**Programming Fundamentals**

|  |  |
| --- | --- |
| Lab 01 | |
| **Topic** | ITC-Revision |
| **Objective** | * Introduction & Review   + Program Execution Flow, Concept of TYPE, Identifier, Declaration, Initialization, Expression, Assignment, Selection, Loop, Arrays * Array   + Integer, Character, C-String   + Declaration, Initialization, one by one Input/Output. |

**Lab Description:**

This lab is basically design for the revision of ITC concepts.

**First of all, let’s discuss about the basic concepts of ITC.**

As we know that a computer can understand only machine language. Therefore, in order to run any program successfully, the code must first be translated into machine language. Let’s have a quick review of the steps required to execute programs written in C++.

The following steps, are necessary to process a C++ program.

1. You use a text editor to create a C++ program following the rules, or syntax, of the high-level language. This program is called the source code, or source program. The program must be saved in a text file that has the extension .cpp. For example, if you saved the preceding program in the file named myProgram, then its complete name is myProgram.cpp.

**Source program:** A program written in a high-level language.

1. The C++ program given in the preceding section contains the statement #include. In a C++ program, statements that begin with the symbol # are called preprocessor directives. These statements are processed by a program called preprocessor.
2. After processing preprocessor directives, the next step is to verify that the program obeys the rules of the programming language—that is, the program is syntactically correct—and translate the program into the equivalent machine language. The compiler checks the source program for syntax errors and, if no error is found, translates the program into the equivalent machine language. The equivalent machine language program is called an object program.

**Object program:** The machine language version of the high-level language program.

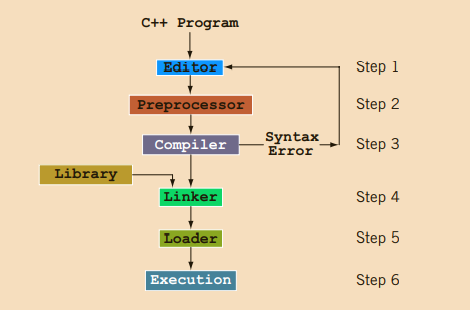
1. The programs that you write in a high-level language are developed using an integrated development environment (IDE). The IDE contains many programs that are useful in creating your program. For example, it contains the necessary code (program) to display the results of the program and several mathematical functions to make the programmer’s job somewhat easier. Therefore, if certain code is already available, you can use this code rather than writing your own code. Once the program is developed and successfully compiled, you must still bring the code for the resources used from the IDE into your program to produce a final program that the computer can execute. This prewritten code (program) resides in a place called the library. A program called a linker combines the object program with the programs from libraries.

**Linker:** A program that combines the object program with other programs in the library and is used in the program to create the executable code.

1. You must next load the executable program into main memory for execution. A program called a loader accomplishes this task.

**Loader:** A program that loads an executable program into main memory.

1. The final step is to execute the program.



The smallest individual unit of any program written in any language is called a token. C++’s tokens are divided into special symbols, word symbols, and identifiers. Following are some of the special symbols:

**Special symbols:**

Special symbols are the operator which are used to perform different operations like:

1. Arithmetic operators (+, -, \*, /, %)
2. Logical operators (&&, ||, !)
3. Relational operators (>, <, >=, <=, ==, !=)

**Word symbols:**

Word symbols are the reserve words, which are used for different purpose. Reserve word are also known as key words.

Some of the commonly used key words are: int, float, double, char, void, return, if, else, while, for.

**Identifiers:**

C++ identifier consists of letters, digits, and the underscore character (\_) and must begin with a letter or underscore, no other symbols are permitted to form an identifier. C++ is case sensitive—uppercase and lowercase letters are considered different. Thus, the identifier NUMBER is not the same as the identifier number. Similarly, the identifiers X and x are different. It’s a good practice to define meaningful identifier name, like if u need to store salary of a person than identifier name should be salary not a, b or c.

**Data Type:**

Data type is a set of values together with a set of allowed operations. Integral, which is a data type that deals with integers, or numbers without a decimal part. Floating point, which is a data type that deals with decimal numbers. Some of the data types which are commonly used are: integer(int), floating point(float), character(char), Boolean(bool).

