Операции

Аритметични и логически изрази.

Приоритет на операциите в C#.

Оператор за присвояване.

Условен оператор.

Sofia, 2022

В тази тема ще стане дума за:

- Структура и организация на C# програма
- Операции над променливи от различни типове
- Аритметични и логически изрази
- Оператор за присвояване
- Оператори за контрол на изпълнението или логически оператори

Операции и изрази

- Манипулацията на данните в езика C# се извършва с ***операции***. Операциите, най-често са ***двуаргументни***, рядко ***едноаргументни*** и имаме един екзотичен случай на операция с ***три аргумента***. Най-добре познати са аритметичните операции.
- Всяка операция завършва с ***резултат***.
- Знакът на двуаргументните (двуместните) операции се поставя, с малки изключения, между двата аргумента – ***инфиксен запис***. При едноаргументите знакът може да се постави пред аргумента – ***префиксен запис*** или след него – ***постфиксен*** запис.
- След като над някакво множество от стойности са дефинирани операции, както и в математиката, можем да строим със стойностите от множеството и знаците за операциите ***изрази***. Например, ***аритметични изрази, логически изрази***.
- При достигане до израз в програмата, този израз ***се пресмята/оценява*** до получаване на резултата. Например, при аритметичните операции резултатът е число от съответно множество.

Операции и изрази

- Пресмятането/оценяването на изрази се извършва в **указан от програмиста ред** на прилагане на участващите операции. Това може да стане с употребата на скобите (и), както в математиката.
- За да се избегне прекомерната употреба на скоби, в тежки изрази се използват два механизма – ***приоритет*** и ***асоциативност*** на операциите.
- Всички операции на езика са подредени по приоритет, като има ***групи от операции*** с еднакъв приоритет. При решаване коя от две или повече операции да се изпълни, се избира тази с най-висок приоритет.
- За всяка група от операции с еднакъв приоритет е дефинирана асоциативност на групата, която може да е ***лява*** или ***дясна***. Когото имаме последователност от ляво асоциативни операции с еднакъв приоритет, тогава те се изпълняват от ляво надясно, а при дясно асоциативни операции с еднакъв приоритет – от дясно наляво.
- Например a + b + c – d е еквивалентен на ((a + b) + c) – d, тъй като събиране и изваждане са ляво асоциативни и с еднакъв приоритет.

Приоритет на операциите

- В един израз могат да бъдат използвани операции с различен приоритет. **Например:**

```
3 + 9 % 2;
```

В горния пример стойността на израза е 4, защото приоритетът на операцията % е по-висок от приоритета на операцията +, следователно винаги се изпълнява преди него.

- С използване на скоби задаваме в явен вид реда на изпълнение на операциите. В много сложни изрази това увеличава четливостта на кода.

Наример:

```
5 + (7 % 2);     // стойност 6
(5 + 7) % 2;     // стойност 0
```

Аритметични операции

Езиците C, C++, C#, Java и др. предлагат повече от 40 операции. Ще разгледаме най-често употребяваните.

- ***Аритметичните операции***, които могат да се прилагат еднакво към произволни числови стойности, са добре познатите: двуаргументните ***събиране*** (знак `+`), ***изваждане*** (знак `-`), ***умножение*** (знак `*`), както и едноаргументната ***смяна на знака*** на числото (също със знак `-`).
- Двуаргументната операция ***деление*** (със знак `/`) се извършва по различен начин при цели и дробни числа. В първия случай резултатът е цяло число – цялата част на резултата от делението (***целочислено деление***), а във втория – истинския резултат от делението. Например, `5 / 2 = 2`, а `5. / 2 = 2.5`
- За компенсация на неточността при целочислено деление, езикът предоставя двуаргументната операция за ***получаване на остатъка*** при целочисленото деление със знак `%`. Например `5%2 = 1`
- Следващият слайд е обобщение на казаното за аритметичните операции.

