBox **CObject/CCmdTarget/CWnd/CListBox**

С този клас може да се създаде списък, които в допълнение към свойствата на обикновените списъци позволява да се придвижват елементите в него, като се променя тяхната последователност. Определението на класа се съдържа във файла <afxcmn.h>

CDumpContext **базовият клас отсъства**

Този клас поддържа потоково извеждане на диагностична информация в текстова форма. За извеждане на диагностичен дъмп в дебъгваната версия на проекта може да се използва предварително обявеният CDumpContext-обект с име afx-Dump. Определението на класа се съдържа във файл <afx.h>

CDWordArray **CObject**

Този клас служи за управление на динамични масиви от двойни думи. Определението на класа се съдържа във файл <afxcoll.h>
Вместо класа CDWordArray в MFC-програмите се препоръчва да се използват шаблони на контейнерни класове (например, CAггay)

CEdit **CObject/CCmdTarget/CWnd**

Този клас служи за създаване на управляващия елемент „Поле за въвеждане". Той представлява правоъгълен дъщерен прозорец, в който може да се въвежда текст, който да се редактира, избира, да се копира в буфера за обмен и да се извлича от буфера за обмен. Определението на класа се съдържа във файл <afxwin.h>

CEditView **CObject/CCmdTarget/CWnd/CView/CCtrlView**

Гози клас е предназначен за организиране на представяне с функционалността на прост многоредов редактор, който може да се използва като управляващ елемент в диалоговия прозорец или за представяне на документ. Определението на класа се съдържа във файл <afxext.h>

CEvent **CObject/CSyncObject**

Обоктът от клас CEvent се използва за създаване на обект, които уведомява приложението за настъпването на дадено събитuе. Уведомителните съобщения за събитието са необходими за информиране на съответната нишка при изпълнение на задачата (например, когато са налице нови данни за получаване или за снемане на забрана от друг поток).
Определението на класа се съдържа във файл <afxmt.h>

CException **CObject**

Базов клас за обработка на изклкзчения. Определението на класа се съдържа във файл <afx.h>

CFieldExchange **базовият клас отсъства**

Този клас служи за организиране на обмен на данни между източника на данни и неговото представяне в приложенията с помощта на механизма RFX (Record Field eXchange - обмен на полета от записи). Класът CFieldExchange е подобен на класа CDataExchange, който изпълнява обмен на данни с диалогови прозорци (DDX). Определението на класа се съдържа във файл <afxdb.h>

CFile **CObject**

Обектът от клас CFile предоставя интерфейс за изпълнение на операции на небуфериран вход/изход. Разглежданият клас (съвместно с CArchive) е необходим за поддръжка сериализацията на MFC-обекти. Той включва операции за управление на файлове, като например създаване, отваряне, затваряне и премахване на файлове, както и за определяне на режима за отваряне на файла. Определението на класа се съдържа във файл <afx.h>

CFileDialog **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog**

Този клас служи за създаване на два стандартни диалогови прозореца: за избор на файл с последващо отваряне и избор на мястото на записване и името на файла при неговото съхраняване. За настройка на параметрите на стандартните диалогови прозорци се използва променливата m_ofn от тип OPENFILENAME Диалоговият прозорец се показвана екрана с обръщение към метода DoModal( )
С методите от класа след затваряне на диалоговия прозорец може да се получи информация за избраните от потребителя настройки.
Определението на класа се съдържа във файл <afxdlgs.h>

CFileException **CObject/CException**

Обектите от класа CFileException са предназначени за обработка на изклкючения при работа с файлове. Създаването и достъпът към обектите се извършва с методите от класа CFile.
Определението на класа се съдържа във файл <afx.h>

CFileFind **CObject**

Този клас служи за търсене на файлове в локалната система. Той е базов за класове CGopherFileFind u CFtpFileFind. Определението на класа се съдържа във файл <afx.h>

CFindReplaceDialog **CObject/CCmdTarget/ CWnd/CDialog/CCommonDialog**

Този клас служи за създаване на стандартен диалогов прозорец за търсене и замяна на фрагменти в текст. За настройки на параметрите на стандартния диалогов прозорец се използа свойството на клас m_fr. Диалоговият прозорец се показва на екрана с обръщение към метода Create( ). С методите от класа след затваряне на диалоговия прозорец може да се получи информация за избраните от потребителя настройки. Определението на класа се съдържа във файл <fxdlgs.h>

CFileStatus **базовият клас отсъства**

Структурата CFileStatus се използва от методи CFile: :SetStatus( ) u CFile::GetStatus( ) за въвеждане или определяне на информация за състоянието на файла. В структурата CFileStatus се съдържат датата и часа на създаване на файла, размерът на файла (в байтове), атрибутите на файла и пълната пътека към файла.

CFont **CObject/CGdiObject**

Този клас служи за създаване и настройка на GDI-шрифт
Определението на класа се съдържа във файл <afxwin.h>

CFontDialog **CObject/CCmdTarget/CWnd/ CDialog/CCommonDialog**

Този клас е предназначен за създаване на стандартен диалогов прозорец за избор на шрифт.
За настройка на параметрите на стандартния диалогов прозорец се използва свойството на класа m_cf с тип CHOOSEFONT. Диалоговият прозорец се показва на екрана с обръщение към метода DoModal()
С методите от класа след затваряне на диалоговия прозорец може да се получи информация за избраните от потребителя настройки. Определението на класа се съдържа във файл <afxdlgs.h>

CFontHolder **базовият клас отсъства**

Този клас реализира функционалните възможности на обекта-шрифт в Windows и OLE-интерфейса IFont. Обектите от класа се използват за реализиране на стандартното свойство Font. Определението на класа се съдържа във файл <afxctl.h>

CFormView **CObject/CCmdTarget/CWnd/ CView/CScrollView**

Този клас се използва за създаване на представяне, което може да съдържа управляващи елементи, разположението на които се задава с помощта на шаблона на ресурса за диалогов прозорец. Определението на класа се съдържа във файл <afxext.h>

CFrameWnd **CObject/CCmdTarget/CWnd**

Обектът от този клас представлява главният прозорец на приложение с еднодокументен интерфейс (SDI - Single Document Interface)
Определението на класа се съдържа във файл <afxwin.h>

CFtpConnection **CObject/CInternetConnection**

Обектът от този клас реализира връзка с FTP-сървър, управление на FTP-връзката и достъп до директориите и файловете на сървъра. Обектите от класа CFtpConnection се създават с обръщение към метода GetFtpConnection() от класа ClnternetSession. Определението на класа се съдържа във файл <afxinet.h>

CFtpFileFind **CObject/CFileFind**

Спомагателен клас, предназначен за търсене на файлове на ftp-сървъри. Определението на класа се съдържа във файл <afxinet.h>

CGdiObject **CObject**

ТозиI клас е базов за GDI-обектите на Windows, като например двоичната (битовата) карта (CBitmap), областта (CRgn), четката (CBrush), перото (CPen), палитрата (CPalette) и шрифта (CFont)
Определението на класа се съдържа във файл <afxwin.h>
Създаваните от потребителя класове за обекти на GDI са производни от класове CPen или CBrush, които, на свой ред, наследяват свойствата и методите на CGdiObject

CGopherConnection **CObject/CInternetConnection**

Обектът от този клас е предназначен за връзка с Gopher-сървъри. Обектите от класа CGopherConnection се създават с обръщение към метода GetGopherConnection() от класа CInternetSession
Определението на класа се съдържа във файл <afxinet.h>

CGopherFile **CObject/CFile/CStdioFind/CInternetFile**

Този клас служи за търсене и четене на файлове на Gopher сървъри.
Определението на класа се съдържа във файл <afxinet.h>

CGopherFileFind **CObject/CFileFind**

Този клас съдържа средства за търсене на файлове на Gopher-сървъри в Интернет. Класът включва методи, които започват търсенето, определят местоположението на файл и връщат URL-адреса на файл. Определението на класа са съдържа във файл <afxinet.h>

CGopherLocator **CObject**

С този клас се създава обект-локатор, с помощта на който се изпълнява заявка за информация на Gopher-сървър. Определението на класа се съдържа във файла <afxinet.h>

CHeaderCtrl **CObject/CCmdTarget/CWhd**

Този клас служи за създаване на управляващия елемент „Заглавие". Той се използва най-често съвместно със списъците, които представляват обекти от класа CListCtrl
Заглавието обикновено е разделено на части, които се наричат заглавни елементи. Всяка такава част всъщност представлява заглавието на съответната колона. Разделителите между елементите на заглавието може да се придвижват, като се изменя ширината на колоните. Този управляващ елемент се поддържа в операционите системи Win95 u WinNT(версии 3.5 и следващи).
Определението на класа се съдържа във файл <afxcmn.h>

CHotKeyCtrl **CObject/CCmdTarget/CWnd**

Този клас се използва за създаване на управляващ елемент, който може да се зададе клавишна комбинация и да се провери нейната допустимост. Този управляващ елемент се поддържа в операционите системи Win95 u WinNT и следващите...
Определението на класа се съдържа във файл <afxcmn.h>

CHtmlEditCtrl **CObject/CCmdTarget/CWnd**

Този клас поддържа функционирането на ActiveX-елемента на Web-браузъра. Създаваният елемент се поставя автоматично в режим на редактиране.
Определението на класа се съдържа във файл <afxhtml.h>

CHtmlEditCtrlBase **отсъства**

Този клас представлява редактируем HTML-компонент.
Определението на класа се съдържа във файл <afxhtml.h>

CHtmlEditDoc **CObject/CCmdTarget/CDocument**

Този клас поддържа средствата за редактиране на Web-браузъра в контекста на архитектурата документ/представяне на MFC. Определението на класа се съдържа във файл <afxhtml.h>

HtmIEditView **CObject/CCmdTarget/CWnd/ CView/CScrollView/CFormView**

Този клас поддържа средствата за редактиране на Web браузъра в контекста на архитектурата документ/представяне на MFC. Определението на класа се съдържа във файл <afxhtml.h>

CHtmlView **CObject/CCmdTarget/CWnd/ CView/CScrollView/CFormView**

Класът CHtmlView реализира функционалните възможности на управляващия елемент WebBrowser в контекста на MFC-архитектурата документ/представяне. Управляващият елемент WebBrowser представлява прозорец, в който потребителят може да преглежда Web сайтове и папки, при това както на локалния, така и на отдалечен компютър от мрежата. Управляващият елемент WebBrowser поддържа работа с хипервръзки(hyperlinks=html links), търсене no URL-agpec и съхранява списък на посещаваните адреси. Определението на класа се съдържа във файл <afxhtml.h>

CHttpArg **базовият клас липсва**

Структурата CHttpArg съхранява информацията за една двойка параметър-стойност, изпращана към HTTP-сървъра. Определението на класа се съдържа във файл <afxisapi.h>

CHttpArgList **базовият клас липсва**

Обектът CHttpArgList съхранява колекцията от структури CHttpArg и осигурява необходимите за номерирането й средства. Определението на класа се съдържа във файл <afxisapi.h>

CHttpConnection **CObject/CInternetConnection**

Обектът от този клас реализира връзка с HTTP-сървър. Обектите от клас CHttpConnection се създават с обръщение към метода GetHttpConnection( ) на класа CInternetSession
Определението на класа се съдържа във файл <afxinet.h>

CHttpFile **CObject/CFile/CStdioFile/CInternetFile**

Този клас е предназначен за търсене и четене на файлове на HTTP-сървъри. Определението на класа се съдържа във файл <afxinet.h>

CHttpFilter

Този клас е предназначен за създаване и управление на филтри, които изпращат избраните HTTP заявки към ISAPI сървър. Определението на класа се съдържа във файл <afxisapi.h>

CHttpFilterContext **базовият клас липсва**

Този клас управлява съдържанието на HTTP филтрите и е предназначен за обработка на паралелните заявки от обекта от клас CHttpFilter. Определението на класа се съдържа във файл <afxisapi.h>

CHttpServer **базовият клас липсва**

Обектът от този клас разширява функционалните възможности на ISAPI сървъра. Той изпълнява обработка на заявките от клиента. Включително и с CGI (Common Gateway Interface)
Определението на класа се съдържа във файл <afxisapi.h>

CHttpServerContext **базовият клас отсъства**

Този клас управлява съдържанието на разширения ISAPI сървър. Той е предназначен за обработка на конкуриращи се заявки към обекти от класа ChttpServer
Определението на класа се съдържа във файл <afxisapi.h>

CImageList **CObject**

Този клас служи за създаване на управляващ елемент от тип спиисък с изображения
Той се използва за работа с големи набори от двоични масиви или икони, достъпът до които се осъществява не по идентификатор, а по номер. Класът поддържа функции за рисуване, създаване и премахване на целия списък, добавяне и унищожаване на съдържащите се в него елементи, тяхното обединение и местене.
Определението на класа се съдържа във файл <afxcmn.h>

CInternetConnection **CObject**

Клас, който се използва съвместно с класа CinternetSession за свързване към Интернет сървър.
Определението на класа се съдържа във файл <afxinet.h>

CInternetException **CObject/CException**

Обектите от класа CinternetException представят изключителни ситуации, свързани с операциите в Интернет. Определението на класа се съдържа във файл <afxinet.h>

CInternetFile **CObject/CFile/CStdioFile**

Абстрактен базов клас за класове CHttpFile u CGopherFile.
Определението на класа се съдържа във файл <afxinet.h>

CInternetSession **CObject**

Клас, който се използва съвместно с класа CInternetConnection за установяване на връзка с Интернет сървър. Определението на класа се съдържа във файл <afxinet.h>

CIPAddressCtrl **CObject/CCmdTarget/CWnd**

Управляващият елемент „IP-Адрес" е аналогичен на управляващия елемент „поле за въвеждане". Той е предназначен за въвеждане и редактиране на числов адрес във формат IP (Internet Protocol – протокол за Интернет)
Класът CIPAddressCtrl реализира функционалните възможности на управляващия елемент „IP-Адрес". Даденият управляващ елемент (CIPAddressCtrl) е достъпен само в програмите, управлявани от Microsoft Internet Explorer версия 4.0 и следващи.
Определението на класа се съдържа във файл <afxcmn.h>

CLinkCtrl **CObject/CCmdTarget/CWnd**

Този клас предлага удобни средства за вграждане на хипертекстови връзки в прозореца.
Определението на класа се съдържа във файл <afxcmn.h>

CList **CObject**

Шаблонът за клас template <class TYPE, class ARG_TYPE> class CList създава двойносвързан списък. Аргументът TYPE определя типа данни на съхраняваните елементи. Аргументът ARG_TYPE е тип, който се използва за достъп до обектите от списък, и се прилага за указване на параметъра TYPE.
Определението на класа се съдържа във файл <afxtempl.h>

CListBox **CObject/CCmdTarget/CWnd**

С този клас се създава управляващият елемент „Списък". Той представлява правоъгълник, в който се съдържа последователност от елементи (например, текстови), които потребителят може да преглежда и избира. Списъкът може да изпраща на родителския прозорец няколко вида известяващи съобщения. За всички тях са предвидени макроси, с които могат да се добавят функции за обработка в картата за съобщения на прозореца. Определението на класа се съдържа във файл <afxwin.h>

CListCtrl **CObject/CCmdTarget/CWnd**

Този клас е предназначен за създаване на управляващия елемент „Преглеждане на списък".
Той представя подреден списък с елементи, всеки от които може да има структура, състояща се от няколко полета. С всеки запис могат да се свържат до три икони с различен размер, външният вид на които може да носи допълнителна информация за елемента от списъка.
Определението на класа се съдържа във файл <afxcmn.h>

CListView **CObject/CCmdTarget/CWnd/CView/CCtrlView**

Този клас се използва за създаване на представяне, което съдържа управляващ елемент „Списък".
Той енкапсулира функционалните възможности на дадения елемент в контекста на архитектурата MFC-документ/представяне. Определението на класа се съдържа във файл <afxcview.h>

CLongBinary **CObject**

Класът CLongBinary е предназначен за обработка в база от данни на обекти за данни от тип BLOBS (Binary Large Objects - големи двоични обекти). Обектът от клас CLongBinary управлява участъка от паметта, отделен за даден BLOBS-обект, и определя неговия размер.
Определението на класа се съдържа във файл <afxdb.h>

