CC-301 Programming Fundamentals

Lecture 4

Engr. Waseem Ullah Khan

Operators

Operators are used to perform operations on variables and values

Example: **int x = 100 + 50;** // + operator is used

The + operator is often used to add two values, it can also be used to add together variable and value, or variable and another variable:

- int sum1 = ***100 + 50;*** // 150 (100 + 50)
- int sum2 = ***sum1 + 250;*** // 400 (150 + 250)
- int sum3 = ***sum2 + sum2;*** // 800 (400 + 400)

Operator Types

Type	Operators	Usage
Arithmetic	'+' '-' '*' '/' '%'	a+b, a-b, a*b, a/b, a%b
Arithmetic Assignment	'+=' '-=' '*=' '/=' '%='	a+=b is same as a=a+b a-=b, a*=b, a/=b, a%=b
Logical	'&&' '||'	A && B
Relational	'<' '>' '<=' '>=' '==' '!='	x == y, x != y, x < y
Increment and Decrement	'++' '--'	a++ is same as a=a+1 a-- is same as a=a-1

Arithmetic Operator

Used to perform arithmetic operations on variables

Operator	Name	Description	Example
+	Addition	Adds two values	a + b
-	Subtraction	Subtracts one value from another	a - b
*****	Multiplication	Multiplies two values	a * b
/	Division	Divides one value by another	a / b
%	Modulus	Returns the division remainder	a % b

The Modulus Operator

- % is known as the Modulus Operator or the Remainder Operator
- It calculates the remainder of two variables
- **3 % 2 = 1**
- **6 % 3 = 0**
- **8 % 5 = 3**

Arithmetic Operator - Example

arithmetic.cpp

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

int main() {
    int num_1 = 10, num_2 = 6;

    // printing the sum of num_1 and num_2
    cout << "num_1 + num_2 = " << num_1 + num_2 << endl;

    // printing the difference of num_1 and num_2
    cout << "num_1 - num_2 = " << num_1 - num_2 << endl;

    // printing the product of num_1 and num_2
    cout << "num_1 * num_2 = " << num_1 * num_2 << endl;

    // printing the division of num_1 by num_2
    cout << "num_1 / num_2 = " << num_1 / num_2 << endl;

    // printing the modulus of num_1 by num_2
    cout << "num_1 % num_2 = " << num_1 % num_2 << endl;

    return 0;
}
```

```
num_1 + num_2 = 16
num_1 - num_2 = 4
num_1 * num_2 = 60
num_1 / num_2 = 1
num_1 % num_2 = 4
```

Arithmetic Assignment Operator

Used to assign values to variables

- a = 5; // assign 5 to a

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of **A + B into C**
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to **C = C + A**
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to **C = C – A**
***=**	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to **C = C * A**
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to **C = C / A**
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to **C = C % A**

Assignment Operator - Example

assignment.cpp

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

int main() {
    int num_1 = 10, num_2;

    num_2 = num_1;
    cout << "\"=\" operator, value of num_2 = " << num_2 << endl;

    num_2 += num_1;
    cout << "\"+=\" operator, value of num_2 = " << num_2 << endl;

    num_2 -= num_1;
    cout << "\"-=\" operator, value of num_2 = " << num_2 << endl;

    num_2 *= num_1;
    cout << "\"*=\" operator, value of num_2 = " << num_2 << endl;

    num_2 /= num_1;
    cout << "\"/=\" operator, value of num_2 = " << num_2 << endl;

    num_2 %= num_1;
    cout << "\"%=\" operator, value of num_2 = " << num_2 << endl;

    return 0;
}
```

```
"=" operator, value of num_2 = 10
"+=" operator, value of num_2 = 20
"-=" operator, value of num_2 = 10
"*=" operator, value of num_2 = 100
"/=" operator, value of num_2 = 10
"%=" operator, value of num_2 = 0
```

Logical Operator

Logical operators are carried out on statements, e.g. statement1 && statement 2, etc

Type	Operators	Usage
Logical	'&&' '||'	A && B

Logical AND (&&)

- false && false= false
- false && true = false
- true && false= false
- true && true = true

Logical OR (||)

- false || false = false
- false || true = true
- true || false = true
- true || true = true

Logical NOT (!)

