ОСОБЕНОСТИ НА ЕЗИКА C++

1. Цел на упражнението

Запознаване с особеностите на езика C++ - организация и управление на вход/изхода, работа с потоци. Разглеждат се някои по-важни стандартни функции на C++.

2. Пространство на имената

Писането на все повече и повече библиотеки, модули или компоненти може да създаде сериозни конфликти при тяхното идентифициране (именоване). Причината за това е, че е възможно модулите и библиотеките да използват един и същ идентификатор за различни неща. Пространствата на имената (*namespaces*) решава този проблем. Чрез пространство на имената може да се дефинират компоненти, които са разпределени в различни физически модули.

Едно пространство на имената групира идентификаторите в именована област на действие. Дефинирайки всички идентификатори в съответно пространство, името на пространството е един глобален идентификатор, който може да бъде в конфликт с другите глобални символи. За да се дефинира един символ в съответно пространство на имената, е необходимо съответният идентификатор да бъде предхождан от името на пространството, разделени чрез оператора за принадлежност **::**, по следния начин:

```
// дефиниране на идентификатор в пространството alfa
namespace alfa
{
     class Person;
     void myGlobalFunc();
     …
}
// използване на съответното пространство
alfa::File     obj;
```

Не е необходимо да се дефинира пространство на имената за функции, ако един или повече от типовете на аргументите са дефинирани в пространството на функцията. Например:

```
// дефиниране на идентификатор в пространството alfa
namespace alfa
{
     class Person;
     void myGlobalFunc(const File &);
     …
}
// използване на съответното пространство
alfa::File     obj;
myGlobalFunc(obj); // открива се alfa::myGlobalFunc();
```

Използвайки директивата `using`, може да се избегне многократното повторение на идентификатора на пространството. Например,

```
using alfa::File;
```

прави `File` локален синоним в текущата област на действие, което съответства на `alfa::File`.

Директивата `using` прави всички имена от пространството налични, така както те биха били декларирани извън тяхното пространство. Може да възникне конфликт с обикновения идентификатор. Например, директивата

```
using namespace alfa;
```

прави `File` и `myGlobalFunc()` глобални в текущия обхват. Компилаторът ще съобщи за двусмисленост (неопределеност), ако съществува и идентификатор `File` или `myGlobalFunc()` в глобалната област на действие и потребителят използва име без съответен определител.

Следователно, не трябва да се използва директивата* `using`*, ако съответният контекст не е ясен. Директивата може да промени областта на действие на идентификаторите, в резултат на което може да се получи различно поведение от очакваното, ако разработеният код се използва от друг модул. Използването на* `using` *в заглавни файлове, библиотеки и модули е лоша практика.

Стандартната библиотека на C++ дефинира всичките идентификатори в пространството `std`. Това е един типичен пример за необходимостта от използване пространство на имената.

Съгласно концепциите, свързани с пространството на имената, съществуват три възможности при използване на идентификатор от стандартната библиотека:

1. Идентификаторът може да се определи директно. Например, може да се напише `std::ostream` вместо `ostream`. Ето как би изглеждал един пълен оператор:
   ```
   std::cout << std::hex << 3.4 << std::endl;
   ```
2. Може да се използва *декларацията* `using`. Например, следният фрагмент дава възможността за пропускане на **`std::`** за `cout` и `endl`.
   ```
   using std:cout;
   using std:endl;
   ```
   Така че горният пример може да бъде написан по следния начин:
   ```
   cout << std::hex << 3.4 << endl;
   ```
3. Може да се използва *директивата* `using`. Това е най-лесният начин. Чрез директивата `using` всички идентификатори от пространството `std` ще бъдат налични, сякаш те са декларирани глобално.
   ```
   using namespace std;
   ```
   Това позволява горният пример може да бъде написан по следния начин:
   ```
   cout << hex << 3.4 << endl;
   ```

3. Заглавни файлове

Използването на пространството на имената `std` за всички идентификатори на стандартната библиотека на C++ беше въведено при утвърждаването на новия стандарт на C++. Тази промяна не води до

