عملگرها و عبارات

هر برنامه از statement ها تشکیل شده است و statement از عبارات (expressions) و عملگر (operators) تشکیل شده است. در این بخش یاد خواهید گرفت statement و expression چیست، عملگرهای C++ کدامند و الویت آنها چگونه است.

Statement
Expression
عملگرها
الویت عملگرها
تبدیل نوع

Statements

یک جمله (statement) یک خط منفرد از کد است که عملي را انجام مي دهد. در C++ معمولا هر جمله در یک خط نوشته می شود، البته بعضی از جملات به چند خط تقسیم می شوند.

جملات همیشه به یک سمیکولن (;) ختم می شوند. (به استثنای راهنماهای کامپایلر مانند #define و #include).

مثال. چند جمله در C++.

```
int i_accnum;
i_accnum=55555;
i_accnum=i_accnum+5;
```

White space

خطوط خالی و فاصله ها در برنامه white space نام دارند. کامپایلرهای C++ نسبت به خطوط خالی حساس نیستند و وقتی کد برنامه را می خوانند در جستجوی کاراکترهای جمله و ختم شدن به سمیکولن هستند و فاصله ها را ندیده می گیرند (ثابت های رشته ای مستثنا هستند). White space امکان فرمت بندی کد برنامه با یک طرح استاندارد برای فاصله گذاری را می دهد که باعث می شود کد برنامه خواناتر شود بدون اینکه روی اجرای آن تاثیر بگذارد. هر برنامه نویسی استیل خود را ممکن است داشته باشد مهم این است که کد برنامه خوانا باشد.

مثال. جملات زیر با هم معادل هستند.

```
x=2+3;

x = 2 + 3;

x        =
2
+
3;
```

White space ها در ثابت های رشته ای که بین علامت گیومه (") قرار می گیرند مجاز نیستند. اگر می خواهید یک رشته را به دو خط بشکنید باید از کاراکتر (\) استفاده کنید.

مثال. جمله زیر زیر درست است.

```
cout<<"Hello \
world!";
```

جملات پوچ

یک سمیکولن تنها در یک خط خالی یک جمله پوچ (null statement) را می سازد که عملی انجام نمی دهد. گاهی جملات پوچ می توانند مفید باشند.

جملات ترکیبی

جمله ترکیبی (compound statement) که بلاک (block) هم نامیده می شود از یک یا چند جمله که بین آکولاد محصور شده اند تشیل شده است.

بلاک می تواند جایگزین هر جمله ای در برنامه بشود. برای شروع بهتر است آکولاد بلاک در خط جداگانه ای باشد تا تشخیص اینکه جا افتاده است آسان تر باشد.

مثال. یک بلاک از کد.

```
{
    cout<<"Hello, ";
    cout<<"world!";
}
```

Expressions

یک عبارت (expression) ارزیابی شده و نتیجه آن به متغیری نسبت داده می شود. هر عبارت ممکن است شامل متغیر، ثابت، عملگر و پرانتز باشد.

عبارات ساده

ساده ترین عبارت از یک بخش تشکیل شده است: یک متغیر، ثابت واقعی یا ثابت سمبلیک.

مثال. هرکدام از موارد زیر یک عبارت ساده هستند.

PI
20
rate
-1.25

عبارات کامل

یک عبارت کامل (Complex expression) از عبارات ساده که با عملگر به هم مربوط شده اند تشکیل شده است.

مثال. چند عبارت کامل.

```
2 + 8
1.25 / 8 + 5 * rate + rate * rate / cost

x = a + 10;
y = x = a + 10;
x = 6 + (y = 4 + 5);
```

وقتی عبارتی شامل چند عملگر است ارزیابی عبارت وابسته به الویت عملگرهاست.

عملگرها

عملگرها (operator) سمبل هائی هستند که به کامپیوتر فرمان می دهند عملی روی یک یا چند عملوند انجام دهد. عملگرها به یک ،دو یا سه عملوند (operand) نیاز دارند تا روی آن عمل کنند.

عملگرهای C++ در چند گروه قرار می گیرند:

- عملگر انتساب
- عملگرهای ریاضی
- عملگرهای رابطه ای
- عملگرهای منطقی

وقتی عبارتی حاوی بیش از یک عملگر باشد به قواعدی نیاز است که ترتیب انجام عملگرها را تعیین کند. این قواعد الویت عملگر (operator precedence) نامیده می شود. هر عملگر الویت خاصی دارد. عملگرهائی که الویت بیشتر دارند اول اجرا می شوند.

