Тема 6
Аритметични, логически и побитови оператори. Пресмятане на изрази и преобразуване на типове

1. Логически оператори и отношения

В булевата алгебра резултатът от изпълнението на сравнения (по-голямо, по-малко, равно, различно и др.) и логически операции (логическо И, логическо ИЛИ, логическо НЕ) е логически израз, чиято стойност може да е вярно твърдение (***true***) или невярно твърдение (***false***). В програмните езици са предвидени оператори, които да изпълняват тези операции. Резултатът от тяхното изпълнение е от логически тип. За разлика от други езици за програмиране, първоначално в език С не е предвиден логически тип данни. За вярно и невярно твърдение се използват аритметичните стойности: 0 или различно от 0. Ако отношението не е вярно, то стойността на резултата е 0, което съответства на логическа стойност ***true***. Ако отношението е вярно, стойността на резултата е различна от 0, което съответства на логическа стойност ***false***. В последствие в езика е добавен булевия тип ***bool***. MS Visual C++ 5.0 и следващите версии поддържат вградения тип ***bool***, чиито константни стойности са две: ***true*** и ***false***.

В език C/C++ са предвидени следните оператори за сравнения:

> по-голямо

>= по-голямо или равно

< по-малко

<= по-малко или равно

== равно

!= различно

Логическите оператори в С и С++ са:

&& логическо И

|| логическо ИЛИ

! логическо НЕ

Действието на всеки логически оператор се представя с т.нар. ***таблица на истинност***. В нея се представят резултатите от изпълнението на оператора при всяка една различна комбинация от входни данни.

Таблицата на истинност на логически оператор И е следната:

x	y	x&&y
false	false	false
false	true	false
true	false	false
true	true	true

Таблицата на истинност на логически оператор ИЛИ е следната:

x	y	x||y
false	false	false
false	true	true
true	false	true
true	true	true

Таблицата на истинност на логически оператор НЕ е следната:

x	!x
false	true
true	false

В случая, в таблиците се използват стойности ***true*** и ***false*** за *вярно* и *невярно твърдение*. Както беше споменато, в C/C++ стойностите **!0** (различно от нула) и **0** (нула) се интерпретират като true и false, съответно.

Примери за действието на логически оператори и отношения са следните: нека са дефинирани променливите a и b като са инициализирани със стойности 4 и 7:

int a=4,b=7;

При изпълнението на изразите се получава следното:

Резултатът от изпълнение на израз **a<0** е false

Резултатът от изпълнение на израз **b>=0** е true

Резултатът от изпълнение на израз **(a>b)&&(a>0)** е false

Резултатът от изпълнение на израз **!a** е false

Резултатът от изпълнение на израз **(a<0)||((b>0)** е true

Логическите изрази и отношения се използват при задаване на условия в конструкциите, които реализират цикли и разклонения и се записват в програмния код, където е необходимо.

2. Побитови оператори

В C/C++ е предвидена възможност за реализиране на операции с отделните битове на операндите т.нар. **поразредни** (**побитови**) операции. Побитовите операции могат да се прилагат само върху данни от тип ***int*** и неговите модификации ***short, long, unsigned***.

Операторите, които реализират поразредните операции са:

<< преместване на битовете наляво с определен брой позиции, като освободените битове се допълват с нули (0)

>> преместване на битовете надясно с определен брой позиции, като освободените битове се допълват с нули (0)

& поразрядно И

| поразрядно ИЛИ

^ поразрядно изключващо ИЛИ

~ инвертиране битовете на операнда

За разлика от логическите оператори И (&&) и ИЛИ (||), побитовите оператори И (&) и ИЛИ (|) изпълняват действия не върху стойността на операндите като цяло, а върху всеки отделен бит от стойностите на операндите. Подобно на логическите, действието на побитовите оператори И, ИЛИ, изключващо ИЛИ и инвертиране на битове може да се илюстрира чрез таблици на истинност, в които операнди са отделни битове:

x	y	x&y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x|y
0	0	0
0	1	1
1	0	1
1	1	1

x	y	x^y
0	0	0
0	1	1
1	0	1
1	1	0

x	~x
0	1
1	0

За използването на побитовите оператори могат да бъдат посочени следните примери:

- **Преместване наляво**

```
x=0x4f;          // представено като двойно число x=0100 1111₂
x=x<<2;          // резултатът е: x = 1 0011 1100₂ =  0x13c
```

Всяко преместване с 1 разряд наляво е равносилно на умножение по 2. Пример:

```
y=1;
y=y<<1;          // представено като двойно число y=010₂=2
y=y<<1;          // y=100₂=4
```

- **Преместване надясно**

```
x=0xf8;          // представено като двойно число x=1111 1000₂
x=x>>2;          // резултатът е: x = 0011 1110₂ =  0x3e
```

Всяко преместване с 1 разряд надясно е равносилно на целочислено деление на 2. Пример:

```
y=4;
y=y>>1;          // представено като двойно число y=010₂=2
y=y>>1;          // y=001₂=1
```

- **Поразрядно И**

Пример за използване на побитов оператор И е нулиране на определени битове. Например, нека да бъде нулиран 5 бит на даден 16 разряден регистър. Стойността, с която ще се нулира бит 5 от 16-разрядният регистър е: 1101 1111 (0xdf) т.е. 0 в 5-ти бит и всички останали битове в установени в 1. Подобна стойност е прието да се нарича **маска**.

```
mask=0xdf;       // mask= 1101 1111₂
reg=0xf8;        // reg=1111 1000₂
reg=reg & mask;  // резултатът е reg = 1101 1000₂ (0xd8)
```

