Week 1: Lecture 2

Structural Programming Review in C++ Environment

Variable : Memory Concept

Programs shown in the previous section print text on the screen. This section will introduce the concept of variable so that our program can perform calculation on data.

Program : Adding two numbers

We'll solve this problem in C++ with the following steps:

STEP 1 : Allocate memory for storing three numbers
STEP 2 : Store first number in computer memory
STEP 3 : Store second number in computer memory
STEP 4 : Add these two numbers together and store the result of the addition in a third memory location
STEP 5 : Print the result

STEP 1 : Now first we'll allocate memory for storing numbers. Location of the computer memory which is used to store data and is given a symbolic name for reference is known as variable. We need three variables, two for storing input and third for storing result. Before a variable is used in a program, we must declare it. This activity enables the compiler to make available the appropriate type of location in the memory.

Following statements declare three variables of type integer to store whole numbers.

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

You can declare more than one variable of same type in a single statement like

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

STEP 2 : Following statement stores value in first variable

```
x = 25;
```

STEP 3 : Following statement stores value in second variable

```
y = 10;
```

STEP 4 : Now, add these two numbers together and store the result of the addition in third variable

```
z = x + y;
```

STEP 5 : Print the result

```
cout << "The sum is ";
cout << sum;
```

You can combine above two statements in one statement

```
cout << "The sum is " << sum;
```

Here, is the complete program

```
#include <iostream>
using namespace std;
int main()
{
//declare variables of integer type
int x;
int y;
int z;
//storing value in variables
x = 25;
y = 10;
//adding numbers and store the result in sum
z = x + y;
//print the result
cout << "The sum is ";
cout << z;
return 0;
}
```

Output : The sum is 35

Identifiers

Symbolic names can be used in C++ for various data items used by a programmer in his program. A symbolic name is generally known as an identifier. The identifier is a sequence of characters taken from C++ character set. In previous program x, y and z are identifiers of variables. The rule for the formation of an identifier are:

- An identifier can consist of alphabets, digits and/or underscores.
- It must not start with a digit
- C++ is case sensitive that is upper case and lower case letters are
- considered different from each other.
- It should not be a reserved word.

Keywords

There are some reserved words in C++ which have predefined meaning to compiler called keywords. These words may not be used as identifiers. Some commonly used Keywords are given below:

asm	auto	bool	break	case
catch	char	class	const	const_cast
continue	default	delete	do	double
dynamic_cast	else	enum	explicit	export
extern	false	float	for	friend
goto	if	inline	int	long
mutable	namespace	new	operator	private
protected	public	register	reinterpret_cast	return
short	signed	sizeof	static	static_cast
struct	switch	template	this	throw
true	try	typedef	typeid	typename
union	unsigned	using	virtual	void
volatile	wchar_t	while		

Basic Data Types

C++ supports a large number of data types. The built in or basic data types supported by C++ are integer, floating point and character. C++ also provides the data type bool for variables that can hold only the values True and false.

Some commonly used data types are summarized in table along with description.

Type Description

Type	Description
int	Small integer number
long int	Large integer number
float	Small real number
double	Double precision real number
long double	Long double precision real number
char	A Single Character

Arithmetical operators

Arithmetical operators +, -, *, /, and % are used to performs an arithmetic (numeric) operation. You can use the operators +, -, *, and / with both integral and floating-point data types. Modulus or remainder % operator is used only with the integral data type.
Operators that have two operands are called binary operators.

Relational operators

The relational operators are used to test the relation between two values. All relational operators are binary operators and therefore require two operands. A relational expression returns zero when the relation is false and a non-zero when it is true. The following table shows the relational operators.

Relational Operators	Meaning
<	Less than
<=	Less than or equal to
==	Equal to
>	Greater than
>=	Greater than or equal to
! =	Not equal to

Logical operators

The logical operators are used to combine one or more relational expression.
The logical operators are

Operators	Meaning
||	OR
&&	AND
!	NOT

Unary operators

C++ provides two unary operators for which only one variable is required.

