Глава 3

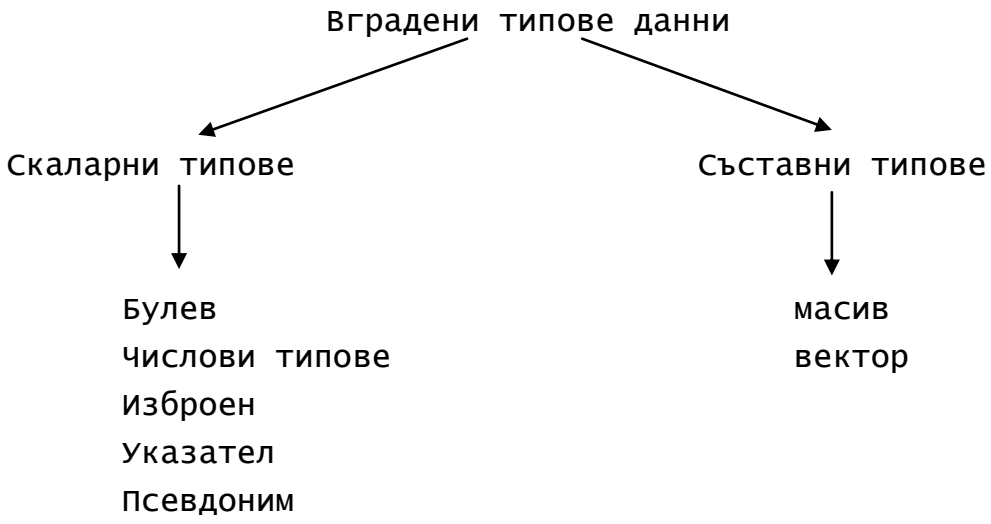
Скаларни типове данни

Езикът C++ е изключително мощен по отношение на типовете данни, които притежава. Най-общо, типовете му могат да бъдат разделени на: **вградени** и **абстрактни**.

Вградените типове са предварително дефинирани и се поддържат от неговото ядро.

Абстрактните типове се дефинират от програмиста. За целта се определят съответни класове.

Една непълна класификация на вградените типове данни е дадена на Фиг. 1.



Фиг. 1.

Скаларни са типовете данни, които се състоят от една компонента (число, знак и др.).

Съставни типове са онези типове данни, компонентите на които са редици от елементи.

Типът указател дава средства за динамично разпределение на паметта.

В тази глава ще разгледаме само някои скаларни типове данни.

Всеки тип се определя с **множество от допустими стойности** (множество от стойности) и **операции и вградени функции**, които могат да се прилагат над елементите от множеството от стойностите му.

3.1. Логически тип

Нарича се още булев тип в чест на Дж. Бул, английски логик, поставил основите на математическата логика.

Типът е стандартен, вграден в реализацията. За означаването му се използва запазената дума bool (съкращение от boolean).

Множество от стойности

Състои се от два елемента – стойностите true (истина) и false (лъжа). Тези стойности се наричат още **булеви константи**.

<булева_константа> ::= true | false.

Променлива величина, множеството от допустимите стойности, на която съвпада с множеството от стойности на типа булев, се нарича **булева** или **логическа променлива** или **променлива от тип булев**. Дефинира се по обичайния начин.

Примери:

```
bool b1, b2;
bool b3 = false;
```

Дефиницията свързва булевите променливи с множеството от стойности на типа булев или с конкретна стойност от това множество като отделя по 1 байт оперативна памет за всяка от тях. Съдържанието на

тази памет е неопределено или е булевата константа, свързана с дефинираната прменлива, в случай, че тя е инициализирана.

След дефиницията от примера по-горе, имаме:

ОП

```
b1       b2       b3
-        -        false
 1 байт   1 байт    1 байт
```

Съдържанието на паметта, именувана с b1 и b2 е неопределено, а това на именуваната с b3 е false. В същност, вместо false в паметта е записан кодът (вътрешното представяне) на false - 0.

Вътрешните представяния на булевите константи са:

```
false    0
true     1
```

Операции и вградени функции

Логически операции

Конюнкция (логическо умножение)

Тя е двуаргументна (бинарна) операция. Означава се с and или && (за Visual C++, 6.0) и се дефинира по следния начин:

A	B	A and B
false	false	false
false	rue	false
true	false	false
true	true	true

Операцията се поставя между двата си аргумента. Такива операции се наричат **инфиксни.**

Дизюнкция (логическо събиране)

Тя е бинарна, инфиксна операция. Означава се с or или || (за Visual C++, 6.0) и се дефинира по следния начин:

A	B	A or B
false	false	false
false	true	true
true	false	true
true	true	true

Логическо отрицание

Тя е едноаргументна (унарна) операция. Означава се с not или ! (за Visual C++, 6.0) и се дефинира по следния начин:

A	not A
false	true
true	false

Поставя се пред единствения си аргумент. Такива оператори се наричат **префиксни**.

Забележка: Може да няма разделител между оператора ! и константите true и false, т.е. записите !true и !false са допустими.

Допълнение: Смисълът на операторите and, or и not е разширен чрез разширяване смисъла на булевите константи. Прието е, че true е всяка стойност, различна от 0 и че false е стойността 0.

Операции за сравнение

Над булевите данни могат да се извършват следните инфиксни операции за сравнение:

== - за равно

!= - за различно

```
>     - за по-голямо
>=    - за по-голямо или равно
<     - за по-малко
<=    - за по-малко или равно
```

Сравняват се кодовете.