با پرانتزها می توان الویت عملگرها را درون عبارت تغییر داد. ارزیابی از داخلی ترین پرانتز شروع می شود. از پرانتز برای روشن تر شدن ترتیب ارزیابی عملگرها استفاده کنید.

اگر عبارت شامل چند عملگر با الویت یکسان باشد عملگرها از چپ به راست ارزیابی می شوند.

بهتر است برای جلوگیری از ابهام یک عبارت پیچیده به چند عبارت ساده تر شکسته شود.

عملگر انتساب

عملگر انتساب (assignment operator) علامت مساوی (=) است. به معنی این است که عبارت سمت راست مساوی (یا rvalue) را به متغیر سمت چپ مساوی (یا lvalue) تخصیص بدهد. فرم کلی آن به شکل زیر است:

```
variable = expression;
```

عملگرهای ریاضی

عملگرهای ریاضی (mathematical operators) عملیات محاسباتی مانند جمع و تفریق را انجام می دهند.

C++ دو عملگر یکتائی و پنج عملگر دوتائی دارد. عملگرهای یکتائی تنها یک آرگومان دارد مانند ++ و - - یعنی تنها روی یک متغیر عمل می کنند. عملگرهای دوتائی روی دو مقدار کار می کنند مثل + و - .

عملگرهای ریاضی یکتائی C++

عملگر	سمبل	عمل	مثال
Increment	++	عملوند را یک واحد افزایش می دهد	++x, x++
Decrement	--	عملوند را یک واحد کاهش می دهد	--x, x--

عملگرهای ریاضی دوتائی C++

عملگر	سمبل	عمل	مثال
Addition	+	جمع دو عملوند	x + y
Subtraction	-	تفریق عملوند اول از عملوند دوم	x - y
Multiplication	*	ضرب دو عملوند	x * y
Division	/	تقسیم عملوند اول بر علموند دوم	x / y
Modulus	%	باقیمانده صحیح عملوند اول بر عملوند دوم	x % y

عملگرهای افزایش و کاهش تنها با متغیرها می توانند استفاده شوند و با ثابت مجاز نیستند. این عملگرها یک واحد از عملوند کم یا یک واحد به آن اضافه می کنند. یعنی عبارت ++x; معادل x=x+1; و عبارت --y; معادل y=y-1; است.

توجه کنید که عملگرهای افزایش و کاهش می توانند قبل یا بعد از علموند خود قرار گیرند. الویت ++ و - - بستگي به محل متغیر دارد. وقتی عملگر بعنوان پیشوند استفاده می شوند عملوند خود را قبل از استفاده تغییر می دهند و وقتی به عنوان پسوند استفاده می شوند عملوند خود را بعد از استفاده از آن تغییر می دهند.

مثال. به دو مثال زیر توجه کنید.

```
x = 10;
y = x++;      // x equals 11 and y equals 10
x = 10;
y = ++x;      // x and y equal 11
```

عملگر منهای یکتائی (-) عملوند خود را منفی می کند. عملگر جمع یکتائی (+) تاثیری روی عملوند خود ندارد. کامپایلر از نحوه بکار بردن این عملگرها د ر عبارت متوجه منظور شما می شود.

مثال.

```
x = -a;
x = a * -b;
x = a * (-b);     //This way is better
```

الویت عملگرهای ریاضی

الویت	عملگرها
1	++ --
2	* / %
3	+ -

مثال. حاصل عبارت 2 * 5 % 12 برابر با 4 است.
مثال. حاصل عبارت زیر 3- است.

```
x = 25 - (2 * (10 + (8 / 2)));
```

عملگرهای رابطه ای

عملگرهای رابطه ای (relational operators) در عبارات مقایسه ای استفاده می شوند و یک مقدار بولین تولید می کنند. عبارتی که دارای یک عملگر رابطه ای است به صورت true و false ارزیابی می شوند. اگر رابطه درست باشد مقدار true یا 1 و اگر غلط باشد مقدار false یا 0 تولید می شود.

بیشترین کاربرد عبارات رابطه ای در جملات شرطی است.