For Example
```
a = - 50;
a = + 50;
```

Here plus sign (+) and minus sign (-) are unary because they are not used between two variables.

Assignment operator

The assignment operator '=' is used for assigning a variable to a value. This operator takes the expression on its right-hand-side and places it into the variable on its left-hand-side. For example:
```
m = 5;
```

The operator takes the expression on the right, 5, and stores it in the variable on the left, m.
```
x = y = z = 32;
```

This code stores the value 32 in each of the three variables x, y, and z.

Input/Output (I/O)

The standard C++ library includes the header file iostream, which can be used to feed new data into the computer or obtain output on an output device such as: VDU, printer etc. The following C++ stream objects can be used for the input/output purpose.

cout console output
cin console input

cout object

cout is used to print message on screen in conjunction with the insertion operator <<

```
cout << "Hello World"; // prints Hello world on screen
cout << 250; // prints number 250 on screen
cout << sum; // prints the content of variable sum on screen
```

To print constant strings of characters we must enclose them between double quotes (").
If we want to print out a combination of variables and constants, the insertion operator (<<) may be used more than once in a single statement

```
cout << "Area of rectangle is " << area << " square meter" ;
```

If we assume the area variable to contain the value 24 the output of the previous statement would be:

Area of rectangle is 24 square meter

cin object

cin can be used to input a value entered by the user from the keyboard. However, the extraction operator >> is also required to get the typed value from cin and store it in the memory location.

Let us consider the following program segment:

```
int marks;
cin >> marks;
```

In the above segment, the user has defined a variable mark of integer type in the first statement and in the second statement he is trying to read a value from the keyboard.

```
// input output example
#include <iostream>
using namespace std;
int main ()
{
int length;
int breadth;
int area;
cout << "Please enter length of rectangle: ";
cin >> length;
cout << "Please enter breadth of rectangle: ";
cin >> breadth;
area = length * breadth;
cout << "Area of rectangle is " << area;
return 0;
}
```

Output :

```
Please enter length of rectangle: 6
Please enter breadth of rectangle: 4
Area of rectangle is 24
```

You can also use cin to request more than one input from the user:

```
cin >> length >> breadth;
```

is equivalent to:

```
cin >> length;
cin >> breadth;
```

cin and strings

We can use cin to get strings with the extraction operator (>>) as we do with fundamental data type variables:

```
cin >> mystring;
```

However, cin extraction stops reading as soon as if finds any blank space character, so in this case we will be able to get just one word for each extraction.

For example if we want to get a sentence from the user, this extraction operation would not be useful. In order to get entire lines, we can use the function getline, which is the more recommendable way to get user input with cin:

```
// cin and strings
#include <iostream>
#include <string>
using namespace std;
int main ()
{
string name;
cout << "Enter your name";
getline (cin, name);
cout << "Hello " << name << "!\n";
return 0;
}
```

Output:

```
Enter your name : Rashid Ayan
Hello Rashid Ayan!
```

Conditional Statements

Sometimes the program needs to be executed depending upon a particular condition. C++ provides the following statements for implementing the selection control structure.