CMap **CObjecf**

Шаблонът за клас template <class KEY, class ARG_KEY, class VALUE, class ARG_VALUE> class CMap генерира асоциативен списък, който се състои от двойки ключ/стойност. Предимствата на асоциативните списъци са в бързото съхраняване и търсене на стойности по ключ.
Аргументът KEY определя типа данни на ключа, а аргументът ARG_KEY - типа, използван за достъп до аргументите KEY (обикновено съвпадат с типа KEY или с указателя към тип KEY)
Аргументът VALUE определя типа данни на стойността, а аргументът ARG_VALUE - типа, използван за достъп до аргументите VALUE (обикновено съвпада с VALUE или с указател към VALUE)
Определението на класа се съдържа във файл <afxtempl.h>

CMapPtrToPtr **CObject**

Клас за асоциативен списък, който използва указател от тип void като ключ за търсене на данни, типът на които е зададен с друг void-указател. Определението на класа се съдържа във файл <afxcoll.h>
Вместо класа CMapPtrToPtr в MFC-програмите трябва да се използва шаблон за контейнерен клас (например клас CMap)

CMapPtrToWord **CObject**

Клас за асоциативен списък, който използва указател от тип void като ключ за търсене на данни от тип DWORD. Определението на класа се съдържа във файл <afxcoll.h>

CMapStringToOb **CObject**

Клас за асоциативен списък, който използва обект от тип CString като ключ за търсене на указатели към обекти от класа CObject или производните от него. Определението на класа се съдържа във файл <afxcoll.h>

CMapStringToPtr **CObject**

Клас за асоциативен списък, който използва обект от тип CString като ключ за търсене на указател от тип void. Определението на класа се съдържа във файл <afxcoll.h>
В MFC-програмите вместо класа CMapStringToPtr трябва да се използва шаблон за контейнерен клас (например класът CMap)

CMapStringToString **CObject**

Клас за асоциативен списък, който използва обект от тип CSiring като ключ за търсене на друг обект от класа CString. Определението на класа се съдържа във файл <afxcoll.h>

CMapWordToOb **CObject**

Клас за асоциативен списък, който използва обект от тип WORD за търсене на указателя на обект от класа CObject или производен от него. Определението на класа се съдържа във файл <afxcoll.h>

CMapWordToPtr **CObject**

Клас за асоциативен списък, който използва обект от тип WORD за търсене на указател от тип void. Определението на класа се съдържа във файл <afxcoll.h>
Вместо класа CMapWordToPtr в MFC-програмите трябва да се използва шаблон за контейнерен клас (класа CMap)

CMDIChildWnd **CObject/CCmdTarget/CWnd/CFrameWnd**

Класът CMDIChildWnd се използва за генериране на дъщерен прозорец на MDI-прилоЖение (MDI - Multiple Document Interface - многодокументен интерфейс). Този дъщерен прозорец не може да бъде изтеглен извън рамките на работната област на родителския прозорец на MDI-приложението, няма собствени менюта, но има собствена рамка и може да използва менютата на родителския прозорец. Цветът на лентата за заглавие на дъщерния прозорец показва неговото състояние (активно, неактивно). Определението на класа се съдържа във файл <afxwin.h>

CMDIFrameWnd **CObject/CCmdTarget/CWnd/CFrame Wnd**

Класът CMDIFrameWnd се използва за създаване на главния прозорец на MDI-приложение. Определението на класа се съдържа във файл <afxwin.h>

CMemFile **CObject/CFile**

Този клас реализира поддръжка на разположени (включени) в паметта файлове. Тези файлове са аналогични на файловете, разположени на харддиска, само че се съхраняват в оперативната памет. Разположените в паметта файлове се използват за временно съхранение на данни, за прехвърляне на необработени байтове или за преобразувани в последователна форма обекти между независимите процеси. Определението на класа се съдържа във файл <afx.h>

CMemoryException **CObject/CException**

Този клас е предназначен за обработка на изключенията, които възникват при работа с паметта, когато липсва необходимия обем свободна оперативна памет (Out-of-Memory-изключения). Обектите от класа CMemoryException се създават и викат при работата на оператора new. Определението на класа се съдържа във файл <afx.h>

CMemoryState **липсва базов клас**

Този клас се използва за откриване на т.нар. „изтичане" на памет в програмата. „Изтичането" на памет възниква, когатo паметта за даден обект е била отделена в „купчината", без да бъде освободена при излизане от процедурата.“Изтичането" на голям обем памет може да доведе до възникване на грешки при резервиране на паметта. Определението на класа се съдържа във файл <afx.h>

CMenu **CObject**

Този клас реализира интерфейс за достъп до менюто на приложението. Обектът от този клас позволява да се манипулира динамично менюто по време на изпълнение на приложението. Определението на класа се съдържа във файл <afxwin.h>

CMetaFileDC **CObject/CDC**

Обект от този клас е т.нар. контекст на устройството за Windows-метафайлове. Метафайловете съхраняват графичните обекти във вид на последователности от GDI команди, които са небходими за тяхното изобразяване. Определението на класа се съдържа във файл <afxext.h>

CMiniFrameWnd **CObject/CCmdTarget/CWnd/CFrameWnd**

Този клас се използва за създаване на прозорец с рамка и два пъти по-малка от нормалната височина, който се използва обикновено за плаващи палитри с инструменти. Обектите от класа се държат аналогично на обикновените прозорци с рамки, но тяхното меню и бутон не изпълняват операциите на минимизация/максимизация на прозореца. Те могат да бъдат премахвани с просто щракане с мишката върху бутон от системното меню
Определението на класа се съдържа във файл <afxwin.h>

CMonikerFile **CObject/CFile/COIeStreamFile**

Обектът от клас CMonikerFile представя потока от данни (IStreams), който се идентифицира от обекта IMoniker. Определението на класа се съдържа във файл <afxole.h>

CMonthCalCtrl **CObject/CCmdTarget/CWnd**

Обектът от клас CMonthCalCtrl енкапсулира функционалните възможности на управляващия елемент „Календар". Той предлага на потребителя удобен интерфейс за избор на дата с възможности за помесечно прелистване на календара напред и назад, избор на текущата дата с щракане върху текста „Today" („Днес"), избор на месец и година от изскачащо меню
Определението на класа се съдържа във файл <afxdtctl.h>

CMultiDocTemplate **CObject/CCmdTarget/CDocTemplate**

Този клас определя шаблон на документ за MDI-интерфейс,който позволява да се отворят няколко документа в различни дъщерни прозорци на главния прозорец на приложението.
За да се обработят едновременно няколко формата на документа в MDI-приложението, за всеки формат трябва да се подготови съответен шаблон за документа. Определението на класа се съдържа във файл <afxwin.h>

CMultiLock **б. клас отсъства**

Този клас служи за управление на обектите от класове CCriticalSection,CEvent, CMutex u CSemaphore, които поддържат синхронизация в многонишкова среда. Определението на класа се съдържа във файл <afxmt.h>

CMultiPageDHtmlDialog **CObject/CCmdTarget/CWnd/ CDialog/CDHtmlDialog**

Многостраничен прозорец, който показва няколко последователни HTML-страници и обработва събитията за всяка от страниците. Определението на класа се съдържа във файл <afxdhtml.h>

CMutex **CObject/CSyncObject**

Този клас създава обект за взаимно изключване, който предоставя на отделна нишка правото на изключителен достъп до ресурс или сектор от програмния код. Взаимните изключения се използват, когато само една от нишките може да променя данните или да се обръща към контролирания ресурс, например, към централния възел в свързан списък (CCriticalSection, CSemaphore). Определението на класа се съдържа във файл <afxmt.h>

CNotSupportedException **CObject/CException**

Този клас е предвиден за обработка на изклкоченията, които възникват при опит да се използват неподдържани свойства. Определението на класа се съдържа във файл <afx.h>

CObArray **CObject**

Клас, който управлява CObject-указател в динамичен масив
Определението на класа се съдържа във файл <afxcoll.h>
Вместо класа CObArray в MFC-програмите трябва да се използва шаблон за контейнерен клас (класа CArray)

CObject **базовият клас отсъства**

Този клас е на практика базов за всички останали класове.
Той съдържа методи за сериализация на данните и подготовка на информация за обекта по време на изпълнение на програмата. Определението на класа се съдържа във файл <afx.h>

CObList **CObject**

Този клас управлява CObject-указателя в двойносвързани списъци. Определението на класа се съдържа във файл <afxcoll.h>
Вместо класа CObList в MFC-програмите е необходимо да се използва шаблон за контейнерен клас (например класа Clist)

COccManager **базовият клас отсъства**

Този клас се използва за управление на различни потребителски управляващи сайтове. Определението на класа се съдържа във файл <afxocc.h>

COleBusyDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog/COleDialog**

Този клас създава прозорец за съобщения, които ви информират, че виканото приложение-сървър е или заето, или не отговаря на заявката от клиента. Такъв прозорец за съобщения обикновено се генерира от обекти от класа COleMessageFilter
Определението на класа се съдържа във файл <afxodlgs.h>

COleChangeIconDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog/COleDialog**

Обектът от този клас представлява диалогов прозорец, от който може да се избере друга икона при вграждане или свързването на OLE-обект. Определението на класа се съдържа във файл <afxodlgs.h>

COleChangeSourceDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog/COleDialog**

Обектът от този клас представлява диалогов прозорец, от който може да се избере източника за свързан OLE-обект
Определението на класа се съдържа във файл <afxodlgs.h>

COleClientItem — **CObject/CCmdTarget/CDocItem**

Този клас е базов за елементите от страната на клиента при свързване или вграждане на OLE-елементи. Определението на класа се съдържа във файл <afxole.h>

COleCmdUl — **CCmdUI**

Този клас съдържа методи, които позволяват да се модифицира състоянието на обектите на потребителския интерфейс, свързани с IOleCommandTarget-управляеми св-ва на приложението. Определението на класа се съдържа във файл <afxdocobj.h>

COleControl — **CObject/CCmdTarget/CWnd**

Този клас е базов при разработка на управляващи елементи от тип OLE. Той наследява от класа CWnd всички функционални възможности на обекта прозорец в Windows. Класът притежава и ред допълнителни функционални възможности,специфични за OLE, като например иницииране на събитие, достъп до методите и свойствата за поддръжка. Определението на класа се съдържа във файл <afxctl.h>

COleControlContainer — **CObject/CCmdTarget**

Изпълнява ролята на контейнер за управляващ елемент от тип ActiveX. Определението на класа се съдържа във файл <afxocc.h>

COleControlModule — **CObject/CCmdTarget/CWinThread/CWinApp**

Изпълнява ролята на класа CWinApp в проект за управляващ елемент от тип ActiveX. Този клас предоставя методите, необходими за инициализация на модула за управляващия елемент. Определението на класа се съдържа във файл <afxctl.h>

COleControlSite — **CObject/CCmdTarget**

Този клас предоставя средства за поддръжка на потребителски интерфейси за управление (от страната на клиента). Определението на класа се съдържа във файл <afxocc.h>

OleConvertDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog/COleDialog**

Обектът от този клас представлява диалогов прозорец, с помощта на който OLE-обектите могат да бъдат преобразувани от един тип в друг. Определението на класа се съдържа във файл <afxodlgs.h>

COleCurrency — **базовият клас отсъства**

Съдържа типа данни CURRENCY, който се използва при автоматизацията. Определението на класа,се съдържа във файл <afxdisp.h>

COleDataObject — **отсъства**

Този клас се използва за възстановяване на данни с различен формат от буфера за обмен при свързване или вграждане на OLE-елементи. Източникът на данните е обект от клас ColeDataSource

Всеки път, когато приемащата приложна програма получава данни чрез свързване или нарежда да се изпълни операция за копиране в документа от буфера за обмен, се създава обект от клас COleDataObject С този клас може да се определи предварително дали данните в желания формат съществуват. Той позволява да се получи информация за достъпните формати за данни. По този начин може да се провери достъпен ли е желаният формат преди да се копират данните.
Определението на класа се съдържа във файл <afxole.h>

COleDataSource **CObject/CCmdTarget**
Обектът от този клас изпълнява ролята на област за кеширане. Приложната програма записва данни в нея при изпълнение на операция за предаване на данни през буфера за обмен или прехвърляне.
Определението на класа се съдържа във файл <afxole.h>

COleDateTime **базовият клас отсъства**
Съдържа типа данни DATE, който се използва при автоматизация. Определението на класа се съдържа във файл <ATLComTime.h>

COleDateTimeSpan **базовият клас отсъства**
Изчислява разликата между две стойности от тип COleDateTime. Определението на класа се съдържа във файл <ATLComTime.h>

COleDBRecordView **CObject/CCmdTarget/CWnd/ CView/CScrollView/CFormView**
Обектът от клас COleDBRecordView представлява представяне, непосредствено свързано с обект от класа CRowSet , което показва записите от база данни в управляващите елементи. Представянето се генерира на базата на шаблона за диалоговия прозорец. Обектът от класа COleDBRecordView използва обмена на данни за диалоговия прозорец (DDX) и функционалните възможности за преход между записите, вградени в CRowSet. Те позволяват да се автоматизира преминаването между записите в управляващите елементи на диалоговия прозорец и между избраните полета.
Обектът от клас COleDBRecordView вклкзчва и зададена по подразбиране реализация на методите за преход към първия, следващия, предишния или последния запис, а също и интерфейс за обновяване на текущия визуализиран запис. Определението на класа се съдържа във файл <afxoledb.h>

COleDialog **CObject/CCmdTarget/CWnd/ CDialog/CCommonDialog**
Този клас се използва за стандартното оформление на всички OLE-диалогови прозорци. Определението на класа се съдържа във файл <afxodlgs.h>

COleDispatchDriver **базовият клас отсъства**
Този клас предлага поддръжка на автоматизация на приложения-клиенти и предоставя достъп до методите и свойствата на обекта. Методите от класа се използват за присъединяване, разделяне, създаване и изпълнение на връзки IDispatch, а също и за обръщение към метода IDispatch::Invoke Определението на класа се съдържа във файл <afxdisp.h>

COleDispatchException **CObject/CException**
Този клас е предназначен за обработка на изклкзченията в интерфейса OLE IDispatch, който е основна част на OLE-автоматизацията. Определението на класа се съдържа във файл <afxdisp.h>

COleDocObjectItem **CObject/CCmdTarget/ CDocItem/COleClientItem**
Обектът от клас COleDocObjectItem съдържа активен документ. В MFC обработката на активните документи се извършва подобно на обработката на вградени документи в режим <<на място>> Съществуват обаче и следните особености:
• класовете, производни от COleDocument, както и преди съхраняват списъка с текущите вградени обекти, но тези обекти могат да бъдат производни и от класа COleDocObjectItem;
• активният документ заема цялата работна област на активния прозорец за преглеждане;
• активният контейнер на документа управлява изцяло менюто Help, което съдържа точките от менюто както за контейнера на активния документ, така и за сървъра
Определението на класа се съдържа във файл <afxole.h>

COleDocument **CObject/CCmdTarget/CDocument**

COleDocument е базов клас за OLE-документи, който поддържа визуално редактиране. Той е производен от класа CDocument, което позволява на OLE-приложенията да използват архитектурата документ/представяне, реализирана в библиотеката Microsoft Foundation Class. Определението на класа се съдържа във файл <Bafxole.h>

COleDropSource **CObject/CCmdTarget**

Този клас съдържа функции за прехвърляне на данни в OLE приложенията. Класът позволява да се определи кога започва и кога завършва операцията на прехвърляне и реализира обратна връзка по време на операцията. Определението на класа се съдържа във файл <afxole.h>

COleDropTarget **CObject/CCmdTarget**

Този клас реализира механизъм за връзка между прозорците и OLE-библиотеката. Създаването на обект от този клас позволява на прозореца да приема данни с механизма на OLE прехвърлянето. Определението на класа се съдържа във файл <afxole.h>

COleException **CObject/CException**

Обектите от този клас обработват изклключенията, възникващи при работа с OLE-обекти. Викането на обект от класа обикновено се изпълнява с функция AfxThrowOleException( )
Класът COleException се използва както при OLE-сървъри, така и при контейнери. Определението на класа се съдържа във файл <afxdisp.h>

COleInsertDialog **CObject/CCmdTarget/CWnd/ CDialog/CCommonDialog/COleDialog**

Обектът от този клас представлява диалогов прозорец, от коймо потребителят може да избере OLE-обект за вграждане или свързване. Определението на класа се съдържа във файл <afxodlgs.h>