съвместимост в обратна посока със старите заглавни (header) файлове, в които идентификаторите от стандартната библиотека на C++ са декларирани с глобален обхват на действие. Някои интерфейси на класовете се промениха при въвеждане на новия стандарт, но целта беше да се запази съвместимост в обратна посока, ако това е възможно. Така че, беше въведен нов стил в имената на заглавните файлове. Това позволява да се осъществи обратна съвместимост чрез осигуряване на възможност за използване на старите заглавни файлове. Дефинирането на нови имена за стандартните заглавни файлове се осъществяваше чрез въвеждането на различни разширения (`.h, .hpp,.hxx`). Имената на заглавните файлове по новия стандарт нямат разширения. Ето как ще изглеждат някои от директивите `include` за стандартните заглавни файлове:

```
#include <iostream>
#include <string>
```

Това се прилага също и за заглавните файлове, приети от стандарта на езика C (стария стандарт). Имената на заглавните файлове от езика C по новия стандарт имат нов префикс `c`, вместо старото разширение .h:

```
#include <cstdlib>          //беше: <stdlib.h>
#include <cstring>          //беше: <string.h>
```

В тези заглавни файлове всички идентификатори са декларирани в пространството `std`.

Едно от предимствата на тази схема на именоване е, че може да се направи разлика между стария файл `string`, който се отнася за функциите на C `char *` от новия заглавен файл `string` за стандартния клас на C++ `String`.

```
#include <string>           //C++ class string
#include <cstring>          //char* functions from C
```

За поддържане на съвместимост старите заглавни файлове на C все още съществуват. Ако е необходимо все още може да се използва следната декларация:

```
#include <stdlib.h>
```

В този случай идентификаторите са декларирани като глобални и в пространството `std`.

За заглавните файлове на C++ от стария формат (`<iostream.h>`) няма спецификация в новия стандарт, те не се поддържат. Но повечето производители на компилатори на C++ ги включват в своите конфигурации, за да осигурят съвместимост в обратна посока.

<u>Следователно, или трябва да се използват старите имена на заглавни файлове или да се възприеме изцяло новият стандарт.</u>

4. Потоци

Потоците са нова техника в C++ за управление на входно/изходни операции. В повечето случаи техниката на използване на потоци е доста по-различна (и по-добра) от използването на ANSI C функциите. В ANSI C се използва метод, базиран на файлове, файлови указатели и файлови дескриптори. Защо разработчиците на езика промениха частта за В/И на C++? Основната причина беше входно/изходните операции да се представят в

обектно-ориентиран вид, като се слеят (вмъкнат) с останалата част на езика. Входно/изходните операции в C++ се осъществяват в по-голямата си част чрез използване на предефинирани оператори, а не чрез функции.

4.1. Недостатъци на `stdio` *подхода*

Езикът C беше някак си особен за своето време. В него не бяха вградени ключови думи за много от операциите, които са най-често срещани в повечето програми, като вход/изход, работа с низове и др. Разработчиците на езика решиха да поставят тези операции в динамична библиотека. Полученият компилатор беше по-малък и по-преносим в сравнение с другите съвременни компилатори. Входно/изходните операции са особено деликатна област за всеки език, тъй като те са тясно свързани с операционната система и хардуера. В ANSI C много от проблемите по преносимостта на В/И бяха решени чрез реализирането на специална `runtime` библиотека, която позволява на програма на C да използва всяко В/И устройство по еднотипен начин – било диск, терминал или принтер – сякаш то е файл. От програмиста се изисква да предостави на компилатора име, показващо типа устройство, с което иска да работи.

В/И система в ANSI C е добра, но не е идеална. Използването на "стандартни" файлове за вход, изход и грешки осигурява някои ценни характеристики, като пренасочване на В/И и използването на канали (абстрактни файлове). Проблемът е в това, че макар методът да е елегантен, той не успява да вземе под внимание понякога огромните разлики в обслужването на устройствата от гледна точка на програмиста, тъй като устройствата могат да се разделят на две категории: блокови устройства и символни устройства. Например, дисковете са блокови устройства, а терминалите са символни устройства. Блоковите устройства изискват буфериране, за да се извършва ефективно четенето и записа на данни, докато при символните устройства такова изискване не съществува. Периферните устройства явно притежават много повече различия от току-що посочените. Няколко изключения бяха въведени в библиотеката, за да се обслужват специални устройства. Третирането на външните устройства като файлове беше новаторска идея за края на 60-те години, но тя притежава доста недостатъци. Прекрасно е да бъде установена еднотипност в извикването на В/И функции, но в повечето случаи програмистите не управляват периферните устройства по начина, по който биха обработвали файл.