Аритметични операции (+, -, *, /, %)

- Аритметичните операции, които се поддържат от C#, са показани в таблицата. Числата в скоби като горен индекс показват броя на аргументите.
- Най-висок приоритет от тези операции има смяната на знака, която е дясно асоциативна. Всички останали са ляво асоциативни.
- Операциите умножение, двете деления и намиране на остатъка са с втори приоритет, а събирането и изваждането – с трети.
- Изразите, съставени с числови стойности и аритметични операции, наричаме ***аритметични изрази*** и те се пресмятат до получаване на числова стойност.

Операция	Описание
-(1)	Смяна на знака
*(2)	Умножение
/(2)	Целочислено деление
/(2)	Деление
%(2)	Остатък при целочислено деление
+(2)	Събиране
-(2)	Изваждане

Присвояване

- Присвояването на стойност на променлива в езика C# е ***операция!!!*** Нейният знак е =. Операцията е двуаргументна, като левият операнд е **променлива**, а десният – **израз,** който се пресмята до някаква **стойност** от типа на променливата.
- При изпълняване на тази операция променливата получава като стойност резултата от пресмятането на израза (при несъответствие на типа на променливата с типа на стойността, стойността се преобразува до типа на променливата, ако това е възможно).
- Присвояването е с **по-нисък приоритет** от този на аритметичните и повечето от другите операции, а асоциативността му е **дясна**!!!
- **Резултат** от операцията е присвоената стойност. Затова изразът

  `a = b = c = d = 0` е еквивалентен на `a = (b = (c = (d = 0))`,

  а резултат от пресмятането му е, че четирите променливи получават стойност `0`.

  Такава е и стойността на целия израза, който също наричаме присвояване.
- Знакът на операцията присвояване може да създаде някакъв дискомфорт, ако се гледа на равенството, така както е познато от математиката. Така присвояването `a = a + 1` изглежда абсурдно от математическа гледна точка, но трябва да се свикне с тази особеност.

Оператор за присвояване

- Конструкцията

  присвояване; //Например a=b;

т.е. изписването на ***знака за край*** на оператор след присвояване, го превръща в ***оператор за присвояване***. При изпълняване на оператора за присвояване се извършват зададените от израза присвоявания.

- Други операции за присвояване са ***комбинираните*** (с аритметична операция) ***присвоявания*** `+=, -=, *=, /=, %=.` Например `a += 10` е еквивалентен израз на `a = a + 10, a -= 10` е еквивалентен израз на `a = a – 10` и т.н.
- Специфичен вид операции за присвояване са ***автоинкрементните*** `a++` и `++a`, и ***автодекрементните*** `a--` и `--a`, като префиксният и постфиксният запис имат различен смисъл:
  - ✓ `a++` и `++a` са еквивалентни на израза `a = a + 1`, като в постфиксен запис стойността на `a` първо се използва за пресмятане на съдържащия я израз, след което се увеличава с 1, а в префиксен – обратно. Приоритетът е по-голям от този на аритметичните операции. Конструкции от вида `++a++` **е добре да се избягват**.
  - ✓ `a--` и `--a` са еквивалентни на израза `a = a - 1`, като разликата между префиксен и постфиксен запис е аналогична.
- От комбинираните присвоявания, автоинкрементните и автодекрементни операции също **се получават операции** с поставяне на знака за край на оператор.

Пример

```
using System;

namespace AritmOp{
  class Program{
   static void Main(){
    int a, b;          // a:?,  b:?
    a = 10;            // a:?,  b:?
    b = 4;             // a:?,  b:?
    a = b;             // a:?,  b:?
    b = 7;             // a:?,  b:?

    Console.WriteLine("a:"+a);
    Console.WriteLine("b:"+b);
   }
  }
}
```

- Изход

```
a:4 b:7
```

- Програмата отпечатва на екрана финалните стойности на **a** и **b** (съответно 4 и 7).
- Обърнете внимание, че стойността на променливата **a** не е повлияна от последната промяна на **b**, въпреки че в по-горна команда сме декларирали, че a = b – **a** приема текущата стойност на **a**. Стойностите на двете променливи не се уеднаквяват.