عملگرهای رابطه ای C++

عملگر	سمبل	عمل	مثال
Equal	==	آیا عملوند اول مساوی عملوند دوم است	x == y
Greater than	>	آیا عملوند اول بزرگتر از عملوند دوم است	x > y
Less than	<	آیا عملوند اول کوچکتر از عملوند دوم است	x < y
Greater than or equal to	>=	آیا عملوند اول بزرگتریا مساوی عملوند دوم است	x >= y
Less than or equal to	<=	آیا عملوند اول کوچکتریا مساوی عملوند دوم است	x <= y
Not equal	!=	آیا عملوند اول نامساوی عملوند دوم است	x != y

مثال. عبارات زیر رابطه ای هستند.

عبارت	نتیجه
5 == 1	0 (false)
5 > 1	1 (true)
5 != 1	1 (true)
(5 + 10) == (3 * 5)	1 (true)

نکته. عملگر رابطه ای == را با عملگر = اشتباه نگیرید. یکی از متداولترین اشتباهات برنامه نویسان استفاده از علامت انتساب به جای عملگر مساوی در عبارات شرطی است.
نکته. عباراتی که برابر مقداری غیر صفر می شوند به عنوان true (1) درنظر گرفته می شوند.

مثال. با اجرای جمله زیر همیشه پیغام نمایش داده می شود زیرا مقدار x برابر با 5 می شود و شرط همیشه درست می شود.

```
if (x = 5)
   cout<<"x is equal to 5";
```

الویت عملگرهای رابطه ای

الویت	عملگر
1	< <= > >=
2	!= ==

عملگرهای رابطه ای الویت کمتری نسبت به عملگرهای ریاضی دارند.

مثال. دو شرط زیر یکسان است.

```
if (x + 2 > y)

if ((x + 2) > y)
```

عملگرهای منطقی

عملگرهای منطقی (logical operators) اجازه می دهند دو یا چند عبارت رابطه ای را به یک عبارت که حاصل آن بولین است تبدیل کنید.

عملگرهای منطقی C++

عملگر	سمبل	مثال
AND	&&	*exp1* && *exp2*
OR	||	*exp1* || *exp2*
NOT	!	!*exp1*

عباراتی که توسط عملگرهای منطقی بهم مربوط می شوند مطابق جدول زیر ارزیابی می شوند.

عبارت	نتیجه
(*exp1* && *exp2*)	اگر هردو عبارت exp1 و exp2 درست باشند نتیجه true (1) است
(*exp1* || *exp2*)	اگر عبارت exp1 یا exp2 یا هردو درست باشد نتیجه true (1) است
(!*exp1*)	اگر exp1 درست باشد نتیجه false (0) و اگر غلط باشد نتیجه true (1) است

مثال.

نتیجه	عبارت
True (1)	(5 == 5) && (6 != 2)
True (1)	(5 > 1) || (6 < 1)
False (0)	(2 == 1) && (5 == 5)
True (1)	!(5 == 4)

یک سوال شرطی را توسط عملگرهای منطقی به چندین طریق می توان نوشت.

الویت عملگرهای منطقی

الویت	عملگر
1	!
2	&&
3	||

عملگرهای بیتی

عملگرهای بیتی اجازه تغییر بیتهای یک عدد صحیح را می دهند. عملگرهای بیتی مطابق با جبر بولی روی بیتهای عملوندها کار می کنند.

عملگرهای بیتی C++

عملگر	سمبل	عمل	مثال
And	&	عملوند اول را بیت به بیت با عملوند دوم and می کند	*exp1 & exp2*
Or	|	عملوند اول را بیت به بیت با عملوند دوم or می کند	*exp1 | exp2*
Xor	^	عملوند اول را بیت به بیت با عملوند دوم xor می کند	*exp1 ^ exp2*
1's Complement	~	مکمل ۲ عملوند را می گیرد	*~exp1*
Left Shift	<<	عملوند اول را به اندازه no به سمت چپ شیفت می دهد	*exp1 << no*
Right Shift	>>	عملوند اول را به اندازه no به سمت راست شیفت می دهد	*exp1* << no

عملگرهای انتساب ترکیبی

در C++ توسط عملگرهای انتساب ترکیبی روش خلاصه تری برای نوشتن عبارات وجود دارد. به طور کلی عملگرهای انتساب ترکیبی قاعده زیر را دارند:

```
exp1 op= exp2
```

op یک عملگر دوتائی است.