Този проблем доведе до добавянето на все нови и нови функции в езика ANSI C, тъй като програмистите понякога трябваше да знаят дали работят с дисков файл или с хардуерно устройство.

4.2. Потоците в C++

Концепцията за поток не е нова за C++. Всъщност, ANSI C използва така наречените `streams` (потоци), за да обозначава вид абстрактен порт, чрез който неструктурираните изходни данни могат да се пренасят едно- или дву- посочно. В ANSI C под поток се разбира В/И конструкция от ниско ниво, върху която се изгражда структурираната файлова система. C++ е разработен за работа с класове, но езика сам по себе си рядко използва класове и то само чрез библиотеки. Използването на класове чрез достъп до библиотеки запазва компилатора по-малък и по-преносим. Потоците в C++ се различават от повечето

останали части на езика, тъй като са изградени изключително на базата на класове. Потоците са реализирани чрез цяла йерархия от класове – **iostream class library**.

Потоците са реализирани отделно от `stdio` функциите. Поради това използването едновременно на потоци и `stdio` в една програма може да доведе до проблеми. Потоците представляват защитна обвивка, която дава на В/И операции полиморфно поведение и други обектно-ориентирани възможности. Макар че потоците често се асоциират с В/И, в действителност те са една абстракция, широко приложение, за трансфер на данни от един обект към друг, като обикновено се използва механизъм за буфериране. Това означава, че всяка функция, използвана за пренасяне на данни от едно местоположение на паметта в друго, с или без промяна на данните, може да се разглежда като потокова операция. Потоковата библиотека притежава много класове, но всеки от тях се вмества в една от следните три категории: стандартен вход/изход, файлов вход/изход, форматиране в паметта.

4.3. Структура на библиотеката за вход-изход

Езикът C++ не притежава вградени езикови конструкции за входно-изходни операции. Те се осъществяват чрез библиотечни функции, организирани в стандартната библиотека `iostream`. Нейната основна задача е да даде на програмиста готови средства за входно-изходни операции, като процесът на въвеждане и извеждане на информация е унифициран. Библиотеката позволява по един и същ начин да се оперира, както с вградени типове данни, така и с типове, дефинирани от програмиста.

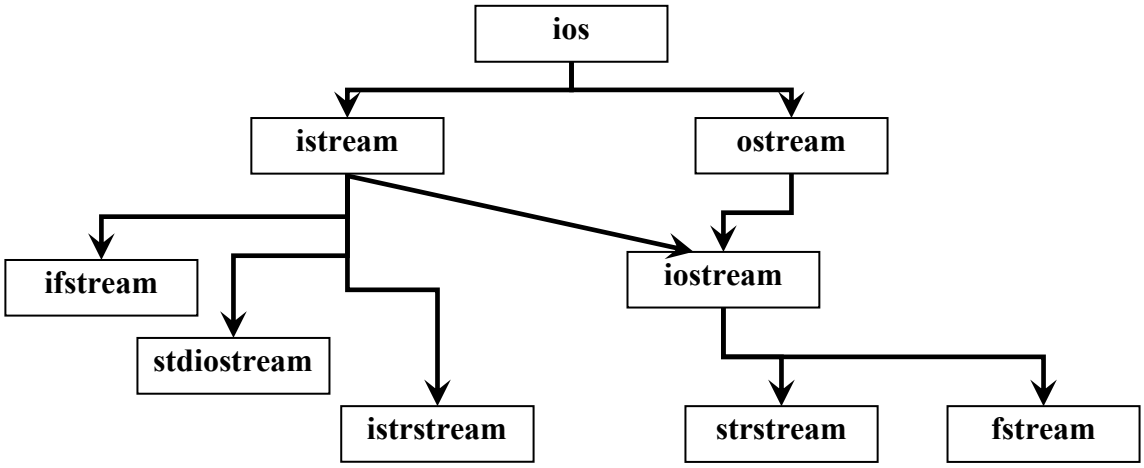
Библиотеката **iostream** е изградена на основата на входно-изходен поток. Входните и изходните данни се организират в поток, представляващ подредена последователност от байтове. Потокът може да представлява двоични данни или да се разглежда като последователност от ASCII символи. Когато потокът се прехвърля от програмата във файл (или обратно) като двоични данни без преобразувания, се говори за неформатиран двоичен вход/изход. Когато прехвърлянето на данните се извършва съгласно зададен формат, се казва, че има форматиран вход/изход.