Комбинирано присвояване

- Операциите за комбинирано присвояване (`+=, -=, *=, /=, %=, както и не споменатите още >>=, <<=, &=, ^=, |=`) ни спестяват изписването на променливата от лявата част на присвояването в дясната.
- Еквивалентността на автоинкрементните и автодекрементните операции, посочена в таблицата, е на **логическо ниво**. В повечето компютърни архитектури има **много по-бързи** схеми за извършване на тези операции, различни от по-бавните схеми за събиране и изваждане!!!

Израз	Еквивалент
`y += x;`	`y = y + x;`
`x -= 5;`	`x = x - 5;`
`x /= y;`	`x = x / y;`
`price *= units + 1;`	`price = price * (units+1);`
`a++;`	Използвай a и след това `a=a+1;`
`++a;`	`a=a+1;` и след това използвай a
`a--;`	Използвай a и след това `a=a-1;`
`--a;`	`a=a-1;` и след това използвай a

Пример

- **Сорс код:**

```
// комбинирано присвояване
using System;
namespace AritmOp{
 class Program{
  static void Main(){
  int a, b=3;
   a = b;
   a+=2;
 //еквивалентно на израза a=a+2
   Console.WriteLine(a);
  }
 }
}
```

- **Изход:**

```
5
```

Инкремент и декремент (++, --)

- Внимателно използвайте префиксната и постфиксната форма на автоинкрементните и автодекрементните операции.

Пример 1	Пример 2

`x = 3;`
`y = ++x;`
`// x съдържа 4, y съдържа 4`
	`x = 3;`
	`y = x++;`
	`// x съдържа 4, y съдържа 3`

- В Пример 1 стойността, която се свързва с y, е стойността на x след като е увеличено.
- В Пример 2 стойността, която се свързва с y, е стойността на x преди да е увеличено.

Сравнения и логически изрази

- ***Операциите за сравняване***: == (еквивалент на = в математиката), **!=** (еквивалент на ≠), >, <, >= (еквивалент на ≥) и <= (еквивалент на ≤) са двуаргументни, като двата аргумента трябва да са изрази със сравними стойности (например числови). Резултатът от сравняване е **булева стойност**.
- Приоритетът на >, <, >= и <= е по-нисък от този на аритметичните операции и по-висок от този на == и **!=**.
- Аргументите на ***логическите операции*** **!** (***отрицание***, едноаргументна с префиксен запис), && (двуаргументната ***конюнкция/“и“***) и || (двуаргументната ***дизюнкция/“или“***) са булеви стойности и резултатът е булева стойност.
- Приоритетът на отрицанието е по-висок от приоритета на аритметичните операции, на конюнкцията – по-нисък от приоритета на операциите за сравняване, а на дизюнкцията – по нисък от приоритета на конюнкцията.
- Изразите, построени с операциите за сравняване и логическите операции, наричаме ***логически изрази***. Те се изчисляват до булева стойност и имат съществена роля за създаването на сложните конструкции на езика – условните оператори и операторите за цикъл.

Сравнения и логически изрази

- Два израза могат да се сравнят с използването на операциите за сравняване, когато стойностите им са от типове, позволяващи сравняване .
- Освен да се сравняват числа, може да се сравняват и низове като за целта се използва лексикографската (речниковата) подредба.
- Например, когато ни се налага да проверяваме дали две стойности са равни или една е по-голяма от друга стойност.
- Резултатът от такива операции е или истина или неистина (булевите стойности true и false).

