CCA-101: Fundamentals of IT & Programming

Assignment - 2

Q1. What is the difference between Machine Language and High Level Language?

Programming Languages

A program is a set of instructions that tells a computer what to do in order to come up with a solution to a particular problem. Programs are written using a programming language. A **programming language** is a formal language designed to communicate instructions to a computer. There are two major types of programming languages: low-level languages and high-level languages.

Low-Level Languages

Low-level languages are referred to as 'low' because they are very close to how different hardware elements of a computer actually communicate with each other. **Low-level languages** are machine oriented and require extensive knowledge of computer hardware and its configuration. There are two categories of low-level languages: machine language and assembly language.

Machine language, or **machine code**, is the only language that is directly understood by the computer, and it does not need to be translated. All instructions use binary notation and are written as a string of 1s and 0s. A program instruction in machine language may look something like this:

```
10010101100101001111101010011011100101
```

Technically speaking, this is the only language computer hardware understands. However, binary notation is very difficult for humans to understand. This is where assembly languages come in.

An assembly language is the first step to improve programming structure and make machine language more readable by humans. An **assembly language** consists of a set of symbols and letters. A translator is required to translate the assembly language to machine language. This translator program is called the 'assembler.' It can be called the second generation language since it no longer uses 1s and 0s to write instructions, but terms like `MOVE`, `ADD`, `SUB` and `END`.

Many of the earliest computer programs were written in assembly languages. Most programmers today don't use assembly languages very often, but they are still used for applications like operating systems of electronic devices and technical applications, which use very precise timing or optimization of computer resources. While easier than machine code, assembly languages are still pretty difficult to understand. This is why high-level languages have been developed.

High-Level Languages

A **high-level language** is a programming language that uses English and mathematical symbols, like +, -, % and many others, in its instructions. When using the term 'programming languages,' most people are actually referring to high-level languages. High-level languages are the languages most often used by programmers to write programs. Examples of high-level languages are C++, Fortran, Java and Python.

To get a flavor of what a high-level language actually looks like, consider an ATM machine where someone wants to make a withdrawal of $100. This amount needs to be compared to the account balance to make sure there are enough funds. The instruction in a high-level computer language would look something like this:

```
x = 100
if balance  x:
 print 'Insufficient balance'
else:
 print 'Please take your money'
```

This is not exactly how real people communicate, but it is much easier to follow than a series of 1s and 0s in binary code.

There are a number of advantages to high-level languages. The first advantage is that high-level languages are much closer to the logic of a human language. A high-level language uses a set of rules that dictate how words and symbols can be put together to form a program. Learning a high-level language is not unlike learning another human language - you need to learn vocabulary and grammar so you can make sentences. To learn a programming language, you need to learn commands, syntax and logic, which correspond closely to vocabulary and grammar.

The second advantage is that the code of most high-level languages is portable and the same code can run on different hardware. Both machine code and assembly languages are hardware specific and not portable. This means that the machine code used to run a program on one specific computer needs to be modified to run on another computer. Portable code in a high-level language can run on multiple computer systems without modification. However, modifications to code in high-level languages may be necessary because of the operating system. For example, programs written for Windows typically don't run on a Mac.

A high-level language cannot be understood directly by a computer, and it needs to be translated into machine code. There are two ways to do this, and they are related to how the program is executed: a high-level language can be compiled or interpreted.

Q2. Discuss about different data types of C programming Language.

Ans. Data Type

A data type is a type of data. Of course, that is rather circular definition, and also not very helpful. Therefore, a better definition of a data type is a data storage format that can contain a specific type or range of values.

When computer programs store data in variables, each variable must be assigned a specific data type. Some common data types include integers, floating point numbers, characters, strings, and arrays. They may also be more specific types, such as dates, timestamps, boolean values, and varchar (variable character) formats.