Потокът за въвеждане/извеждане на данни е модел, който се използва на ниво програма и този модел е машинно-независим. Програмистът не се интересува от специфичните особености на външните устройства. Той записва и чете данните във файлове. Системният модел за обработка на потоци може да бъде описан по следния начин:

- чрез предварително дефинирани операции в стандартната библиотека **iostream** програмистът активира съответните входно-изходни действия. Операнди на операцията са обектите, които представляват външните устройства и данни от входно-изходния поток. Освен предварително дефинираните операции могат да се използват и методи на класовете от библиотеката **iostream**. Формален параметър на методите е входно-изходният поток;
- библиотечните функции прехвърлят данните от програмата в буфер на файла. Тези операции са машинно-независими;

- операционната система прехвърля данните от буфера във външното устройство. Тези операции са машинно-независими и се извършват от драйверите на външните устройства.

В разгледания модел се използва буфериране на информацията. Като цяло това позволява да се повиши надеждността и ефективността на работа. Стандартната библиотека `iostream` е изградена на основата на повече от 20 класа. По-долу е представена една опростена йерархия на тези класове.



Поради сложността си библиотеката е разделена на пет части и връзката с нея се осъществява чрез следните пет заглавни файла:

- **<iostream>** - Използва се във всички в/и операции. Съдържа описания на основни класове. Във файла са дефинирани стандартните обекти `cin, cout, cerr, clog`;
- **<fstream>** - Използва се при работа с дискови файлове;
- **<strstream>** - Използва се при работа със символни масиви;
- **<stdiostream>** - Използва се при смесване на програми на C и C++, когато C модулите използват файловия дескриптор FILE;
- **<iomanip>** - Използва се при форматиране чрез манипулатори.

5. Операция за стандартен изход

Операцията за стандартен изход е дефинирана за всички вградени типове и се прилага директно за различните типове.

```
int k = 10;
double f = 12.18;
char c = 'A';
char *s = "This is a string";
cout << "Стойността на k = " << k << endl;
cout << "Стойността на f = " << f << endl;
cout << "Стойността на c = " << c << endl;
cout << "Стойността на s = " << s << endl;
```

Манипулаторът `endl` вмъква управляващия символ за нов ред в изходния поток и изчиства буфера (използването на манипулатори ще бъде по-подробно обяснено в секция ... ***"Управление формата на входно-изходния поток***".

Задача 1:

Да се състави програма, която дефинира и инициализира променливи от различни типове, включително указатели и извежда стойностите им на екрана с операция <<.

Решението на тази задача е представено във файла `H:\PROBLEMS\EX2_1e.CPP`.

Да се създаде проект с име `EX21`.

Файлът `EX2_1e.CPP` да се добави към създадения проект (виж "Добавяне на .cpp файл към съществуващ проект")

Проектът да се компилира и изпълни.

Да се анализират получените резултати.

Извеждането на информация може да се осъществи освен чрез операцията << и с помощта на методи на класа `iostream`. Функциите имат следните прототипи:

- **`ostream& put(char);`** - извеждане на символ;
- **`ostream& write(const char *ptr, int number);`** - извеждане на `number` на брой последователни байта, сочени от ptr;
- **`ostream& flush();`** - изчистване на изходния буфер.

