ХИМИКОТЕХНОЛОГИЧЕН И МЕТАЛУРГИЧЕН УНИВЕРСИТЕТ – СОФИЯ

ИНФОРМАТИКА

част първа

Основи на програмирането на C++

лектор: гл. ас. д-р Стефан М. Панов

Катедра “Информатика”

Лекция 3

Основни елементи на езика C++

Ключови думи в C++

В стандарта C++ (версия C++98) са определени 63 ключови думи, показани на таблица 3.1.

Ключовите думи съвместно със синтаксиса на операторите и разделителите образуват определението (дефиницията) на езика C++.

N.B. Длъжни сме да запомним, че в C++ се различава написаното с главни и малки букви. Ключовите думи не правят изключение, т.е. те всички трябва да бъдат писани с малки букви.

Например, думата RETURN (или Return) няма да бъде разпозната в качеството на ключовата дума return.

asm	auto	bool	break
case	catch	char	class
const	const__class	continue	default
delete	do	double	dinamic_cast
else	enum	explicit	export
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespace
new	operator	private	protected
public	register	reinterpret_cast	return
short	signed	sizeof	static
static_cast	struct	switch	template
this	throw	true	try
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	wchar_t	while	

Таблица 3.1 – Ключовите думи в C++

Идентификатори в C++

В C++ идентификаторът представлява име, което се присвоява на функция, променлива или друг елемент, определен от потребителя. Идентификаторите се състоят от един или повече символа. **Имената на променливите трябва да започват с буква или символа за подчертаване „_“. Следващият символ може да бъде буква, цифра или символа за подчертаване**.

Символът за подчертаване се използва за по-добра читаемост на имената. Можем да разграничим два основни стила - например first_sum или firstSum. Отново: В C++ главните и малки букви се възприемат като различни символи, т.е. **myvar** и **MyVar** са две различни имена.

Ето няколко примера за допустими идентификатори:

Pyrvi first a Abbccdd MaxLoad

name67 topdown my_var matrica_3_na_3

В C++ не могат да се използват ключови думи и имена на стандартни функции (примерно функцията за втори корен sqrt) в качеството си на идентификатори. Разбира се, всеки е свободен да назовава променливите и другите програмни елементи по свое усмотрение, но се препоръчва идентификаторът да отразява смисловата характеристика на елемента, към който той принадлежи.

Запомнете, че идентификаторът не трябва да започва с цифра. Например: **97_year** е недопустим идентификатор.

Стандартната C++ библиотека

В примера по-нагоре, се спомена функцията **sqrt()**. По същество тя не се явява част от езика C++, но всеки C++ компилатор я „познава“. Тая функция, както и множество други, влиза в състава на стандартната библиотека. В примерите на лекции и упражнения ние ще разгледаме за какво служат и как се използват немалко C++ библиотечни функции.

Стандартната C++ библиотека съдържа множество вградени функции, които програмистите могат да използват в своите програми.

В C++ е определен доста голям набор от функции, които се съдържат в стандартната библиотека. Те са предназначени за изпълнение на

често срещащи се задачи, особено следните 3 групи: входно-изходни операции, математични изчисления и обработка на низове.

При използване от програмиста на една библиотечна функция компилаторът автоматично свързва обектния код на тази функция с обектния код на програмата.

Библиотечните функции може да оприличим на строителните блокове, от които се издига една сграда. Ако ние сами напишем функция, която ще използваме в доста програми, също може да я поместим в библиотека.

Заб. Отделно от стандартната, **всеки** C++-компилатор също така съдържа друга **обектно-ориентирана библиотека с класове** и трета **стандартна библиотека с шаблони** (Standard Template Library— STL). **В нашия курс ние няма да ги разглеждаме**.

Първо запознаване с променливите

Една от най-важните конструкции във всеки език за програмиране е присвояване на стойност на променлива.

Променлива —именована област от паметта, в която могат да се съхраняват различни стойности. Самият термин "променлива" подсказва, че стойността, която се съхранява в нея може да се променя **по време на изпълнение на програмата**.