Примери:

```
false < true   е true
false > false  е false
true >= false  е true
```

Въвеждане

Не е възможно въвеждане на стойност на булева променлива чрез оператора >>, т.е. операторът

```
cin >> b1;
```

е недопустим, където b1 е булевата променлива, дефинирана по-горе.

Извеждане

Осъществява се чрез оператора

```
cout << <булева_константа>;
```

или по-общо

```
cout << <булев_израз>;
```

където синтактичната категория <булев_израз> е определена по-долу.

Извежда се кодът на булевата константа или кодът на булевата константа, която е стойност на <булев_израз>.

Булеви изрази

Булевите изрази са правила за получаване на булева стойност. Дефинират се *рекурсивно* по следния начин:

- Булевите константи са булеви изрази.
- Булевите променливи са булеви изрази.
- Прилагането на булевите оператори not (!), and (&&), or (||) над булеви изрази е булев израз.

- Прилагането на операциите за сравнение ==, !=, >, >=, <, <= към булеви изрази е булев израз.

Примери: Нека имаме дефиницията

```
bool b, b1, b2;
```

Следните изрази са булеви:

```
true     b     b1       b2       !false      !!b      !b1 || b2
!!!b && !!!!!b2      b < !b2  false >= b  b1 == b2 > b   b != b1
```

Тази дефиниция е непълна. Ще отбележим само, че сравнението на аритметични изрази чрез изброените по-горе операции за сравнение, е булев израз. Освен това, аритметичен израз, поставен на място, където синтаксисът изисква булев израз, изпълнява ролята на булев израз. Това е резултат от разширяването смисъла на булевите константи true и false, чрез приемането всяка стойност, различна от 0 да се интерпретира като true и 0 да се интерпретира като false.

Засега отлагаме разглеждането на семантиката на булевите изрази. Това ще направим след разглеждане на аритметичните изрази.

3.2. Числени типове

3.2.1. Целочислени типове

Ще разгледаме целочисления тип int.

Типът е стандартен, вграден в реализацията на езика. За означаването му се използва запазената дума int.

Множество от стойности

Множеството от стойности на типа int зависи от хардуера и реализацията и не се дефинира от ANSI (American National Standarts Institute). Състои се от целите числа от някакъв интервал. За реализацията Visual C++ 6.0, това е интервалът [-2147483648, 2147483647].

Целите числа се записват като в математиката, т.е.

<цяло_число> ::= [+|-]<цяло_без_знак>
<цяло_без_знак> ::= <цифра>|
<цифра><цяло_без_знак>
<цифра> ::= 0| 1| … |9.
Обикновено знакът + се пропуска.

Допълнение: Целите числа могат да са в десетична, осмична и шестнадесетична позиционна система. Осмичните числа започват с 0 (нула), а шестнадесетичните - с 0x (нула, x).

Елементите от множеството от стойности на типа int се наричат **константи от тип int**.

Променлива величина, множеството от допустимите стойности, на която съвпада с множеството от стойности на типа int, се **нарича цяла променлива** или **променлива от тип int**.

Дефинира се по обичайния начин. Дефиницията свързва променливите от тип int с множеството от стойности на типа int или с конкретна стойност от това множество като отделя по 4 байта (1 дума) оперативна памет за всяка от тях. Ако променливата не е била инициализирана, съдържанието на свързната с нея памет е неопределено, а в противен случай – съдържа указаната при инициализацията стойност.

Примери:

```
int i;
int j = 56;
```

След тези дефиниции, имаме:

```
ОП
i           j
-           56
4 байта   4 байта
```

Операции и вградени фунции

Аритметични операции

Унарни операции

Записват се пред или след единствения си аргумент.

+, - са префиксни операции. Потвърждават или променят знака на аргумента си.

Примери: Нека

```
int i = 12, j = -7;
```

Следните означения съдържат унарна операция + или -:

```
-i    +j    -j    +i    -567
```

Бинарни операции

Имат два аргумента. Следните аритметични операции са инфиксни:

```
+     - събиране
-     - изваждане
*     - умножение
/     - целочислено деление
%     - остатък от целочислено деление.
```

Примери:

```
15 - 1235 = -1220    13 / 5 = 2
15 + 1235 =  1250    13 % 5 = 3
-15 * 123 = -1845    23 % 3 = 2
```

Забележка: Допустимо е използването на два знака за аритметични операции, но единият трябва да е унарен. Например, допустими са 5-+4, 5+-4, имащи стойност 1, а също 5*-4, равно на –20.

Логически операции

Логическите операции конюнкция, дизюнкция и отрицание могат да се прилагат над целочислени константи. Дефинират се по същия начин, като целите числа, които са различни от 0 се интерпретират true, а 0 – като false.

Примери:

```
123 and 0   е false
0 or 15     е true
```

not 67 e false

Операции за сравнение

Над целочислени константи могат да се извършват следните инфиксни операции за сравнение:

==	- за равно	!=	- за различно
>	- за по-голямо	>=	- за по-голямо или равно
<	- за по-малко	<=	- за по-малко или равно.

Наредбата на целите числа е като в математиката.

Примери:

123 < 234 e true
-123456 > 324 e false
23451 >= 0 e true

Вградени функции

В езика C++ има набор от вградени функции. Обръщението към такива функции има следния синтаксис:

<име_на_функция>(<израз>, <израз>, …, <израз>)

и връща стойност от типа на функцията.

Тук ще разгледаме само едноаргументната целочислена функция abs.

abs(x) – намира |x|, където x е цял израз
(в частност цяла константа).

Примери:

abs(-1587) = 1587 abs(0) = 0 abs(23) = 23

За използването на тази функция е необходимо в частта на заглавните файлове да се включи директивата:

#include <math.h>

Библиотеката math.h съдържа богат набор от функции. В някой реализации тя има име math или cmath.