6. Операция за стандартен вход

Операцията >> е дефинирана за всички вградени типове. По подразбиране входът за операцията >> е от клавиатурата. Входният поток се обработва по следния начин. Предшестващите празни символи (\n, \t и интервал) се пропускат. Въвеждането на съответната стойност завършва при достигането на празен символ. Потребителят е отговорен за правилното оформяне на входния поток. Ако във входния поток се срещне символ, който не отговаря на типа на въвежданата стойност, въвеждането се прекратява.

За въвеждане на информация могат да се използват и методите на съответните класове. Функциите имат следните прототипи:

- **`istream& get(char& c);`** - служи за въвеждане на символ от входния поток (включително и празни символи), който се присвоява на променливата c. (например `cin.get(c)`);
- **`int get();`** - служи за въвеждане на символ от входния поток, който се връща като резултат. Въвежда празни символи и също символ за край на файла EOF (CTRL/Z). Обикновено символът EOF се определя да бъде равен на -1 и се дефинира като int;

```
int k;
while ((k=cin.get()) != EOF)
 ……..;
```

- **istream& get(char *ptr, int len, char delim='\n');** - служи за въвеждане на символи от входния поток, включително и празни символи. Въвежданите символи се записват в символен масив, указан от `ptr`. Въвеждането завършва при изпълнение на едно от следните условия: прочетени са `len-1` символа; прочетен е ограничителят `'\n';` входният поток е изчерпан. Въвеждането завършва със записването на '\0';
- **istream& getline(char *ptr, int len, char delim='\n');** - напълно аналогичен на метода `get(char *,int,char)`. Различава се по начина на обработване на ограничителя '\n'. Ограничителят се извлича от входния поток, но не се записва в масива;
- **istream& read(char *ptr, int number);** - служи за въвеждане на `number` на брой последователни символа, които се записват в символния масив, указан чрез `ptr`.

7. Управление състоянието на входно-изходния поток

Библиотеката **iostream** поддържа набор от флагове за контрол на входно-изходните операции. Те са описани в класа **ios** като стойности на изброимия тип **io_state**:

```
enum io_state { goodbit, eofbit, failbit, badbit, hardfail }

ios::goodbit        0x00 Операциите са успешни
ios::eofbit         0x01 Прочетен е край на файла
ios::failbit        0x02 Последната в/и операция е неуспешна
ios::badbit         0x04 Невалидна операция
ios::badbit         0x80 Непоправима операция
```

Могат да се използват следните методи за определяне състоянието на потока:

- **int eof()** - връща стойност, различна от нула (true), ако е прочетен край на файла;
- **int bad()** - връща true, ако някоя операция е неуспешна. Не е възможно поправяне на грешката;
- **int fail()** - връща true, ако някоя операция е неуспешна. Възможно е поправяне на грешката. В/И поток е използваем и по-нататък, ако флагът за грешка се изчисти;
- **int good()** - връща true, ако няма установен флаг за грешка;
- **int rdstate()** - връща текущото състояние на потока;
- **void clear(int state=0)** - изчиства флаговете на състоянието. При използване на параметър установява флаговете на състоянието.

8. Управление формата на входно-изходния поток

Библиотеката **iostream** поддържа форматиран вход/изход. Форматирането на потока се управлява по два начина. Първият начин е чрез методи на библиотечните класове, които трябва да се активират чрез съответния

обект (например `cout.width(20);`) и след това трябва да се изпълни стандартната операция << или >>. Вторият начин позволява форматирането да се извършва чрез функции, които са операнди на стандартните операции << и >>. Тези функции се наричат ***манипулатори***.

```
cout << манипулатор1 << променлива<< манипулатор2 << променлива
```

Методите за форматиране и манипулаторите в много случаи имат едно и също предназначение. Манипулаторите са дефинирани в заглавните файлове **<iostream>** и **<iomanip>**. Управлението на формата е организирано по следния начин. Всеки обект-поток (cin, cout, и т.н.) има четири променливи за управление на формата:

- **long flags** – съдържа флаговете за формат на потока;
- **int precision** – определя точността при извеждане на реални стойности;
- **int width** – определя ширината на полето за извежданата/въвежданата стойност;
- **int fill** – показва незначещите символи.