Оператор	Описание
==	Равно на
!=	Различно от
<	по-малко от
>	по-голямо от
<=	по-малко от или равно на
>=	по-голямо от или равно на

Сравнения и логически изрази

- Например:

```
(7 == 5)      // оценява се на false
(5 > 4)       // оценява се на true
(3 != 2)      // оценява се на true
(6 >= 6)      // оценява се на true
(5 < 5)       // оценява се на false
```

- Позволява се да се сравняват, както числови константи, така и променливи.
- Нека предположим, че `a=2, b=3 и c=6`, следователно:

```
(a == 5)      // = false, тъй като a не е равно на 5
(a*b >= c)    // = true, тъй като (2*3 >= 6) е истина
(b+4 > a*c)   // = false, тъй като (3+4 > 2*6) не е истина
```

- Каква ще бъде стойността на израза `((b=2) == a)` за `a = 1` и `a = 2`?

Логически операции (!, &&, ||)

- Операцията && е булевата логическа операция ***конюнкция***, наричана още ***И***, която е истина, само тогава, когато и двата операнда са `true`, и `false` във всички останали случаи.
- Операцията || е булевата логическа операция ***дизюнкция***, наричана още ***(включващо) ИЛИ***, която е истина, ако поне един от операндите е `true`, и е `false` само когато двата операнда са `false`.
- В таблиците са показани резултатите от прилагане на двете операции при оценяване на логически изрази.

&& операция (И)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false
|| операция (ИЛИ)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Логически Операции (!, &&, ||)

- Операцията ! в C# е логическата операция отрицание (NOT), която четем „не е вярно, че ... “.
- Тя е едноаргументна, т.е. има само един операнд – логически израз и се изписва в префиксна форма. Резултатът от изпълнението на отрицанието е `false`, ако стойността на операнда е `true`, и `true` ако стойността на операнда е false.
- Пример:

```
!(5 == 5)    // = false, тъй като (5 == 5) е true
!(6 <= 4)    // = true, тъй като (6 <= 4) е false
!true        // = false
!false       // = true
```

```
C# не конвертира int to bool!!!
```

- За разлика от C++ в C# за представяне на логическите стойности `true` и `false` не се използват цели числа – 0 за `false` и различно от нула за `true`, **целите числа не могат да участват в логически изрази и да се интерпретират като булеви стойности.**

Логически Операции (!, &&, ||)

- Пример:

```
( (5 == 5) && (3 > 6) )
// оценява се на false (true && false)
( (5 == 5) || (3 > 6) )
// оценява се true (true || false)
```

- Когато използваме логически операции, C# оценява само това, което е необходимо от ляво на дясно, за да стигне до резултат, като игнорира всичко останало.
- Ето защо в последния пример ((5 == 5) || (3 > 6)), C# първо оценява стойността на израза 5 == 5, който е true и не проверява дали 3 > 6 е true или не.
- Това е известно като ***съкратено оценяване*** или ***късо-свързване***.

Операция	Съкратено оценяване или късо-свързване
&&	Ако изразът от ляво е false, целият резултат е false (изразът от дясно никога не се оценява).
||	Ако изразът от ляво е true, целият резултат е true (изразът от дясно никога не се оценява).

Пример за *съкратено оценяване*

```
class Program    {
        public static bool Abc()      {
            Console.WriteLine("Now in abc!!!");
            return true;
        }
        static void Main()  {
            int a=10, b=4;
        if((a > b) && Abc()) Console.WriteLine("Exp is true!");
            Console.WriteLine("b:"+b);
            Console.ReadKey();
        }
    }
```

- Това е известно като ***съкратено оценяване*** или ***късо-свързване***.

Логически Операции (!, &&, ||)

- Съкратеното оценяване е особено важно, когато изразът от дясно съдържа операция със странични ефекти, като промяна на стойности:

```
if ( (i < 10) && (++i < n) ) { /*...*/ }
// обърнете внимание, че в логическия израз i се инкрементира
```

- В примера, комбинираният логически израз ще увеличи стойността на i с едно, но само ако условието от ляво на знака && е true, в противен случай, условието от дясно (++i < n) никога не се оценява.
- Ако желаем да се пресметне целия логически израз, вместо || използваме | и вместо && използваме &.

Условна операция (... ?... :...)