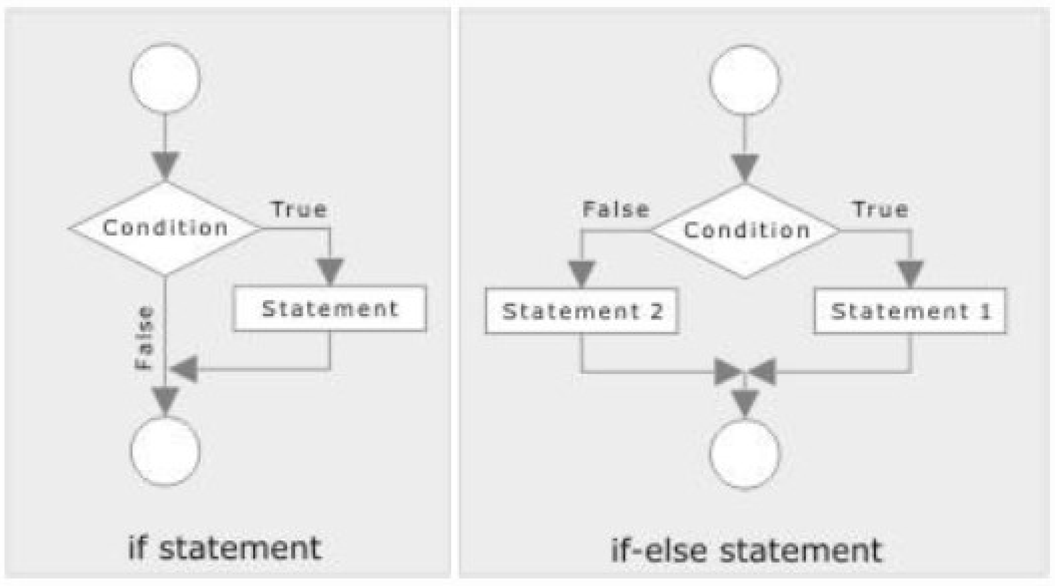
- if statement
- if else statement
- nested if statement
- switch statement

if statement

syntax of the if statement

```
if (condition)
{
statement(s);
}
```

From the flowchart it is clear that if the if condition is true, statement is executed; otherwise it is skipped. The statement may either be a single or compound statement.



if else statement

syntax of the if - else statement

```
if (condition)
statement1;
else
statement2;
```

From the above flowchart it is clear that the given condition is evaluated first. If the condition is true, statement1 is executed. If the condition is false, statement2 is executed. It should be kept in mind that statement and statement2 can be single or compound statement.

if example	if else example

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

Nested if statement

The if block may be nested in another if or else block. This is called nesting of if or else block.

Syntax of the nested if statement

```
if(condition 1)
{
if(condition 2)
{
statement(s);
}
}
if(condition 1)
statement 1;
else if (condition 2)
statement2;
else
statement3;
```

if-else-if example

```
if(percentage>=60)
      cout<<"Ist division";
else if(percentage>=50)
      cout<<"IInd division";
else if(percentage>=40)
      cout<<"IIIrd division";
else
      cout<<"Fail" ;
```

switch statement

The if and if-else statements permit two way branching whereas switch statement permits multiple branching. The syntax of switch statement is:

```
switch (var / expression)
{
case constant1 : statement 1;
break;
case constant2 : statement2;
break;
. .
default: statement3;
break;
}
```

The execution of switch statement begins with the evaluation of expression. If the value of expression matches with the constant then the statements following this statement execute sequentially till it executes break. The break statement transfers control to the end of the switch statement. If the value of expression does not match with any constant, the statement with default is executed.

Some important points about switch statement

- The expression of switch statement must be of type integer or character type.
- The default case need not to be used at last case. It can be placed at any place.
- The case values need not to be in specific order.

Week 2: Lecture 3

Structural Programming Review in C++ Environment

Flow of Control Statements

Statements are the instructions given to the computer to perform any kind of action. Action may be in the form of data movement, decision making etc. Statements form the smallest executable unit within a C++ program. Statements are always terminated by semicolon.

Compound Statement

A compound statement is a grouping of statements in which each individual statement ends with a semi-colon. The group of statements is called block. Compound statements are enclosed between the pair of braces ({}.). The opening brace ({) signifies the beginning and closing brace (}) signifies the end of the block.

Null Statement

Writing only a semicolon indicates a null statement. Thus ';' is a null or empty statement. This is quite useful when the syntax of the language needs to specify a statement but the logic of the program does not need any statement. This statement is generally used in for and while looping statements.