- **Поразрядно ИЛИ**

Побитов оператор ИЛИ се използва при установяване в 1 на определени битове. Например, нека да бъде установен в 1, бит 5 на даден 16 разряден регистър. Маската е: 0010 0000 (0x20) т.е. 1 в бит 5 и 0 за всички останали битове.

```
mask=0x20;          // mask=0010 0000₂
reg=0x88;           // reg=1000 1000₂
reg=reg | mask;     // резултатът е reg = 1010 1000₂ (0xa8)
```

- **Поразрядно изключващо ИЛИ**

Побитов оператор изключващо ИЛИ се използва когато е необходимо да бъдат инвертирани определи битове. Например, нека да бъде инвертиран бит 5 на даден 16 разряден регистър. Маската е: 0010 0000 (0x20) т.е. 1 в бит 5 и 0 за всички останали битове.

```
mask=0x20;          // mask=0010 0000₂
reg=0x88;           // reg=1000 1000₂
reg=reg ^ mask;     // резултатът е reg = 1010 1000₂ (0xa8)
reg=reg ^ mask;     // резултатът е reg = 1000 1000₂ (0x88)
```

3. Изрази. Оператор за присвояване

Израз е правило за изчисляване на стойност т.е. всяка валидна за езика комбинация оператори и операнди (данни за операторите), по която се пресмята стойност. В резултат на изчислението на даден израз се получава стойност – **резултат от израза**. Изразите участват в управляващите конструкции, както и в оператора за присвояване. Изразите също имат тип като това е типът на получения резултат.

Когато в даден израз има повече от един оператори, те се изпълняват съгласно техния **приоритет**. Отчитането на приотитета на операторите е важно при съставянето на изразите. Например, **събиране** и **изваждане** са с по-висок приоритет от **умножение** и **деление**. В Приложение 2 е дадена таблица с приоритетите на операторите в C/C++. Най-висок приоритет имат **скобите ()**.

Оператор за присвояване служи за задаване на стойности на променливи, масиви, елементи на структури и други модифицируеми компоненти на програмите. Общият вид (***синтаксис***) на оператор присвояване е:

променлива = израз;

Действието на оператора е: пресмята се стойността на израза отдясно и се присвоява на променливата отляво.

В C/C++ съществува ***съкратена форма на оператор присвояване***:

операнд1 *операция* = операнд2;

Тази форма е еквивалента на:

операнд1 = (операнд1) *операция* (операнд2);

Например: вместо ***p=p+2;*** записът е ***p+=2;***

Като операции в съкратения оператор за присвояване могат да се използват бинарните аритметичните оператори + - * / %, както и побитовите оператори << >> & | ^. Комбинациите на оператора за присвояване с останалите оператори са следните: +=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=. Операндите в съкратения оператор за присвояване са в скоби. Това означава, че запис: ***x*=y+1;*** е еквивалентен на: ***x=x*(y+1);*** а не на: *x=x*y+1;*

Операторът за присвояване в C++ може да се използва и каскадно т.е. да се използва повече от вежнъж в един и същ израз. В този случай присвояванията се извършват отдясно наляво.

Пример:

```
int x, y;
x = y = 0;
```

В случая, стойност 0 ще се присвои първо на променливата y, а после на променливата x.

4. Преобразуване на типове

В програмните езици често се налага един тип да бъде преобразуван към друг тип. Преобразуването на типове цели да бъдат уеднаквени типовете на операндите в даден израз, както и типа на резултата в оператора за присвояване. Преобразуването на типове е или ***явно***, чрез използване на специален оператор, или ***неявно*** – автоматично, по съответните правила. Явното преобразуване на типове е възможно само в език C++.

4.1. Неявно преобразуване на типове

Неявното преобразуване на типове се осъществява автоматично от компилатора в следните случаи:

- при изчисляване на изрази;
- при присвояване;
- при връщане стойности от функции.

При изчисляването на изрази, преобразуването на типове се регламентира по следната схема:

char	--> int
unsigned char	--> int
signed char	--> int
short	--> int
unsigned short	--> unsigned int
float	--> double

При изчисляването на изрази се прилага следното правило за уеднаквяване на операндите: Ако единият от операндите е от тип: **long double, double, float, unsigned long, long, unsigned**, то останалите операнди се преобразуват до същия тип.

Вторият случай на автоматично преобразуване на типове е свързан с оператора за присвояване, като типът на израза се преобразува до типа на променливата в лявата част.

Пример:

```
int x;
float y=2.4;
x=y+1;          // типът на x се преобразува като float
```

Автоматичното преобразуване на типове не винаги е възможно. В случай че има несъответствие на типовете и не е възможно тяхното преобразуване, компилаторът издава съобщение за грешка.

4.2. Явно преобразуване на типове *(typecasting)*

Явното преобразуване на типове се осъществява чрез оператор ***привеждане (typecast)***. Синтаксисът на оператора е:

(тип)израз;

Действието на оператор ***typecast*** е: привежда типа на израза към типа, указан в скобите.

Пример:

```
int x=5;
int y=2;
cout<<"x/y="<<(float)x/(float)y;       // резултатът от делението е 2.5
```

В случая типът на променливите **x** и **y** явно се преобразува към float.
Необходимо е да се отбележи, че типовете на променливите се преобразуват само за изчисляването на израза, а не като цяло.

Относно въпроси по темата на адрес: ln_zh_st@yahoo.com