В следващата програма се създава променлива с име **x**, на която се присвоява стойност 2019, след което на екрана се извежда съобщение и накрая се извежда стойността на **x**.

```
// Programa №2 – Izpolzvane na promenliva.
#include <iostream>
using namespace std;
int main()
{
	int x; // obiaviavane na promenliva.
	x = 2019; // Na x se prisvoiava chisloto 2019
	cout << "Taia programa izvejda stojnostta na promenlivata x: ";
	cout << x; // Izvejdane na chisloto 2019.
	return 0;
}
```

Кое е новото в нашата втора програма? Първо, инструкцията

```
int x; // obiaviavane na promenliva.
```

обявява променлива с име x от целочислен тип. **В C++ всички променливи трябва да бъдат обявени преди тяхното използване**. При обявяването е необходимо да се укаже не само името, но и стойности от какъв тип ще може да съхранява променливата, т.е. да се обяви нейният тип.

Второ, при изпълнение на следващата инструкция на **x** се присвоява конкретна стойност:

```
x = 2019; // Na x se prisvoiava chisloto 2019
```

В C++ операторът за присвояване е знакът за равенство (=). Неговото действие се състои в копиране на стойността, разположена отдясно на оператора в променливата намираща се вляво от него. Т.е. след изпълнение на нашата инструкция променливата **x** ще съдържа числото 2019.

Третата нова за нас инструкция извежда стойността на променливата **x**:

```
cout << x; // Izvejdane na chisloto 2019.
```

В общия случай за извеждане на стойността на една променлива е достатъчно в инструкцията cout да поставим нейното име вдясно от оператора <<. (**т.е. не ползваме кавички, както е при низове**.) Понеже в нашия случай **x** съдържа числото 2019, именно то ще се види на екрана.

Основни типове данни

В C++ са дефинирани седем основни типа данни: **символен, символен двубайтов, целочислен, с плаваща точка, с плаваща точка с двойна точност, логически (наречен още булев) и "нямащ стойност"**. За обявяване на променливи от тези типове съответно се използват **ключовите думи char, wchar_t, int, float, double, bool и void**. **Типичните** размери в битове и обхватът на числата на всеки от седемте типа е представен на таблица 3.2.

Тип	Размер в битове	Обхват
char	8	-128÷127 или 0÷255 (2^8-1)
wchar_t	16	0 ÷ 65535 (2^{16}-1)
int	32	-2^{31} ÷ 2^{31}-1
float	32	1.2E-38 ÷ 3.4E+38
double	64	2.2E-308 ÷ 1.8E+308
bool	8	true или false
void	-	без стойност

Таблица 3.2 - Основните типове данни в C++.

Запомнете, че размерът и обхватът при някои компилатори могат да се различават от показаните в таблицата тук.

Променливите от типа **char** се използват за съхранение на 8-битови ASCII-символи (например буквата D или цифрата (т.е. символа) 4 или за произволни други 8-битови стойности). **За да зададем даден символ трябва да го поставим в единични кавички**.

За езици като китайския където има много голям брой символи (йероглифи) 8 битовото им представяне с типа **char** не е възможно. Решението е добавяне на друг тип **wchar_t** където двата байта са достатъчни.

Променливите от типа **int** както вече видяхме позволяват да се съхраняват целочислени стойности (не съдържащи дробна компонента).

Променливите от типа **float** и **double** се използват или за обработка на числа с дробна част или при необходимост от изпълнение на операции над много големи или много малки числа. Типовете **float** и **double** силно се различават по стойността на най-голямото и най-малкото число, които могат да се представят.

Типът **bool** е предназначен за съхраняване на логически (булеви) стойности т.е. ИСТИНА/ЛЪЖА. В C++ са определени две булеви константи: **true** и **false**, представляващи единствените стойности, които могат да имат променливи от типа bool.

Типът **void**, както ще се запознаем по-късно, се използва за обявяване на функции, които не връщат никаква стойност.

Второ запознаване с променливите

Общият формат на инструкцията за обявяване на променливи изглежда така:

тип списък_от_променливи;

Тук **тип** означава всеки допустим в C++ тип данни, а **списък_от_променливи** може да се състои от един или няколко имена (идентификатори), разделени със запетаи. Ето няколко примера за обявяване на променливи:

```
int  i, j, k, m;  char  ch, chr; // две инструкции на един ред

float  f, prom1;

double  d1; // обявяването може да запишем и на два или повече реда
```

Съгласно C++ стандарта първите 1024 символа на едно име (вкл. и име на променлива) са значими. Това означава, че ако две имена се различават поне с един символ от първите 1024 компилаторът ги разглежда като различни.