**How to define an identifier?**

Whenever we use an identifier first we need to define it otherwise system creates a syntax error (undefined identifier used). So we need to learn how to create an identifier.

**The generic way of define an identifier is:** dataType IdentifierName;

When we declare an identifier it holds a value which is not assigned by user so for user this value is considered as garbage value. So, it is a good practice to assign a value to an identifier at the time of creation which is called initialization.

Initialization statement:

Assigning a value at the time of declaration of a variable is called an initialization statement.

**The generic way of initialize an identifier is:** dataType IdentifierName = value;

In case of integer, float, double:

int noOfStudentsInClass=45;

float averageOfClass=60.5;

double length=5; or double length=6.85;

but in case of character and boolean the way of initialization is different.

In case of character assigned value is written in single quotes.

char grade=’A’;

In case of Boolean assigned value is written as true or false. But we can also write it as 0 or 1 form. 0 represent as false and 1 represent as true.

bool isNumberFound=true or bool isNumberFound=1;

**Can we change the value of an identifier?**

Yes, we can change the value of an identifier. There are multiple ways of doing this. We can change the value by assigning method and input method.

**Assigning method:**

There are two ways which are used as assigning method assigning value and assigning another identifier.

**Assigning a value:**

Identifier= value;

**Assigning an identifier:**

Identifier=identifier;

**Is there any difference between assigning statement and initialization statement?**

Yes, in initialization we assign a value at the time of creation of an identifier. On another hand in assigning statement we assign a value to an already defined identifier.

**Input method:**

We can change the value of an already define variable by taking the input from user. Input statement is used for this purpose.

cin>> identifier;

it’s a good practice to display an input message on console before an input statement.

**How to display output on console?**

in C++ we can display different type of data on console using output statement.

**Message output:**

cout<<”anything written in double quotes display as it is as a mesaage”;

**Identifier output:**

cout<<identifierName;

In case of identifier the value of identifier displays on console.

**Expression output:**

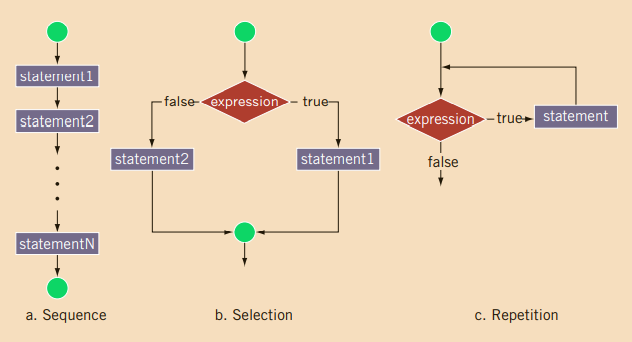
cout<< combinataion of operand and operators;

In case of expression the result value of expression displays on console.

**Processing a program:**

There are three ways of process a program.

* Sequence
* Selection
* Repetition



**Sequence:**

In sequence way of processing code is executed line by line. Each line of code executed in a sequence in which those instructions are defined.

**Selection:**

In selection, the program executes particular statements depending on some conditions. There are multiple selection statements:

**if:**

syntax:

if(expression)

{

Body of if(statement);

}

In this selection statement a set of instruction are dependent on the condition/expression of selection statement if condition/expression is true body of selection statement will be executed. If condition/expression is false body of selection statement will be skipped and next instructions which are defined after selection statement will be executed.

**if else:**

syntax:

if(expression)

{

Body of if(statement);

}

else

{

Body of else(statements);

}

In this selection statement a set of instruction are dependent on the condition/expression of selection statement if condition/expression is true body of selection statement will be executed. If condition/expression is false body of selection statement will be skipped and body of else will be executed. Else part doesn’t have any condition/expression.

**if else if:**

syntax:

if(expression)

{

Body of if(statement);

}

else if(expression)

{

Body of else(statements);

}

In this selection statement a set of instruction are dependent on the condition/expression of selection statement if condition/expression is true body of selection statement will be executed. If condition/expression is false, then next selection statement’s condition/expression will be tested and so on. If any of the selection statement is tested as true than no further statement will be test and the control of program will be shifted after those selection statement.