COleIPFrameWnd **CObject/CCmdTarget/ CWnd/CFrameWnd**

Този клас е базов за прозорците за редактиране „на място". Класът създава и настройва управляващите области в прозореца на документа за приложението-контейнер. Той обработва също и информационните съобщения, които се генерират от вградения обект от класа COleResizeBar при промяна на размерите на прозореца за редактиране на място отпотребителя. Определението на класа се съдържа във <Bafxole.h>

COleLinkingDoc **CObject/CCmdTarget/CDocument/ COleDocument**

Този клас съдържа инфраструктура за свързване на OLE-обекти. За поддръжка на свързани и вградени обекти контейнерните приложения трябва да създават своите класове за документи като производни от класа COleLinkingDoc, не от класа COleDocument. Определението на класа се съдържа във файл <afxole.h>

COleLinksDialog **CObject/CCmdTarget/CWnd/ CDialog/CCommonDialog/COleDialog**

Обектът от този клас представлява диалогов прозорец, от който могат да се въвеждат параметрите на свързан с документа обект. Определението на класа се съдържа във файл <afxodlgs.h>

COleMessageFilter **CObject/CCmdTarget**

Този клас регулира взаимодействието между OLE-приложенията. При приложенията-сървъри той се използва за присвояване на специален признак за заетост. Това дава възможност входящите заявки от други приложения-контейнери да бъдат отменени или повторени по-късно. С негова помощ се определя и действието, което ще предприеме викащата програма, в случай че виканото приложение е заето. Определението на класа се съдържа във файл <afxole.h>

COleObjectFactory **CObject/CCmdTarget**

Този клас се използва за организиране на фабрики от OLE-класове за създаване на такива OLE-обекти, като сървъри, обекти за автоматизация и документи. Класът COleObjectFactory извършва и регистрация на обекти. Определението на класа се съдържа във файл <Bafxdisp.h>

COlePasteSpecialDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog/COleDialog**

Обектът от този клас представлява стандартен диалогов прозорец Paste Special (Специално възстановяване). Определението на класа се съдържа във файл <afxodlgs.h>

COlePropertiesDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog/COleDialog**

Обектът от този клас представлява стандартен диалогов прозорец за свойства на обект, в който се показват и могат да бъдат променяни свойствата на OLE-обекти. Диалоговият прозорец съдържа информация за файла, в който се съхранява обекта, параметрите за мащабиране и свързване. Определението на класа се съдържа във файл <afxodlgs.h>

COlePropertyPage — **CObject/CCmdTarget/CWnd/CDialog**

Обектът от този клас представлява вложена (припокриваща се) страница със свойствата на управляващ елемент от тип ActiveX. Определението на класа се съдържа във файл <afxctl.h>

COleResizeBar — **CObject/CCmdTarget/CWnd/CControlBar**

Обектът от този клас представлява особен тип управляваща област, която служи за промяна размерите на OLE-елементи. Обектите от класа COleResizeBar се показват във вид на защрихована рамка с маркери за изменение на размера. Обектите COleResizeBar обикновено са вградени елементи за обекта прозорец с рамка, производен от класа COlelPFrameWnd.
Определението на класа се съдържа във файла <afxole.h>

COleSafeArray — **базовият клас отсъства**

Клас за работа с масиви от всякакъв тип и размерност. Определението на класа се съдържа във файл <afxdisp.h>

COleServerDoc — **CObject/CCmdTarget/CDocument COleDocument/COleLinkingDoc**

Класът е базов за класовете за документи на OLE-сървъра.
Документите на сървъра могат да съдържат обекти от клас COleServerItem, които реализират интерфейса на сървъра с вградените или свързани елементи. При стартиране на приложение-сървър от приложение-контейнер при редактиране на вграден елемент последният се зарежда като собствен документ на сървъра; обектът COleServerDoc съдържа само един обект от клас COleServerItem, който вклкзчва целия документ. При стартиране на приложение-сървър от приложението-контепнер за редактиране на свързан елемент, съществуващият документ се зарежда от диска; при това част от съдържанието на документа се осветява, за да се отдели свързаният елемент. Определението на класа се съдържа във файл <afxole.h>

COleServerItem — **CObject/CCmdTarget/CDocItem**

Този клас обезпечава интерфейса на сървъра с OLE-елементи. Всеки свързан елемент може да представя част или целия документ на сървъра. Вградените елементи винаги представят целия документ на сървъра. Класът COleServerItem определя няколко презареждаеми метода, които се викат от OLE-библиотеките (DLL), обикновено в отговор на заявките от приложение-контейнер. Тези методи позволяват на приложението-контейнер да управлява елемента индиректно по най-различни начини, например, като показва или възстановява данни с различни формати. Определението на класа се съдържа във файл <afxole.h>

COleStreamFile — **CObject/CFile**

Обектът от клас COleStreamFile представя поток данни (IStream) в съставния документ като част от структурирана 0LE2 област за съхранение. Определението на класа се съдържа във файл <Bafxole.h>

COleTemplateServer — **CObject/CCmdTarget/COleObjectFactory**

Този клас се използва при OLE-сървъри за визуално редактиране, сървъри за автоматизация и контейнери за свързване (приложни програми, които поддържат както свързване, така и вграждане) Класът COleTemplateServer използва обекта CDocTemplate за управление на документите на сървъри.

Определението на класа се съдържа във файл <afxdisp.h>

COleUpdateDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog/COleDialog/COleLinksDialog**

Обектът от клас COleUpdateDialog представлява специален диалогов прозорец Edit Link. Той се използва в документа само когато е необходимо да се модифицират съществуващи свързани или вградени обекти. Определението на класа се съдържа във файл <afxodlgs.h>

COleVariant — **базовият клас отсъства**

Съдържа типа данни VARIANT, който се използва при автоматизация. Определението на класа се съдържа във файл <afxdisp.h>

CPageSetupDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog**

Този клас служи за създаване на стандартен диалогов прозорец за настройка на параметрите на страниците. Свойството на класа m_psd от типа PAGESETUPDLG се използва за настройка на параметрите на стандартния диалогов прозорец. Диалоговият прозорец се показва на екрана с обръщение към метода DoModal()
След като диалоговият прозорец бъде затворен, с методите от този клас може да получите информация за избраните от потребителя настройки. Определението на класа се съдържа във файл <afxdlgs.h>

CPaintDC — **CObject/CDC**

Обекmume от този клас се използват само в програмата за обработка на съобщения WM_PAINT OnPaint() , генерирани в рузултат на обръщението към функция UpdateWindow() или RedrawWindow( )
Определението на класа се съдържа във файл <afxwin.h>
При създаване на обект от клас CPaintDC в конструктора се вика функция BeginPaint( ), а при неговото разрушаване в деструктора - функция EndPaint( ). Те изпълняват необходимите операции за подготовка и завършване на процесса на графично извеждане.

CPalette — **CObject/CGdiObject**

Обектът от този клас съдържа палитрата с цветове на GDI, за да се използва като интерфейс м/у приложението и устройството за избеждане, способно да възпроизвежда цветове. Този интерфейс позволява на приложението да оползотвори напълно възможностите на устройството за показване на цветовете и същевременно минимизира конфликтите с представянето на цветовете от другите приложения. Класът включва методи за обработка на палитрата.
Определението на класа се съдържа във файл <afxwin.h>

CPen — **CObject/CGdiObject**

Този клас служи за създаване на GDI-перо, което може да бъде избрано за текущо в контекста на устройството. Определението на класа се съдържа във файл <afxwin.h>

CPictureHolder — **базовият клас отсъства**

Този клас съчетава функционалните възможности на обектите-изображения на Windows и интерфейса OLE IPicture. С обектите от класа се реализира свойството Picture в управляващите елементи от тип OLE. Определението на класа се съдържа във файл <afxctl.h>

CPoint

Този клас съдържа Windows-структурата POINT, в която се съхранява двойката координати x и y Класът CPoint се използва за предаване на координатите на указателя на мишката от методите за обработка на съобщенията от мишката. Определението на класа се съдържа във файл <atltypes.h>

CPrintDialog — **CObject/CCmdTarget/CWnd/CDialog/CCommonDialog**

Този клас се използва за създаване на стандартни диалогови прозорци, от които може да се прави настройка на параметрите за печат и принтера. Свойството от класа m_pd и тип PRINTDLG се използва за настройка параметрите на стандартния диалогов прозорец. Диалоговият прозорец се показва на екрана с обръщение към метода DoModal()
След затваряне на диалоговия прозорец с методите от класа, може да се получи информация за избраните от потребителя настройки. Определението на класа се съдържа във файл <afxdlgs.h>

CPrintDialogEx **CObject/CCmdTarget/CWind/CDialog/ CCommonDialog**

Този клас енкапсулира средствата, достъпни в прозореца за свойства Print в Windows 2000. Определението на класа се съдържа във файл <afxdlgs.h>

CPrintInfo **базовият клас отсъства**

Обектът от този клас се създава при избор на команда Print (Печатане) или Print Preview (Предварителен преглед на печатане). Той съхранява информацията за заданието за печат или предварителен преглед. След изпълнението на командата обектът от клас CPrintInfo се разрушава. Обектът от клас CPrintInfo съдържа информация както за всички задания за печат (вкл. и броя печатани страници), така и за текущото състояние на печата (за текущо печатаната страница). Част от информацията като например стойностите, зададени от потребителя в диалоговия прозорец Print (Печат), се съхраняват в съответния обект от клас CPrintDialog
Класът CPrintInfo се използва по време на обработката на заданието за печат за обмен на информация между класа на главния прозорец на приложението и класа на представянето. Определението на класа се съдържа във файл <afxext.h>

CProgressCtrl **CObject/CCmdTarget/CWnd**

С този клас се създава управляващият елемент „Индикатор", който се използва в приложенията, когато трябва да се следи процеса на изпълнение на някоя дълга операция.
Определението на класа се съдържа във файл <afxcmn.h>

CPropertyPage **CObject/CCmdTarget/CWnd/CDialog**

Този клас служи за създаване на отделни припокриващи се страници в прозореца за свойства. Всяка припокриваща се страница от диалоговия прозорец представлява обект от клас, производен от CPropertyPage. Определението на класа се съдържа във файл <afxdlgs.h>

CPropertyPageEx **CObject/CCmdTarget/CWnd/CDialog CPropertyPage**

В MFC 7.0 класът CPropertyPageEx отсъства, защото се съдържа функционално в своя родителски клас CpropertyPage

CPropertySheet **CObject/CCmdTarget/CWnd**

Този клас се използва за създаване структурата на прозорец за свойства с набор от припокриващи се страници. Всеки прозорец за свойства се състои от един обект от клас CPropertySheet и един или няколко обекта от класа CPropertyPage.
Определението на класа се съдържа във файл <afxdlgs.h>

CPropertySheetEx **CObject/CCmdTarget/CWnd/ CPropertySheet**

В MFC *7.0* класът CPropertySheetEx липсва, тъй като се съдържа функционално в своя родителски клас CpropertySheet::CPropertySheetEx

CPropExchange **базовият клас отсъства**

Този клас се използва при управляващи елементи от тип OLE за обмен на информация за състоянието на управлението, обикновено представяна със свойствата на класа, между управлението и средата. Определението на класа се съдържа във файл <afxctl.h>

CPtrArray **CObject**

Обектите от този клас представляват масиви с указатели от тип void. Определението на класа се съдържа във файл <afxcoll.h>
Вместо класа CPtrArray в MFC-програмите трябва да се използва шаблон за контейнерен клас (CArray)

CPtrList **CObject**

Обектите от този клас представляват двойносвързани списъци с указатели от тип void. Определението на класа се съдържа във файл <afxcoll.h>

CReBar **CObject/CCmdTarget/CWnd/CControlBar**

Обектите от класа CReBar представляват специални области от управляващи елементи (rebar), които могат да съхраняват информация за разположението и състоянието на управляващите елементи. Всяка такава област може да включва ред дъщерни прозорци, които като правило са също управляващи елементи (прозорец за въвеждане, ленти с инструменти, списъци и т. н.). Размерът на областта с управляващи елементи може да се променя както от приложната програма (автоматично), така и от потребителя (с изтегляне). Определението на класа се съдържа във файл <afxext.h>

CReBarCtrl **CObject/CCmdTarget/CWnd**

Класът CReBarCtrl енкапсулира функционалните възможности на област с управляващи елементи (rebar), която представлява контейнер на дъщерен прозорец. Приложната програма, в която се намира областта с управляващи елементи, свързва дъщерния прозорец на дадения управляващ елемент с отрязък от областта с управляващи елементи. Дъщерният прозорец (като правило) е управляващ елемент. Областта с управляващи елементи съдържа един или повече отрязъка. Всеки отрязък може да се състои от различни компоненти, като например заглавие, растерно изображение, текст и дъщерен прозорец. Отрязъкът може да съдържа само по един от изброените компоненти. Определението на класа се съдържа във файл <afxcmn.h>

CRecentFileList **базовият клас отсъства**

Този клас се използва за управление на списъка с последните обработени файлове (MRU - Most Recently Used). Определението на класа се съдържа във файл <afxadv.h>

CRecordset **CObject**

Обектът от този клас съдържа множество от записи, избрани от източника на данни (текущия набор). Наборите от класа CRecordset могат да бъдат както динамични (Dynaset), така и статични (Snapshot). Статичните набори отразяват състоянието на базата данни в момента на избирането. Динамичните се обновяват и синхронизират с базата данни, отразявайки и всички изменения, внесени от другите потребители. Определението на класа се съдържа във файл <afxdb.h>

CRecordView **CObject/CCmdTarget/CWnd/CView/ CScrollView/CFormView**

Този клас служи за организиране на представяне-форма, което чрез механизма DDX е непосредствено свързано с обект от тип база данни. По този начин може да се извършва обмен на информация между текущия набор и управляващите елементи на представянето. Определението на класа се съдържа във файл <afxdb.h>

CRect **базовият клас отсъства**

Този клас съдържа Windows-структурата RECT, предназначена за съхраняване на координатите на правоъгълна област. Методите от класа CRect позволяват да се обработват обектите от класа CRect и структурата RECT. Определението на класа се съдържа във файл <atltypes.h>

CRectTracker

Обектът от този клас показва рамка, която позволява да се променят размерите или да се местят отделни обекти (като правило, OLE-обекти). Определението на класа се съдържа във файл <afxext.h>

CResourceException **CObject/CException**

Обектите от този клас се използват за обработка на изклкюченията, които възникват при зареждане на Windows ресурси (например, когато ресурсите не са намерени или не могат да бъдат създадени) Определението на класа се съдържа във файл <afxwin.h>

CRgn **CObject/CGdiObject**

Обектите от този клас представляват GDI-области с елиптична, многоъгълна или произволна форма вътре в прозореца. Този род области се използват от различните функции за отсичане от клас CDC. Определението на класа се съдържа във файл <afxwin.h>

CRichEditCntrItem **CObject/CCrndTarget/CDocItem/ COleClientItem**

Управляващият елемент „Разширено поле за въвеждане" (Rich Edit) представлява прозорец, в който потребителят може да въвежда и редактира текст, като прилага форматиране на символи и абзаци, а

също и вградени OLE-обекти...съвместно с класове CRichEditDoc u CRichEditView класът CRichEditCntrItem реализира функциите на управляващия елемент <<Разширено поле за въвеждане>> в контекста на архитектурата документ/представяне на библиотеката MFC.
Класът CRichEditView поддържа работа с форматиран текст, а класът CRichEditDoc - със списъка с OLE-елементи- клиенти в представянето. Класът CRichEditCntrItem осигурява достъпа на приложения-контейнери към OLE-обекти-клиенти. Определението на класа се съдържа във файл <afxrich.h>

CRichEditCtrl **CObject/CCmdTarget/CWnd**
Стандартен управляващ елемент на Windows „Разширено поле за въвеждане", който представлява многоредово поле за въвеждане, в което потребителят може да въвежда и редактира текст, като прилага форматиране на символи и абзаци, както и да използва OLE-обекти.Определението на класа се съдържа във файл <afxcmn.h>