Препоръка: По-добре да обявяваме по една променлива на ред с коментар

Заб. Трябва да правим разлика между един ред от изходния код и една инструкция. Инструкцията може да заема повече от ред ред, но може да има и няколко инструкции на един ред.

В зависимост от мястото на обявяване променливите се наричат локални, глобални и формални параметри. Третите ще разгледаме подробно по-късно.

- **Локални променливи**

Променливи, които се обявяват вътре във функция, се наричат локални. **Те могат да се използват само в инструкции, принадлежащи към тялото на същата функция.** Локалните променливи са неизвестни за външните функции.

- **Глобални променливи**

Променливи, които се обявяват извън функциите, се наричат глобални. **Те могут да се използват във всички функции.** Единственото условие е всяка променлива да бъде обявена преди нейното използване. Ето защо е най-добре глобалните променливи да се обявяват в началото на програмата, но след **#include** редовете. Да разгледаме следния пример:

```
#include <iostream>  //nachalo na nashata chetvyrta programa
using namespace std; //tretata e izvyn lekciata
int fun1() ;  int fun2();
int count; // Tova e globalna promenliva.
int main() {
      int i=5; // Tova e lokalna promenliva. Inicializirana.
      count = 2; // Tova e globalnata promenliva count.
      cout << "stojnostta na izraza e ="<< 2*i*fun1();
      return 0;
}
int fun1() {
      cout << "count = " << count<< "  "; // Tova e globalnata promenliva count.
      return fun2()*count;
} //kolko shte e stoinostta na izraza v main(), ako vav fun2 imame  int i=2; count = 7;
int fun2() {
      int i=1+1, count = 2*i+3; // Tova sa 2 lokalni promenlivi. Inicializirani.
      return count*i;
}
```

В тая програма освен функцията **main()** са определени 2 други функции с избрани от нас имена **fun1()** и **fun2()**. Ключовата дума **int** (съкращение от **integer**), стояща пред името на двете означава, че тези функции ще връщат (посредством **return**) стойности от типа **int**.

Локалната променлива **i** във **main()** няма нищо общо с **i** от **fun2().** Независимо, че променливата **count** не е обявена нито в **main()**, нито във **fun1()** и двете функции могат да я използват. Но във функцията **fun2()** е обявена локална променлива със същото име **count**. Тук при обръщането към **count** след **return** се осъществява достъп към локалната, а не към глобалната променлива. **Важно е да се знае, че ако глобална и локална променливи имат еднакви имена, всички обръщения към "спорното" име се отнасят към локалната вътре във функцията, където локалната променлива е обявена**.

Инициализация на променливите

При обявяване на една променлива може да ѝ се присвои някаква стойност, т.е. да я инициализираме. Общият формат на инициализацията е следния:

тип име_на_променлива = стойност; (или **израз** вместо **стойност**)

Ето няколко примера:

char ch = 'a'; // забележете единичните кавички при символа a

int first = 0; // инициализация на целочислена променлива с нула

float balance = 12.23; // иниц-я с реално число на пром. от тип float

Когато имаме обявени няколко променливи в една инструкция, както е в нашия пример във функцията **fun2(),** инициализацията също е възможна:

int i=2, count = 7;

Независимо, че променливите обикновено се инициализират с константи, C++ позволява и **динамична инициализация**, т.е. **с помощта на произволен израз**, стига той да е действителен в момента на инициализацията. Т.е. следният ред също би бил верен в нашата програма:

```
int i=1+1, count = 2*i + 3*1;
```

Глобалните променливи се инициализират само в началото програмата. Те **се инициализират по подразбиране с нулеви стойности**, ако не е направена явна инициализация. Локалните променливи се инициализират при всяко влизане във функцията, в която те са обявени. **Неинициализираната локална променлива ще има неизвестна стойност** до първата инструкция за присвояване, в която тя е използвана.

Инструкцията за присвояване има вида **име_на_променлива = израз;**

Т.е. инициализацията на променлива е съчетания от обявяване на променлива и инструкция за присвояване.