Conditional Statements

Sometimes the program needs to be executed depending upon a particular condition. C++ provides the following statements for implementing the selection control structure.

1. if statement
2. if else statement
3. nested if statement
4. switch statement

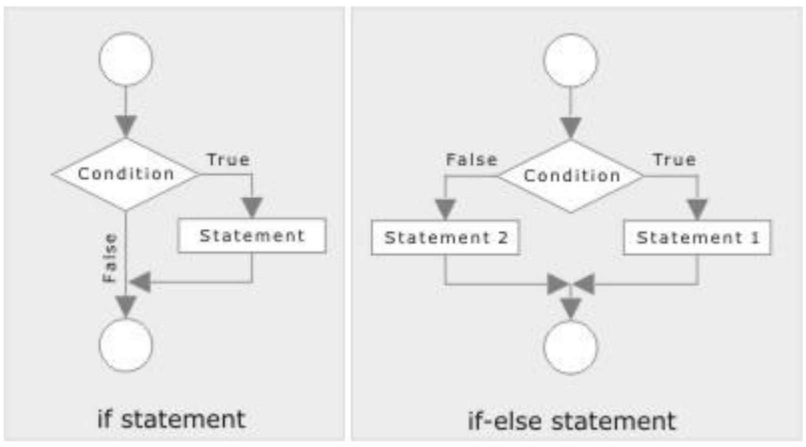
if statement

syntax of the if statement

```
if (condition) { statement(s); }
```

From the flowchart it is clear that if the if condition is true, statement is executed; otherwise it is skipped. The statement may either be a single or compound statement.

From the flowchart it is clear that if the if condition is true, statement is executed; otherwise it is skipped. The statement may either be a single or compound statement.



if else statement

syntax of the if - else statement

```
if (condition) statement1; else statement2;
```

From the above flowchart it is clear that the given condition is evaluated first. If the condition is true, statement1 is executed. If the condition is false, statement2 is executed. It should be kept in mind that statement and statement2 can be single or compound statement.

if example	if else example

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

Nested if statement

The if block may be nested in another if or else block. This is called nesting of if or else block.

syntax of the nested if statement

```
if(condition 1)
{
      if(condition 2)
      {
             statement(s);
      }
}

if(condition 1)
      statement 1;
else if (condition 2)
      statement2;
else
      statement3;
```

if-else-if example

```
if(percentage>=60)
    cout<<"Ist division";
else if(percentage>=50)
    cout<<"IInd division";
else if(percentage>=40)
    cout<<"IIIrd division";
else
    cout<<"Fail" ;
```

switch statement

The if and if-else statements permit two way branching whereas switch statement permits multiple branching. The syntax of switch statement is:

```
switch (var / expression)
{
      case constant1 :
            statement 1;
            break;
      case constant2 :
            statement2;
            break;
      . . .
      default:
            statement3;
            break;
}
```

The execution of switch statement begins with the evaluation of expression. If the value of expression matches with the constant then the statements following this statement execute sequentially till it executes break. The break statement transfers control to the end of the switch statement. If the value of expression does not match with any constant, the statement with default is executed.

Some important points about switch statement

- The expression of switch statement must be of type integer or character type.
- The default case need not to be used at last case. It can be placed at any place.
- The case values need not to be in specific order.

Week 2: Lecture 4

Structural Programming Review in C++ Environment

Looping statement

It is also called a Repetitive control structure. Sometimes we require a set of statements to be executed a number of times by changing the value of one or more variables each time to obtain a different result. This type of program execution is called looping. C++ provides the following construct