CRichEditDoc **CObject/CCmdTarget/CDocument/ COleDocument/COleLinkingDoc/COleServerDoc**
Управляващият елемент „Разширено поле за въвеждане" (Rich-Edit) представлява прозорец, в който потребителят може да въвежда и редактира текст, като прилага форматиране на символи и абзаци, както и да използва вградени OLE-обекти. Съвместно с класове CRichEditCntrItem u CRichEditView класът CRichEditDoc реализира функциите на управляващия елемент <<Разширено поле за въвеждане>> в контекста на архитектурата документ/представяне на библиотека MFC.
Класът CRichEditView поддържа работа с форматиран текст, а класът CRichEditDoc - със списъка от OLE-елементи-клиенти в представянето. Класът CRichEditCntrItem осигурява достъп на приложения контейнери към OLE-обекти-клиенти. Определението на класа се съдържа във файл <afxrich.h>

CRichEditView **CObject/CCmdTarget/CWnd/ CView/CCtrlView**
Съвместно с класове CRichEditCntrItem и CRichEditDoc класът CRichEditView реализира функционалността на управляващия елемент „Разширено поле за въвеждане" в контекста на архитектурата документ/представяне на библиотеката MFC. Класът CRichEditView поддържа работа с форматиран текст, а класът CRichEditDoc - със списъка на OLE-елементите-клиенти в представянето. Класът CRichEdit- CntrItem осигурява достъпа на приложението-контейнер към OLE-обектите-клиенти. Определението на класа се съдържа във файл <afxrich.h>

CRuntimeClass **базовият клас отсъства**
Всеки клас, производен на клас CObject, е свързан със структурата CRuntimeClass, предназначена за съхраняване на информация за обекта или неговия базов клас по време на изпълнението на програмата. Възможността за определяне на класа за обекта за времето на изпълнение е необходима при реализирането на допълнителен контрол за съответствие на типовете за параметрите на функцията или при създаване на специализиран код за отделен клас. Определението на класа се съдържа във файл <afx.h>

CScrollBar **CObject/CCmdTarget/CWnd**
Този клас служи за създаване на управляващия елемент „лента за придвижване". За разлика от останалите управляващи елементи, лентите за придвижване изпращат известия за няколко различни събития и смени на състоянието в рамките на едно съобщение. Определението на класа се съдържа във файл <afxwin.h>

CScrollView **CObject/CCmdTarget/CWnd/CView**
Този клас се използва за създаване на представяне с възможност за автоматично придвижване на показваните данни. Определението на класа се съдържа във файл <afxwin.h>

CSemaphore **CObject/CSyncObject**
Обектите от клас CSemaphore представляват синхронизиращи обекти, които разрешават достъп до определени ресурси на ограничен брой нишки от един или няколко процеса. Обектите управляват и броя нишки, на които е разрешен достъп (CCriticalSection, CMutex)
Определението на класа се съдържа във файл <afxmt.h>

CSharedFile **CObject/CFile/CMemFile**
Обектът от този клас представлява разположен в паметта файл. Той допуска съвместен достъп и е

предназначен за обмен на данни между независимите Win32-процеси. Определението на класа се съдържа във файл <afxadv.h>

CSingleDocTemplate **CObject/CCmdTarget/ CDocTemplate**

Този клас определя шаблон за документ със SDI-интерфейс, който позволява да се отваря само един документ в главния прозорец на приложението. Определението на класа се съдържа във файл <afxwin.h>

CSingleLock **базовият клас отсъства**

Този клас организира механизъм за контрол на достъпа, който се използва от класове CCriticalSection, CEvent, CMutox и CSemaphore за управление на ресурси в многонишкова среда. Определението на класа се съдържа във файл <afxmt.h>

CSize

Този клас съдържа Windows-структурата SIZE и е предназначен за съхраняване на размерите или относителните координати на позицията. Определението на класа се съдържа във файл <atltypes.h>

CSliderCtrl **CObject/CCmdTarget/CWnd**

Този клас служи за създаване на управляващия елемент „плъзгам", който се използва за избор на дискретни стойности от зададен диапазон. Определението на класа се съдържа във файл <afxcmn.h>

CSocket **CObject/CAsyncSocket**

Този клас е предназначен за създаване на т.нар. Windows сокети и се използва за синхронизация на операции. При предаването на данни обектите от дадения клас използват обекти от класове CSocketFile u CArchive. Определението на класа се съдържа във файл <afxsock.h>

CSocketFile **CObject/CFile**

Този клас предоставя интерфейс към класа CFile за Windows сокемите. Определението на класа се съдържа във файл <afxsock.h>

CSpinButtonCtrl **CObject/CCmdTarget/CWnd**

Този клас е предназначен за създаване на управляващ елемент,състоящ се от двойка бутони със стрелки, с които се намалява или увеличава съответната числова стойност със зададена стъпка. Определението на класа се съдържа във файл <afxcmn.h>

CSplitterWnd **CObject/CCmdTarget/CWnd**

Обектът от клас CSplitterWnd представлява прозорец, разделен на няколко области, размерите на който могат да бъдат променяни от потребителя. Областите представляват обекти от класове, производни от CView. Определението на класа се съдържа във файл <afxext.h>

CStatic **CObject/CCmdTarget/CWnd**

Този клас се използва за създаване на статични управляващи елементи. Той позволява да се извеждат на екрана неизискващи редактиране текстови низове, икони, двоични масиви и разширени метафайлове. Определението на класа се съдържа във файл <afxwin.h>

CStatusBar **CObject/CCmdTarget/CWnd/CControlBar**

Обектите от този клас са т.нар. ленти за състояние. Те съдържат поле за извеждане на текст и индикатори, в които се извежда информация за състоянието. Определението на класа се съдържа във файл <afxext.h>

CStatusBarCtrl **CObject/CCmdTarget/CWnd**

Този клас реализира функционалността на стандартния управляващ елемент в Windows „Лента за състояние". Той представлява хоризонтален прозорец, най-често разположен в долния край на родителския прозорец, който показва информация за състоянието. Определението на класа се съдържа във файл <afxcmn.h>

CStdioFile **CObject/CFile**

Обектите от този клас се използват за буферирано четене и запис на файлове, файловете могат да бъдат отваряни в текстов или двоичен режим. Обектите от класа CStdioFile функционират аналогично на файловете, отваряни с помощта на функция fopenf()
Определението на класа се съдържа във файл <afx.h>

CStringArray **CObject**

Обектите от този клас представляват динамични масиви от низове (обекти от класа CString)
Определението на класа се съдържа във файл <afxcoll.h>
Вместо класа CStringArray в MFC-програмите се препоръчва да се използва шаблона за контейнерен клас (CArray)

CStringList **CObject**

Обектите от този клас представляват двойно свързани списъци, елементите на които са обекти от класа CString
Определението на класа се съдържа във файл <afxcoll.h>
Вместо класа CStringList в MFC-програмите се препоръчва да се използва шаблон за контейнерен клас (Clist)

CSyncObject **CObject**

Това е абстрактен базов клас, който организира общите функции наWin32-класовете за синхронизация (CCriticalSection, CEvent, CMutex u Csemaphore)
Определението на класа се съдържа във файл <afxmt.h>

CTabCtrl **CObject/CCmdTarget/CWnd**

Този клас се използва за създаване на управляващите елементи, с които се реализират припокриващи се страници.
Определението на класа се съдържа във файл <afxcmn.h>

CTime

Обектите от класа CTime позволяват да се съхраняват дати и часове и използващите ги функции.
Определението на класа се съдържа във файл <atltime.h>

CTimeSpan **базовият клас отсъства**

Определя разликата между две стойности, съхранявани в обекти от класа CTime. Определението на класа се съдържа ввфайл <atltime.h>

CToolBar **CObject/CCmdTarget/CWnd/CControlBar**

Обектите от класа CToolBar са т.нар. ленти с инструменти (т.е. области с набор бутони в тях). Бутоните могат да бъдат представени по традиционния начин с растерни изображения върху тях, или във вид на полета за маркировка или радиобутони. Определението на класа се съдържа във файл <afxext.h>

CToolBarCtrl **CObject/CCmdTarget/CWnd**

Обектът от този клас реализира функционалността на стандартния управляващ елемент на Windows „Лента с инструменти". Определението на класа се съдържа във файл <afxcmn.h>

CToolTipCtrl **CObject/CCmdTarget/CWnd**

Този клас служи за създаване на управляващия елемент „Подсказващ прозорец". Той представлява неголям плаващ прозорец с кратко текстово описание на обекта, върху който в дадения момент е курсорът на мишката. Определението на класа се съдържа във файл <afxcmn.h>

CTreeCtrl **CObject/CCmdTarget/CWnd**

С този клас се създава управляващият елемент за преглеждане на дървовидна структура. Той се използва за показване на информация, която има йерархична структура. Всеки запис от структурата се състои от текстов низ и растерно изображение и може да има един или повече подзаписи
Определението на класа се съдържа във файл <Bafxcmn.h>

CTreeView **CObject/CCmdTarget/CWnd/CView/CCtrlView**
Обектът от този клас представлява представяне, за показване на йерархично подредени икони и символни низове. Определението на класа се съдържа във файл <afxcview.h>

CTypedPtrArray **определя се от потребителя**
Шаблонът за класа template <class BASISCLASS, class TYPE> class CTypedPtrArray представлява шаблон на клас за масиви от указатели (CObArray или CPtrArray)
Определението на класа се съдържа във файла <afxtempl.h>

CTypedPtrList **определя се от потребителя**
Шаблонът за клас template <class BASISCLASS, class TYPE> class CTypedPtrList представлява шаблон на клас за списъци от указатели (CObList или CPtrList). Определението на класа се съдържа във файл <afxtempl.h>

CTypedPtrMap **определя се от потребителя**
Шаблонът за класа template <class BASISCLASS, class TYPE> class CTypedPtrMap представлява шаблон на клас за асоциативни списъци от указатели (CMapPtrToPtr, CMapPtrToWord CMapWordToPtr или CMapStringToPtr). Определението на класа се съдържа във файл <afxtempl.h>

CUIntArray **CObject**
Този клас поддържа обработката на динамични масиви от целочислени елементи без знак.
Определението на класа се съдържа във файл <afxcoll.h>
Вместо класа CUIntArray в MFC-програмите се препоръчва да се използва шаблон за контейнерен клас (CArray)

CUserException **CObject/CException**
Обектите от този клас изпълняват обработка на специфични за приложението изклкючения, които се викат при спиране на операции, започнати от потребителя. Обръщението към обекта от класа CUserException обикновено се изпълнява след обръщение към глобалната функции AfxMessageBox(), която уведомява потребителя, че операцията не е била изпълнена. Ако искате да инициирате самостоятелно обект от класа CUserException, съобщете на потребителя за възникването на изключителна ситуация, а след това извикайте глобалната функция AfxThrowUserException()
Определението на класа се съдържа във файл <Bafxwin.h>

CView **CObject/CCmdTarget/CWnd**
Този клас е базов за класовете от представяния, определяни от потребителя.
Класът за представяне е прозорец, предназначен за показване на текст и графика, данните за който се съхраняват в свързания с представянето обект от класа за документа.
Класът за представяне служи като посредник между документа и потребителя. Той визуализира данните от документа на екрана (или на принтера) и интерпретира въвежданата от потребителя информация като операции над документа. Определението на класа се съдържа във файл <afxwin.h>

CWaitCursor **базовият клас отсъства**
Обектът от този клас се използва за показване и премахване на указателя на мишката, оформен като пясъчен часовник. Определението на класа се съдържа във файл <afxwin.h>

CWinApp **CObject/CCmdTarget/CWinThread**
Този клас е базов за създаване на обекти-приложения на Windows.
Обектите от класа CWinApp съдържат методи за инициализация и изпълнение на приложението.
До указателя към обект от клас CWinApp може да получите достъп с помощта на глобалната функция AfxGetApp( )
Определението на класа се съдържа във файл <afxwin.h>

CWindowDC **CObject/CDC**
Обектът от този клас представлява контекстът на цялата област, принадлежаща към прозореца, включително и неговата рамка u управляващи елементи (за разлика от класа CClientDC)
Определението на класа се съдържа във файл <afxwin.h>

CWinThread **CObject/CCmdTarget**

Класът CWinThread е базов за класа CWinApp. Той служи за организиране на основната нишка на приложението. Допълнително обектите от класа CWinThread може да се използват и за създаване в рамките на приложението на няколко нишки. Определението на класа се съдържа във файл <afxwin.h>

CWnd **CObject/CCmdTarget**

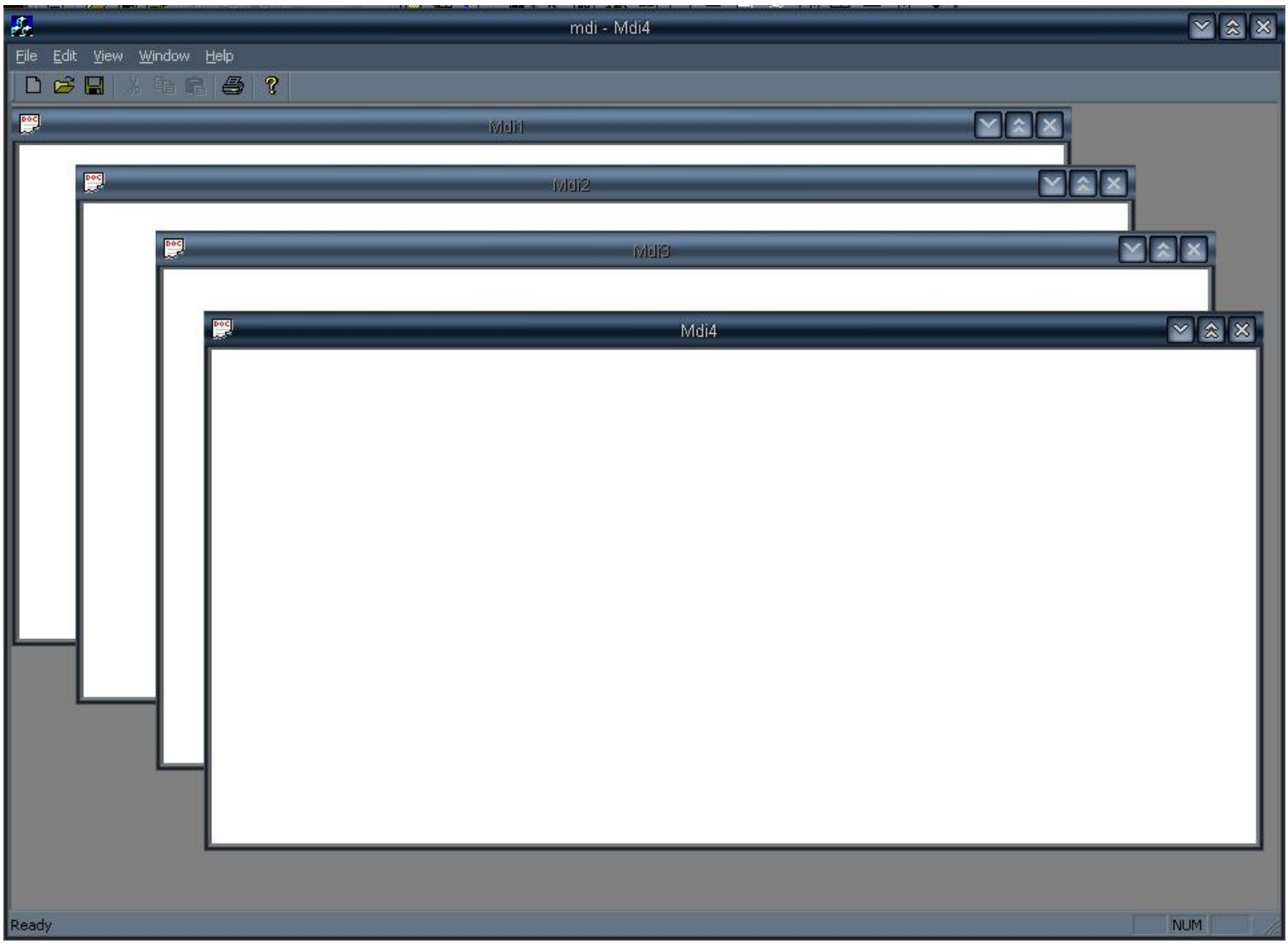
Класът CWnd е базов за всички класове за прозорци (прозорец с рамка, прозорец за представяне, диалогов прозорец, прозорец на управляващия елемент)
Обектът от клас CWnd не е прозорец на Windows. Обектът от клас CWnd се създава с конструктора за клас CWnd и се премахва от деструктора CWnd
Прозорецът в Windows представлява вътрешна структура от данни за клас CWnd, създавана с метода Create( ) и разрушавана от виртуалния деструктор за клас CWnd. Класът CWnd и картата за обработка на съобщения от MFC използват API-функцията WndProc, за да разпределят постъпващите съобщения към съответните OnMessage-методи на класа CWnd или производните му класове. Определението на класа се съдържа във файл <afxwin.h>

