Programming (1)
C++

Faculty of Computers and Information

Fayoum University

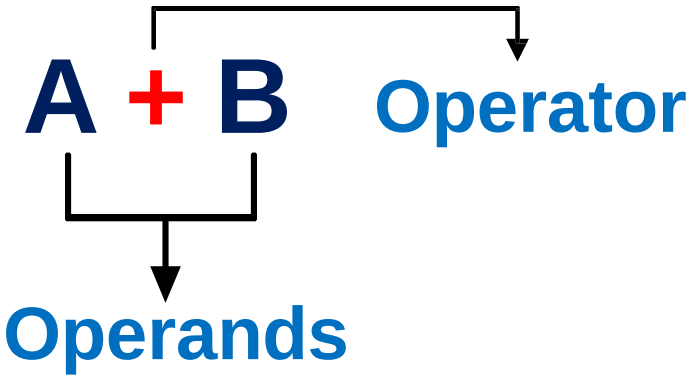
2020-2021

Operators and Operands

- Almost all of the programs use some sort of data stored in variables.
- In a program different types of operations are performed on that data.
- The data on which operations are performed are known as operands and the types of the operations performed are known as operators.

Operators and Operands

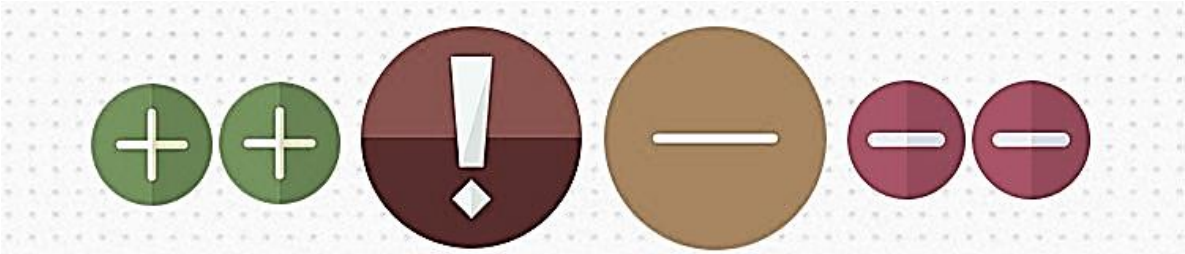
- For Example:

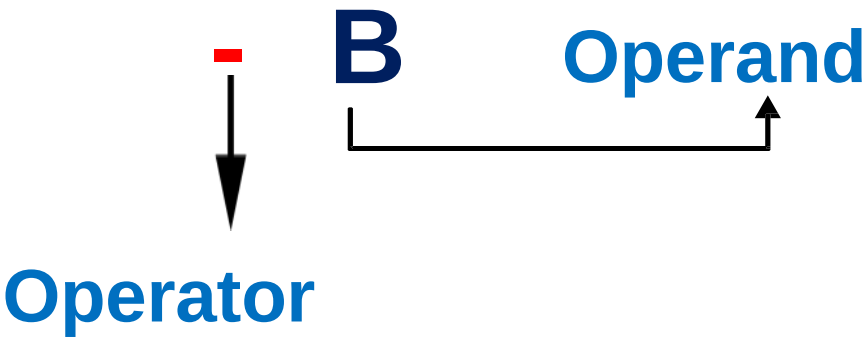


Types of Operators

Unary Operators

- Perform operation on one operand





Types of Operators

Binary Operators

- Perform operation on two operands

A * B Operator

Operands

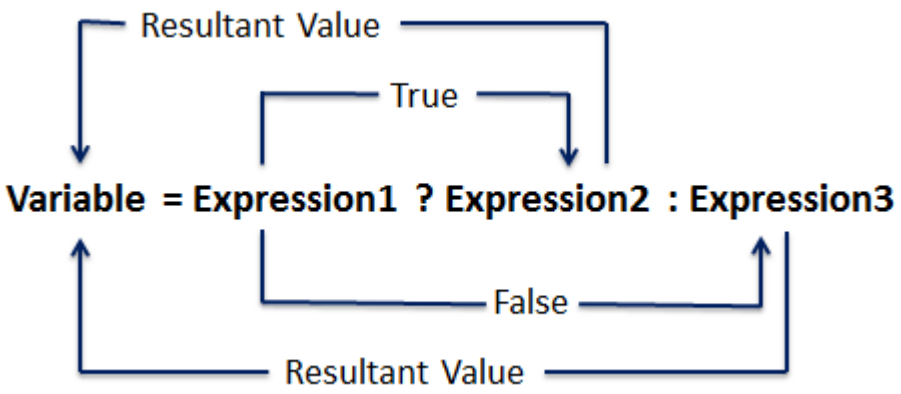
A % B Operator

Operands

Types of Operators

Ternary Operators [Conditional Operator]

