Introduction to C++

Facebook Group

www.facebook.com/groups/pst.community.eg


The ACM
The ACM


Registration
Started
ECPC 2012

Agenda

- Intro.
- Conditions.

Introduction

Phases of C++ Program:

- **Edit:** write your program in any IDE (integrated development environment) like Microsoft Visual Studio, Eclipse,…etc. Editing produces a .cpp file.

- **Preprocess:** C++ preprocessor obeys special commands called ***preprocessor directives***. This usually consists of including other files to compile with our project files.
  e.g.: # include <iostream> is a preprocessor directive that include I/O functions from library iostream to our project. The preprocessor is invoked by the compiler before the program is converted to machine language.

- **Compile:** The compiler translates the .cpp file into machine language code, also called object code (.obj file). The compiler also indicates syntax errors in the program.

Introduction (cont.)

- **Link:** a C++ program may contain reference to functions defined elsewhere by other programmers or vendors. Linker links the .obj file with the code of the functions to produce a .exe file, ready for execution.

- **Load and execute:** Finally go and run the program through loading the program and its data into memory and running the .exe file.

Important Terms

- A **source program** (.cpp file type) consists of the program statements comprising a C++ or other programming language program.
- An **object program** (.obj file type) is the result of compiling a source program.
- **Header files** (.h file type) contain constant, variable, and function declarations needed by a program.
- An **executable program** is a program that can be run by a computer.
- **Linking** adds code from libraries to your file. Collects the object code from all files in the workspace and puts them into one executable program.
- A **compiler** is a program that translates a source program into an object program.
- An **interpreter** is a program that translates individual source program statements, one at a time, into executable statements. Each statement is executed immediately after translation.

Compiler vs. Interpreter

- A compiler first takes in the entire program, checks for errors, compiles it and then executes it.
- Languages like Assembly Language, C, C++, Fortran, Pascal are compiled into machine code.
- Longer process than that of an interpreter but program runs faster.

- An interpreter does this line by line, so it takes one line, checks it for errors and then executes it.
- Languages like Basic, VbScript and JavaScript are interpreted.
- interpreted programs run much slower than compiled programs, as much as 5-10 times slower as every line of code has to be re-read, then re-processed.

Errors

- **Syntax Errors – Typing Errors**
  - Errors in programming language rules.
  - You can use the compiler or interpreter to uncover syntax errors.
  - You must have a good working knowledge of error messages to discover the cause of the error.

- **Semantic Errors – Logic or Meaning Errors**
  - Errors that indicate the logic used when coding the program failed to solve the problem.
  - You do not get error messages with logic errors.
  - Your only clue to the existence of logic errors is the production of wrong solutions.

- **Run-time Errors (Exceptions)**
  - Code does something illegal when it is run (hence runtime)
    E.g., divide by zero

C++ is case sensitive

“google” and “GoogLe” are different

C++ Hello World!

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello World!" << endl;
    return 0;
}
```

Common mistakes

find the error(s):

```
#include “iostream”
using namespace std
int main {
    cout << "Hello World!" << endl;
    return 0;
```

Variables

- Used to store values in memory.
- When naming a variable you should consider the following:
  - is a sequence of one or more letters (a-z) (A-Z), digits (0-9) or underscore characters (_).
  - Neither spaces nor punctuation marks or symbols can be part of a variable name.
  - In addition, variable names always have to begin with a letter.
  - Another rule that you have to consider when naming a variable is that they cannot match any keyword of the C++ language nor your compiler's specific ones.

Data Types

Name	Description	Size*	Range*
`char`	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
`short int (short)`	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
`int`	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`long int (long)`	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`bool`	Boolean value. It can take one of two values: true or false.	1byte	`true` or `false`
`float`	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
`double`	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`long double`	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`wchar_t`	Wide character.	2 *or* 4 bytes	1 wide character

Declaring Variables

- int a;
- double d;
- int x, y, z; same to:

```
int x;
int y;
int z;
```

- unsigned short int usi;

Declaring and Initializing Variables

Initialization: is to give your variable an initial value at the beginning of the program.