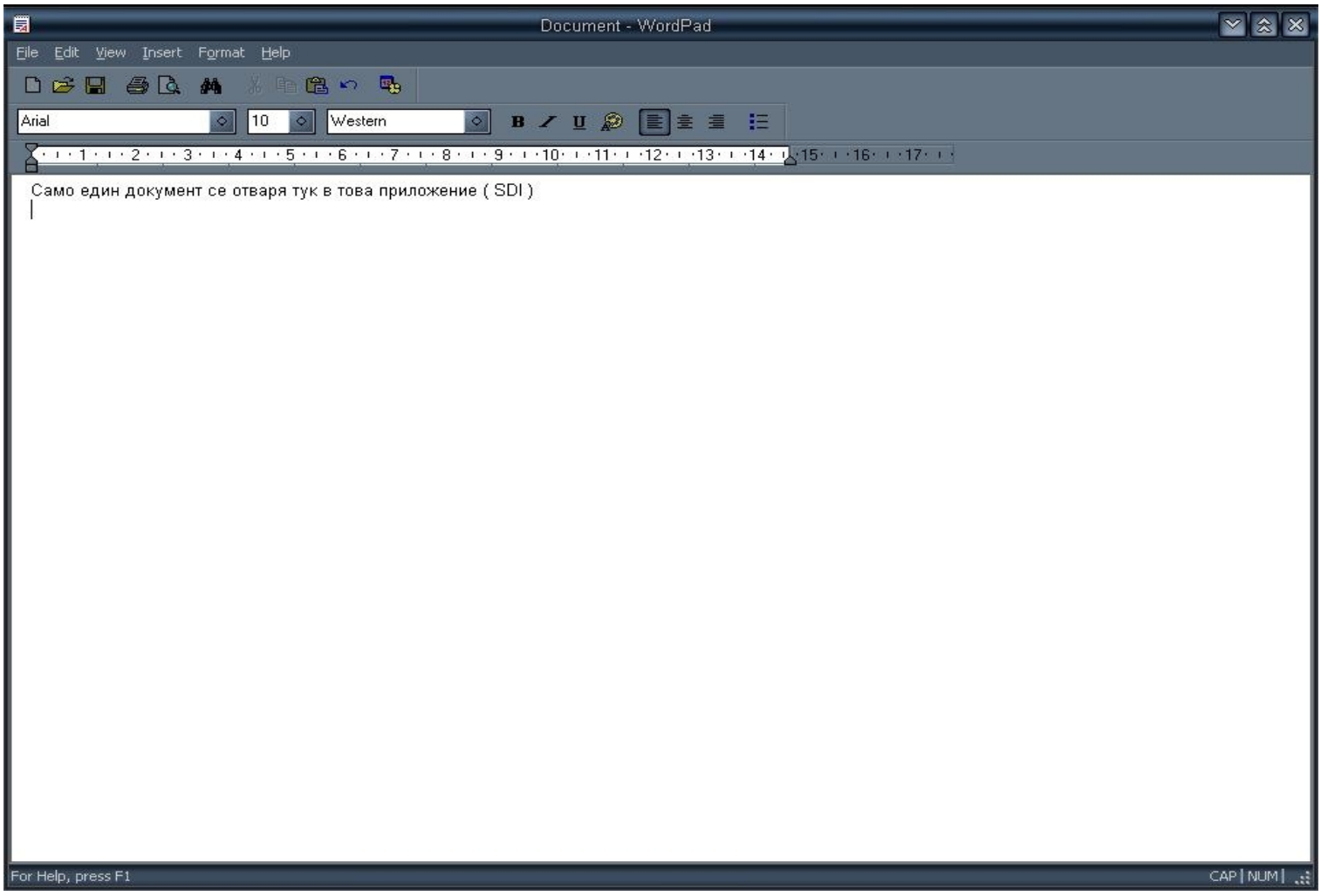
CWordArray **CObject**

Този клас се използва за управление на динамични масиви с елементи от тип WORD. Определението на класа се съдържа във файл <afxcoll.h>

SDI и MDI приложения:

Освен диалогово базирани приложения съветника AppWizard на MFC може да генерира фреймуорк и за два типа документ/изглед приложения: приложения с еднодокументен интерфейс (SDI) и приложения с многодокументен интерфейс (MDI)
SDI приложенията позволяват само един единствен прозорец рамка за документ. Приложения като Paint WordPad са типични примери за SDI приложения.
MDI приложенията позволяват множество документни прозорци в една инстанция на приложението. В едно MDI приложение потребителя може да отвори множество дъщерни MDI прозорци в главния прозорец. Тези дъщерни прозорци също се явяват прозорци рамки и всеки си съдържа отделен документ
Пример за MDI приложения са Microsoft Word,Exel ...

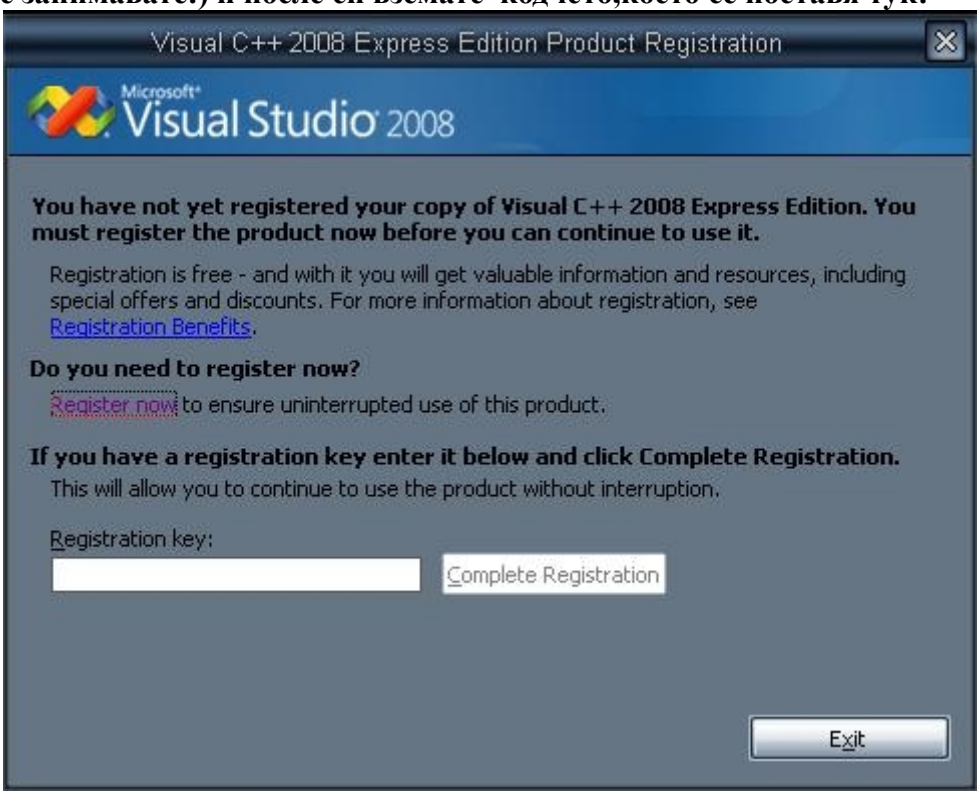

mdi - Mdi4
File Edit View Window Help
Mdi1
Mdi2
Mdi3
Mdi4
Ready
NUM


Document - WordPad
File Edit View Insert Format Help
Arial
10
Western
Само един документ се отваря тук в това приложение (SDI)
For Help, press F1
CAP NUM

Нека сега видим как е в програмната среда Visual C++ 2008 Express Edition

...прескачам другите(предишните) среди:) като: Visual C++ 2002 (Visual v7.0); ... Visual C++ 2005 (v8.0) и пристъпваме към Visual C++ 2008 (v 9.0 с кодово название „Orcas“ ... това е едно островче:) всъщност цялото Visual Studio 2008 (което си включва компилатори на доста програмни езици) беше под това кодово наименование:)
Добре е да знаете, че повечето програмисти работят с Visual Studio Profesional 2008 и Expression Blend Те разбира се не са безплатни:) Затова за нас бедните програмисти са създадени продуктите: Express Editions , които са си доста осакатени(орязани) , но пак де не е зле:) линка е за 2005,ако искате по-старите среди... а ето и връзки към 2008 :
http://www.microsoft.com/express/product/default.aspx
http://www.microsoft.com/express/download/default.aspx

ще свалите малък файл, при стартирането на който инсталац. файлове ще бъдат свалени от Интернет Express Editions изискват да имате регистрация... e-mail в hotmail.com
става много бързо регването като се отмятат там в сайта им разни отговори относно това с кои програмни езици се занимавате:) и после си вземате кодчето,което се поставя тук:

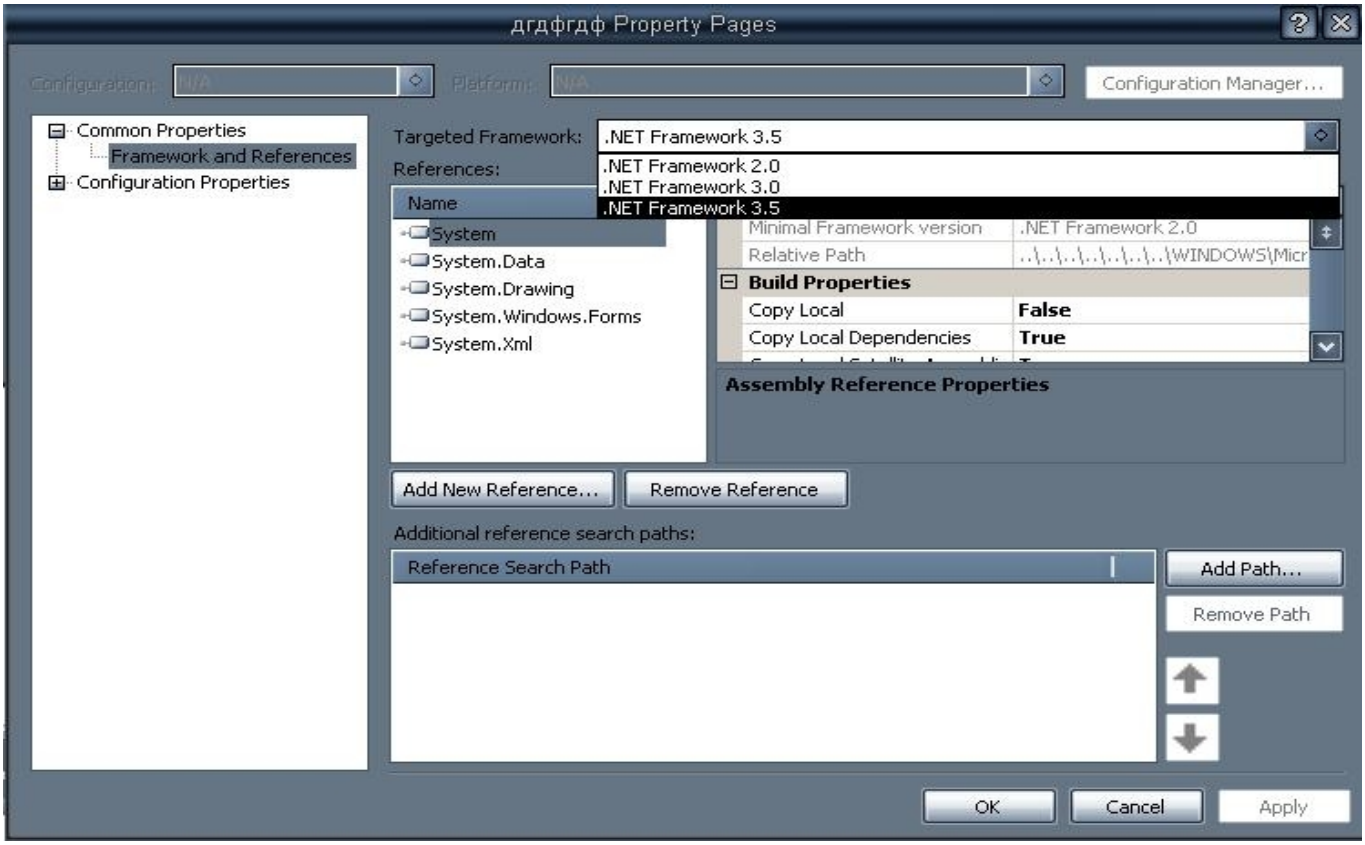


разбира се отначало може да не се регвате ... известно време(30дни) ще може да си ползвате продукта...
http://www.microsoft.com/express/samples/

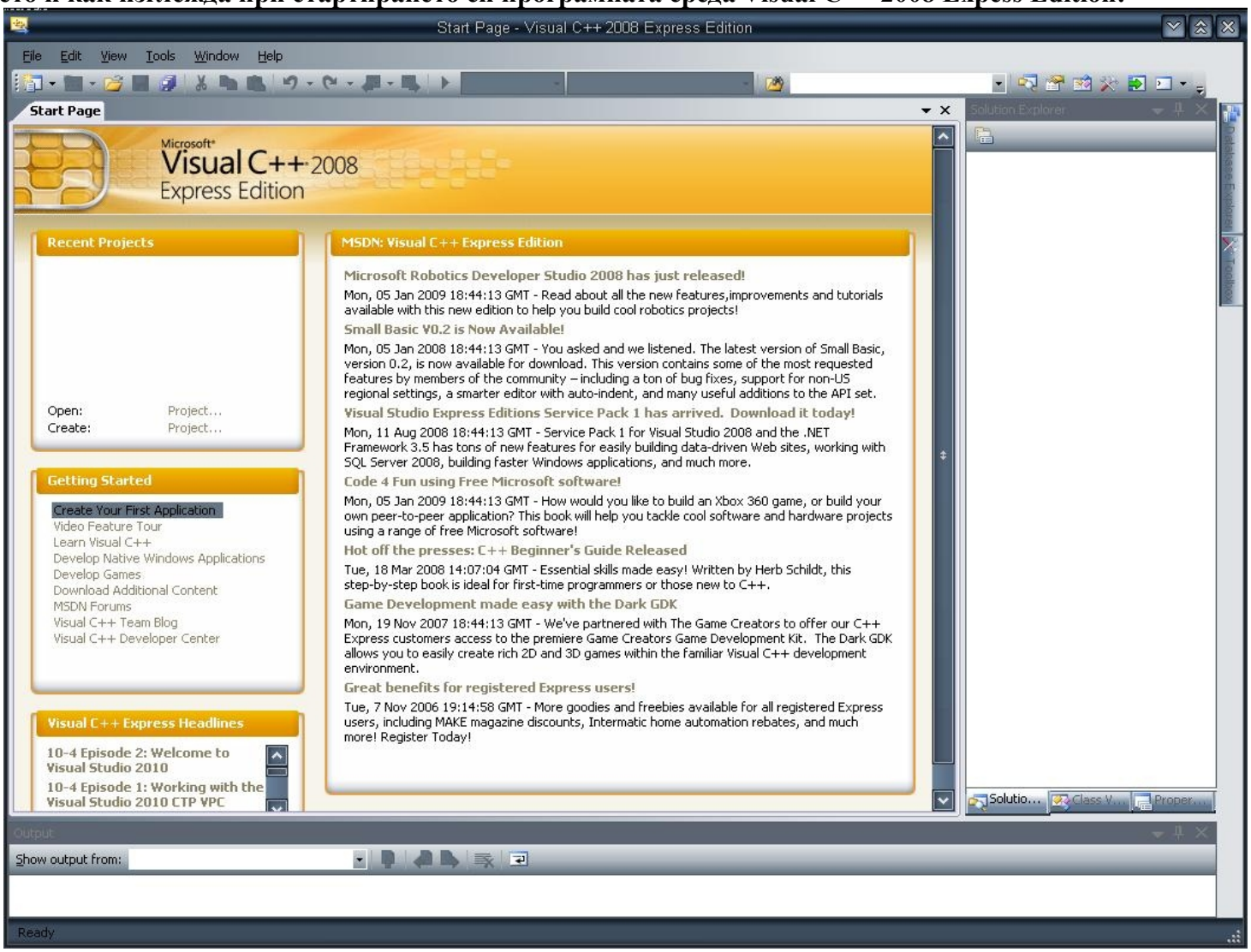
Доста са нововъведенията в този нов продукт на Майкрософт :)
Има толкова много различия м/у Visual 6.0 и Visual C++ 2008 ,че трябва да напиша още един e-book:) Добавени са много повече контроли,подобрения в редактора,възможно е автоматично трансформиране на стар код към новата среда(от VC++6 към v9.0) ,разликите между VS 2005 и VS 2008 далеч не са толкова фрапиращи...по-добър интерфейс и разни подобрения в бързодействието: компилиране и прекомпилиране,дебъгване, зареждането на решения (solutions) е по-бързо...Box с intellisense допълнения може да стане полупрозрачен, ако се натисне Ctrl...и много др. удобства:) сканиране на кода... допълнения към Code Analysis...дебъгването на многонишкови приложения е значително подобрено...
В последното Visual Studio 2008 може да се разработват приложения за различни версии на .NET framework било то 2.0/3.0/3.5 ...така е и с Visual C++ 2008 Express Edition
http://www.microsoft.com/express/support/faq/
от този линк ще разберете какво съдържа и не съдържа тази програмна среда...