- ***Условната операция*** :

```
условие ? резултат1 : резултат2
```

оценява логическия израз (или ***условие***), и дава в резултат една стойност, ако стойността на условието е true и друга, ако е false. Ако стойността на *условие* е true, стойност на израза е *резултат1*, в противен случай – *резултат2*.

```
7==5 ? 4 : 3
// стойност 3, тъй като 7 не е равно на 5.

7==5+2 ? 4 : 3
// стойност 4, тъй като 7 е равно на 5+2.

5>3 ? a : b
// стойността на a, тъй като 5 е по-голямо от 3.

a>b ? a : b
// стойността на по-голямото от a и b.
```

Условна операция (... ? ... : ...)

Пример:

```
// conditional operator
static void Main()
{
  int a,b,c;
  a=2;
  b=7;
  c = (a>b) ? a : b;
  Console.WriteLine(c);
}
```

Изход:

```
7
```

- В примера променливата **a** има стойност 2, а **b** – стойност 7, следователно при оценка на израза `(a > b)` резултатът е `false`.
- Затова първата стойност, която е специфицирана след въпросителната, се пренебрегва, за сметка на втората (изразът след двоеточието), която е **b** (със стойност 7).

Вложена условна операция

// Условният оператор е дясно-асоциативен, тоест вложен израз от вида:

a ? b : c ? d : e

се оценява по следния начин:

a ? b : (c ? d : e)

Можем да използваме следното мнемонично правило, за да запомним как се оценява условният оператор:

true ли е условието ? yes : no

Още операции

- ***Операцията за смяна на типа*** е двуаргументна, записана по специален начин – (*тип*) *израз*. Първият аргумент е всеки допустим от езика *тип*. След пресмятането на *израз*, стойността му се преобразува в указания тип. Например, ако искаме да превърнем резултата от пресмятане на целочислен израз в дробния тип double, можем да направим това с оператора:

```
double c = (double) b * b – 4 * a * c;
```

- Друг пример

```
int i; float f = 3.14;
i = (int) f;
```

  В примера стойността 3.14 се преобразува в цяло число (3); дробната част е загубена.

  Тази операция често се нарича „кастване“ от англ. **cast**

Неявно преобразуване на типа

- Когато се пресмятат изрази, в които участват променливи от различни (числови) типове, компилаторът добавя автоматично ***неявно преобразуване*** на типа на променливите.
- Преобразуването става винаги към по-общия тип.
- Пример:

```
3.14 + 7 – 2 / 3* 2.1
```

- Тъй като в израза участват както цели стойности, така и дробни, при оценяването му, всички цели ще се преобразуват в по-общия дробен тип.

Неявно преобразуване на тип

- Правила, по които се преобразуват типовете в C#:

Тип на данни	Преобразува се до тип на данни
char	int
unsigned char	int
short	int
unsigned short	unsigned int
enum	int
float	double

- Също така трябва да се има предвид, че ако в една операция участват два операнда, като единият е от целочислен тип, а другият е от реален, тогава целочислената стойност се преобразува до реално число в типа на реалния операнд.

Неявно преобразуване на типа

- В следващия пример променливата i е целочислена, f – реална от типа float и d – реална от типа double.
- За да се извърши операцията събиране, стойността на i ще се преобразува до типа float, след това ще се извърши събирането, след което, за да се извърши присвояването, резултатът от израза (число във float) ще се преобразува до типа double и тогава ще се присвои на d.

```
int i = 1;
float f = 5.56;
double d = 0.0;
d = f + i;
```

Неявно преобразуване на типа

- В езика C# е позволено и преобразуването на типовете от „по-големите” типове към „по-малките”, но в този случай е възможна загуба на информация.
- Например, ако искаме да присвоим стойност от тип double в променлива от тип int, ще загубим дробната част, ако има такава и ако цялата част на числото надхвърля допустимите стойности за типа int, то и тя няма да може да се запише.
- В такива случаи компилаторът извежда предупреждения (warnings).
- В следващия програмен текст са дадени трите варианта, които могат да се случат, при такова преобразуване.

