Аритметични операции. Изрази. Преобразуване на типове

В тази част, ще разгледаме аритметичните и логически операции, както, и включването им в изрази.

В следващата таблица са дадени всички възможни операции в езикът C и C++.

Символ	Предназначение
Аритметични оператори	
+	Събиране
-	Изваждане
*	Умножение
/	Деление
%	Делене по модул
Оператори за отношения	
<	По-малко
<=	По-малко или равно
>	По-голямо
>=	По-голямо или равно
==	Равно
!=	Не равно
Присвояване	
=	Присв яване
+=	Увеличаване с число и присвояване
-=	Намаля с число и присвояване
*=	Умножаване с число и присвояване
/=	Делене на число и присвояване
%=	делене по модул с число и присвояване
<<=	Изместване в ляво и присвояване
>>=	Изместване в дясно и рисвояване
&=	Двоично AND с присвояване
^=	Двоично изключващо OR (XOR) с присвояване
|=	Двоично OR с присвояване
Инкрементиране и декрементиране	
++	Инкрементиране
--	Декрементиране
Побитови операции	
&	Побитов AND
^	Побитово XOR
|	Побитово OR
<<	П еместване на ляво
>>	Преместване на дясно
~	Допълване до 1
Логически операции	
&&	Логическо AND
||	Логическо OR
!	Логическо NOT

Адресни операции	
&	Адрес на променлива
*	Деклариране на указател или връщане на съдържанието на клетка сочена от указател
**	Деклариране на указател към указател
:>	Базов адрес Пример: myseg:>bp На указателя bp се присвоява отместване и сегмент специ ицирани в myseg.
Логическо оператор за присвояване	
? :	Логическо присвояване **Пример:** (val >= 0) ? val:-val; Ако val е > 0 резултатът ще е истина. Иначе резултатът ще бъде неистина.
,	Последователно изпълнение на изрази
Специални символи	
( )	Групиране в изрази и описание на функции
[ ]	Индексиране и описание на масиви
.	Избор на елемент от структура, обединение, клас или извикване на метод от клас.
->	клас чрез указател.
(type)	Промяна на тип
sizeof	Размер в байтове
Друго оператори в C++	
::	Деклариране на принадлежност
&	Псевдоним
.*	Указател към елемент
->*	Указател към елемент

Прецеденти при присвояването на операндите.

В тази таблица ще бъдат показани прецедентите възникващи при присвояването на стойност. По подразбиране присвояването се извършва от дясно на ляво, но има случаи когато това правило не важи. В следващата таблица са посочени тези прецеденти.

Трябва да се отбележи, че приоритетът на операциите намаля в посока от първият елемент от таблицата към последния.

Символ	Предназначение	Присвояване
::	Принадлежност	Няма
++	Постфиксно инкрементиране	От ляво на дясно
--	Постфиксно декрементиране	
( )	Извикване на функ ия	
[ ]	Елемент на масив	

->	Указател към елемент от структура	
.	Към структура или обединение	
++	Префиксно инкрементиране	От дясно на ляво
--	Префиксно декрементиране	
:>	База	
!	Логическо NOT	
~	По битово NOT	
-	Унарен минус	
+	Унарен плюс	
&	Адрес	
*	Съдържание на клетка сочена от указател	
sizeof	Размер в байтове	
New	Заделяне на памет	
delete	Освобождаване на динамично заделената памет	
(type)	Смяна на тип [пример, (**float**) I]	
.*	Указател(Обект)	От ляво на дясно
->*	Указател(Указа ел)	
*	Умножение	От ляво на дясно
/	Деление	
%	делене по модул	
+	Събиране	От ляво на дясно
-	Изваждане	
<<	Преместване на ляво	От ляво на дясно
>>	Преместване на дясно	
<	По-малко	От ляво на дясно
<	По-малко или равно	
>	По-голямо	
>=	По-голямо или равно	
==	Равно	От ляво на дясно
!=	Не равно	
&	Побитово AND	От ля о на дясно
^	Побитово XOR	От ляво на дясно
|	Побитово OR	От ляво на дясно
&&	Логическо AND	От ляво на дясно
||	Логическо OR	От ляво на дясно
? :	Логическо присвояване	От дясно на ляво
=	Присвояване	От дясно на ляво
*=, /=,	Комбинирано присвояване	
%=, +=,		
-=, <<=,		
>>=,		
&=, ^=,		
|=		
,	Запетая	От ляво на дясно