Важно е да се отбележи,че тази среда(Visual C++ 2008 Express Edition) не поддържа(не са включени) MFC и ATL !!!

...от Project ->Properties и си избирате: или .NET Framework 2.0/3.0/3.5



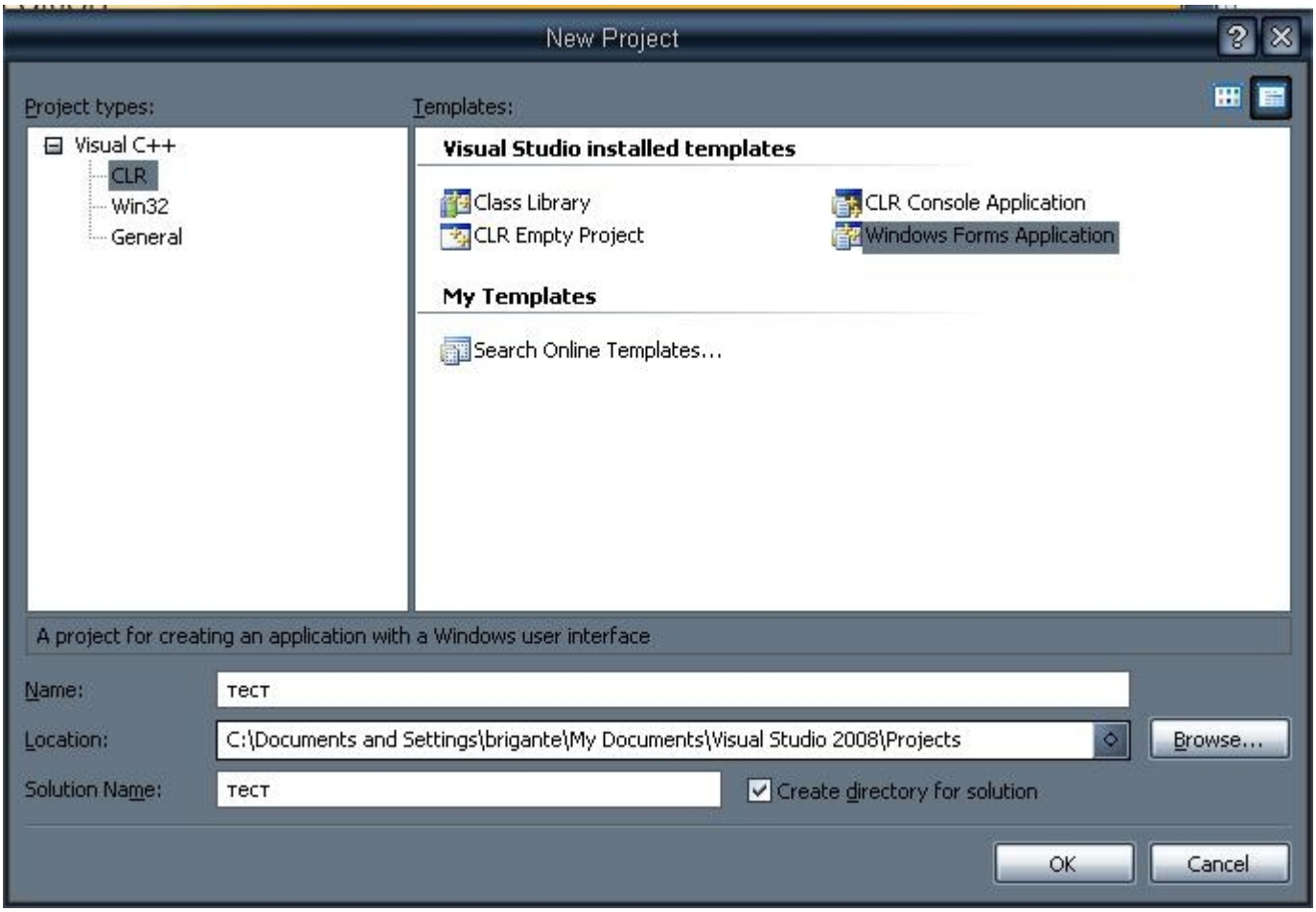
А ето и как изглежда при стартирането си програмната среда Visual C++ 2008 Express Edition:



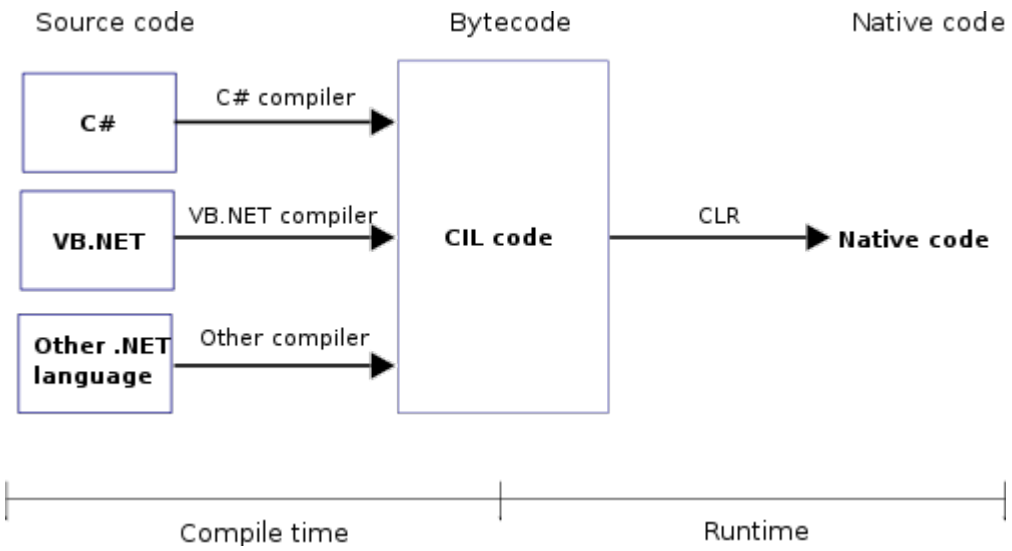
Solution Explorer може отначало да е в левия край,но за удобсво може да се мести:)просто го хванете и завлачвайте и ще видите едни стрелки...поставяте го на част от стрелките...
Забелязвате,че в стартовата страница си има браузър с rss msdn новинки:) кликайки там ще се отвори сайта и ще можете да четете най-новите неща...
В ляво горе има разделче наречено Recent Projects (текущи проекти=създадени вече проекти,които ще се заредят при кликане)
Под него се намира раздела Getting Started: там са част от помощните уроци от сайта на Майкрософт
В раздела Visual C++ Express Headliness научаваме за водещите главни новинки...ако искаме да сме в крак с новостите,то си е задължително да се влиза там:)

Нека сега създадем първият си проект(апликация,приложение)
Създадените приложения се съхраняват в:
C:\Documents and Settings\името ви\My Documents\Visual Studio 2008\Projects

...от File -> New -> Project (или с клавишна комбинация от Ctrl+Shift+N)



CLR е Common Language Runtime = Общ език за изпълнение



Кодът, който се създава под тази развойна програмна среда се нарича управляван код...
Изходът на компилатора включва метаданни (metadata), които са информация, описваща обектите на апликацията като:
Типове данни и техните зависимости
Обекти и техните членове
Референции към изисквани компоненти
Информация за компонентите и ресурсите, използвани за построяване на апликацията
Метаданните при CLR се използват за:
Управление на локациите памет
Локализиране и зареждане на инстанциите на класовете
Управление на обектните референции и преобразуване на колекцията за боклук
Решаване на обръщенията на методите
Генериране на естествен код
Гарантиране, че апликацията е в коректната версия и с необходимите компоненти и ресурси
Задействане на защитата (сигурността)
Метаданните в компилирания софтуерен компонент правят компонентите само-описващи се (self-describing). Това допринася компонента, дори и написан на друг език да може да общува с даден компонент директно. Обекти, чийто мениджмънт е при CLR се наричат управлявани данни (managed data). (Допустимо е да се използват и неуправлявани данни в апликациите)
Common Language Runtime се състои от:
Common Type System - обезпечава поддържането на типовете и операциите над тези типове
Metadata - описва и свързва типовете дефинирани от CTS; обезпечава общия обменен механизъм.
Virtual Execution System - зарежда и стартира програми, написани под CLR; използва метаданни за изпълняване на управляван (managed) код; осъществява обслужвания като garbage collection

Win32 вече знаете,че е за създаване на конзолно приложение:) програмката ще се стартира в команд промпт...

General е за създаване на мейкфайлове и на празни проекти (изграждане от самото начало)

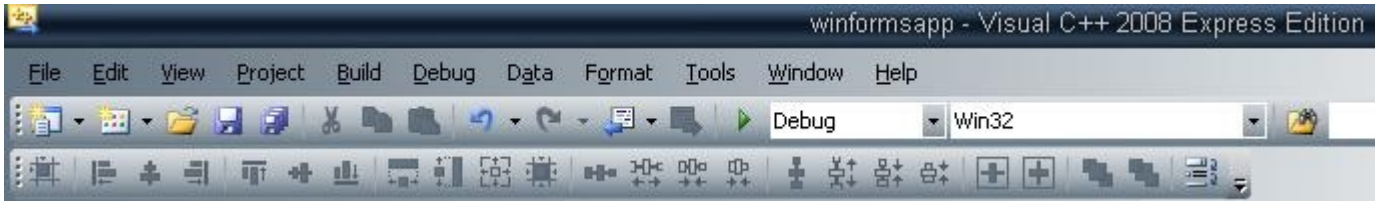
Когато създадете проект и после искате да го унищожите,то просто го изтрийте от директорията...
C:\Documents and Settings\името\My Documents\Visual Studio 2008\Projects
и после няма да го има в Текущи проекти...

Хайде сега се опитайте сами да си създадете тази апликацийка там в сайта на Майкрософт:
кликнете тук

Вече може би сте забелязали,че в тази нова програмна среда вместо така наречените Controls(контроли,инструменти) имаме така нареченото Toolbox (кутийка с инструменти:)
Оттам ще можете да си разполагате бутоните и др. контроли...
Можете да си нагласите Toolbox както Ви е удобно...най-добре е да е на AutoHide (автоматично да се скрива и при допир с мишката да се показват всички(е не всички) контроли:)

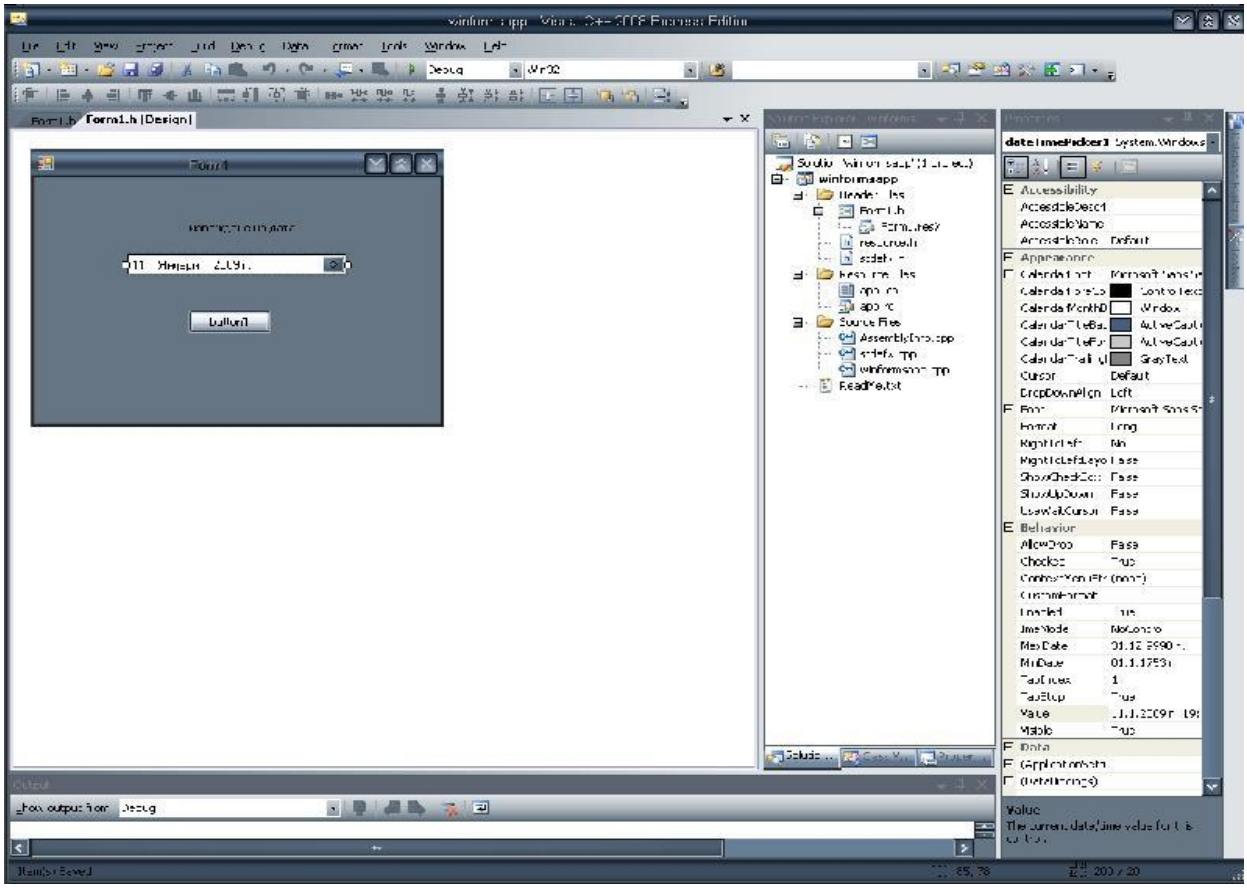
Поставянето на контролите става просто като кликнете веднъж в/у тях и после отидете с мишката до работната форма и кликате там, и така контрола вече е добавен...

Ако се опитате да компилирате,за изградите приложението това става в режим Debug (създава се и такава директория в проекта Ви) Обикновено програмката ще е с по-голям размер,отколкото в крайния режим Release ... а той се задава от главното меню:



...там където по подразбиране е Debug отметнете на Release и създаденият изпълнителен файл ще е с по-малък размер...

Добра практика е постоянно като правите промени по проекта си да си го сейвате(съхранявате)
След като си разположите контролите както искате да са, кликнете два пъти на всеки и разгледайте кода...разбира се ще се добава програмен код м/у конструкцията...
и всичко това забележете вече се случва във форм хедъра:)
Form1.h ,ако си е и при Вас по подразбиране в началото:)
и има също друг Form1.h [Design] ,където си разполагаме контролите...и дума няма да обелим за сравнение м/у версия 6.0 и тази нали:)))
Сами разбирате,че вече летим с тази програмна среда:)))скоростта за създаване на приложения е космическа!!!