Въвеждане

Реализира се по стандартния и разгледан вече начин.

Пример: Ако

```
int i, j;
```

операторът

```
cin >> i >> j;
```

въвежда стойности на целите променливи i и j. Очаква се въвеждане от стандартния входен поток на две цели константи от тип int, разделени с интервал, знаците за хоризонтална или вертикална табулация или знака за преминаване на нов ред.

Извеждане

Реализира се чрез оператора

```
cout << <цяла_константа>;
```

или по-общо

```
cout << <цял_израз>;
```

В текущата позиция на курсора се извежда константата или стойността на целия израз. Използва се минималното количество позиции, необходими за записване на цялото число.

Пример: Нека имаме дефиницията

```
int i = 1234, j = 9876;
```

Операторът

```
cout << i << j << "\n";
```

извежда върху екрана стойностите на i и j, но слепени

12349876

Този изход не е ясен. Налага се да се извърши **форматиране** на изхода. То се осъществява чрез подходящи **манипулатори**.

Манипулатор setw

Setw е вградена функция.

Синтаксис

setw(<цял_израз>)

Семантика

Стойността на <цял_израз> задава широчината на полето на **следващия** изход.

Пример: Операторът

```
cout << setw(10);
```

не извежда нищо. Той “манипулира” следващото извеждане като указва, че в поле с широчина 10 отдясно приравнена, ще бъде записана следващата извеждана цяла константа.

Забележка: Този манипулатор важи само за първото след него извеждане.

Пример: Нека

```
ОП
i          j
1234     9876
```

Операторът

```
cout << setw(10) << i << j << “\n”;
```

извежда отново стойностите на i и j слепени, като 1234 се предшества от 6 интервала, т.е.

******12349876

където интервалът е означен със знака *.

Операторът

```
cout << setw(10) << i << setw(10) << j << “\n”;
```

извежда

******1234******9876

Манипулатори dec, oct и hex

Целите числа се извеждат в десетична позиционна система. Ако се налага изходът им да е в осмична или шестнадесетична позиционна система, се използват манипулаторите oct и hex съответно. Всеки от тях е в сила, докато друг манипулатор за позиционна система не е указан за следващ извод. Връщането към десетична позиционна система се осъществява чрез манипулатора dec.

dec – манипулатор, задаващ всички следващи изходи (докато не е указан друг манипулатор, променящ позиционната система) на цели числа да са в десетична позиционна система;

oct – манипулатор, задаващ всички следващи изходи (докато не е указан друг манипулатор, променящ позиционната система) на цели числа да са в осмиична позиционна система;

hex - манипулатор, задаващ всички следващи изходи (докато не е указан друг манипулатор, променящ позиционната система) на цели числа да са в шестнадесетична позиционна система.

Пример: Нека имаме

ОП

```
i     j
12    23
```

След изпълнението на операторите

```
cout << setw(10) << dec << i << setw(10) << j << "\n";
cout << setw(10) << oct << i << setw(10) << j << "\n";
cout << setw(10) << hex << i << setw(10) << j << "\n";
```

имаме:

```
********12********23
********14********27
********c********17
```

Забележка: Преди използване на манипулаторите е необходимо да се включи заглавният файл iomanip.h, т.е. в частта за заглавни файлове да се запише директивата

```
#include <iomanip.h>
```

Други целочислени типове

Други цели типове се получават от int като се използват модификаторите short, long, signed и unsigned. Тези модификатори доопределят някои аспекти на типа int.

За реализацията Visual C++ 6.0 са в сила:

Тип	Диапазон	Необходима памет
short int	-32768 до 32767	2 байта
unsigned short int	0 до 65535	2 байта
long int	-2147483648 до 2147483647	4 байта

unsigned long int	0 до 4294967295	4 байта
unsigned int	0 до 4294967295	4 байта

Запазената дума int при тези типове се подразбира и може да бъде пропусната. Типовете short int (или само short) и long int (или само long) са съкратен запис на signed short int и signed long int.

3.2.2. Реални типове

Ще разгледаме реалния тип double.
Типът е стандартен, вграден във всички реализации на езика.

Множество от стойности

Множеството от стойности на типа double се състои от реалните числа от $-1.74*10^{308}$ до $1.7*10^{308}$. Записват се във два формата – като числа с фиксирана и като числа с плаваща запетая (експоненциален формат).

```
<реално_число> ::= <цяло_число>.<цяло_число_без_знак>|
                   <цяло_число>E<порядък> |
                   <цяло_число>.<цяло_число_без_знак>E<порядък> |
<порядък> ::= <цяло_число>
```

При експоненциалния формат може да се използва и малката латинска буква e.

Примери: Следните числа

123.3454 -10343.034 123E13 -1.23e-4

са коректно записани реални числа.

Смисълът на експоненциалния формат е реално число, получено след умножаване на числото пред E (e) с 10 на степен числото след E (e).

Примери: 12.5E4 е реалното число 125000.0, а –1234.025e-3 е реалното число –1.234025.

Елементите от множеството от стойности на типа double се наричат **реални константи** или по-точно **константи от реалния тип double**.

Променлива величина, множеството от допустимите стойности, на която съвпада с множеството от стойности на типа double, се нарича **реална променлива** или **променлива от тип double.**

Дефинира се по обичайния начин. Дефиницията свързва реалните променливи с множеството от стойности на типа double или с конкретна стойност от това множество, като отделя по 8 байта оперативна памет за всяка от тях. Ако променливата не е била инициализирана, съдържанието на свързната с нея памет е неопределено, а в противен случай – съдържа указаната при инициализацията стойност.