- !false = true
- !true = false

Logical Operator - Example

logical_operator.cpp

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

int main()
{
    int x = 5;

    // Logical AND (&&)
    // returns true (1) because 5 is greater than 3 AND 5 is less than 10
    cout << "Logical AND (&&) = ";
    cout << (x > 3 && x < 10) <<endl;

    // Logical OR (||)
    /* returns true (1) because one of the conditions are true
    (5 is greater than 3, but 5 is not less than 4) */
    cout << "Logical OR (||) = ";
    cout << (x > 3 || x < 4) <<endl;

    // Logical NOT (!)
    // returns false (0) because ! (not) is used to reverse the result
    cout << "Logical NOT (!) = ";
    cout << (!(x > 3 && x < 10));

    return 0;
}
```

```
Logical AND (&&) = 1
Logical OR (||) = 1
Logical NOT (!) = 0
--------------------------------
```

Relational Operators

A relational operator is used to check the relationship between two operands

For Example: ***a > b;*** // checks if a is greater than b

Here, > is a relational operator. It checks if a is greater than b or not

If the relation is true, it returns 1 whereas if the relation is false, it returns 0

relational.cpp

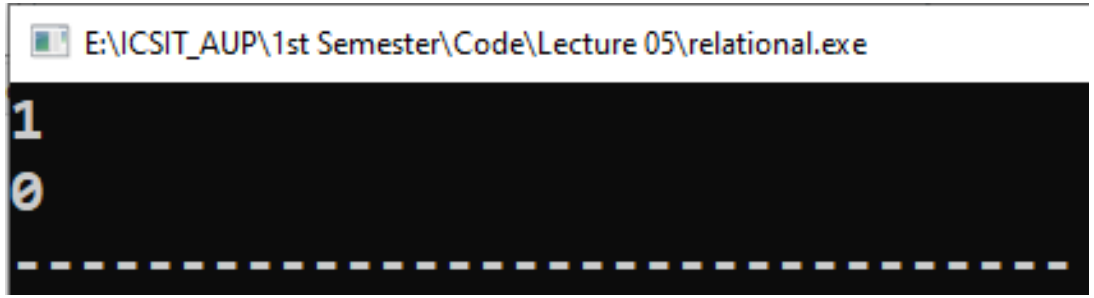
```cpp
#include<iostream>
using namespace std;

int main()
{
    int x = 10;
    int y = 5;

    // returns 1 (true) because 10 is greater than 5
    cout << (x>y) << endl;

    // returns 0 (false) because 10 is greater than 5
    cout << (x<y);

    return 0;
}
```



List of Relational Operators

Operator	Name	Description	Example
==	Is Equal To	Checks if the values of two operands are equal or not, if yes then condition becomes true	3 == 5 gives us **false**
!=	Not Equal To	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true	3 != 5 gives us **true**
>	Greater Than	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true	3 > 5 gives us **false**
<	Less Than	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true	3 < 5 gives us **true**
>=	Greater Than or Equal To	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true	3 >= 5 give us **false**
<=	Less Than or Equal To	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true	3 <= 5 gives us **true**

Increment and Decrement Operators

C++ also provides increment and decrement operators: ++ and -- respectively

- ++ increases the value by 1
- -- decreases the value by 1

Type	Operators	Usage
Increment and Decrement	'++' '--'	++a means a=a+1 --a means a=a-1

increment_decrement.cpp

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

int main()
{
    int x = 10, y = 50, result_1, result_2;

    // incrementing x by 1 and storing the result in result_1
    result_1 = ++x;
    cout << "Result_1 = " << result_1 << endl;

    // decrementing y by 1 and storing the result in result_2
    result_2 = --y;
    cout << "Result_2 = " << result_2 << endl;

    return 0;
}
```