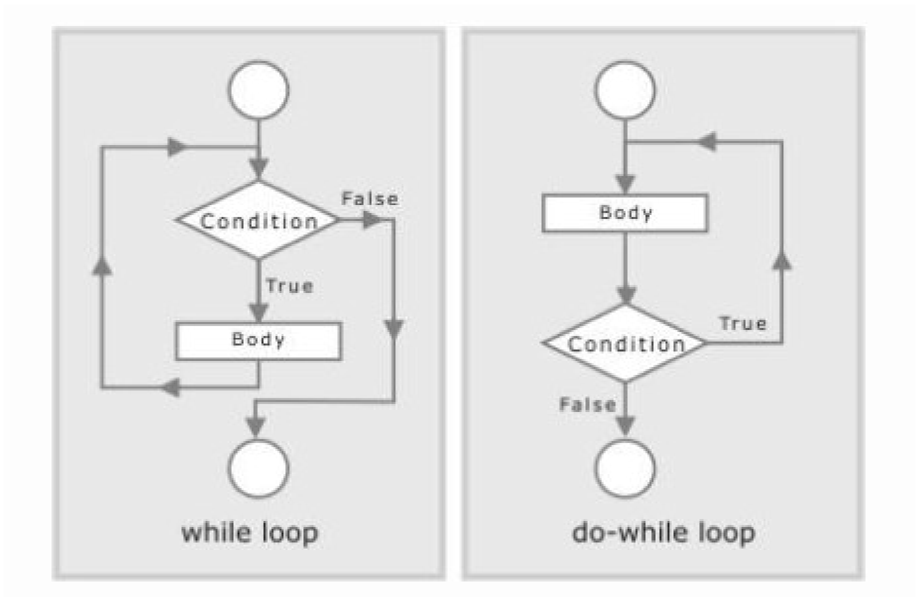
- while loop
- do-while loop
- for loop

While loop

Syntax of while loop

```
while(condition)
{
statement(s);
}
```

The flow diagram indicates that a condition is first evaluated. If the condition is true, the loop body is executed and the condition is re-evaluated. Hence, the loop body is executed repeatedly as long as the condition remains true. As soon as the condition becomes false, it comes out of the loop and goes to the statement next to the 'while' loop.



do-while loop

Syntax of do-while loop

```
do
{
statements;
} while (condition);
```

Note : That the loop body is always executed at least once. One important difference between the while loop and the do-while loop the relative ordering of the conditional test and loop body execution. In the while loop, the loop repetition test is performed before each execution the loop body; the loop body is not executed at all if the initial test fail. In the do-while loop, the loop termination test is Performed after each execution of the loop body. hence, the loop body is always executed least once.

for loop

It is a count controlled loop in the sense that the program knows in advance how many times the loop is to be executed.
Syntax of for loop

```
for (initialization; decision; increment/decrement)
{
statement(s);
}
```

The flow diagram indicates that in for loop three operations take place:

- Initialization of loop control variable
- Testing of loop control variable
- Update the loop control variable either by incrementing or decrementing.

Operation (i) is used to initialize the value. On the other hand, operation (ii) is used to test whether the condition is true or false. If the condition is true, the program executes the body of the loop and then the value of loop control variable is updated. Again it checks the condition and so on. If the condition is false, it gets out of the loop.

Jump Statements

The jump statements unconditionally transfer program control within a function.

- goto statement
- break statement
- continue statement

The goto statement

goto allows to make jump to another point in the program.

```
goto pqr;
```

`pqr:` pqr is known as label. It is a user defined identifier. After the execution of goto statement, the control transfers to the line after label pqr.

The break statement

The break statement, when executed in a switch structure, provides an immediate exit from the switch structure. Similarly, you can use the break statement in any of the loop. When the break statement executes in a loop, it immediately exits from the loop.

The continue statement

The continue statement is used in loops and causes a program to skip the rest of the body of the loop.

```
while (condition)
{
Statement 1;
If (condition)
continue;
statement;
}
```

The continue statement skips rest of the loop body and starts a new iteration.

The exit ( ) function

The execution of a program can be stopped at any point with exit ( ) and a status code can be informed to the calling program. The general format is exit (code) ; where code is an integer value. The code has a value 0 for correct execution.

The value of the code varies depending upon the operating system.