Но какво означава **влизане във функция**? **Всеки път, когато една функция се извика - в main() или от друга функция - се изпълняват инструкциите в нейното тяло (в нейния код).** Т.е. всяко извикване на функция води до влизане в нейното тяло за изпълнение на инструкциите ѝ.

Функциите се пишат, за да бъдат ползвани, а това става **когато една функция се извика**. **Те се извикват в израз** (що е **израз** ще се разгледа по-късно). В нашата четвърта програма по-нагоре е показано как се извикват функции, които нямат параметри - **име_на_функция()**. Не е задължително отварящата скоба да е залепена до името.

Редът от нашата програма **cout << "stojnostta na izraza e ="<< 2*i*fun1();**

също изисква подробно разглеждане. Вместо употребата на единична променлива, както е направено във тялото на fun1() за променливата **count**, тук в инструкцията за извеждане на данни е използван израз. В нашия случай изразът е **2*i*fun1().**

Втората особеност на реда е, че **операторът за извеждане „<<”** е използван повече от един път в една инструкция за извеждане! Първо ще бъде изведен символният низ, а после стойността на израза, който преди извеждането се изчислява. **В общия случай една инструкция за извеждане може да съдържа произволно количество оператори за извеждане „<<” като след всеки следва елементът, който искаме да изведем**. **Под елемент се разбира низ, израз или изходен манипулатор** (последното ще разгледаме по-късно).

Същото, което заявихме за инструкцията за извеждане е в сила и за инструкцията за връщане от функция **return** – **вместо единична променлива или константа след ключовата дума return можем да използваме израз**. Така е направено и в двете функции от нашия пример.

Прототип на функция

При извикване на една функция също важи правилото, че компилаторът трябва да „познава“ функцията преди нейното извикване. Но да си представим следната ситуация: имаме три функции – **fun1()**, **fun2()** и **fun3()** – където **fun1()** извиква **fun2()**, **fun2()** извиква **fun3()**, а **fun3()** вика **fun1()**. Излиза, че тялото на втората функция трябва да е преди тялото на първата, тялото на третата преди това на втората, а тялото на първата преди това на третата, което няма как да стане!

Решението е прототип. **Прототип на функция е обявяването на функцията преди нейното определения**.

Прототипът включва типа на стойността, връщана от функцията, името на функцията, а също броя и типа на формалните параметри, които тя може да има.

Компилаторът е длъжен да знае тая информация преди първото викане на дадена функция. **Затова прототипите на функциите се разполагат в началото на изходния файл, т.е. преди тялото (определението) на първата функция от програмата. Единствената функция, която не изисква прототип е main(), защото тя е вградена в езика C++.**

В нашата програма са обявени прототипи и за двете функции **fun1()** и **fun2()**. Направено е на един ред, защото синтаксисът на езика го позволява с цел да съберем програмата на една страница.

Заб. 1. Ако една функция не връща стойност типът преди нейното име (както в прототипа, така и при определението на функцията) се задава с ключовата дума void. В тоя случай след return няма стойност, т.е. винаги е return ;

Заб. 2. Посредством формалните параметри към функциите могат да се предават стойности. Формалните параметри ще бъдат разгледани по-късно. Нашите функции от третия пример нямат формални параметри .

Модификатори на основните типове

В C++ пред типовете данни **char**, **int** и **double** е разрешено да се използват **модификатори**. Модификаторът служи за изменение на стойността на базовия тип така, че той по-точно да съответства на конкретната ситуация. **Модификаторите на типове са 4 - signed, unsigned, long и short.**

И 4-те модификатори могат да се прилагат към целочисления тип **int**. За **int** всяко от **signed или unsigned може да се комбинира с long или short.** Освен това, модификаторите **signed** и **unsigned** могат да се използват с типа **char**, а модификаторът **long**— с типа **double**. Примери:

```
unsigned short int  x; //променливата x от типа unsigned short int
```

Вместо да се помнят наизуст типичните размери в битове и обхвати по-добре е да се запомни таблица 3.2, дадена по-горе и да си припомним от първата лекция, че най-голямото положително число, което може да се представи в

n бита е 2^n -1, а най-малкото отрицателно е -2^n. И понеже размерът на повечето основни типове в битове зависи пак от конкретния компилатор най-добре е да видим как е при нашия компилатор с помощта на един специален оператор на име **sizeof**. Използва се така: **sizeof(type)** където **type** е валиден тип и **връща размера на посочения тип в байтове**. Примерно **sizeof(short int)** в инструкцията за извеждане на данни ще върне 2.