Примери:

```
double i;
double j = 5699.876;
```

След тази дефиниция, имаме:

```
ОП
   i              j
   -           5699.876
8 байта        8 байта
```

Операции и вградени функции

Аритметични операции

Унарни операции

+, - Префиксни са. Потвърждават или променят знака на аргумента си.

Примери: Нека

```
double i = 1.2, j = -7.5;
```

Следните конструкции съдържат унарна операция + или -:

```
-i    +j    -j    +i    -56.7
```

Бинарни операции

Имат два аргумента. Следните аритметичните оператори са инфиксни:

+ - събиране
- - изваждане

* - умножение
/ - деление (поне единият аргумент е реален)

Примери:
15.3 – 12.2 = 3.1 13.0 / 5 = 2.6
15 + 12.35 = 27.35 13 / 5.0 = 2.6
-1.5 * 12.3 = -18.45

Логически операции

Логическите операции конюнкция, дизюнкция и отрицание могат да се прилагат над реални константи. Дефинират се по същия начин, като реалните числа, които са различни от 0.0 се интерпретират като true, а 0.0 – като false.

Примери:
123.6 and 0.0 е false
0.0 or 15.67 е true
not 67.7 е false

Операции за сравнение

Над реални данни могат да се извършват следните инфиксни операции за сравнение:

== - за равно != - за различно
> - за по-голямо >= - за по-голямо или равно
< - за по-малко <= - за по-малко или равно.

Наредбата на реалните числа е като в математиката.

Примери:
123.56 < 234.09 е true
-123456.9888 > 324.0098 е false
23451.6 >= 0 е true

Допълнение: Сравнението за равно на две реални числа x и y се реализира обикновено чрез релацията: $|x - y| < \varepsilon$, където $\varepsilon = 10^{-14}$ за тип double. По-добър начин е да се използва релацията:

$$\frac{|x - y|}{\max\{|x|, |y|\}} \leq \varepsilon$$

Вградени функции

При цял или реален аргумент, следните функции връщат реален резултат от тип double:

Sin(x) - синус, sin x, x е в радиани
cos(x) - косинус, cos x, x е в радиани
tan(x) - тангенс, tg x, x е в радиани
aSin(x) - аркуссинус, arcsin x $\in [-\pi/2, \pi/2]$, x $\in [-1, 1]$
acos(x) - аркускосинус, arccos x $\in [0, \pi]$, x $\in [-1, 1]$
atan(x) - аркустангенс, arctg x $\in (-\pi/2, \pi/2)$
exp(x) - експонента, e^x
log(x) - натурален логаритъм, ln x, x > 0
log10(x) - десетичен логаритъм, lg x, x > 0
sinh(x) - хиперболичен синус, sh x
cosh(x) - хиперболичен косинус, ch x
tanh(x) - хиперболичен тангенс, th x
ceil(x) - ([x]+1), преобразувано в тип double
floor(x) - [x], преобразувано в тип double
fabs(x) - абсолютна стойност на x, |x|
sqrt(x) - корен квадратен от x, x ≥ 0
pow(x, n) - степенуване, x^n (x и n са реални от тип double).

Примери: ceil(12.345) = 13.0 ceil(-12.345) = -12.0
ceil(1234) = 1234.0 ceil(-1234) = -1234.0
floor(12.345) = 12.0 floor(-12.345) = -13.0
floor(123) = 123.0 floor(-123) = -123.0
fabs(123) = 123.0 fabs(-1234) = 1234.0
sin(PI/6) намира $\sin(30^\circ)$, където PI = 3.141592

Тези функции се намират в библиотеката math.h и за да бъдат използвани е необходимо в частта на заглавните файлове да бъде включена директивата:

```
#include <math.h>
```

Задача 4. Да се напише програма, която намира функцията скобка от дадено реално число.

Следната програма решава задачата:

```
#include <iostream.h>
#include <math.h>
int main()
{cout << “x= “;
 double x;
 cin >> x;
 int y;
 y = floor(x);
 cout << y << “\n”;
 return 0;
}
```

Компилаторът издава предупреждение, че на линия 8 се извършва **преобразуване** на типа double в тип int и това може да доведе до загубване на информация.

Въвеждане

Реализира се по стандартния начин.

Пример: Ако

```
double x, y;
```

операторът

```
cin >> x >> y;
```

въвежда стойности на x и y.

За разделител се използват: интервалът, знаците за вертикална и хоризонтална табулация и знакът за преминаване на нов ред. Ако за някоя реална променлива е въведена цяла константа, извършва се конвертиране в реален тип, след което полученото реално число се записва в отделената за променливата памет.

Извеждане

Реализира се чрез оператора

```
cout << <реална_константа>;
```

или по-общо

cout << <реален_израз>;

В текущата позиция на курсора се извежда реалната константа или стойността на реалния израз. Използва се минималното количество позиции, необходими за записване на реалното число.

Пример: Нека имаме дефиницията

double x = 12.34, y = 9.876;

Операторът

cout << x << y << “\n”;

извежда върху екрана стойностите на x и y слепени

12.349.876

Този изход не е ясен. Налага се да се извърши форматиране на изхода. То се осъществява чрез подходящи манипулатори.

Манипулатор setw

Setw е функция.

Синтаксис

setw(<цял_израз>)

Семантика

Задава широчината на **следващото** изходно поле.

Пример: Операторът

cout << setw(12);

не извежда нищо. Той “манипулира” следващото извеждане като указва, че в поле с широчина 12 отдясно приравнена, ще бъде записана следващата извеждана реална константа.