- Perform operation on three operands



Categories of Operators in C++

- **Arithmetic Operators (+ , - , / , * ,%)**
- **Assignment Operator (=)**
- **Increment and Decrement Operators (++ , - -)**
- **Arithmetic Assignment Operators (+= , -= , /= , *=, %=)**
- **Relational Operators (> , <, >= , <= , == , !=)**
- **Logical Operators (&& ,| | , !)**
- **Bitwise Operators (& , | , ~, ^, >> , <<)**
- **Conditional Operator (?:)**

Arithmetic Operators in C++

- They perform arithmetic operations on the program data.

Operation	Description	Type	Example
+	Addition	Binary	a + b
-	Subtraction	Binary	a - b
/	Division	Binary	a / b
*	Multiplication	Binary	a * b
%	Modulus / Remainder	Binary	a % b

- +, -, *, and / can be used with integral and floating-point data types

Arithmetic Operators in C++

```
#include<iostream>


using namespace std;

int main()
{
    int a=5, b=8;
    float c=9.8, d=15.4;

    cout<< a + b <<endl;
    cout<< b - d <<endl;
    cout<< a * b <<endl;
    cout<< c / d <<endl;
    cout<< b % a;


    return 0;
}
```

Arithmetic Operators in C++

- Parentheses are used in C++ expressions in the same manner as in algebraic expressions.

- For example, to multiply **a** times the quantity **b + c**

```
a * ( b + c )
```

- There is no arithmetic operator for exponentiation in C++, so X^2 is represented as **x * x**.

Precedence of arithmetic operations

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. [*Caution:* If you have an expression such as `(a + b) * (c - d)` in which two sets of parentheses are not nested, but appear "on the same level," the C++ Standard does *not* specify the order in which these parenthesized subexpressions will be evaluated.]
*, /, %	Multiplication, Division, Modulus	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

For example,

`2 + 3 * 5` and `(2 + 3) * 5`

both have different meanings

Precedence of arithmetic operations

Example :

Step 1.	`y = 2 * 5 * 5 + 3 * 5 + 7;` `2 * 5 is 10`	*(Leftmost multiplication)*
Step 2.	`y = 10 * 5 + 3 * 5 + 7;` `10 * 5 is 50`	*(Leftmost multiplication)*
Step 3.	`y = 50 + 3 * 5 + 7;` `3 * 5 is 15`	*(Multiplication before addition)*
Step 4.	`y = 50 + 15 + 7;` `50 + 15 is 65`	*(Leftmost addition)*
Step 5.	`y = 65 + 7;` `65 + 7 is 72`	*(Last addition)*
Step 6.	`y = 72`	*(Last operation—place* `72` *in* `y`*)*

Precedence of arithmetic operations

- ? = **1 + 2 * (3 + 4)**

- **5 * 2 + 9 % 4**

- ? = **5 * 2 % (7 – 4)**

Precedence of arithmetic operations

- ? = **1 + 2 * (3 + 4)**
  - ? =Evaluated as **1 + (2 * (3+4))** and the result is **15**
- **5 * 2 + 9 % 4**
- ? = **5 * 2 % (7 – 4)**

Precedence of arithmetic operations

- ? = **1 + 2 * (3 + 4)**
  - ? =Evaluated as **1 + (2 * (3+4))** and the result is **15**
- **5 * 2 + 9 % 4**
  - Evaluated as **(5*2) + (9 % 4)** and the result is **11**
- ? = **5 * 2 % (7 – 4)**

Precedence of arithmetic operations

- ? = **1 + 2 * (3 + 4)**
  - ? =Evaluated as **1 + (2 * (3+4))** and the result is **15**
- **5 * 2 + 9 % 4**
  - Evaluated as **(5*2) + (9 % 4)** and the result is **11**
- ? = **5 * 2 % (7 – 4)**
  - Evaluated as **(5 * 2) % (7 – 4)** and the result is **1**