```
int sum = 0;
int multiplication = 1;
int counter = 100;
```

Example

```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

```
4
```

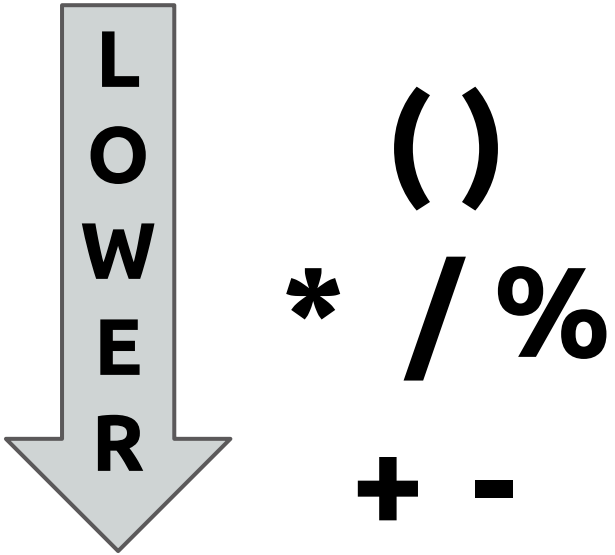
Assignment (=)

The assignment operator assigns the value of the right side to the variable and only one variable of the left side.

```
sum = 10;
sum = sum + 10 * 7 / 2;
```

Arithmetic Operators (+ - * / %)

Order of precedence of arithmetic operators:



List of escape sequences

\n	Newline
\r	carriage return
\t	Tab
\v	vertical tab
\b	Backspace
\f	form feed (page feed)
\a	alert (beep)
\'	single quote (')
\"	double quote (")
\?	question mark (?)
\\	backslash (\)

cout << “Hello” << endl;

same to

cout << “Hello\n”;

Compound Assignment (+= | -= | %= | *= | /=)

expression	is equivalent to
`value += increase;`	`value = value + increase;`
`a -= 5;`	`a = a - 5;`
`a /= b;`	`a = a / b;`
`price *= units + 1;`	`price = price * (units + 1);`

and the same for all other operators. For example:

```
1  // compound assignment operators
2
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8    int a, b=3;
9    a = b;
10   a+=2;                 // equivalent
11 to a=a+2
12   cout << a;
13   return 0;
   }
```

```
5
```

Increment and Decrement (++ | --)

++ increments the value by 1

-- decrements the value by 1

++B differs from B++

Example 1	Example 2
B=3; A=++B; // A contains 4, B contains 4	B=3; A=B++; // A contains 3, B contains 4

Standard Input (Cin)

```
int age;
cin >> age;
int a, b;
cin >> a >> b;
same to:
    cin >> a;
    cin >> b;
```

Quiz :’D

Write a program to calculate the sum, multiplication, division, remainder of two given numbers.

Input:

2 7

3 0

Calculator Program

```
#include <iostream>
# include <string>
using namespace std;
void main(){
      int n1,n2;
      cout<<" please enter two numbers \n";
      cin>>n1>>n2;
      int isum=0, imulti=1, iquotient=1, iRemainder=0;
      isum= n1+n2;
      cout<<"Sum="<<isum<<endl;
      imulti=n1*n2;
      cout<<"Multiplication="<<imulti<<endl;
      iquotient = n1/n2;
      cout<<"Quotient="<<iquotient<<endl;
      iRemainder = n1%n2;
      cout<<"Remainder="<<iRemainder<<endl;
}
```

Notes

1. Try to enter the second number 0, A Handled Exception will appear.
2. iquotient results is always integer:
    a. That's because of integer division.
    b. To solve the problem, declare the variable to be of type float:
    float fquotient;
    c. Typecast the division process to be of type float, since the two numbers divided by each other are integers, this **typecasting** urges the result to be a float number and thus the correct result will appear.
    fquotient = (float) n1/n2; à this called typecasting.

Conditions

Enter two numbers and display a message telling which is bigger.

Conditions

Enter two numbers and display a message telling which is bigger.

Conditions (cont.)

```
if(cond 1 is true) {
        //      execute code
}
if(cond 2 is true) {
        //      execute code
}
if(cond 3 is true) {
        //      execute code
}
if(cond n is true) {
        //      execute code
}
```

```
if(cond 1 is true) {
        //      execute code
}
else if(cond 2 is true) {
        //      execute code
}
else if(cond n is true) {
        //      execute code
}
else {
        //      execute code
}
```

Comparison Operators

==

!=

<

>

<=

>=

```
if (x == 100)
        cout<< “X = 100” <<endl;
```

Here (==) is used for comparison which is unlike (=) used for assignment.