Този манипулатор важи само за първото след него извеждане.

Пример: Нека

ОП

x	y
1.56	-2.36

Операторът

cout << setw(10) << x << y << “\n”;

извежда отново стойностите на x и y слепени, като 1.56 се предшества от 6 интервала, т.е.

******1.56-2.36

където интервалът е означен със знака *.

Операторът

cout << setw(10) << x << setw(10) << y << “\n”;

извежда

******1.56*****-2.36

При реалните данни се налага използването и на манипулатор за задаване на броя на цифрите след десетичната точка.

Манипулатор setprecision

Setprecision е функция.

Синтаксис

setprecision(<цял_израз>)

Семантика

Стойността на аргумента на тази функция задава броя на цифрите, използвани при извеждане на следващите реални числа, а в съчетание със загадъчното обръщение setiosflags(ios::fixed), задава броя на цифрите след десетичната точка на извежданите реални числа.

Пример 1: Операторът

cout << setprecision(3);

не извежда нищо. Той указва, че следващите реални константи ще се извеждат с 3 значещи цифри.

Нека в ОП имаме:

ОП

X

21.5632

Операторът

cout << setprecision(3) << setw(10) << x << “\n”;

извежда

******21.6

като извършва закръгляване.

А оператора

cout << setprecision(2) << setw(10) << x << “\n”;

извежда

********22

Пример 2: Операторът

cout << setiosflags(ios::fixed) << setprecision(3);

не извежда нищо. Той указва, че следващите реални константи ще се извеждат с точно 3 цифри след десетичната точка.

Нека в ОП имаме:

ОП

X

21.5632

Операторът

```
cout << setiosflags(ios::fixed)
     << setprecision(3)
     << setw(10) << x << “\n”;
```

извежда

****21.563

А операторът

```
cout << setiosflags(ios::fixed)
     << setprecision(1)
     << setw(10) << x << “\n”;
```

извежда

******21.6

като извършва закръгляване.

Забележка: За щастие, манипулаторите setprecision() и setiosflags () са в сила не само за първата извеждана константа, но и за всички следващи, докато с нов setprecition() не се промени точността и с resetiosflags(ios::fixed) не се отмени setiosflags(ios::fixed).

Пример: Изпълнението на програмата

```
#include <iostream.h>
#include <iomanip.h>
int main()
{double a = 209.5, b = 63.75658;
 cout  << setprecision(3)
       << setiosflags(ios::fixed);
 cout << setw(10) << a << "\n";
 cout << setw(10) << b << "\n";
 cout << resetiosflags(ios::fixed);
 cout << setw(20) << a << "\n";
 cout << setw(20) << b << "\n";
 return 0;
```

```
    }
```
извежда
```
***209.500
****63.757
*****************210
****************63.8
```

Допълнение: Точността на типа double е около 15 значещи цифри. За да се убедите в това, изпълнете следната програма:

```
#include <iostream.h>
int main()
{ double a = 5e14;
  double b = a - 0.1;
  double c = a - b;
  cout << c << "\n";
  return 0;
}
```

Лесно се вижда, че стойността на променливата с трябва да бъде 0.1, а програмата намира за такава 0.125. Причината е в броя на значещите цифри на b.

Други реални типове

В езика C++ има и друг реален тип, наречен float. Различава се от типа double по множеството от стойностите си и заеманата памет.

Множеството от стойности на типа float се състои от реалните числа от диапазона от $-3.4*10^{38}$ до $3.4*10^{38}$. За записване на константите от този диапазон са необходими 4 байта ОП.

Броят на значещите цифри при този тип е около 7. За да се убедите в това, изпълнете следната програма:

```
#include <iostream.h>
int main()
{ float a = 5e6;
  float b = a - 0.1;
  float c = a - b;
  cout << c << "\n";
```

```
  return 0;
}
```

Лесно се вижда, че стойността на с трябва да е 0.1, а след изпълнение на горната програма се получава 0. Компилаторът предупреждава, че на линия 4 става преобразуване от тип float в тип double.

Реална константа от диапазона на тип float, но с повече от 7 значещи цифри се приема от компилатора за реално число от тип double и присвояването му на променлива от тип float издава предупреждението, че се извършва преобразуване от тип double в тип float, което може да доведе до загуба на точност.

Точността е причината заради, която препоръчваме използването на типа double.

3.2.3. Аритметични изрази

Аритметичните изрази са правила за получаване на числови константи. Има два вида аритметични изрази: цели и реални.

<аритметичен_израз> ::= <цял_израз> | <реален_израз>

Цели аритметични изрази

Целите аритметични изрази са правила за получаване на константи от тип int или разновидностите му. Дефинират се рекурсивно по следния начин:

- Целите константи са цели аритметични изрази.

Примери: 123 −2345 −32767
0 22233345 −87
са цели аритметични изрази.

- Целите променливи са цели аритметични изрази.

Примери: Ако имаме дефиницията:

```
int i, j;
short p, q, r;
```

i j
са цели аритметични изрази от тип int, а
p q r
са цели аритметични изрази от тип short.

- Прилагането на унарните операции + и – към цели аритметични изрази е цял аритметичен израз.

Примери: -i +j -j
са цели аритметични изрази от тип int, а
+p -p +r -q
са цели аритметични изрази от тип short.

- Прилагането на бинарните аритметични операции +, -, *, / и % към цели аритметични изрази, е цял аритметичен израз.

Пример: i % 10 + j * i - p -i + j / 5
са цели аритметични изрази от тип int,
r – p / 12 – q r % q – p r + p – q
са цели аритметични изрази от тип short.