E:\ICSIT_AUP\1st Semester\Code\Lecture 05\increment_decrement.exe

```
Result_1 = 11
Result_2 = 49

--------------------------------
```

Prefix vs Postfix

In **Prefix**, the value present in the variable is incremented or decremented first and then it is used in the program i.e., **++a; --a;**

In **Postfix**, the value present in the variable is assigned first and then it is incremented or decremented i.e., **a++; a--;**

prefix_postfix.cpp

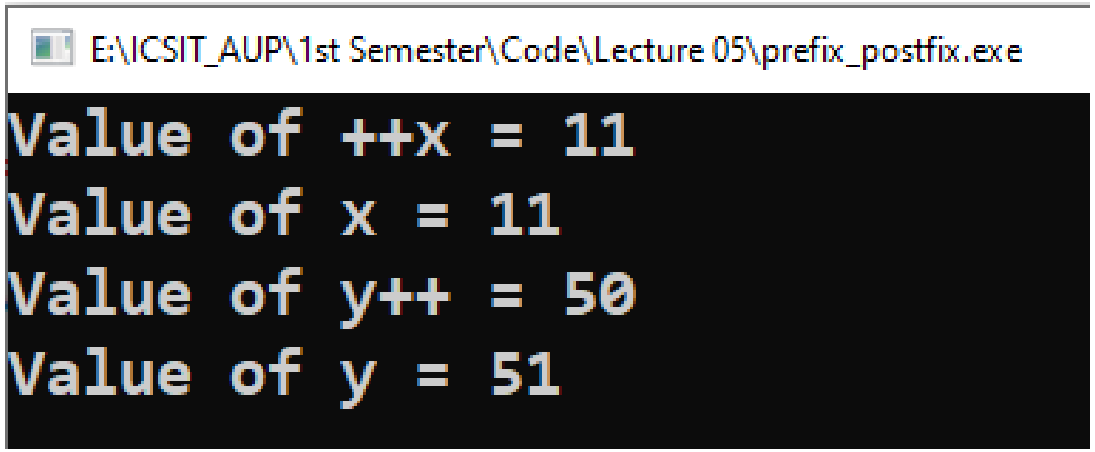
```cpp
#include<iostream>
using namespace std;

int main()
{
    int x = 10, y = 50, result_1, result_2;

    // Value of x will be increased before assignment
    result_1 = ++x;
    cout << "Value of ++x = " << result_1 << endl;
    cout << "Value of x = " << x << endl;

    // Value of y will not be increased before assignment
    result_2 = y++;
    cout << "Value of y++ = " << result_2 << endl;
    cout << "Value of y = " << y << endl;

    return 0;
}
```



Exercise

Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	True
(5 != 6)	True
(5 < 3) && (6 <= 6) || (5 != 6)	True
(5 < 3) && ((6 <= 6) || (5 != 6))	False
!((5 < 3) && ((6 <= 6) || (5 != 6)))	True

What is an Expression in C++

An expression is a valid arrangement of variables, constants, and operators

In C++, each expression can be evaluated to compute a value of a given type

In C++, an expression can be:

- 7 /* Constant */
- x /* Variable */
- x + 7 /* with operator */
- x = x + 7 /* with operators */
- x = x + 7; /* Simple statement */

Arithmetic Expression

Used to express the mathematic expression in C++

$b^2 - 4ac$	$b*b-4*a*c$
$x(y+z)$	$x*(y+z)$
$\frac{1}{x^2+x+3}$	$1/(x*x+x+3)$
$\frac{a+b}{c-d}$	$(a+b)/(c+d)$

Precedence of Operators

Operators	Operations	Precedence (order of Evaluation)
()	Parentheses	Evaluated first
*****	Multiplication	Evaluated second. If there are several, they are evaluated from left to right
/	Division	
%	Modulus	
+	Addition	Evaluated last. If there are several, they are evaluated from left to right
-	Subtraction	

What is the result of 10 + 10 * 5 ?

What is the result of (10 + 10) * 5 ?