Работим по тази апликацийка там в сайта на Майкрософт...не сте забравили нали:)
...просто си разположете 3-те контрола:
Label , DateTimePicker , Button

на лейбълчето изпишете примерно Избери дата: или Изведи датата:
...това става като от Properties(Свойства) (ако не е видим,то значи не е актично в момента и трябва от View на главното меню да си го извикате(да се покаже) и да си го направите или да е показано постоянно или да е на автопоказване/скриване(AutoHide) режим на работа...препоръчвам да е винаги показан в работната среда... там намерете поле Text и си напишете този низ примерно Избери дата: или Изведи датата:
...съхранявайте си проекта във всеки един по-важен момент от работата...

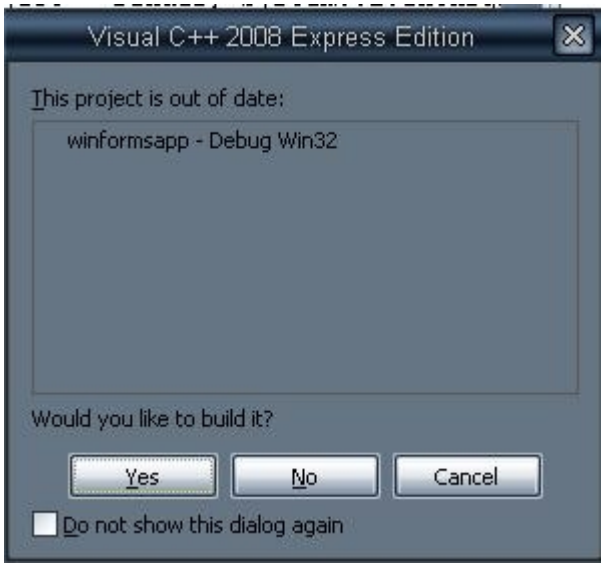
същото е и с контрола button1 ... надпишете си го както искате ...примерно Изход или Затвори
Следва писането на програмния код за обработването на събитията...

за бутона ни разгледайте в кода ... System::Void button1_Click ...
кода ще е : Application::Exit();
просто в конструкцията само това ще напишем,за да се затвори приложението...

а за контрола dateTimePicker кода е както го виждате
label1->Text=String::Format("Новата избрана дата е: {0}", dateTimePicker1->Text);

сега натиснете F7 или от главното меню Build->Build Solution

или от стартиране под режим дебъгване (зеленката стрелка)



натиснете Yes :)
можете да отидете до:
C:\Documents and Settings\името\My Documents\Visual Studio 2008\Projects
име на проекта и в папка=директория Debud да си стартирате приложението и от там...
изградете същото приложение и в режим Release

Добра практика за разучаване на средата е просто да натискате смело и безотговорно абсолютно всичко каквото виждате :) създайте си проектче и започнете да си тествате всеки един контрол ...няма какво да повредите:)

Може да гледате видео урока(на английски е) за въведение в работната среда Visual C++ 2008
http://msdn.microsoft.com/bg-bg/beginner/bb964629(en-us).aspx
...и също така да гледате видео уроците от раздела: How Do I?
http://msdn.microsoft.com/bg-bg/visualc/bb496952(en-us).aspx
дава се също и сорс кода (пише се source code или src)
...още видео урочета могат да бъдат намерени и в сайта ми...кликни тук!

Ако имате чужд сорс код,проект и искате да го отворите,то кликнете в/у името на файла с разширение .vcproj или .sln (solution) и ще се зареди целия проект...

Основни стъпки при създаване на програма:

Всички файлове, които са свързани с разработваната програма, се включват в един общ проект на Visual C++ . NET ... няколко проекта могат да бъдат обединени в едно общо решение (solution), което има обща конфигурация...
За да заредите програма в IDE, отворете файла със съответстващия й проект, като използвате команда File /Open (Отваряне) /Project
При създаване на нова програма е необходимо да се организира нов проект. За тази цел изпълнете команда File /New /Project u в показания диалогов прозорец си изберете Projects(Проекти на Visual C++) От десния списък изберете типа на проекта. После въведете името на проекта в поле Name от същата страница, а в поле Location укажете папката, където искате той ще бъде записан(оставете го по подразбиране) С командите от меню Projects могат да се включват нови (с Add New Item) или вече съществуващи (с Add Existing Item) файлове...

Отваряне на файлове с изходен код в прозореца за редактиране: За да отворите файл с изходен код в прозореца за редактиране, щракнете двукратно върху елемента, който съответства на файла в прозореца за проекта..

Отваряне на файлове без включване в проекта: Ако желаете да преглеждате или редактирате файлове, но без да ги вклкзчвате в проекта, използвайте команда File/Open / File

Добавяне на ресурси: Описанията на ресурсите на Windows-приложенията (менюта, диалогови прозорци, ленти с инструменти и т. н.) се съдържат във файловете с ресурси. Създаването и включването в проекта на нов ресурс става с команда Project /Add Resource (Добави ресурс)
За да създадете нов ресурс, от показания прозорец изберете типа на ресурса и щракнете New
За да включите в проекта съществуващ ресурс, и да го заредите в прозореца на съответния редактор на ресурси, използвайте бутона Import (Импорт) от същия прозорец...

Построяване на изпълним файл на проекта: За да създадете изпълним файл на проекта, използвайте команда Build (Построй) / Build име_на_проекта...

Дебъгване на програмата: Дебъгерът се стартира с команда от менюто Debug /Start

Управление на прозорците в IDE

Един от основополагащите принципи на интегрираните среди за разработка се състои в това, че менютата, лентите с инструменти и прозорците на работната област (областта за редактиране, прозорецът на проекта, прозорецът за съобщения) са „вградени" - т.е. се намират в рамките на главния прозорец на Visual C++ . NET Ако се скрият част от лентите с инструменти, на екрана се освобождава допълнително място, което може да се използва за прозорците за редактиране на файловете с изходния код на проекта...
По подразбиране прозорците за редактиране заемат цялата свободна работна област в главния прозорец.
Препоръки за управление на прозорците в IDE :
Работните параметри и показваните в прозореца на IDE компоненти се настройват от страниците на групата Environment от диалоговия прозорец Options, който се показва с команда Tools /Options

С командите от меню View се показват различните прозорци (прозорецът за ресурси, прозорецът за съобщения, прозорецът със списъка от задачи и т.н.)...
Всички отворени прозорци (припокриващи се страници) за редактиране са включени в списъка от меню Window(Прозорец)
Ако изберете някой от тях в списъка, ще го покажете на по-преден план от всички останали прозорци. Същото ще се случи и ако щракнете върху етикета на страницата над областта за редактиране.

Команда Window /Split (Раздели) разделя активния прозорец на подпрозорци. Това дава възможност да се преглеждат едновременно няколко фрагмента от файла, без да се увеличава броят на отворените прозорци за редактиране...
Разделителната линия в прозореца за редактиране може да се премества с мишката. Ако искате да затворите някой от подпрозорците, придвижете разделителната му линия до разположената по-горе от нея...

Система от менюта:

Ще опиша само най-често използваните от тях...

<u>Меню File (Файл)</u> се използва за общо управление на файлове и на областта на проекта
New (Нов) тази команда показва подменю с команди, с които се отварят диалогови прозорци за създаване на нови файлове, проекти и решения. Оттук се стартират различните помощници за разработка на проекти
Open (Отвори) командата показва подменю, което служи за отваряне на различни файлове, файловете, отворени с тази команда, не се включват в текущия проект
Close (Затвори) командата затваря активния прозорец за редактиране
Add New Item (Добави нов компонент) тази команда създава нов компонент от избрания тип и го добавя към проекта
Add Existing Item (Добави съществуващ компонент) тази команда добавя към проекта нов (вече съществуващ) компонент от указания файл
Add Project (Добави проект) командата създава нов проект от избрания тип в рамките на дадено решение, или добавя в дадено решение вече съществуваш, проект
Open Project/Solution (Отвори проект/решение)
Close Solution (Затвори решение) съхранява включените в решението проекти...
Save (Съхрани)
Save as (Съхрани като)

Save all (Съхрани Всичко)
Page Setup (Структура на страница) с тази команда се задават параметрите на страницата при печат на файловете с изходния код
Print (Печат) командата разпечатва файла, който се намира в активния прозорец за редактиране
Recent Files (Наскоро отворени файлове) показва списък с последните отворени файлове.
Дължината на списъка се задава на страница Environment/General (Работна област/Обши) на диалоговия прозорец Options, който се отваря с команда Tools /Options
Recent Projects (Наскоро отворени проекти) показва списък с последните отворени проекти и решения...
Дължината на списъка се задава на страница Environment/ General (Работна област/Общи) от диалоговия прозорец Options, който се отваря с команда Tools /Options
Exit (Изход)

<u>Менюто Edit (Редактиране)</u>

Меню Edit (Редактиране) съдържа команди за редактиране, търсене и замяна на текст в активния прозорец за редактиране
Undo (Отмени) ако се свъркали нещо(изтрили) в кода се поправете с тази команда...
Redo (Върни)
Cut (Изрежи)
Copy (Копирай)
Paste (Вмъкни)
Delete (Изтрий)
Select all (Избери всичко)
Find and replace (Намери и замени) подменю, което позволява да се търси и заменя зададен текстов фрагмент в прозореца за редактиране или в указано подмножество от файлове
Go To (Премини) командата извършва преход към зададен адрес, определение, референция или ред

...тук в тази програмна среда в сравнение с Visual Studio 2008 менютата са малко поорязани...

Bookmarks (позиции за преход) с тази команда се създават специални позиции за преход, към които след това може да се премине с команда Go To (Премини)

<u>Меню View (Изглед)</u>

По-голямата част от командите в менюView показват различните прозорци за преглеждане на информация...
... Solution Explorer (Преглед на решенията) отваря прозореца за преглед на решенията в областта на проекта
Class View отваря прозореца за преглед на класовете от проекта в областта на проекта

за съжаление тук в тази програмна среда
Resource View (Показване на Ресурсите от проекта) липсва:(
Тази команда отваря прозорец със същото име в областта на проекта, в който може да се преглеждат и редактират ресурсите, използвани в текущия проект...и сорс от други чужди проекти...

Properties Menager или Window (Параметри) командата отваря прозореца, в който се показват параметрите на избрания компонент на проекта...

Toolbox (Палитра с инструменти/контроли)
От Other Windows (Други прозорци) към Web Browser (Браузър) се отваря подменю с команди за управление на браузъра (за по-удобен достъп до справочната система)

Tasks List (Задачи) избор на информацията, която ще се показва в областта за съобщения
Toolbars (ленти с инструменти) za избор на показваните ленти с инструменти

Full Screen (Пълен екран) командата превключва текущия прозорец за редактиране в пълноекранен режим. С натискане на клавиша [Esc] можете да се върнете в нормален режим...
Следва двойка команди за навигация (преход към предишното и следващото съдържание на областта за редактиране)

Property Pages (Параметри на страницата) командата показва на екрана диалогов прозорец с информация за проекта, който е избран от областта за проекта

Меню Project (Проект)

Меню Project съдържа команди за създаване и управление на проекта.
Add Class (Създай клас) тази команда показва на екрана диалогов прозорец, от който може да се зададе нов клас, като за него се укажат базови класове и файлове с изходен код

В програмната среда Visual Studio 2008 има
Add Resource (Добави ресурс) ... при нас липсва:)

Add New Item (Добави нов компонент) с тази команда се създава и включва в проекта нов компонент от указания тип

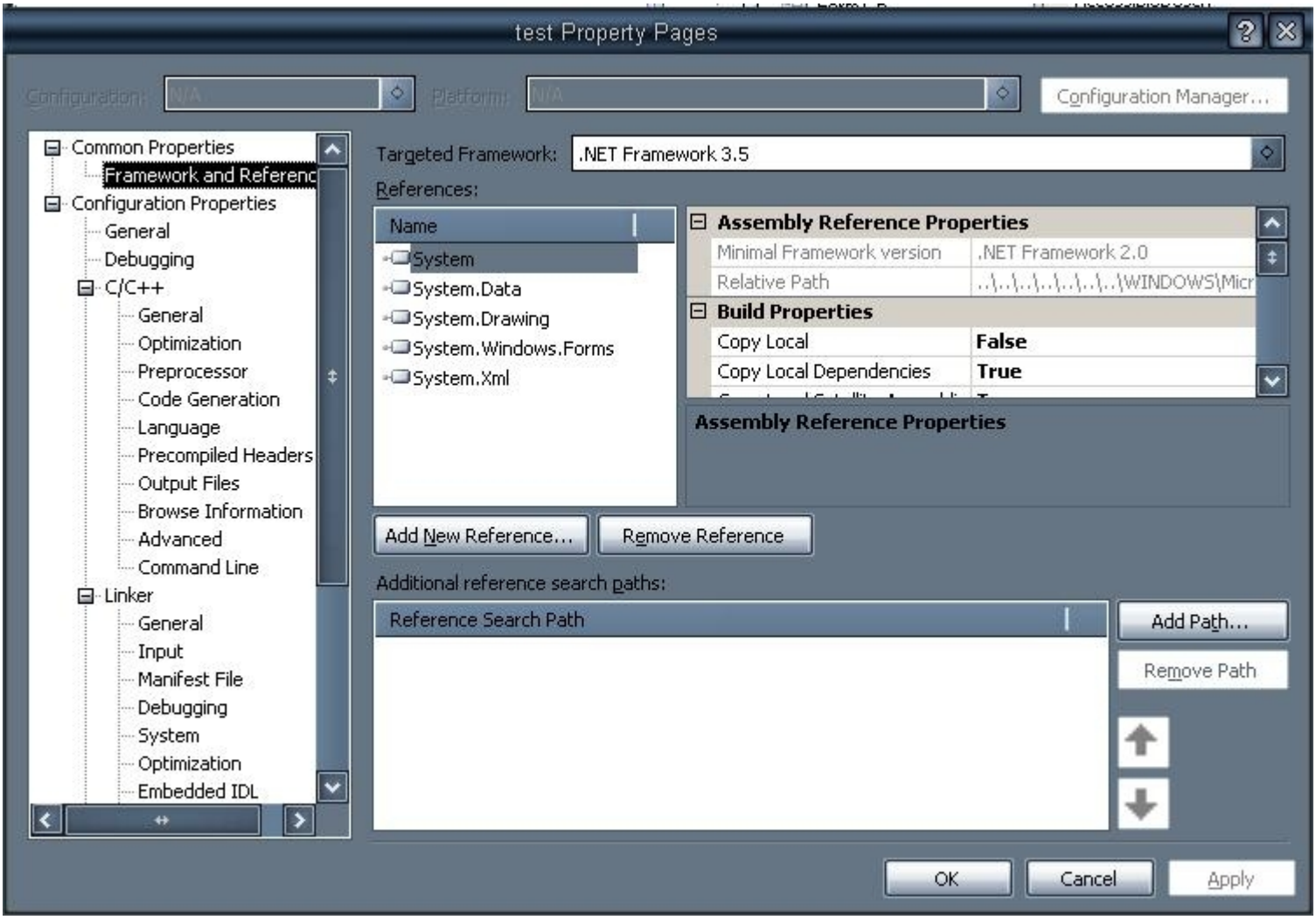
Add Existing Item (Добави съществуващ компонент) командата добавя към проекта нов компонент от избрания файл