- Цели функции, приложени над цели аритметични изрази, са цели аритметични изрази.

Примери: abs(i+j) е цял аритметичен израз от тил int, а abs(p-r) е цял аритметичен израз от тип short.

Реални аритметични изрази

Реалните аритметични изрази са правила за получаване на константа от тип double или float. Дефинират се рекурсивно по следния начин:

- Реалните константи са реални аритметични изрази.

Примери: 1.23e-3 –2345e2 –3.2767 0.0
222.33345 –8.7009
са реални аритметични изрази.

Забележка: Реална константа от диапазона на тип float, но с повече от 7 значещи цифри се приема от компилатора за реално число от тип double.

- Реалните променливи са реални аритметични изрази.

Примери: Ако имаме дефинициите:

```
double i, j;
float p, q, r;
```

```
i     j
```

са реални аритметични изрази от тип double, а

```
p     q     r
```

са реални аритметични изрази от тип float.

- Прилагането на унарните операции + и – към реални аритметични изрази е реален аритметичен израз.

Примери: -i +j -j

са реални аритметични изрази от тип double, а

+p -p +r -q

са реални аритметични изрази от тип float.

- Прилагането на бинарните аритметични операции +, -, * и / към аритметични изрази, поне един от които е реален, е реален аритметичен израз.

Пример: i % 10 + j*i - p -i + j/5

са реални аритметични изрази от тип double, а

–p/12 – q r%q – p r + p – q

са реални аритметични изрази от тип float.

- Реални функции, приложени над реални или цели аритметични изрази, са реални аритметични изрази.

Примери: fabs(i+j) sin(i-p) cos(p/r-q) floor(p)
ceil(r-p+i) exp(p) log(r-p*q)

са реални аритметични изрази от тип double.

Семантика на аритметичните изрази

За пресмятане на стойностите на аритметичните изрази се използват следният приоритет на операциите и вградените функции:

1. Вградени функции

2. действията в скобите
3. Операции в следния приоритет
- +, - (унарни) най-висок
- *, /, %
- +, - (бинарни)
- <<, >> най-нисък

Забележка 1: Операторите >> и << са за побитови измествания надясно и наляво съответно. Те са предефинирани за входно/изходни операции. В случая имаме предвид тази тяхна употреба.

Забележка 2: Инфиксните операции, които са разположени на един и същ ред са с еднакъв приоритет. Тези оператори се изпълняват отляво надясно. Унарните операции се изпълняват отдясно наляво.

Пример: Нека имаме дефиницията

```
double x = 23.56, y = -123.5;
```

Изпълнението на оператора

```
cout << sin(x) + ceil(y) * x - cos(y);
```

ще се извърши по следния начин: отначало ще се пресметнат стойностите на sin(x), ceil(y) и cos(y), след това ще изпълни операцията * над стойността на ceil(y) и x, полученото ще събере със стойността на sin(x), след което от полученото реално число ще се извади пресметнатата вече стойност на cos(y). Накрая върху екрана ще се изведе получената реална константа.

Аритметичните изрази могат да съдържат операнди от различни типове. За да се пресметне стойността на такъв израз, автоматично се извършва преобразуване на типовете на операндите му. Без загуба на точността се осъществяват следните преобразувания:

Тип	Преобразува се до тип
bool	всички числови типове
short	int
unsigned short	unsigned int
float	double

т.е.конвертира се от “по-малък” тип (в байтове) към “по-голям” тип.

За да се пресметне стойността на аритметичен израз с операнди от различни типове, последователно се прилагат правилата по-долу, докато се уеднаквят типовете (ако е възможно).

Ако има операнд от тип:	Другите операнди се преобразуват до:
double	double
float	float
unsigned int	unsigned int
int	int
unsigned short	unsigned short
short	short

Семантика на булевите изрази

Булевите изрази са правила за получаване на булева стойност. За пресмятане на стойностите им се използва следният приоритет на операциите и вградените функции:

1. Вградени функции
2. Действията в скобите
3. Операции в следния приоритет

- !, not, +, - (унарни) най-висок
- *, /, %
- +, - (бинарни)
- >> << (вход/изход)
- <, <=, >, >=
- ==, !=
- &&
- || най-нисък

Забележка: Инфиксните операции, които са разположени на един и същ ред са с еднакъв приоритет. Тези оператори се изпълняват отляво надясно, т.е. лявоасоциативни са. Унарните оператори се изпълняват отдясно наляво, т.е. дясноасоциативни са.

Примери: а) Нека имаме дефинициите:

```
double x = 23.56, y = -123.5;
bool b1, b2, b3;
b1 = true;
b2 = !b1;
```

```
   b3 = b1||b2;
```
Изпълнението на оператора
```
   cout << sin(x) + ceil(y) * x > 12398;
```
ще сигнализира грешка – некоректни аргументи на <<. Това е така, заради нарушения приоритет. Операторът << е с по-висок приоритет от този на операторите за сравнение. Налага се вторият аргумент на << да бъде ограден в скоби, т.е.операторът
```
   cout << (sin(x) + ceil(y) * x > 12398);
```
вече работи добре.

б) Изпълнението на оператора
```
   cout << b1 && b2 || b3 << "\n";
```
също съобщава грешка – некоректни аргументи на <<. Отново е нарушен приоритетът на операциите. Налага се аргументът b1 && b2 || b3 на << да се огради в скоби, т.е.
```
   cout << (b1 && b2 || b3) << "\n";
```
вече работи добре.

Задачи върху типовете булев, цял и реален

Задача 5. Кои от следните редици от знаци са числа в C++?