مثال. دو جمله زیر معادل هم هستند.

```
x = x + 5;

x += 5;
```

مثال. حاصل متغیرهای x و z هردو برابر با 14 می شود.

```
x = 12;
z = x += 2;
```

عملگر شرطی

عملگر شرطی تنها عملگر سه تائی در C++ است که سه عملوند دارد و فرم کلی آن به صورت زیر است:

```
exp1 ? exp2 : exp3;
```

اگر حاصل عبارت exp1 درست باشد (غیر صفر باشد) exp2 و اگر exp1 غلط باشد (برابر با صفر باشد) عبارت exp3 به عنوان نتیجه ارزیابی می شود.

مثال. اگر y غیرصفر باشد مقدار 1 و اگر صفر باشد مقدار 100 به x اختصاص داده می شود.

```
x = y ? 1 : 100;
```

مثال. عملگر شرطی مشابه یک جمله شرطی عمل می کند.

```
z = (x > y) ? x : y;
```

که مشابه جمله شرطی زیر است

```
if (x > y)
   z = x;
else
   z = y;
```

مثال. در عبارت زیر a برابر با b می شود اگر نتیجه کاهش b غیرصفر باشد. اگر نتیجه صفر شود a و b هردو برابر با مقدار -99 می شوند

```
a = --b ? b : (b = -99);
```

عملگر کاما

در حالت های خاصی کاما (,) در نقش یک عملگر کار می کند نه یک جداکننده. یک عبارت را می توان با مربوط کردن دو زیرعبارت توسط کاما شکل داد. هر دو عبارت ارزیابی می شود، ابتدا عبارت سمت چپ محاسبه می شود و نتیجه عبارت سمت راست بعنوان نتیجه کلی عبارت ارزیابی می شود.

مثال. جمله زیر به a یکی اضافه می کند و مقدار b را در x قرار می دهد سپس آنرا افزایش می دهد.

```
x = (a++ , b++);
```

عملگر sizeof

تعداد بایت های مورد استفاده هرمتغیریا نوع داده ای را می توان توسط عملگر sizeof بدست آورد.

مثال. برنامه زیر تعداد بایت های نوع های double و char را نمایش می دهد.

```
#include <iostream>
int main() {
  cout << "sizeof(double) = " << sizeof(double);
  cout << ", sizeof(char) = " << sizeof(char);
}
```

توجه داشته باشید که sizeof یک تابع نیست و می تواند بدون پرانتز هم استفاده شود.

```
int main() {
  int x;
  int i = sizeof x;
}
```

عملگرهای آدرس (&) و اشاره گر (* و <-) در بخش اشاره گرها توضیح داده خواهند شد.

جدول الویت عملگرها

الویت	عملگرها
1	() [] -> .
2	! ~ ++ -- * *(indirection)* & *(address-of)* (*type*)
	sizeof + *(unary)* - *(unary)*
3	* / %
4	+ -
5	<< >>
6	< <= > >=
7	== !=
8	*(bitwise* AND) &
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= %= &= ^= |= <<= >>=
15	,

تبدیل نوع

کامپایلر به طور خودکار یک نوع داده را درصورت نیاز به دیگری تبدیل می کند. مثلا اگر یک عدد int در متغیر float ذخیره شود کامپایلر مقدار را به float تبدیل می کند. همیشه نوع کوچکتر به نوع بزرگتر تبدیل می شود. اگر یک عدد float با double جمع شود با هردو به عنوان double برخورد می شود.

اگر سعی کنید مقدار بزرگتری را در متغیر کوچکتر ذخیره کنید بخشی از داده ممکن است از دست برود و احتمالا با یک پیغام هشدار مواجه خواهید شد.

مثال.

```
char a='2';
int b= a+ 9;

const int big=110232343;
const short int small=big;
```

اگر برنامه نویس بخواهد صریحا عمل تبدیل نوع را انجام دهد از casting استفاده می کند. برای این کار نوع داده مورد نظر را درون پرانتز سمت چپ مقدار قرار دهید. مقدار می تواند متغیر، ثابت، حاصل یک عبارت یا مقدار برگشتی یک تابع باشد.

مثال.

```
int b = 200;
unsigned long a = (unsigned long int)b;
```

در C++ راه دیگری هم برای تبدیل نوع وجود دارد. در این روش مانند فراخوانی توابع پرانتز اطراف مقدار قرار می گیرد.

مثال.

```
float a = float(200);
// This is equivalent to:
float b = (float)200;
```

البته در مثال فوق نوشتن 200f به تنهائی کفایت می کند.

www.HPKClasses.ir