Изрази

Израза е конструкция, която връща стойност, след като той е бил оценен. Изразите са съставени от оператори и операнди. Операнда може да бъде константа, или друг израз.

Пример:

```
b + c
( a - b + c ) * cos(b)
sin(a) * cos(a)
```

Оператор за присвояване

Операторът за присвояване задава стойността на израза от дясно на променливата от ляво. Той се записва с единично равно =. Задължително след израза се поставя ;. Като операнд от ляво може да бъде зададена променлива, елемент на масив или входно/изходен порт. Не може от ляво на оператора за присвояване да се поставя константа или израз.

Пример:

```
Var1 = Var2 + 2; // Var1 ⇓ (Var2 + 2)
```

След като се изпълни операцията Var1 ще съдържа стойността на Var2 увеличена с 2

Аритметични операции в C++

Мисля че използването на аритметични операции в програмирането е много по-лесно от колкото в математиката. Нас ни интересуват следните аритметични операции:

Символ	Предназначение	Използване
+	Събиране	Sum = a + b;
-	Изваждане	Sub = a - b;
*	Умножение	Mul = a * b;
/	Деление	Div = a / b;
%	Делене по модул (остатък от деление)	Mod = a % b;

Почти всеки програмен код в C++ използва под една или друга форма аритметични операции.

Следващия пример показва примерно използване на аритметичните операции.

```cpp
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
  float Sum, Sub, Mul, Div, Mod; // Обявяване на променливите чрез
запетая
  float a1; // Отделно деклариране на променливата a1
  float a2; // Отделно деклариране на променливата a2
  setlocale(LC_ALL, "Bulgarian");
  cout << "Въведи първото число: ";
  cin >> a1;
  cout << "Въведи второто число: ";
  cin >> a2;
  Sum  =  a1 + a2;  // операция събиране
  cout << a1 << "+" << a2 << "=" << Sum  << endl;
  Sub  =  a1 - a2;  // операция извеждане
  cout << a1 << "-" << a2 << "=" << Sub << endl;
  Mul  =  a1 * a2;  // операция умножение
  cout << a1 << "*" << a2 << "=" << Mul << endl;
  Div  = a1 / a2;  // операция деление
  cout << a1 << "/" << a2 << "=" << Div  << endl;
  return 0;
}
```

На ред 5 са декларирани променливи с имена `Sum`, `Sub`, `Mul`, `Div` от тип `float` – реален тип данни (то ест тези променливи могат да приемат стойности от вида: 0.99; 3.0; 21.6; –43.15; 345.342).

`Запомнете: Всяка променлива може да бъде използвана едва след като бъде декларирана (обявена)`. Променливите могат да се инициализират по време на декларирането (задаване на начална стойност). Това действие се нарича `"Дефиниране на променливата"`. Например:

```cpp
float Sum = 10;
```

Това означава, че на променлива с име `Sum` и тип `float` се присвоява стойност **10**. Променливите могат да се декларират на един ред, като се отделят със запетая или всяка променлива на отделен ред. На редове 10 и 12 се въвеждат данни от клавиатурата, които се записват в променливите a1 и a2.

```cpp
cin >> a1; // Въвеждане на първото число и записване на в a1

cin >> a2; // Въвеждане на второто число и записване на в a2
```

Символът `>>` е оператор за извличане на данни от входен поток. Дадения символ се използва съвместно с оператора за вход **`cin`**.