Променливата flags съдържа флаговете, съгласно които се формира входно-изходният поток. Стойностите на флаговете са следните:

Име	Стойност	Предназначение
ios::skipws	0x0001	Пропускане на празните символи при въвеждане (по подразбиране)
ios::left	0x0002	Изравняване на изхода отляво
ios::right	0x0004	Изравняване на изхода отдясно (по подразбиране)
ios::internal	0x0008	Показване на незначещите символи след знака
ios::dec	0x0010	Показване на стойностите в десетичен формат
ios::oct	0x0020	Показване на стойностите в осмичен формат
ios::hex	0x0040	Показване на стойностите в шестнайсетичен формат
ios::showbase	0x0080	Показване на използваната бройна система
ios::showpoint	0x0100	Показване на реалните числа с десетична точка
ios::uppercase	0x0200	Използване на главни букви в шестнайсетичното представяне на стойности и за експонентата E
ios::showpos	0x0400	Показване на положителните стойности със знак +
ios::scientific	0x0800	Представяне на стойностите в експоненциална форма
ios::fixed	0x1000	Представяне във формат с плаваща запетая
ios::unitbuf	0x2000	Изчистване на буферите след извеждане
ios::stdio	0x4000	Изчистване на stdout и stderr след извеждане

Флаговете са обединени съгласно предназначението им в следните именовани полета:

```
ios::adjustfield      -    ios::left, ios::right, ios::internal
ios::basefield        -    ios::dec, ios::oct, ios::hex
ios::floatfield       -    ios::scientific, ios::fixed
```

За четене, установяване и изчистване на флаговете за формат са използват следните функции:

- **long flags();**- връща текущата стойност на променливата flags, която съдържа флаговете за формата;
- **long flags(long);** - установява променливата flags (т.е. флаговете) и връща предишната ѝ стойност.

```
long u = cout.flags( ios::oct | ios::left );
```

- **`long setf(long bitFlags);`** - установява флаговете като резултат от побитово ИЛИ на параметъра и текущото състояние, и връща предишното състояние на флаговете.
- **`long setf(long bitFlags, long bitField);`** - bitField е едно от имената на полетата с флаговете. Функцията изчиства полето bitField. Към получената стойност на flags и първия параметър се прилага операцията побитово ИЛИ. Полученият резултат е новото състояние на флаговете за формат. Функцията връща предишното състояние на флаговете за формат.

```
// Избор на шестнайсетичен формат
setf ( ios::hex, ios::basefield);

// Избор на шестнайсетичен формат с показване на бройната система
setf ( ios::hex | ios::showbase, ios::basefield);
```

- **`long unsetf( long );`** - нулира определените от параметъра флагове и връща предишното състояние на флаговете.

За четене и промяна на променливата **`width`** се използват два метода:

- **`int width();`** - връща стойността на променливата **width**;
- **`int width( int );`** - установява минималното поле и връща предишната стойност.

Полето `width` се използва с всяка операция за въвеждане >> и извеждане <<. При въвеждане то определя броя на символите, които ще се въвеждат. При извеждане, ако цялото поле не е необходимо, оставащата част от него се запълва с незначещи символи, определени с функция **`fill()`**. Ако полето е по-късо, за да се изведе съответната числова стойност, се използва такова поле, каквото е необходимо.

След изпълнение на операциите променливата **`width`** се нулира. Ако операция за въвеждане се изпълнява при нулева стойност на полето, от входния поток се извличат толкова символа, колкото са необходими за представяне на въвежданата величина. При извеждане с нулево поле се използва полето, необходимо за извеждане на съответния елемент от данните.

За управление на променливата **`precision`** се използват два метода:

- **`int precision();`** - връща действащия в момента брой значещи цифри при извеждане на реални числа. По подразбиране се използват 6 цифри;
- **`int precision(int);`** - установява чрез параметъра нов брой на значещите цифри и връща стария брой;

За управление на променливата **`fill`** има два метода:

- **`char fill();`** - връща символа, с който се запълват незначещите позиции при извеждане на стойности;
- **`char fill(char);`** - установява чрез параметъра нов символ за запълване на незначещите позиции. Връща стария символ. Подразбиращият се символ за запълване е интервалът.