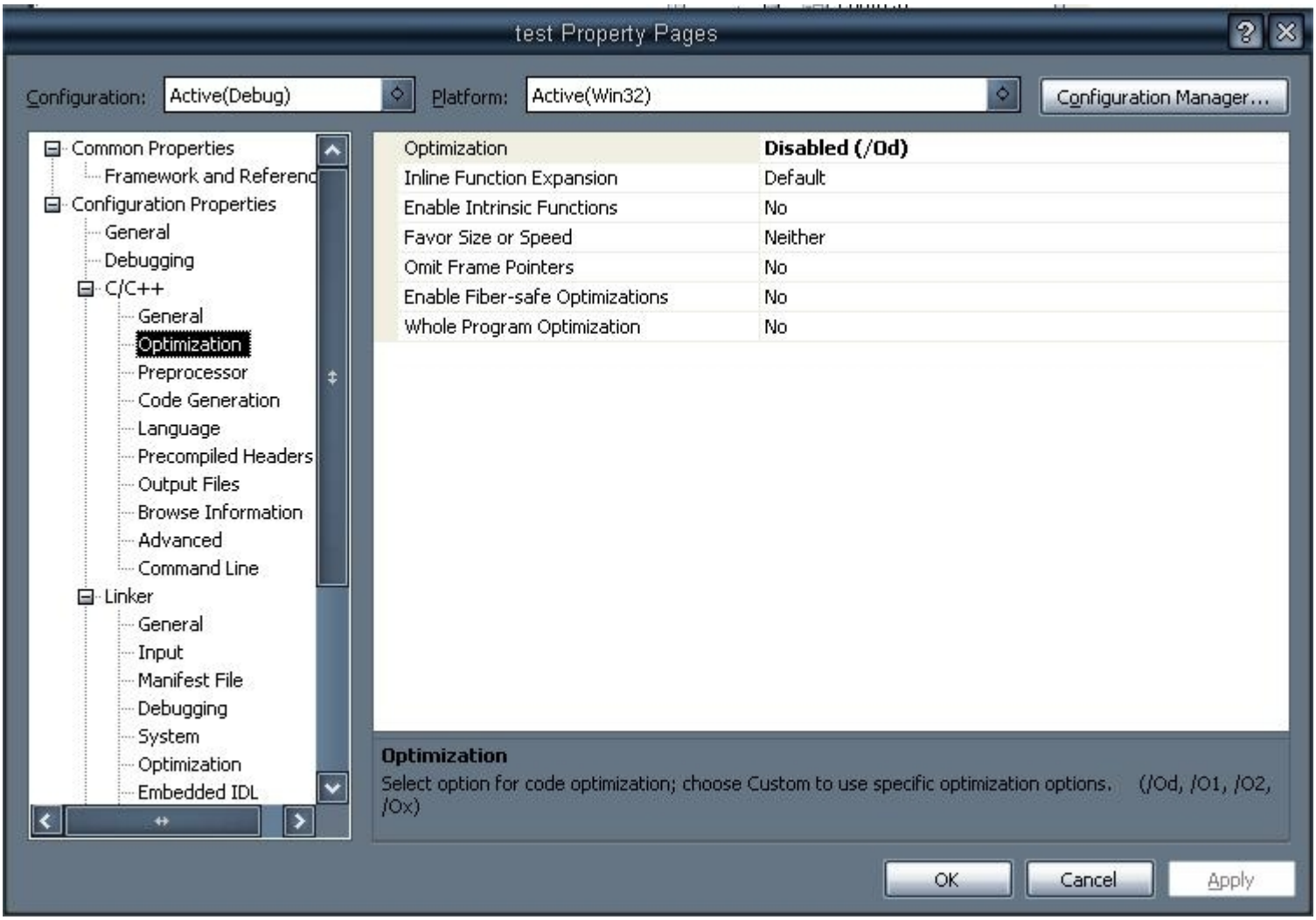
...орязали са ни също така:
Newfolder (Нова папка) добавя нова папка към проекта
Unload Project (Затвори проекта) затваря проекта, като съхранява неговите компоненти
Add Web Reference (Добави Web-референция)
Set As Startup Project (Направи начален проект по подразбиране)
Project Dependencies (Зависимости на проекта)
Project Build Order (Ред на построяване на проектите)

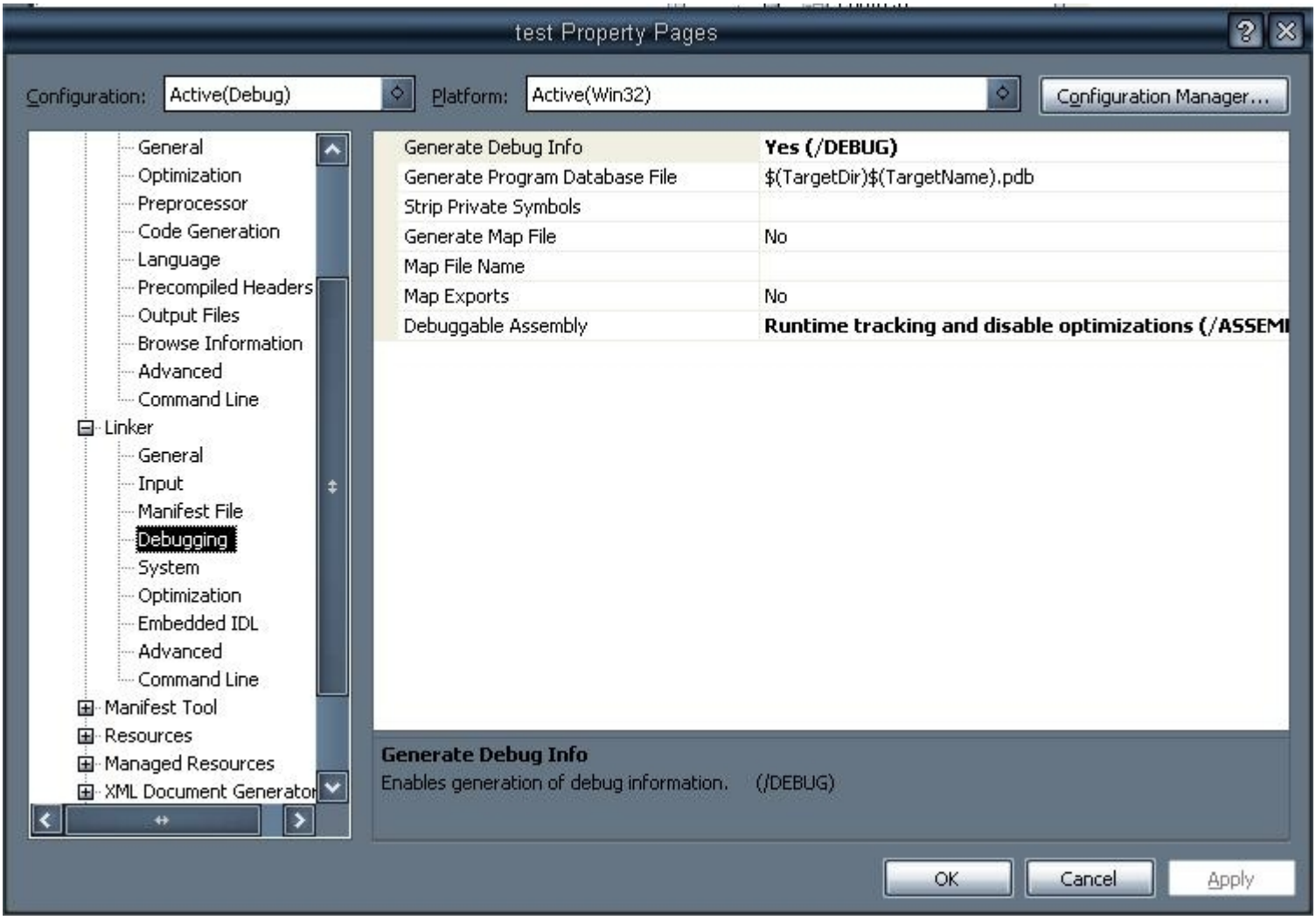
име на проекта Properties (Параметри) командата показва на екрана диалогов прозорец, от който може да настройвате параметрите на проекта...може например да зададете настройки за компилатора, свързващия редактор и дебъгера, да укажете параметрите на командния ред и т.н.

Меню Build (Построй)

Меню Build съдържа функции за разработка на програмата, както и команди за стартиране на вградения дебъгер
За работата на дебъгера в проекта трябва да се включи специална допълнителна информация.
Тя се генерира при компилация и свързване, ако за проекта в диалоговия прозорец името на проекта Property Pages, който се показва с команда Project /Properties са зададени следните параметри:
от +Configuration Properties
C/C++/General
• Стойността на поле Optimizations (Оптимизация) от страница C/C++/Optimization да си е Disabled (Изключено)
• На страница Linker(Свързбане/Дебъг) е маркирано поле Generate Debug info (Генерирай дебъг информация) да си е на YES(DEBUG)
...разгледайте останалите сами:)


test Property Pages
Configuration: N/A
Platform: N/A
Configuration Manager...
Common Properties
Framework and Referenc
Configuration Properties
General
Debugging
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Targeted Framework: .NET Framework 3.5
References:
Name
System
System.Data
System.Drawing
System.Windows.Forms
System.Xml
Assembly Reference Properties
Minimal Framework version .NET Framework 2.0
Relative Path ..\..\..\..\..\..\..\..\WINDOWS\Micr
Build Properties
Copy Local False
Copy Local Dependencies True
Assembly Reference Properties
Add New Reference...
Remove Reference
Additional reference search paths:
Reference Search Path
Add Path...
Remove Path
OK
Cancel
Apply


test Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Common Properties
Framework and Referenc
Configuration Properties
General
Debugging
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Optimization Disabled (/Od)
Inline Function Expansion Default
Enable Intrinsic Functions No
Favor Size or Speed Neither
Omit Frame Pointers No
Enable Fiber-safe Optimizations No
Whole Program Optimization No
Optimization
Select option for code optimization; choose Custom to use specific optimization options. (/Od, /O1, /O2, /Ox)
OK
Cancel
Apply



Build solution (Построй решение) командата построява всички проекти, вкючени в решението. Ако след създаването на изпълнимия модул в някой от файловете с изходен код на проекта са били внесени изменения, поправеният изходен файл ще бъде компилиран отново. Изпълнимият модул също ще бъде свързан отново

Rebuild Solution (Препострой решение) командата първо компилира всички изходни файлове от всички проекти от решението, след което свързва отново и изпълнимите модули на проектите

Clean Solution (Почисти решение) командата премахва всички междинни и резултантни файлове в проектите на решението

Build името_на_изпълним_файл (Построй името_на_изпълним_файл)... тази команда построява указания изпълним файл (на проект)...ако след създаването на изпълнимия модул в някой от файловете с изходен код на проекта са били внесени промени, поправеният изходен файл ще бъде компилиран отново, а изпълнимият модул - съответно свързан още веднъж

Rebuild името_на_изпълним_файл (Препострой проекта) командата компилира всички файлове с изходен код в проекта, след което отново свързва и изпълнимия модул на проекта

Clean името_на_изпълним_файл (Почисти проекта) тази команда премахва всички междинни и резултантни файлове за проекта

Batch build (Пакетна обработка) командата показва на екрана диалогов прозорец, в който може да се изберат и създадат няколко конфигурации на проекта

Configuration Manager (Конфигурации) командата служи за конфигуриране настройките на проекта, които ще се използват при разработка на програми. С нея също може да се зададе и нова конфигурация за текущия проект. Настройките може да се променят в диалоговия прозорец име на проекта Property Pages, който се показва с команда Project / Properties

Compile (Компилация) компилира избрания в областта на проекта изходен файл

Меню Tools (Инструменти)

С командите от менюTools се викат от IDE различни спомагателни програми, а също така може да се настройва интегрираната среда за разработка(програмната среда)

Connect to Database (Свържи се с база данни)

Customize тази команда показва диалогов прозорец със същото име, от който може да си организирате менютата, лентите с инструменти и клавиатурата по начин, който е най-удобен

Options командата отваря диалогов прозорец със същото име, от който може да се настройва интегрираната среда за разработка

Меню Debug
С командите от меню Debug се извършва проверка и оптимизация на програмите от проекта ... няма да обяснявам работата с дебъгера:)

Меню Help (Справочна система)
В меню Help (Справочна система) са включени командите на библиотеката MSDN

Меню Window(Прозорец)
Меню Window включва функции за управление на прозорците

Ако примерно искате приложението да има леко прозрачен изглед (Opacity) може да направите това ето така: оразмерете(модефицирайте) си прозореца както искате като издърпвате долу краищата му...след това отидете на Properties(а ако не е видимо в програмната среда идете към главното меню и от View отидете на Other Windows и в подменюто намерете Properties Window и ще се покаже...добра практика е винаги да е показано в IDE) ... в Properties намерете къде е Opacity и намалете стойността (тя е на 100%) сложете примерно 50-70 % ... компилирайте и забележете разликата в приложението си...

Създаване на менюта, ленти с инструменти, лента за състояние

Потребителите получават достъп до предоставяния набор от команди на приложенията с помощта на менюта. Лентата за менюта обикновено се свързва към прозореца с рамка като ресурс. Отделните изскачащи менюта са не само команди на прозореца с рамката, но и команди на работния прозорец. Обикновено на прозореца с рамка се подчиняват най-общите команди, а по-специфичните се обработват от работния прозорец. Този вариант е най-оптимален, когато приложението работи с различни работни прозорци или дъщерни MDI прозорци и техните ленти с менюта съдържат различни команди. Командите, които се отнасят към определен работен прозорец или дъщерен MDI прозорец, се обработват от тези прозорци. Командите, които се отнасят за приложението като цяло, се обработват от прозореца с рамката (при модела документ/представяне в обработката на командите взема участие и класа за документа)...
Освен лентите с менюта в Windows-приложенията се използват също и ленти (палитри) с инструменти, в които са представени най-важните команди от менютата, както и така нар. лента за състоянието, в която се показва например пояснителен текст за командите от менютата...

Ако проектът Ви е създаден с помощта на съветника за изграждане на MFC-приложения в програмна среда Visual Studio 2008 , в него вече е включена стандартна лента с менюта (а вероятно, също и ленти с инструменти и за състоянието), които можете безпроблемно да настроите за решаване на вашите задачи...но тук в тази програмна среда MFC няма...
Всички ресурси от един и същ тип трябва да имат уникални идентификатори. Ресурсите от различни типове може да имат и еднакви идентификатори. Тези правила се използват в следните ситуации:
На стандартните ресурси за приложението (лента с менюта, клавишни комбинации, икона на

приложението, текстовия ресурс за заглавието на главния прозорец) може да се присвои един и същ идентификатор. Така ще се намали общият брой на идентификаторите за ресурси...така всички едноименни ресурси ще могат да бъдат заредени само с едно обръщение към метода LoadFrame() от класа CFrameWnd или при създаването на обекта CDocTemplate
Удобно е да присвоите един и същ идентификатор на команда от меню, клавишна комбинация и бутон от палитра с инструменти, свързани с едно и също действие...след това е достатъчно да напишете един обработващ метод, който да се вика независимо дали потребителят е избрал съответната команда от менюто, натиснал е еквивалентната й клавишна комбинация или е кликнал(щракнал) върху бутона...

<u>Създаване на меню в Visual C++ 2008 Express Edition:</u>
Редакторът на менюта е предназначен за създаване и редактиране на менюта. Прозорецът му за редактиране показва проектираното меню във вида, в който то ще се появи във вашето приложение (WYSIWYG=What you see is what you get)
Новите елементи от менюто се въвеждат в свободна клетка, разположена на желаното ниво и след това се настройват от областта Properties
За създаване на меню изпълнете следната процедура:
...от Toolbox намерете раздел Menus & Toolbars и изберете MenuStrip ...поставете го в/у работната си рамка...въведете текста за елемента от менюто. За тази цел щракнете в празна клетка с мишката и въведете желания текст за елемента от менюто
В другите полета от областта за свойства може да зададете и останалите параметри, като например началното състояние на елемента от менюто...цвят на менюто,добавяне на малка картинка в менюто...
Задължително трябва да въведете и идентификатор. Той е необходим за обработка на съобщенията, постъпващи от елемента на менюто...
Областта за свойства Properties се показва с едноименната команда от контекстното меню...в момента само си проектираме менюто...после се слага програмния код, необходим за работата на менюто...
Основните стъпки при визуално проектиране са:
• Създаване и настройка на елемент от меню. Въведете текста за елемента от менюто в избраната празна клетка на жееланото ниво. След това настройте елемента от менюто от областта Properties
• За всяка от командите от менюто е задължително да въведете идентификатор. Той е необходим за обработка на съобщенията, постъпващи от елемента на менюто
• Преместване на елемент от менюто. След като въведете елемента от менюто в свободната (т.е. последната) клетка, спокойно можете да го преместите на желаната позиция
• Създаване на функция за обработка на съобщения, постъпващи при избор на командата от менюто. С помощта на помощника за работа с класове създайте функция за обработка на съобщенията при избор на командата от менюто и добавете във функцията необходимия изходен код
• Присвояване на клавишна комбинация...

Ако искате да премахнете някое поле от менюто , изберете в редактора съответния ред и натиснете клавиша [Del] или с десен бутон на мишката и се придвижете до Delete...

...следва продължение:) писна ми:) когато ми кефне тогава:)
в книжката са ползвани доста материали от други e-books:)
-този ел.учебник е създаден за начинаещи

Благодаря на Мирелка за подкрепата!!! {}