  В първия вариант имаме загуба на дробната част на реално число, във втория – нямаме загуба, в третия – губим и дробната част и цялата част на числото, като в променливата sh се записва максималната допустима стойност за типа unsigned short.

```
int pi = 3.14159265;
long l = 12.0;
unsigned short sh = 788999.56;
Console.WriteLine(pi + " " + l + " " + sh;
```

- Изход:

```
3 12 65535
```

Примери

Пример 1:

```
char c = 96; int i = 0; double d = 1;
++c; i++; d--;
//c=97, i=1, d=0 double g = c * ++d - i--;
```

Резултат: c? d? i?

Пример 2:

```
int a = 5, b = 10, c = 15;
bool b=a<b && a<c;
bool h = a < b || c < b;
bool g = true;
b ? h ? !g ?
```

Резултат: ? ? ?

Пример 3:

```
(a >= 0) ? Console.WriteLine(a>=0) : Console.WriteLine(a<0);
```

Размер на типа

- Друга операция, която се записва по особен начин, е операцията за определяне ***размера*** в байтове на необходимата памет за допустим от езика тип. Нейното изписване също е особено `sizeof(`*`име на тип`*`).`
- Например операторът,

```
int a = sizeof(long);
```

  ще присвои на променливата `a` дължината в байтове на стандартния тип `long long`, с която е реализиран този тип в компилатора.

- А след присвояването

```
int x = sizeof(char);
```

  x приема стойност 1, тъй като char е тип, който най-често заема 1 байт в паметта.

- Когато казваме „допустим от компилатора тип“, това означава както стандартните типове на езика, така и създадените от програмиста.

Побитови операции (`&, |, ^, ~, <<, >>`)

- В паметта на компютъра всеки обект се разполага като двоично число (последователност от 0 и 1) в полета от 8, 16, 32 и т.н. бита.
- ***Побитовите операции*** интерпретират полетата **от цели типове** като съвкупност от битове. Те позволяват на програмиста да направи редица важни неща – да провери стойности на отделни битове, да промени стойност на бит или множество битове и т.н.
- Знаците на побитовите (наричани още ***поразрядни***) операции са `&, |, ^, ~, >>, <<`.
- Операциите `>>` и `<<` са зависими от това дали целият тип е знаков или беззнаков. В първия случай знаковият бит не участва в предписаното действие.

Побитови операции

Знак на операция	Значение	Описание
&	AND	Побитово И
|	OR	Побитово включващо ИЛИ
^	XOR	Побитово изключващо ИЛИ
~	NOT	Побитово логическо ОТРИЦАНИЕ
<<	SHL	Побитово изместване на ляво
>>	SHR	Побитово изместване на дясно

Побитови операции

- Правилата за извършване на побитовите операции И, (влючващо ИЛИ) и ОТРИЦАНИЕ са както на логическите И, ИЛИ и ОТРИЦАНИЕ като 0 е `false`, а 1 – `true`.
- Таблицата на изключващото ИЛИ (наричано и ***събиране по модул 2***) е вдясно.

изключващо ИЛИ		
`a`	`b`	`a ^ b`
1	1	0
1	0	1
0	1	1
0	0	0

- При операцията за изместване на ляво `a << k` допустимите за изместване битове се изместват на `k` позиции вляво, а освободените отдясно се запълват с нули.
- При операцията за изместване на дясно `a >> k` допустимите за изместване битове се изместват на `k` позиции вдясно, а освободените отляво битове се запълват с нули.
- Изместването на ляво на `k` бита `a << k` е **много бързо** умножаване на `a` по 2^k.
- Изместването на дясно на `k` бита `a >> k` е **много бързо** деление на `a` на 2^k.