Когато стойността се изравнява отдясно, полето, в което тя се показва, се запълва отляво, а когато изравняването е отляво, запълването е отдясно.

```
cout.fill('@');
cout.setf(ios::internal, ios::adjustfield);
cout.setf(ios::uppercase);
cout.width(15);
int k=10;
cout << hex << k <<endl;
cout.setf(ios::showpos);
cout.width(15);
cout << 28.354e6 << '\n';
```

Много от разгледаните дотук методи за управление на формата имат аналог като манипулатори.

Основни манипулатори за управление на формата

- **dec** Установява десетична бройна система;
- **oct** Установява осмична бройна система;
- **hex** Установява шестнайсетична бройна система;
- **ws** Пропуска празните символи при въвеждане с >>;
- **endl** Вмъква управляващия символ за нов ред в изходния поток и изчиства буфера;
- **ends** Прибавя '\0' за край на низ;
- **flush** Изчиства буфера за извеждане;
- **setfill(char)** Аналог на fill();
- **setiosflags(long)** Аналог на flags();
- **setprecision(int)** Аналог на precision();
- **setw(int)** Предотвратява препълването при въвеждане в символен масив, разделяйки входния низ на порции с размер, определен от параметъра.

Манипулаторите без параметър и с параметър се използват аналогично;

```
int k=10;
cout << dec << k << oct << k << hex << k << dec << endl;
double f = 3.145689156;
cout.fill('@');
cout << setw(10) << setprecision(3) << f << endl;
```

Подразбиращи се стойности за управление на формата

Тип	*Подразбиращи се параметри*
`char`	Извежда се като символ в поле с дължина 1
`int`	Извежда се в поле, необходимо за стойността. Отрицателните стойности се предхождат от знак -
`float` `double`	Извеждат се в поле, необходимо за числовата стойност. Извеждат се във формат с плаваща точка, когато експонентата е по-голяма от –4 и

	по-малка от 6. В противен случай се представят в експоненциална форма. Показват се 6 значещи цифри след десетичната точка.
`char *`	Извежда се в поле, необходимо за низа
`&`	Извежда се в шестнайсетичен формат с означение 0x в началото. Незначещите 0 пред числата и след десетичната точка не се показват

Задача 2:

Да се състави програма, която да извежда названията на три цвята на екрана, форматирани по различен начин. Да се използват манипулатори за ограничаване полето на въвеждане на текст, с което се избягват грешките, свързани с въвеждане на по-голям от предвидения брой символи.

Решението на тази задача е представено във файла `H:\PROBLEMS\EX2_2e.CPP`.

Да се създаде проект с име `EX22`.

Файлът `EX2_2e.CPP` да се добави към създадения проект (виж "Добавяне на .cpp файл към съществуващ проект")

Проектът да се компилира и изпълни.

Да се анализират получените резултати.

Задача 3:

Да се състави програма, която да въвежда координатите на точка от равнината - реални числа, и да изчислява разстоянието от точката до началото на координатната система. Данните и полученият резултат да се изведат на екрана в различен формат, като се използват манипулатори.

Решението на тази задача е представено във файла `H:\PROBLEMS\EX2_3e.CPP`.

Да се създаде проект с име `EX23`.

Файлът `EX2_3e.CPP` да се добави към създадения проект (виж "Добавяне на .cpp файл към съществуващ проект")

Проектът да се компилира и изпълни.

Да се анализират получените резултати.

9. Стандартни математически функции

Всички математически функции са декларирани в заглавния файл **<cmath>**.

`double ceil (double x)` - закръглява стойността на аргумента до най-близкото по-голямо цяло число;

`double floor (double x)` - закръглява стойността на аргумента до най-близкото по-малко цяло число;

`int abs (int x)`

`double fabs (double x)` - връща абсолютната стойност на x;

double fmod (double x, double y) - връща остатъка от делението x/y като число с плаваща запетая със същия знак, като този на x;

double pow (double x, double y) - връща стойността на x повдигната на степен y. Ако стойността на x е по-малка или равна на 0, y трябва да бъде цяло число;

double sqrt (double x) - връща квадратния корен на x. Стойността на x трябва да бъде по-голяма или равна на 0;

double cos (double x) - връща косинуса на ъгъл x, изразен в радиани;

double sin (double x) - връща синуса на ъгъл x, изразен в радиани;

double tan (double x) - връща тангенса на ъгъл x, изразен в радиани;

double exp (double x) - връща стойността на e повдигната на степен x;

double log (double x) - връща натуралния логаритъм на x. Стойността на x трябва да бъде по-голяма от 0;

double log10 (double x) - връща десетичния логаритъм на x. Стойността на x трябва да бъде по-голяма от 0.