Note:- $(b^2 – 4ac)/2a$ *is not the same as* ***b * b – 4 * a * c / 2 * a***

This is wrong as it evaluates to $b^2 – 4ac/2a$ (i.e. 4ac is divided by 2a instead of $(b^2 -4ac)$)

Evaluate

In what order will the expression be evaluated?

In algebra

$$\boldsymbol{m = pr\ \%\ q + w/x\ - y}$$

In C++

$$\boldsymbol{m = p\ *\ r\ \%\ q\ +\ w\ /\ x\ -\ y;}$$

C++ String

Strings are used for storing text

A string variable contains a ***collection of characters*** surrounded by ***double quotes***

Example: Create a variable of type string and assign it a value:

string greeting = "Hello";

To use strings, you must include an additional header file in the source code, the <string> library:

```
#include <iostream>
#include <string>   // Include the string library
using namespace std;

int main ()
{
    // Create a string variable
    string name = "Waseem Khan";

    cout << "Hello my name is " << name;

    return 0;
}
```

C++ String

String Concatenation: The **+** operator can be used between strings to add them together to make a new string. This is called concatenation

String Length: To get the length of a string, use the **length()** or **size()** function

Access Strings: You can access the characters in a string by referring to its index number inside square brackets [ ]

Note: String indexes start with 0: [0] is the first character. [1] is the second character, etc.

User Input Strings: Use the extraction operator >> on cin to display a string entered by a user

C++ String - Example

strcan.cpp

```cpp
// String Concatenation
#include <iostream>
#include <string>   // Include the string library
using namespace std;

int main ()
{
    string firstName = "Waseem";
    string lastName = "Khan";
    string fullName1, fullName2;

    fullName1 = firstName + " " + lastName;

    fullName2 = firstName + lastName;

    cout << "Name with space : " << fullName1 <<endl;
    cout << "Name without space : " << fullName2;

    return 0;
}
```

E:\ICSIT_AUP\1st Semester\Code\Lecture 05\strcan.exe

```
Name with space : Waseem Khan
Name without space : WaseemKhan
--------------------------------
```

string.cpp

```cpp
#include <iostream>
#include <string>   // Include the string library
using namespace std;

int main ()
{
    string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string myname   = "Waseem Khan";
    // String length
    cout << "The length of the alphabet string is : " << alphabet.length() <<endl;
    cout << "The length of the alphabet string is : " << alphabet.size() <<endl;
    cout << "The length of the myname string is : " << myname.length() <<endl;
    // Access Strings
    cout << alphabet[0] <<endl; // outputs: A
    cout << myname[6] <<endl; // outputs: space
    cout << myname[7] <<endl; // outputs: K

    return 0;
}
```

E:\ICSIT_AUP\1st Semester\Code\Lecture 05\string.exe

```
The length of the alphabet string is : 26
The length of the alphabet string is : 26
The length of the myname string is : 11
A

K
```

C++ String - Example

input_string.cpp

```cpp
#include <iostream>
#include <string>   // Include the string library
using namespace std;

int main ()
{
    string firstName;
    cout << "Type your first name: ";
    cin >> firstName; // get user input from the keyboard
    cout << "Your name is: " << firstName <<endl;

    string fullName;
    cout << "Type your full name: ";
    cin >> fullName;
    cout << "Your full name is: " << fullName <<endl;

    return 0;
}
```

E:\ICSIT_AUP\1st Semester\Code\Lecture 05\input_string.exe

```
Type your first name: Waseem
Your name is: Waseem
Type your full name: Waseem Khan
Your full name is: Waseem
```

getline_string.cpp

```cpp
#include <iostream>
#include <string>   // Include the string library
using namespace std;

int main ()
{
    string Name;
    cout << "Type your full name: ";
    getline (cin, Name);
    cout << "Your full name is: " << Name;

    return 0;
}
```

E:\ICSIT_AUP\1st Semester\Code\Lecture 05\getline_string.exe

```
Type your full name: Waseem Khan
Your full name is: Waseem Khan
--------------------------------
```