а) 061 б) –31 в) 1/5 г) +910.009

д) VII е) 0.(3) ж) sin(0) з) 134+12

Решение: а), б), г).

Задача 6. Да се запишат на C++ следните числа:

а) 6! б) LXXIV в) –0,4(6) г) 138,2(38)

д) 11/4 е) π ж) 1,2 .10^{-1} з) –23,(1) .10^{2}

В дробната част да се укажат до 4 цифри.

Решение:

а) 120 б) 74 в) –0.4667 г) 138.2384

д) 2.7500 е) 3.1416 ж) 0.1200 з) –2311.1111

Задача 7. Да се запишат на езика C++ следните математически формули:

а) $a + b.c - a^2b^3c^4$

б) $\frac{a.b}{c} + \frac{c}{a.b}$

в) $(1 + \frac{x}{1!} + \frac{x^2}{2!}).(1 + \frac{x^3}{3!} + \frac{x^5}{5!})$

Решение:

а) a + b * c – a * a * b * b * b * c * c * c * c

г) $\sqrt{1 + \sqrt{2 + \sqrt{3 + \sqrt{4}}}}$

б) (a * b) / c + c / (a * b)

в) (1 + x + x*x/2)*(1 + x*x*x/6 + x*x*x*x*x/120)

или

(1 + x + pow(x,2))/(1 + pow(x, 3)/6 + pow(x, 5)/120)

г) sqrt(1 + sqrt(2 + sqrt(3 + sqrt(4))))

Задача 8. Какво ще бъде изведено след изпълнението на следната програма:

```
#include <iostream.h>
#include <math.h>
int main()
{ cout << "x=";
  double x;
  cin >>x;
  bool b;
  b = x < ceil(x);
  cout << "x=";
  cin >> x;
  b = b && (x < floor(x));
  cout << "b= " << b << "\n";
  return 0;
 }
```

ако като вход бъдат зададени числата

а) 2.7 и 0.8 б) 2.7 и –0.8 в) –2.7 и –0.8.

Решение: а) Тъй като булевият израз 2.7 < ceil(2.7) има стойност true, а true && (0.8 < floor(0.8)) - е false, ще бъде изведено 0 (false).

Задача 9. Какъв ще е резултатът от изпълнението на програмата

```
#include <iostream.h>
int main()
{int a, b;
 cin >> a >> b >> a >> b >> a;
 cout << a << " " << b << " "
```

```
        << a << " " << b << " "
        << a << "\n";
    return 0;
   }
```

ако като вход са зададени числата 1, 2, 3, 4 и 5?

Решение: След обработката на дефиницията int a, b; за променливите a и b са отделени по 4 байта ОП, т.е.

ОП

a b

\- -

а след изпълнението на оператора за четене >>, a и b получават отначало стойностите 1 и 2. След това стойностите им се променят на 3 и 4 съответно. Най-накрая a става 5, т,е,

a b

5 4

Тогава програмата извежда

5 4 5 4 5

Задача 10. Да се запише булев израз, който има стойност true, ако посоченото условие е в сила, и стойност false, ако условието не е в сила.

а) цялото число a се дели на 5;

б) точката x принадлежи на отсечката [2, 6];

в) точката x не принадлежи на отсечката [2, 6];

г) точката x принадлежи на отсечката [2, 6] или на отсечката [-4, -2];

д) поне едно от числата a, b и c е отрицателно;

е) числата a, b и c са равни помежду си.

Решение:

а) a % 5 == 0

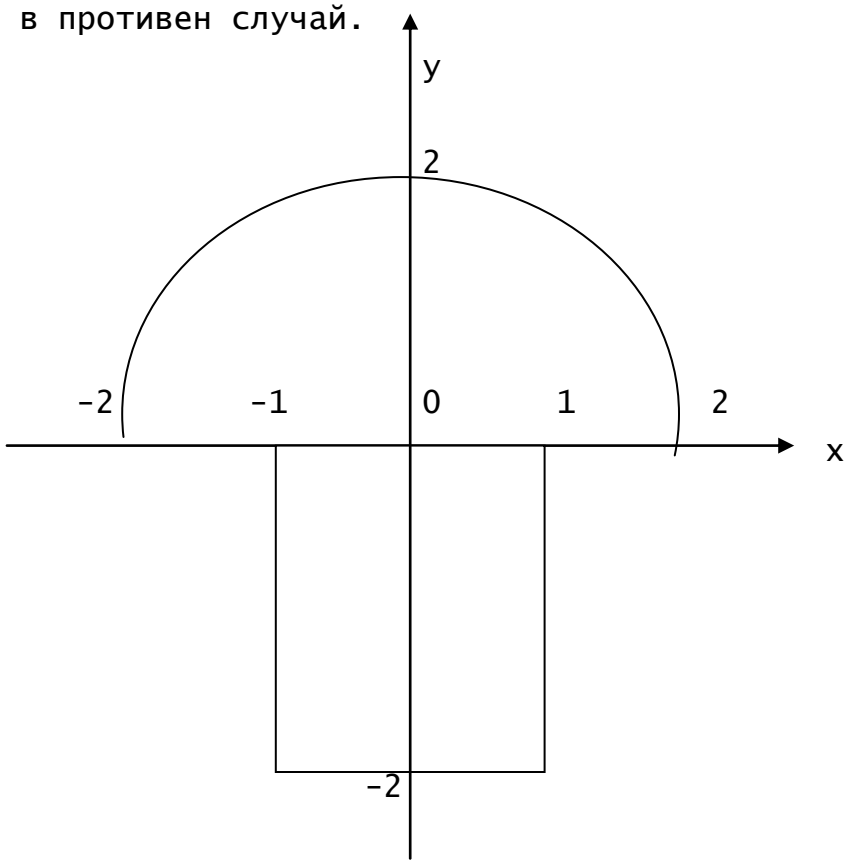
б) x >= 2 && x <= 6

в) x < 2 || x > 6 или !(x >= 2 && x <= 6)

г) x >= 2 && x <= 6 || x >= -4 && x <= -2

е) a == b && a == c

Задача 11. да се напише програма, която въвежда координатите на точка от равнината и извежда 1, ако точката принадлежи на фигурата по-долу и – 0, в противен случай.