10. Стандартни функции за работа със символни низове

Всички функции за работа със символни низове са декларирани в заглавния файл **<cstring>**.

typedef unsigned size_t - тип, използван за размери на обект в паметта и броячи.

size_t strlen (char *str) - изчислява дължината на символен низ, без да се смята нулевия символ.

char *strcat (char *destin, char *source) - добавя копие на символния низ `source` в края на `destin`. Дължината на получения символен низ е `strlen(destin) + strlen(source)`.

int strcmp (char *str1, char *str2) - сравнява `str1` със `str2`. Всички функции за сравнение връщат стойност (<0 ако `str1<str2`; ==0, ако `str1=str2` или >0, ако `str1>str2`) в зависимост от резултата от сравнението. Сравнението е лексикографско.

int stricmp (char *str1, char *str2) - сравнява `str1` със `str2` без да отчита разликата главни - малки букви от латинската азбука.

char *strcpy (char *destin, char *source) - копира символния низ `source` в `destin`. В края се прибавя нулевия символ. Връща destin.

char *strdup (char *str) - копира символен низ в ново отделено за него място. Връща указател към дублирания `str`.

char *strchr (char *str, int ch) - анализира символния низ `str` в права посока, като търси първото появяване на символа `ch` в него. Връща указател към първото появяване на `ch` в `str`. Ако `ch` не е намерен, върнатата стойност е `NULL`. Нулевият символ се счита за част от символния низ, така че `strchr(strs,0)` връща указател към завършващия нулев символ на `strs`.

char *strstr (char *str1, char *str2) - търси в `str1` първото появяване на подмножеството `str2`. Връща указател към елемента от `str1`, съдържащ `str2` (сочи мястото на `str2` в `str1`). Ако `str2` не се среща в `str1`, върнатата стойност е `NULL`.

char *strtok (char *str1, const char *str2) - сканира `str1` за първия знак, който не се съдържа в `str2`. Функцията `strtok` разглежда `str1` като последователност от думи, разделени от символите, изброени в `str2`. Първото извикване на `strtok` връща указател към първия символ от първата дума в `str1` и записва `NULL` в `str1` непосредствено след върнатата дума. Всяко следващо извикване на функцията `strtok` с първи аргумент `NULL` ще продължи да търси в `str1` следващата дума.

Задача 4:

Да се състави програма за въвеждане на символни низове, всеки на отделен ред, за сортиране и извеждането им на екрана. За сортировката да се използва масив от указатели към основния масив от символните низове.

Решението на тази задача е представено във файла `H:\PROBLEMS\EX2_4e.CPP`.

Да се създаде проект с име `EX24`.

Файлът `EX2_4e.CPP` да се добави към създадения проект (виж "Добавяне на .cpp файл към съществуващ проект")

Проектът да се компилира и изпълни.

Да се анализират получените резултати.

Задача 5:

Да се състави програма, която чете текст от думи и печата броя на еднобуквените, двубуквените и т.н. думи. За сканиране на низа да се използва функцията **`strtok()`**.

Решението на тази задача е представено във файла `H:\PROBLEMS\EX2_5e.CPP`.

Да се създаде проект с име `EX25`.

Файлът `EX2_5e.CPP` да се добави към създадения проект (виж "Добавяне на .cpp файл към съществуващ проект")

Проектът да се компилира и изпълни.

Да се анализират получените резултати.

Задача 6:

Да се състави програма, която да съхранява пароли в масив от символи с име pass. Чрез функцията `strcmp()`, да се провери за правилно въведена парола. Да се изведе съответно съобщение.