Expressions

- If all operands are integers
  - Expression is called an integral expression
    - Yields an integral result
    - Example: `2 + 3 * 5`
- If all operands are floating-point
  - Expression is called a floating-point expression
    - Yields a floating-point result
    - Example: `12.8 * 17.5 - 34.50`

Expression

- If operators are *, /, +, or – , then the type of the result will be:
  - integer, if all operands are integer.
    - int A , B;
    - A+ B → Integer.
  - float, If at least one operand is float and there is no double
    - int A ; float B;
    - A + B → float.
  - double, if at least one operand is double
    - int A ; float B; fouble C;
    - A + B + C → double.

Mixed Expressions

- Mixed expression:
  - Has operands of different data types
  - Contains integers and floating-point
- Examples of mixed expressions:
  - `2 + 3.5`
  - `6 / 4 + 3.9`
  - `5.4 * 2 – 13.6 + 18 / 2`

Mixed Expressions (continued)

- Evaluation rules:
  - If operator has same types of operands
    - Evaluated according to the type of the operands
  - If operator has both types of operands
    - Integer is changed to floating-point
    - Operator is evaluated
    - Result is floating-point
  - Entire expression is evaluated according to precedence rules

Expression

int * int;

result int

int + float;

result float

Int + double / float;

result double

int – double;

result double

Expression

- The data type of the target variable is also important
- If the result is a real number and the target variable is declared as integer, only the integer part of the result will be kept, and decimal part will be lost.

Example

```
int avg;
float sum=100.0,  cnt = 6.0;
avg = sum / cnt;
```

The result is calculated as 16.66667

But avg will be 16

Expression

```
float avg;
int sum=100,  cnt = 6;
avg = sum / cnt;
```

The result of the division will be 16
avg will be 16.0

- Only the integer part of the result will be considered if two operands are integer Even when the target variable is float

Type Conversion (Casting)

- Implicit type coercion: when value of one type is automatically changed to another type
- Cast operator: provides explicit type conversion
  - `static_cast<dataTypeName>(expression)`

Type Conversion (Casting)

EXAMPLE 2-9

Expression	Evaluates to
`static_cast<int>(7.9)`	`7`
`static_cast<int>(3.3)`	`3`
`static_cast<double>(25)`	`25.0`
`static_cast<double>(5+3)`	`=static_cast<double>(8)=8.0`

`static_cast<double>(15)/2`	`=15.0/2`
	(because `static_cast<double>(15)=15.0`)
	`=15.0/2.0=7.5`

`static_cast<double>(15/2)`	`=static_cast<double>(7)` (because `15/2=7`) `= 7.0`

`static_cast<int>(7.8 +`
`static_cast<double>(15)/2)`
	`= static_cast<int>(7.8+7.5)`
	`= static_cast<int>(15.3)`
	`= 15`

`static_cast<int>(7.8 +`
`static_cast<double>(15/2))`
	`= static_cast<int>(7.8 + 7.0)`
	`= static_cast<int>(14.8)`
	`= 14`

Type Conversion (Casting)

```
int main()
{
    int  i=5, j=3;
    float div;
    div= i/j;
    cout<< div;
    return o; }
```

The div will be 1.0 and this is not write

```
int main()
{
    int  i=5, j=3;
    float div;
    div=(float) i/j;
    cout<< div;
    return 0; }
```

Type cast: tells the compiler to treat **i** as a float

After type casting , The div will be 1.66667

Assignment Operator

- Assignment operator assigns a constant value, a variable or equation to a single variable.
- There is one assignment operator (**=**). It is binary operator.
- It assigns anything on its right side to its left side.

```
float radius = 6.98 ;
```

```
a = b ;
```

```
c = ( f – 32 ) / 1.8 ;
```

= Assignment Operator

Arithmetic assignment

- C++ has special assignment statements called compound assignments

+= , -= , *= , /= , %=

- Example:

```
x +=5 ; means  x = x + 5;
x *=y;  means  x = x * y;
x /=y;  means  x = x / y;
```

Arithmetic assignment

- They perform arithmetic operations on two operands and assigns the resultant in the same first operand.