**multiple if:**

syntax:

if(expression)

{

Body of if(statement);

}

if(expression)

{

Body of if(statement);

}

In this selection statement a set of instruction are dependent on the condition/expression of selection statement if condition/expression is true body of selection statement will be executed. But it is quite different from above mention statements. Because, if any selection statement is true it will not have skipped any statement. Each and every selection statement will be tested and executed accordingly.

**nested if:**

syntax:

if(expression)

{

if(expression)

{

Body of if(statement);

}

}

In this selection statement a set of instruction are dependent on the condition/expression of selection statement if condition/expression is true body of selection statement will be executed which is another selection statement.

**Repetition:**

In repetition, the program repeats particular statements a certain number of times based on some conditions. Normally three types repetition statement are used:

**Counter control:**

In this kind of repetition, a set of statement repeats specific number of times according to the requirement. You know exactly how many times certain statements need to be executed.

**Sentinel control:**

You do not always know how many pieces of data (or entries) need to be read, but you may know that the last entry is a special value, called a sentinel.

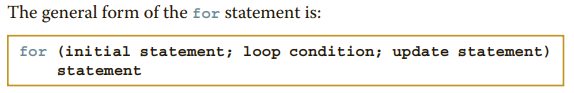
**Flag control:**

A flag variable is a bool variable that indicates whether a condition is true or false.

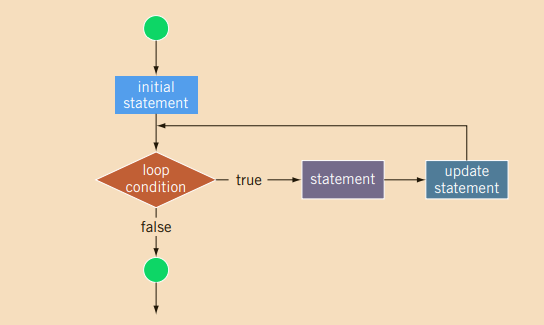
There are multiple repetition statements:

**for:**

Syntax:



Flow of for loop:



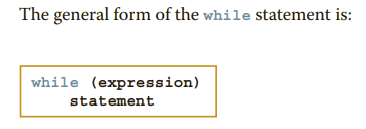
The for loop executes as follows:

1. The initial statement executes.
2. The loop condition is evaluated. If the loop condition evaluates to true:
   1. Execute the for loop statement.
   2. Execute the update statement (the third expression in the parentheses).
3. Repeat Step 2 until the loop condition evaluates to false.

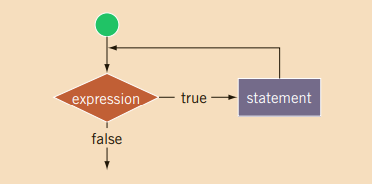
The initial statement usually initializes a variable (called the for loop control, or for indexed, variable). In C++, for is a reserved word.

**while:**

Syntax:



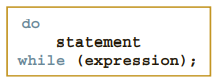
Flow of while loop:



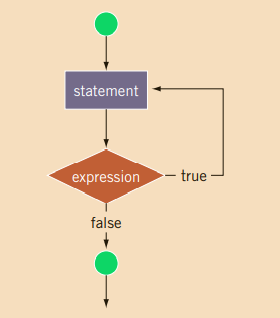
The expression provides an entry condition to the loop. If it initially evaluates to true, the statement executes. The loop condition—the expression—is then reevaluated. If it again evaluates to true, the statement executes again. The statement (body of the loop) continues to execute until the expression is no longer true. A loop that continues to execute endlessly is called an infinite loop. To avoid an infinite loop, make sure that the loop’s body contains statement(s) that assure that the entry condition—the expression in the while statement—will eventually be false. In C++, while is a reserve word.

**do while:**

The general form of do while loop is:



Flow of do while loop:



The statement executes first, and then the expression is evaluated. If the expression evaluates to true, the statement executes again. As long as the expression in a do. . .while statement is true, the statement executes. To avoid an infinite loop, you must, once again, make sure that the loop body contains a statement that ultimately makes the expression false and assures that it exits properly. In C++, do is a reserved word.

Nested Repetition statement:

A repetition statement is defined inside a repetition statement is called nested repetition.

**Array:**

An array is a collection of a fixed number of components (also called elements) all of the same data type and in contiguous (that is, adjacent) memory space.