Побитови операции

- & (поразрядно логическо И):

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

- Пример:

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

- Използва се обикновено за нулиране на отделни битове в числената стойност на резултата.
- Ако някой бит в единия или другия операнд има нулева стойност, съответния по позиция бит в резултата също ще има нулева стойност.

Побитови операции

- | (побитово логическо ИЛИ):

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

- Пример:

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

- Операцията може да се разглежда като обратна на побитовото И, следователно отделни битове в резултата, могат да се направят единици.

Побитови операции

- ^ (побитово логическо ИЗКЛЮЧВАЩО ИЛИ (XOR))

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

- Пример:

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

- Определени битове в резултата се установяват като единица само когато съответните битове в операндите са различни.

Побитови операции

- ~ (побитово ОТРИЦАНИЕ) – едноместна с префиксен запис ~a.
- При изпълнението на операцията, всеки бит на операнда a се променя в противоположната си стойност, т.е. 0 става 1, а 1 става 0.
- От това веднага следва, че двукратното изпълнение на операцията възстановява числото в първоначален вид.
- **Пример**:

```
~11010101          ~11010101
 ----------         ----------
 00101010          ~00101010
                    ----------
                    11010101
```

Побитови операции

- >> (побитово изместване на дясно)
- Това е двуаргументна операция. Битовете на първия операнд (цяло число) се изместват на дясно на брой битове, който е стойността на втория операнд (също цяло число). Ако цялото е със знак, знаковият бит не участва в изместването.
- **Пример**:

```
01111000 >> 3          -1111000 >> 3
--------               --------
00001111               -0001111
```

- Ефектът от изместването надясно е, че все едно три пъти сме разделили числото на две. Например 120 >> 3 ще върне резултат 15 ((120/2)/2)/2=120/2^8.

Побитови операции

- << (побитово отместване на ляво)
- Обратната операция на изместването на дясно. Ако цялото е със знак, знаковият бит не участва в изместването.
- Пример:

```
00001111 << 3          -0001111 << 3
--------               --------
01111000               -1111000
```

- Ефектът от операцията е равносилен на умножение на числото три пъти по две. Например 10 << 3 ще върне резултат 80 ($10*2*2*2 = 10.2^3$).

Побитови операции

- При операциите изместване вляво или вдясно, изместените извън полето битове се губят.
- Следният пример го демонстрира:

```
00001111 >> 1
--------
00000111 << 1
--------
00001110
```

Пример:

```
using System;
namespace OperatorsAppl {
   class Program {
      static void Main(string[] args) {
         int a = 60;            /* 60 = 0011 1100 */
         int b = 13;            /* 13 = 0000 1101 */
         int c = 0;
         c = a & b;             /* 12 = 0000 1100 */
         Console.WriteLine("Line 1 - Value of c is {0}", c );
         c = a | b;             /* 61 = 0011 1101 */
         Console.WriteLine("Line 2 - Value of c is {0}", c);
          c = a ^ b;             /* 49 = 0011 0001 */
         Console.WriteLine("Line 3 - Value of c is {0}", c);
          c = ~a;                /*-61 = 1100 0011 */
         Console.WriteLine("Line 4 - Value of c is {0}", c);
           c = a << 2;       /* 240 = 1111 0000 */
         Console.WriteLine("Line 5 - Value of c is {0}", c);
          c = a >> 2;       /* 15 = 0000 1111 */
         Console.WriteLine("Line 6 - Value of c is {0}", c);
         Console.ReadLine();
      }
   }
}
```

Операции и оператори

- След като нарекохме лексемите **думи** на езика за програмиране, можем да продължим аналогията с текстовете на естествен език и да наречем операторите на езика ***изречения***.
- Съответният английски термин е „statement” – твърдение, изявление.
- Другият възможен превод на statement е „инструкция”, но по традиция, в терминологията на български „инструкция” употребяваме при програмите, написани на асемблер или на машинен език, а запазваме термина „оператор” за програмите, написани на езици от високо ниво.
- Известно объркване може да внесе използването в професионалния жаргон на английския термин „operator”, който означава „операция”, като +, *, и др.
- В лекциите ще се стараем да съблюдаваме следното съответствие на българските и английски термини: „оператор” = „statement”, „израз” = „expression”, „операция” = „operator”.