Operation	Description	Type	Example	Same as
+=	Addition Assignment	Binary	a += b	a = a + b
-=	Subtraction Assignment	Binary	a -= b	a = a - b
/=	Division Assignment	Binary	a /= b	a = a / b
*=	Multiplication Assignment	Binary	a *= b	a = a * b
%=	Modulus / Remainder Assignment	Binary	a %= b	a = a % b

Arithmetic assignment

a += b is same as **a = a + b**

a = a + b

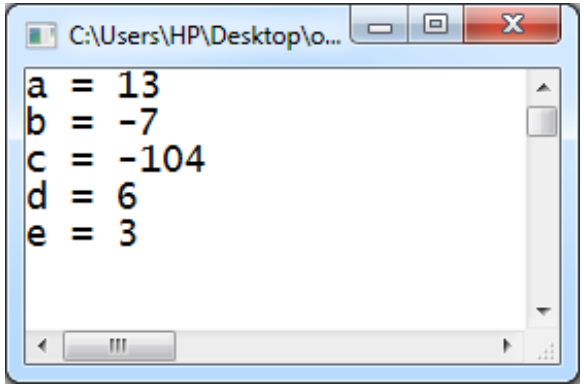
First operand = a

Second operand = b

Result stored back in to first operand i.e. a

Arithmetic assignment

```
#include<iostream>
#include<conio.h>

using namespace std;

int main()
{
    int a=5, b=8, c=-8, d=61, e=15;

    a += b;
    b -= 15;
    c *= a;
    d /= 10;
    e %= 12;

    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
    cout<<"c = "<<c<<endl;
    cout<<"d = "<<d<<endl;
    cout<<"e = "<<e;

    getch();
    return 0;
}
```



Increment and Decrement Operators

- They are unary operators, who require only one operand.
- They increment or decrement the value of operand by one.
- There are two versions of operators: **Prefix** and **Postfix**.

Operation	Description	Type	Example	Operation
++	Prefix Increment	Unary	++a	Increments a, then evaluates a
--	Prefix Decrement	Unary	--a	Decrements a, then evaluates a
++	Postfix Increment	Unary	a++	Evaluates a, then increments a
--	Postfix Decrement	Unary	a--	Evaluates a, then decrements a

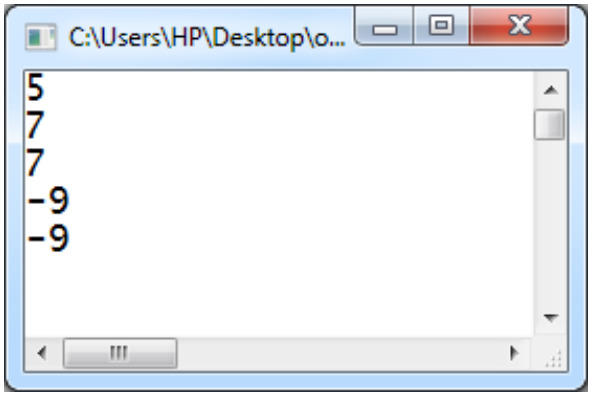
Increment and Decrement Operators

- Increment operator: increment variable by 1
  - Pre-increment: ++variable
  - Post-increment: variable++
- Decrement operator: decrement variable by 1
  - Pre-decrement: --variable
  - Post-decrement: variable --

awkward	easy	easiest
x = x+1;	x += 1	x++
x = x-1;	x -= 1	x--

Increment and Decrement Operators

```
#include<iostream>
#include<conio.h>

using namespace std;

int main()
{
    int a=5, b=8, c=-8;

    cout<< a++ <<endl;
    cout<< ++a <<endl;
    cout<< --b <<endl;
    cout<< --c <<endl;
    cout<< c--;

    getch();
    return 0;
}
```



Increment and Decrement Operators

- If the value produced by ++ or – – is not used in an expression, it does not matter whether it is a pre or a post increment (or decrement).
- When ++ (or – –) is used before the variable name, the computer first increments (or decrements) the value of the variable and then uses its new value to evaluate the expression.
- When ++ (or – –) is used after the variable name, the computer uses the current value of the variable to evaluate the expression, and then it increments (or decrements) the value of the variable.

```
x = 5;
Cout << x++;
```

```
x = 5;
Cout << ++x;
```