Lab #0

GETTING STARTED WITH C++

OBJECTIVE:

_____

_____

_____

C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

C++ is regarded as a **middle-level** language, as it comprises a combination of both high-level and low-level language features.

Use of C++:

C++ is used by hundreds of thousands of programmers in essentially every application domain.

C++ is being highly used to write device drivers and other softwares that rely on direct manipulation of hardware under realtime constraints.

C++ is widely used for teaching and research because it is clean enough for successful teaching of basic concepts.

C++ Program Structure:

- The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header **<iostream>** is needed.
- The line **using namespace std;** tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.

- The next line **// main() is where program execution begins.** is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.
- The line **int main()** is the main function where program execution begins.
- The next line **cout << "This is my first C++ program.";** causes the message "This is my first C++ program" to be displayed on the screen.
- The next line **return 0;** terminates main( )function and causes it to return the value 0 to the calling process.
- Use **endl**, which inserts a new-line character after every line and << operator is being used to pass multiple values out to the screen.

Practice Program:

```
#include<iostream>
usingnamespace std;

// main() is where program execution begins.

int main()
{
   cout <<"Hello World";// prints Hello World
return0;
}
```

Semicolons & Blocks in C++:

In C++, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

For example, following are three different statements:

```
x = y;
y = y+1;
add(x, y);
```

A block is a set of logically connected statements that are surrounded by opening and closing braces. For example:

```
{
   cout <<"Hello World";// prints Hello World
return0;
}
```

C++ does not recognize the end of the line as a terminator. For this reason, it does not matter where on a line you put a statement. For example:

```
x = y;
y = y+1;
add(x, y);
```

is the same as

```
x = y; y = y+1; add(x, y);
```

C++ Keywords:

The following list shows the reserved words in C++. These reserved words may not be used as constant or variable or any other identifier names.

Asm	Else	new	This
Auto	enum	operator	Throw
Bool	explicit	private	True
break	export	protected	Try
case	extern	public	typedef
catch	False	register	Typeid
char	float	reinterpret_cast	typename

class	For	return	Union
const	friend	short	unsigned
const_cast	goto	signed	Using
continue	If	sizeof	virtual
default	inline	static	Void
delete	Int	static_cast	volatile
do	Long	struct	wchar_t
double	mutable	switch	While
dynamic_cast	namespace	template	

From greatest to smallest priority, C++ operators are evaluated in the following order:

Level	Precedence group	Operator	Description	Grouping
1	Scope	::	scope qualifier	Left-to-right
2	Postfix (unary)	++ --	postfix increment / decrement	Left-to-right
		()	functional forms	
		[]	subscript	
		. ->	member access	
3	Prefix (unary)	++ --	prefix increment / decrement	Right-to-left
		~ !	bitwise NOT / logical NOT	
		+ -	unary prefix	
		& *	reference / dereference	
		new delete	allocation / deallocation	
		sizeof	parameter pack	
		(*type*)	C-style type-casting	
4	Pointer-to-member	.* ->*	access pointer	Left-to-right

5	Arithmetic: scaling	* / %	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	+ -	addition, subtraction	Left-to-right
7	Bitwise shift	<<>>	shift left, shift right	Left-to-right
8	Relational	<><= >=	comparison operators	Left-to-right
9	Equality	== !=	equality / inequality	Left-to-right
10	And	&	bitwise AND	Left-to-right
11	Exclusive or	^	bitwise XOR	Left-to-right
12	Inclusive or	|	bitwise OR	Left-to-right
13	Conjunction	&&	logical AND	Left-to-right
14	Disjunction	||	logical OR	Left-to-right
15	Assignment-level expressions	= *= /= %= += -= >>= <<= &= ^= |=	assignment / compound assignment	Right-to-left
		?:	conditional operator	
16	Sequencing	,	comma separator	Left-to-right

Bitwise operators (&, |, ^, ~, <<, >>)

Bitwise operators modify variables considering the bit patterns that represent the values they store.

Operator	Equivalent	Description
&	AND	Bitwise AND
|	OR	Bitwise inclusive OR
^	XOR	Bitwise exclusive OR
~	NOT	Unary complement (bit inversion)

<<	SHL	Shift bits left
>>	SHR	Shift bits right

Relational and comparison operators (==, !=, >, <, >=, <=)

Two expressions can be compared using relational and equality operators. For example, to know if two values are equal or if one is greater than the other.

The result of such an operation is either true or false (i.e., a Boolean value).

The relational operators in C++ are:

operator	Description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

A comment can also **start with //,** extending to the end of the line. For example:

```
#include<iostream>
usingnamespace std;
main()
{
cout <<"Hello World";// prints Hello World
return0;
}
```

When the above code is compiled, it will ignore // **prints Hello World** and final executable will produce the following result:

```
HelloWorld
```

Within a /* and */ comment, // characters have no special meaning. Within a // comment, /* and */ have no special meaning. Thus, you can "nest" one kind of comment within the other kind. For example:

```
/* Comment out printing of Hello World:

cout << "Hello World"; // prints Hello World

*/
```

Variable Definition in C++:

A variable definition means to tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type, and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Here, **type** must be a valid C++ data type including char, w_char, int, float, double, bool or any user-defined object, etc., and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

- The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.
- Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

```
type variable_name = value;
```

Some examples are:

```
externint d =3, f =5;// declaration of d and f.
int d =3, f =5;// definition and initializing d and f.
char x ='x';// the variable x has the value 'x'.
```

Without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables is undefined.

Variable Declaration in C++:

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable.

Example

Try the following example where a variable has been declared at the top, but it has been defined inside the main function:

```
#include<iostream>
usingnamespace std;

// Variable declaration:
externint a, b;
externint c;
externfloat f;

int main ()
{
// Variable definition:
int a, b;
int c;
float f;

// actual initialization
  a =10;
  b =20;
  c = a + b;

  cout << c << endl ;
```

```
  f =70.0/3.0;
  cout << f << endl ;

return0;
}
```

When the above code is compiled and executed, it produces the following result:

```
30
23.3333
```

Local Variables:

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables:

```
#include<iostream>
usingnamespace std;
int main ()
{
// Local variable declaration:
int a, b;
int c;
// actual initialization
  a =10;
  b =20;
  c = a + b;
  cout << c;
return0;
}
```

Global Variables:

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

```
#include<iostream>
usingnamespace std;

// Global variable declaration:
int g;

int main ()
{
// Local variable declaration:
int a, b;

// actual initialization
  a =10;
  b =20;
  g = a + b;

  cout << g;

return0;
}
```

Lab Task

Task No 1

a) What is the output of the following program?

```
#include<iostream>
usingnamespace std;
main()
{
int i =1, j =2, k =3, r;
r =(i, j, k);
cout<<r<<endl;
}
```

a) What is the output of the following program?

```
#include<iostream>
usingnamespace std;
main()
{
intconst a =5;
a++;
cout<<a;}
```

Task 2

a) Write a program that adds and subtracts two integers using variable int.

b) Write a program that generates an array with size 10.