Аритметичните операции се изпълняват от редове:

```
Sum  =  a1 + a2;  // операция събиране
Sub  =  a1 - a2;  // операция извеждане
Mul  =  a1 * a2;  // операция умножение
Div  =  a1 / a2;  // операция деление
```

Извеждането на резултата се извърва от операторите

```
cout << a1 << "+" << a2 << "=" << Sum  << endl;
cout << a1 << "-" << a2 << "=" << Sub << endl;
cout << a1 << "*" << a2 << "=" << Mul << endl;
cout << a1 << "/" << a2 << "=" << Div  << endl;
```

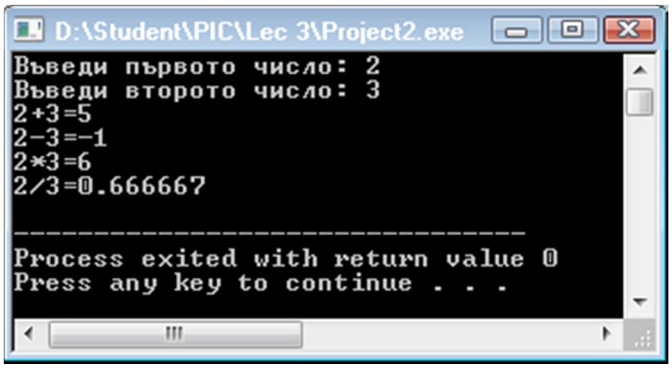
Както в математиката, така и в C++ с помощта на скоби могат да се формира различни математически изрази с различна сложност. например израза `((a+b)*c)-d` се изпълнява в следната последователно:

Първото действие е : `a+b`;

Второто действие е умножението с `c`;

Третото действие е изваждането на `d`;

Резултатът от изпълнението на програмата е показан на фиг. 1.



Фиг. 1. Аритметични операции в C++

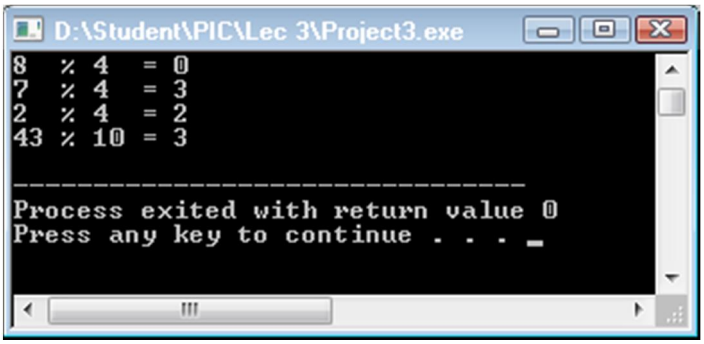
Остатък от целочислено деление `%`

Ще разгледаме още една аритметична операция: Остатък от целочислено деление `%`. За целта ще разгледаме подробно следната примерна програма

```
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
    cout << "8  % 4  = " << 8  % 4   << endl;
    cout << "7  % 4  = " << 7  % 4   << endl;
    cout << "2  % 4  = " << 2  % 4   << endl;
    cout << "43 % 10 = " << 43 % 10  << endl;
    return 0;
}
```

Резултатът от действието на операцията "остатък от целочислено деление" е показан на фиг.2



Фиг. 2. Аритметична операции % C++

·1-ви случай: Четворката може да се помести два пъти в 8 . Следователно остатъка от делението ще бъде 0;

2-ри случай: Четворката може да се помести един път в осмицата. Частта която неможе да се раздели е 3. Тази тройка е резултатът;

·3-ти случай: Четворката е по-голяма от двойката. Следователно резултатът ще бъде равен на делимото, тоест 2;

4-ти случай: В 43 се съдържат четири десетки. Остатъка е 3.