Приоритет и асоциативност

Приоритет	Вид	Знак на операцията	Описание	Асоциативност
1	Област на видимост	`::`	Достъп до пространство от имена	Лява
2	Постфиксни (унарни)	`++ --`	Постфиксен инкремент/ декремент	Лява
		`( )`	Задаване параметри на функция	
		`[ ]`	Индексиране	
		`. ->`	Достъп до член на структура	
3	Префиксни (унарни)	`++ --`	Префиксен инкремент / декремент	Дясна
		`~ !`	Побитово NOT / логическо NOT	
		`+ -`	Унарен префиксен знак	
		`& *`	Референция / дереференция	
		`new delete`	Алокация / делокация	
		`sizeof`	Размер на параметъра	
		`(тип)`	C-стил на препобразуване на типа	
4	Указатели	`.* ->*`	Достъп до указател	Лява

Приоритет на операциите

Приоритет	Вид	Знак на операцията	Описание	Асоциативност
5	Приоритетни Аритметични	* / %	Умножение, деление, остатък при деление	Лява
6	Неприоритетни аритметични	+ -(2)	Събиране, изваждане	Лява
7	Побитово изместване	<< >>	Изместване на ляво и на дясно	Лява
8	Сравняване	< > <= >=	Проверка за наредба	Лява
9	Сравняване	== !=	Проверка за равенство или различие	Лява

Приоритет на операциите

Приоритет	Вид	Знак на операцията	Описание	Асоциативност
10	Побитова	`&`	Побитово И	Лява
11	Побитова	`^`	Изключващо ИЛИ	Лява
12	Побитова	`|`	Включващо ИЛИ	Лява
13	Логическа	`&&`	Логическо И	Лява
14	Логическа	`||`	Логическо ИЛИ	Лява
15	Условна операция	`?:`	Условна операция	
16	Присвоявания	`= *= /= %= += -= >>= <<= &= ^= |=`	Присвояване и комбинирано присвояване	Дясна
17	Обединяваща	`,`	Съчленяване на изрази	Лява

Размяна на стойности

- Размяната на стойности може да се реализира или чрез трета променлива или чрез аритметични или побитови операции.
- Често при решаване на различни задачи, се налага осъществяване на размяна на стойности. Ако не разполагаме с достатъчно памет, за да заделим място за трета променлива, е удачно да използваме аритметични или други операции за тази цел.
- Чрез събиране и изваждане:
```
a=a+b;
b=a-b;
a=a-b;
```
- Чрез умножение и деление:
```
a=a*b;
b=a/b;
a=a/b;
```

!!! ***Има и други варианти. Кои са те? Кой от горните два варианта е по-надежден и по-ефективен и защо?***

Домашна работа

Задача 1. Програма, която отпечатва 1 или 0 в зависимост от това дали зададена точка (Xt,Yt) лежи или не лежи в зададена окръжност (Xc,Yc,R). Използвайте тип на променливите dooble.

Задача 2. Въведете изречение, което съдържа няколко думи, разделени с интервал и завършващо с . (точка). Изведете броя на думите и размера на най-дългата дума.

Задача 3. Използвайте вградената функция Math.Pow за да пресметнете стойността на числото ***double baseNumber***, повдигнато на степен ***double powerNumber.*** Представете примери с положителни и отрицателни стойности на двете променливи.

Задача 4. Въведете 5 цели числа. Изведете сумата на четните и сумата на нечетните.

Задача 5. Въведете 5 цели числа. Изведете броя на числата, които са по-малки от средната стойност и броя на числата по-големи от средната стойност. Какво става с тези, които са равни на средната стойност?

Изгревът е близо

Благодаря за вниманието!

Пано Панов

Готов съм да отговарям на вашите въпроси