Решение:

```
#include <iostream.h>
#include <math.h>
int main()
{cout << "x= ";
 double x;
 cin >> x;
 cout << "y= ";
 double y;
 cin >> y;
 bool b1 = x*x + y*y <= 4 && y >= 0;
 bool b2 = fabs(x) <= 1 && y < 0 && y >= -2;
 cout << (b1 || b2) << "\n";
 return 0;
}
```

Задачи

Задача 1. Да се запишат на езика C++ следните математически формули:

а) $\dfrac{x^2 + y^2}{(|\sin x| + \lg|y|)^{\frac{1}{5}}}$

б) $\dfrac{a - b.c}{(\text{arctg } a - \text{sh } b + \text{ch } c)^4}$

в) $(\text{acos } x - [x] - 1)^3 * e^{x + .,23}$

г) $\ln(x^4 + e^x + 10)$

д) $\log_5(x^2 + x^4 + 3)$

Задача 2. Кои от следните редици от символи са правилно записани изрази на езика C++:

а) 1 + |y|
б) -abs(x) + sin z
в) abs(x) + cos(abs(y - 1.7))
г) 1 + sqrt(sin((u+v)/10))
д) -6 + xy
е) 1/-2 + Beta.

Задача 3. Да се запишат в традиционна (математическа) форма следните изрази, записани в синтаксиса на езика C++:

а) sqrt(a+b) - sqrt(a-b)
б) a + b/(c+d)-(a+b)/c+d
в) x*y/(u+v)-(u-v)/y*(a+b)
г) 1+exp(cos((x+y)/2)).

Задача 4. Да се пресметне стойността на израза:

а) cos(0) + abs(1/(1/3-1))
б) abs(a-10) + sin(a-1), за a = 1;
в) cos(-2+2*x) +sqrt(fabs(x-5)), за x = 1;
г) sin(sin(x*x-1)*sin(x*x-1)) + cos(x*x*x-1)*abs(x-2), за x = 1;
д) sin(sin(x*x-1))+cos(x*x*x-1)*cos(abs(x-2)-1)/y*a + sqrt(abs(y)-x), за x = 1, y = -2, a = 2.

Задача 5. в аритметичния израз

а) a/b*c/d*e/f*h

б) a+b/x-2*y

в) a+b/x-2*y

да се поставят скоби така, че полученият израз да съответствува на математическата формула:

а) $\dfrac{a}{b\,.\,\dfrac{c}{d\,.\,\dfrac{e}{f\,.\,h}}}$ б) $\dfrac{a+b}{x-2.y}$ в) $a+\dfrac{b}{x-2}\,.\,y$

Задача 6. Да се напише израз на езика C++, който да изразява:

а) периметъра на квадрат с лице, равно на a;

б) лицето на равностранен триъгълник с периметър, равен на p.

Задача 7. Да се напише програма, която пресмята стойността на v1, където

а) $v1 = m + \dfrac{n}{p + \dfrac{q}{r + \dfrac{s}{t}}}$ б) $v1 = \dfrac{\sin(\sin(\sin x)) + \cos(\cos(\cos x))}{|\ln x| + |\cos x| + e^x}$

Задача 8. Да се пресметне стойността на израза:

а) pow(x, 2) + pow(y, 2) <= 4	при x = 0.6, y = -1.2
б) p % 7 == p / 5 - 2	при p = 15
в) floor(10*k+16.3)/2 == 0	при k = 0.185
г) !((k+325)%2 == 1)	при k = 28
д) u*v != 0 && v > u	при u = 2, v = 1
е) x || !y	при x = false, y = true.

Задача 9. Да се запише булев израз, който да има стойност истина, ако посоченото условие е вярно и стойност - лъжа, ако условието не е вярно:

а) цялото число p се дели на 4 или на 7;

б) уравнението $a.x^2 + b.x + c = 0$ $(a \neq 0)$ няма реални корени;

в) точка с координати (a, b) лежи във вътрешността на кръг с радиус 5 и център (0, 1).

г) точка с координати (a, b) лежи извън кръга с център (c, d) и радиус f;

д) точка принадлежи на частта от кръга с център (0, 0) и радиус 5 в трети квадрант;

е) точка принадлежи на венеца с център (0, 0) и радиуси 5 и 10;

ж) x принадлежи на отсечката [0, 1];

з) x = max{a, b, c}

и) x != max{a, b, c} (операцията ! да не се използва);

к) поне една от булевите променливи x и y има стойност true;

л) и двете булеви променливи x и y имат стойност true;

м) нито едно от числата a, b и c е положително;

н) цифрата 7 влиза в записа на положителното трицифрено число p;

о) цифрите на трицифреното число m са различни;

п) поне две от цифрите на трицифреното число m са равни помежду си.

Допълнителна литература

1. К. Хорстман, Принципи на програмирането със C++, С., СОФТЕХ, 2000.
2. П. Лукас, Наръчник на програмиста, С., Техника, 1994.
3. Ст. Липман, Езикът C++ в примери, “КОЛХИДА ТРЕЙД” КООП, С. 1993.