Надявам се че тези четири разяснения са били достатъчни за да разберете принципа на работа на оператора. Ако не сте успели да го разберете, заменете някой от числата и вижте какъв ще бъде резултатът.

Операции инкремент и декремент в C++

Инкремент `++` – това е увеличение с единица.

Декремент `--` – това е намаляване с единица.

Операциите декремент и инкремент с лекота се заменят с аритметични операции събиране и изваждане.

Операторите `++` и `--` могат да бъдат записани преди променливата или след променливата. Тяхното положение определя кога ще бъде изпълнено действието при участие в израз.

Възможните синтаксиси са:

```
++VariableName;  // префиксен инкремент (преинкремент)
VariableName++;  // постфиксен инкремент (постинкремент)
--VariableName;  // префиксен декремент (предекремент)
VariableName --; // постфиксен декремент (постдекремент)
```

Ако имаме префиксен инкремент (`++VariableName`) при участие в израз първо се извършва увеличаването с 1 на променливата, след което новата стойност се използва в израза за изчисление.

За разлика от посфиксния инкремент (**++VariableName**) който при участие в израз първо стойността на променливата се използва в израза за изчисление след което се извършва увеличаването с 1 на променливата.

В следващата таблица и променлива се демонстрират тези свойства на операторите ++ и --.

Таблица 2: Операции инкремент и декремент в C++

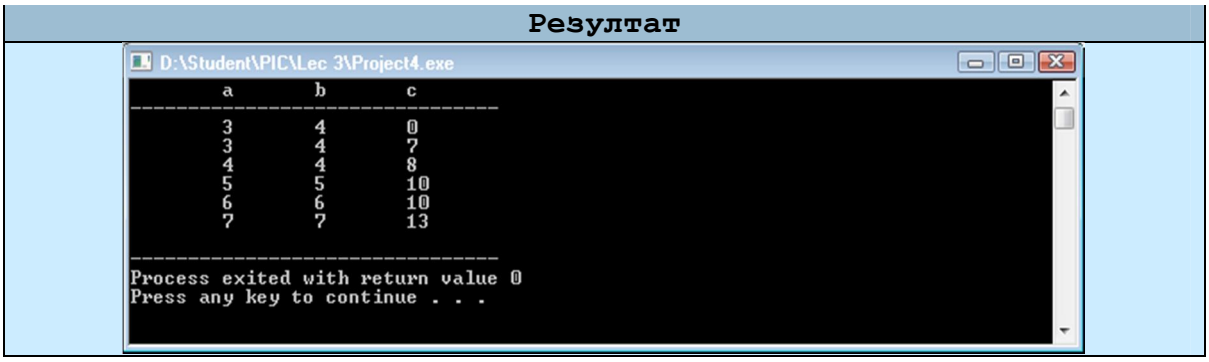
Операция	Символ	Пример	Кратко пояснение
преинкремент	++	cout << ++value;	Първо се увеличава стойността на **value** с 1, след което се отпечатва резултата с **cout**
предекремент	--	cout << --value;	Първо се намалява стойността на **value** с 1, след което се отпечатва резултата с **cout**
постинкремент	++	cout << value++;	Първо се отпечатва стойността на **value** след което се увеличава стойността на **value** с 1
постдекремент	--	cout << value--;	Първо се отпечатва стойността на **value** след което се намалява стойността на **value** с 1

Пример:

Програмен код

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
  int a,b,c; // Деклариране на променливите a, b и c
  a = 3; b = 4; c = 0; // Инициализиране на променливите a, b и c
  cout <<  "\ta" << "\tb" << "\tc" << endl;
  cout << "--------------------------------"<< endl;
  cout << "\t" << a  << "\t" << b << "\t" << c << endl;
  c = a + b;
  cout << "\t" << a  << "\t" << b << "\t" << c << endl;
  c = ++a + b;
  cout << "\t" << a  << "\t" << b << "\t" << c << endl;
  c = ++a + ++b;
  cout << "\t" << a  << "\t" << b << "\t" << c << endl;
  c = a++ + b++;
  cout << "\t" << a  << "\t" << b << "\t" << c << endl;
  c = a++ + ++b;
  cout << "\t" << a  << "\t" << b << "\t" << c << endl;
  return 0;
```

Резултат



На ред 8 от програмата се извеждат началните стойности на променливите a,b и c. В разпечатката на резултата това са символите на първият ред след реда с тиретата.

След изпълнението на израза c = a + b; в **c** се записва сумата на **a** и **c**. По специални са следващите оператори които ще разгледаме по-подробно.

```
c = ++a + b;
```

Този израз е съставен от два по-прости израза. При него първо стойността на **a** се увеличава с 1, след което **a** и **b** се сумират и резултатът се присвоява на **c**.

```
a = a + 1;  c =  a  + b;
```

Изразът

```
c = ++a + ++b;
```

е съставен от следната последователност от изрази:

```
a = a + 1;
b = b + 1;
c = a  + b;
```

Изразът

```
c = a++ + ++b;
```

е съставен от изразите:

```
b = b + 1;
c = a + b;
a = a + 1;
```

Изразът

```
c = a++ + b++;
```

е съставен от изразите:

```
c = a + b;
b = b + 1;
a = a + 1;
```

Действието на оператор **--** е същото като това на **++** с разлика на това, че стойността на операнда се намалява с 1;

Оператор	Действие
+=	Увеличаване с число и присвояване
-=	Намаля с число и присвояване
***=**	Умножаване с число и присвояване
/=	Делене на число и присвояване
%=	делене по модул с число и присвояване

Операторите от таблицата са съставни оператори. Използват се по следният начин:

```
c += 2;  // c = c + 2;
c -= 2;  // c = c - 2;
c *= 2;  // c = c * 2;
c /= 2;  // c = c / 2;
c %= 2;  // c = c % 2;
```

Като коментар са дадени съответстващите разширени аритметични изрази.

Спазването на конвенцията за типове на операторите е много важно при съставянето на изразите. В следващата таблица са показани възможните преобразования на типове данни.

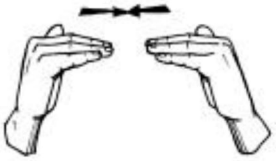
Синтаксис	Тип на операнда			Резултат
	Res	Op 1	Op 2	
Res = 8 / 3	int	int	int	2
Res = 8 / 3	float	int	int	2.0
Res = 8.0 / 3	float	float	int	2.6667
Res = 8.0 / 3	int	float	int	*Error

* Състояние на грешка: Не може да се конвертира **float** в **int**

Както се вижда резултатът зависи от видът на данните.

Важно е от ляво на оператора за присвояване да стои операнд от тип с по-голям размер спрямо операнда намиращ се от дясно.

Left DataTYPE = Right DataTYPE



Конвертиране на типовете данни

С поддържа стандартните механизми за автоматично превръщане на данни от един в друг тип. Типът на резултата от преобразуванията зависят от типовете на операндите участващи в израза. Трябва да се

отбележи, че при промяна типа на данните е възможно да се загуби част от информацията. Това се получава когато се прави преобразуване от тип с по-малък размер в тип с по-голям размер. C++ позволява участието на елементи от различен тип в един израз, което налага да се съблюдават следните правилата за промяна на типа.

- Присвояването на символна константа на обект от цял тип дава пълно 16-битово присвояване, тъй като символните константи (едно- или двусимволни) се представят в 16 бита.
- Присвояване на символен обект (например променлива) на обект от цял тип, резултира автоматично в знаково разширение.
- Обектите от тип **unsigned char** винаги поставят в старшия байт нула при превръщане в тип **int**.
- Стойности от тип **enum** се превръщат в цели без модификация.
- Аналогично стойности тип **int** се превръщат директно в тип **enum**.
- Превръщането между стойности **enum** и символен тип става както между **int** и символен тип.
- Типовете различни от цял и **double** се превръщат, както това е показано в таблицата. Така всеки две стойности, свързани с даден оператор, са от тип **int** (включително модификаторите **long** и **unsigned**) или **double**.
- Ако единият операнд е от тип **double**, то и другият се преобразува в **double**.
- Ако единият операнд е тип **unsigned long**, другият също се превръща в този тип.
- Ако единият операнд е тип **long**, другият също се превръща в тип **long**.
- Ако единият операнд е тип **unsigned** , другият се превръща също в този тип.
- Резултатът от израза е от типа на операндите.

Data TYPE	bool	short	int	long	unsigned short	unsigned	float	double
bool	bool	x	x	x	x	x	x	x
short	X		int	long	unsigned short	unsigned	float	double
int	X	int		long	int	unsigned	float	double
long	X	long	long		long	unsigned	float	double
unsigned short	X	unsigned short	int	long		unsigned	float	double
unsigned	X	unsigned	unsigned	long	long		float	double
float	X	float	float	float	float	float		double
double	X	double	double	double	double	double	double	

Ако променливите имат по същия тип размер, те се конвертират в **"unsigned"** тип.

Пример:

```
INT_Result = INT_Varl  +  SINT_Var2;
  /*[int]       [int]     [short int] */
```

Променливата SINT_Var2 първо се преобразува в **int** след което участва в операцията.

Явно преобразуване на типове

Правилата за преобразуване са еднакви като при typecast. Например:

```
INT_Totaltteight = INTJtfeight1 + INTJtfeightS;
  /*    [int]          [int]           [int] */
```

Това изглежда добро на първи поглед, но стойността на сумата (INT_Weight1 + INT_Weight2) можеше да бъде по-висока отколкото е размерът на **int**. За това е необходимо резултатът да се помести в променлива с по-голям размер а преобразуването да стане явно.

```
DINTJTotalWeight = (long long)INT_Weight1 + (long long)INT_tteight2;
/*     [long long]          [int]                    [int ] */
```

Пример:

```
void main() {
    int a,b;
    long l;
    double d;
    a = 10;             // резултат  signed int
    b = a + 20;         // резултат  signed int
    l = (a + b ) * 0.1; // резултат  long int
    d = 123e6;          // резултат double
    l =(long)( d + l);  // резултат long int
}
```

В първите четири израза присвояванията са коректни и няма загуба на информация. При последният израз операцията (d + l), дава като резултат число от тип **double**. След което се присвоява на променлива от тип **long int**, посредством явна промяна на типа. При тази операция се губи голяма част от информацията поради това, че **double** е с по-голям размер от **long int**. Обърнете внимание на този запис

Съставен оператор в език C/C++.

Разгледаните досега оператори са прости оператори. Те се състоят от една синтактична конструкция и завършват със знака за край на оператор `';'`. Понякога в алгоритмите се налага последователно изпълнение на няколко прости оператора, които образуват една обща логическа част. Обединението на няколко прости оператора в една логическа група води до дефиниране на съставен оператор. В езика C/C++ за задаване на съставен оператор се използват средни скоби `"{}"` В следващия фрагмент е дефиниран един съставен оператор.

```
{
    d = -b*b-4*a*c ;
    d = sqrt(d) ;
}
```

След `}` в съставния оператор не се поставя знакът за край на оператор `';'`. Поставянето му не е грешка в програмите на C/C++, защото се предполага, че е използван празен оператор. **`Празен оператор`** е такъв оператор, който не изпълнява никакви команди и операции. Той съдържа само знака за край на оператор `';'`. Напълно допустим е например запис от типа:

```
d = -b*b-4*a*c ; ; ; Тук има 2 празни оператора.
```