Трябва да се подчертае, че **long и short** могат да променят размера на съответния използван тип, докато **signed или unsigned** указват как да се тълкува най-старшия бит в една променлива от модифицирания тип. **Пример_1**: **unsigned char ch**; указва, че най-старшият бит в променливата **ch** е информационен, т.е. всичките 8 бита се използват за запис на едно число. Оттук най-голямото число е 2^8 -1 = 255. **Пример_2**: **signed char ch**; указва, че най-старшият бит в променливата **ch** е знаков, т.е. 7 бита се използват за запис на едно число. Оттук най-малкото число е $-2^7 = -128$, а най-голямото $2^7-1=127$.

Почти за всички компилатори **signed char** и **char е едно и също**. Аналогично, **signed int** и **int е едно и също**, така че **signed** на практика много рядко се използва. Доколкото обхватът на числата в типът **double** в огромен, **long double** също се използва само в редки, специални случаи.

Следващата програма може да се използва за да видим размера в байтове (за използвания от нас компилатор) на всички интересни за практиката комбинации от типове и модификатори. **Новото в програмата е употребата на комбинацията \n в низове**. В C и C++ последователността **\n се възприема като един символ**, който води до преместване на курсора на нов ред в инструкцията за извеждане на данни.

Ще споменем по темата за модификаторите и това, че където се използва **short int** може да се използва само **short**. Аналогично, навсякъде където се използва **long int** може да се използва само **long**. Примерно **short a;** е валидна конструкция за обявяване на променливата **a**, която ще заема 2 байта.

```
#include <iostream>  // nashata peta programa
using namespace std;
// programata dava razmer v baitove na osnovnite tipove s i bez modifikatori
int main() {
	cout << "Za nashia kompilator:\n\n";
	cout << "tipyt short int e "<< sizeof(short int)<<" baita"<< "\n";
	cout << "tipyt long int e "<< sizeof(long int)<<" baita"<< "\n";
cout << "tipyt long long e "<< sizeof(long long)<<" baita"<< "\n";// izvyn standarta!
	cout << "tipyt  int e "<< sizeof(int)<<" baita"<< "\n\n";
	cout << "tipyt  char e "<< sizeof(char)<<" bait"<< "\n";
	cout << "tipyt  wchar_t e "<< sizeof(wchar_t)<<" baita"<< "\n\n";
	cout << "tipyt  bool e "<< sizeof(bool)<<" bait"<< "\n\n";
	cout << "tipyt  float e "<< sizeof(float)<<" baita"<< "\n";
	cout << "tipyt  double e "<< sizeof(double)<<" baita"<< "\n";
	cout << "tipyt  long double e "<< sizeof(long double)<<" baita"<< "\n";
	return 0;
}
```

Променливите от типа **char** могат да се използват не само за съхранение на ASCII- символи, **но и за съхранения на числови стойности**. Ако знаем, че използваните от нас числа ще са само в обхвата -128÷127 може да използваме променлива от типа **char** вместо от типа **int.** Аналогично, ако използваните от нас числа ще са само в обхвата 0÷255 може да използваме променлива от типа **unsigned char** вместо от типа **unsigned int.**

За да се схване различието в C++ тълкуването на целочислени стойности с и без знак, ще се разгледа следващата програма. При нейното изпълнение на екрана се извеждат две числа: **-5536 60000**

Тънкостта е в това, че битовата комбинация (1110101001100000_2), която представя числото 60000 като късо целочислено без знак, се тълкува в променливата **i** като късо целочислено със знак. Най-старшият (най-левия) бит е 1, значи числото е **отрицателно в допълнителен код**. Оттам лесно се получава съответното положително $0001010110100000_2 = 5536$.

```
#include <iostream> // nashata 6-a programa
using namespace std;
// Taia programa pokazwa razlichieto mejdu
// signed- i unsigned-stoinostite ot  celochislen tip.
int main()
{
	short int i; // kysa int promenliva sys znak
	unsigned short int k; // kysa int promenliva bez znak
	k = 60000;
	i = k ;
	cout << i << " " << k;
	return 0;
}
```