**Declaration statement of an array:**

dataType identifierName[numberOfElementsRequired];

Above statement is used for creating an array. But as we discussed above when a memory location is reserved for this array it holds some values which are not assigned by user. So those values are considered as garbage value. In order to avoid garbage value, it is a good practice to assign values at the time of creation of array.

**Initialization statement of an array:**

Assigning value at the time of declaring of array or variable is called initialization statement. There are multiple ways of initialize an array.

**Initialize an array with 0:**

dataType identifierName[numberOfElementsRequired]={};

OR

dataType identifierName[numberOfElementsRequired]{};

Example:

int arr[5]={}; OR int arr[5]{};

**Full array initialization with different elements:**

dataType identifierName[numberOfElementsRequired]={required # of elements but comma separated values};

Example:

int arr[5]={1,2,3,4,5};

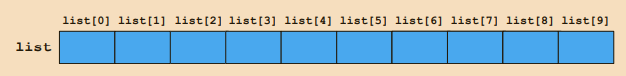
**Partial array initialization with different elements:**

dataType identifierName[numberOfElementsRequired]={less than required # of elements but comma separated values};

int arr[5]={1,2,3};

on remaining indexes 0 will be assigned as initial value in case of partial array initialization.

**Accessing array components:**



**Generic way of accessing an array is:** IdentifierName[index#];

int list[10]; //list[0] is use for accessing first element of an array.

**Array input:**

We can take input from user into an array. We can take input index wise one by one and we can also take input at specific index number.

**Input at specific index:**

cin>> list[0]; //taking input on index 0 which is the first element of an array.

**Input in whole array:**

We use repetition statement(loops) for taking input in whole array. We can take input one by one as mention above but it is not considered as a good practice.

for(int i=0;i<size;i++)// i is used as index number and size is the number element you want to enter.

{

cin>>list[i]; // taking input at specific index i. After every iteration value of i will be updated.

}

**Array output:**

We can display the values of array on console as output to user. We can display index wise one by one and we can also take input at specific index number.

**Output of specific index:**

cout<< list[0]; //display the value of index 0 which is the first element of an array.

**Output of whole array:**

We use repetition statement(loops) for display whole array. We can display elements one by one as mention above but it is not considered as a good practice.

for(int i=0;i<size;i++)// i is used as index number and size is the number element you want to display.

{

cout<<list[i]; // display specific index i. After every iteration value of i will be updated.

}

**Array index out of bound:**

Index of an array is in bounds if index is between 0 and ARRAY\_ SIZE - 1, that is, 0 <= index <= ARRAY\_SIZE - 1. If index is negative or index is greater than ARRAY\_SIZE - 1, then we say that the index is out of bounds.

**Lab Tasks**

**Swap Values**

Program that reads in two integer variables and swaps the value of these variables.

**Power of number**

Program which calculates power of any integer number using for loop.

**Print Odd integers**

Program that uses a for loop to calculate and print the sum of the odd integers from 1 to 49.

**Print even numbers in array**

Program that takes input from user in integer array of length 10 using for loop and only print even numbers entered in input array

**Display Sum and average**

Program which takes input integer array from user of size 7. Your task is to store and calculate the Sum and Average of 7 numbers entered by the user.

**Pattern Printing**

Program to print the following pattern by using nested loop.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 4 | 3 | 2 | 1 |
| - | 5 | 4 | 3 | 2 |
| - | - | 5 | 4 | 3 |
| - | - | - | 5 | 4 |
| - | - | - | - | 5 |

**Pattern Printing**

Program to print the following pattern by using nested loop.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  |  |  |
| 2 | 3 |  |  |
| 4 | 5 | 6 |  |
| 7 | 8 | 9 | 10 |

**Repeated Elements**

Program that takes 10 inputs in integer array and find the duplicate elements of the array and then display it. If no duplicate value exists then display message accordingly

**Sorting**

Program that takes input integer array of size 5 from user and sort it in ascending order.

**Print Vowels**

Program that takes input from user in character array of length 10 using for loop and only print vowels entered in input array

**Length of Character array**

Program that asks user to input character array and then your task is to find the exact length of that character array i.e. how many characters are there in an array. You must not include spaces.

**Reverse of array**

Program that asks user to input character array and then your task is to display the reverse of the array characters.