Example

Enter a number and display if it is equal to 10 or not.

```
int main()
{
	int A;
	cin >> A;

	if ( A == 10 )
	{
		cout << "is equal" << '\n';
		cout << "closing program" << '\n';
	}
	else
	{
		cout << "not equal" << '\n';
		cout << "closing program" << '\n';
	}
	return 0;
}
```

The Same ?

```
if (x == 100)
    cout << "x is 100";
else
    cout << "x is not 100";
```

```
if (x == 100)
    cout << "x is 100";
cout << "x is not 100";
```

Quiz

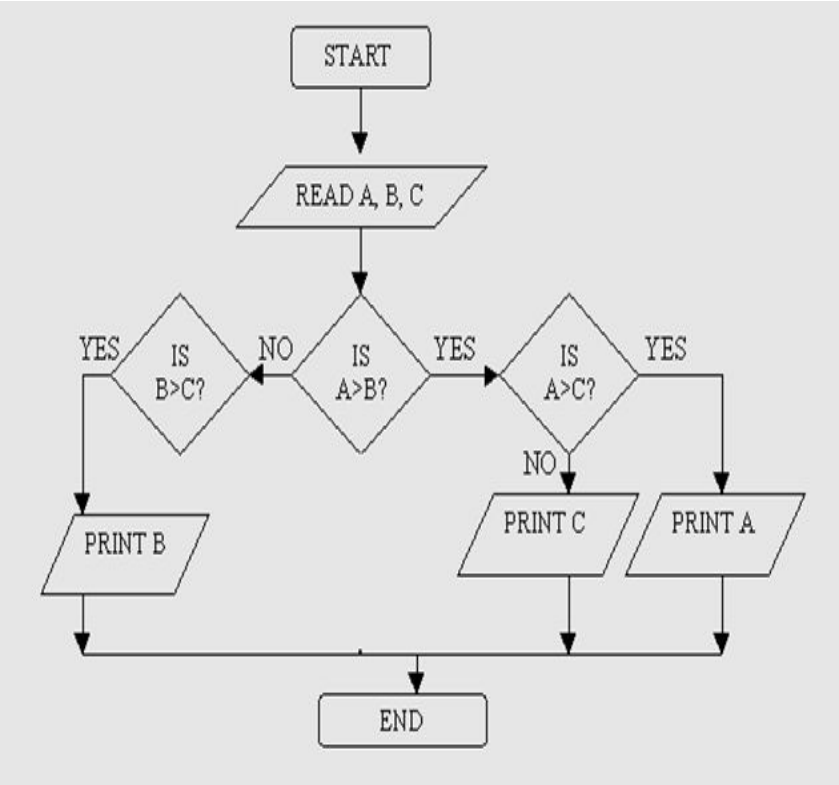
Enter two numbers and display a message telling which is bigger.

```
#include <iostream>
using namespace std;
int main()
{
  // define two integers
    int x = 3;
    int y = 4;
  //print out a message telling which is bigger
   if (x > y)
         cout << "x is bigger than y" << endl;
    else
     cout << "y is bigger than x" << endl;
}
```

Problem

Write a program to find the largest of three numbers A,B, and C.

Answer




```
int a, b, c;
cin >> a >> b >> c;
if (a > b) {
        if (a > c) cout << a << endl;
        else cout << c << endl;
}
else {
        if (b > c) cout << b << endl;
        else  cout << c << endl;
}
```

Can we do any better ?!

and &&

or ||

The new solution

before:

```
int a, b, c;
cin >> a >> b >> c;
if (a > b) {
        if (a > c) cout << a << endl;
        else cout << c << endl;
}
else {
        if (b > c) cout << b << endl;
        else    cout << c << endl;
        }
}
```

after:

```
int a, b, c;
cin >> a >> b >> c;
if (a > b && a > c)
        cout << a << endl;
else  if(b > a && b > c)
        cout << b << endl;
else
        cout << c << endl;
```

Cont.

&& ⟶ We need to check that two conditions are true in order to execute a statement.

```
if (X < max && x > min)
        cout<< “x is within limits”;
```

|| ⟶ We need to check that either of two conditions is true in order to execute a statement.

```
if (G == ‘E’ || G == ‘e’)
        cout << “ Students got Excellent degree” ;
```

Thank You!