Some programming languages require the programmer to define the data type of a variable before assigning it a value. Other languages can automatically assign a variable's data type when the initial data is entered into the variable. For example, if the variable "var1" is created with the value "1.25," the variable would be created as a floating point data type. If the variable is set to "Hello world!," the variable would be assigned a string data type. Most programming languages allow each variable to store a single data type. Therefore, if the variable's data type has already been set to an integer, assigning string data to the variable may cause the data to be converted to an integer format.

Data types are also used by database applications. The fields within a database often require a specific type of data to be input. For example, a company's record for an employee may use a string data type for the employee's first and last name. The employee's date of hire would be stored in a date format, while his or her salary may be stored as an integer. By keeping the data types uniform across multiple records, database applications can easily search, sort, and compare fields in different records.

Data types in C programming language

One of the most important concept in programming is the **variable**. The variable can be seen as the “place” to store “things” as: numerical values, characters, text strings, memory addresses, etc. There are two main concepts regarding variables. The fist concept is the **declaration** of the variable, which basically means setting its **data type**. The second concept is the **definition** of the variable, which means setting its **content**.
In the C programming language every variable used in the source code needs to be declared by setting its **data type**. By assigning a certain data type to a variable we define several aspects linked to the variable:

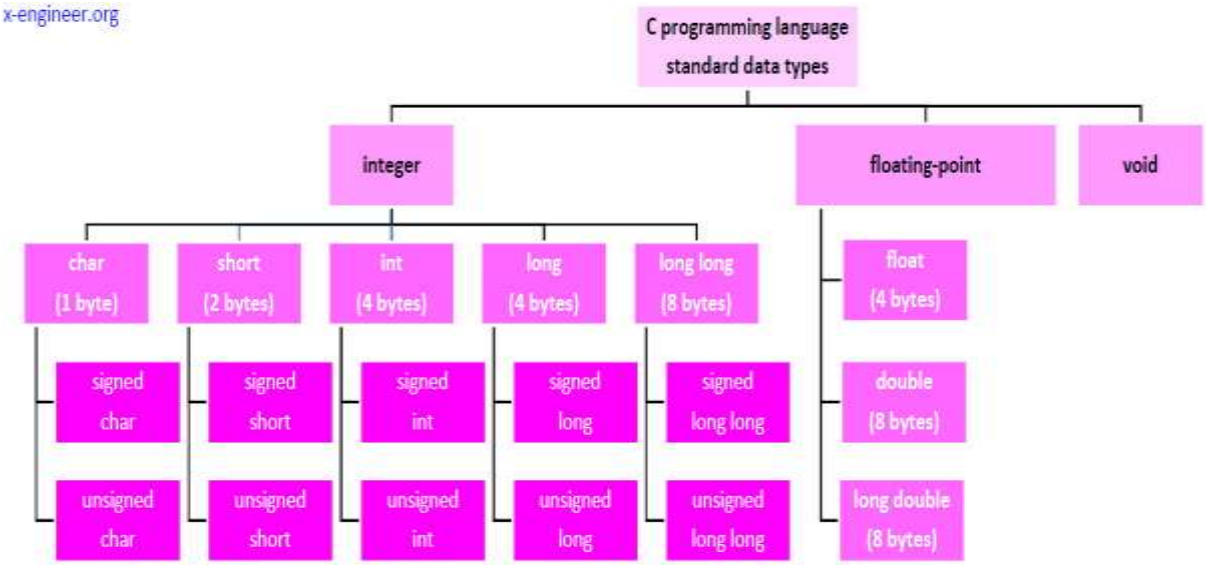
- the memory size to be allocated to store the content of the variable

- the types of operations that can be performed on the variable
- the restrictions which are applied in terms of operations

By the end of this tutorial the reader will know:

- what is the significance of a data type
- how to declare and define a variable
- which are the properties of the standard data types
- what is integer overflow

In the C programming language a variable is **declared** as



Q4. Describe the syntax of the following statements

a) If – else statement

Ans. In the last tutorial we learned how to use if statement in C. In this guide, we will learn how to use if else, nested if else and else if statements in a C Program.

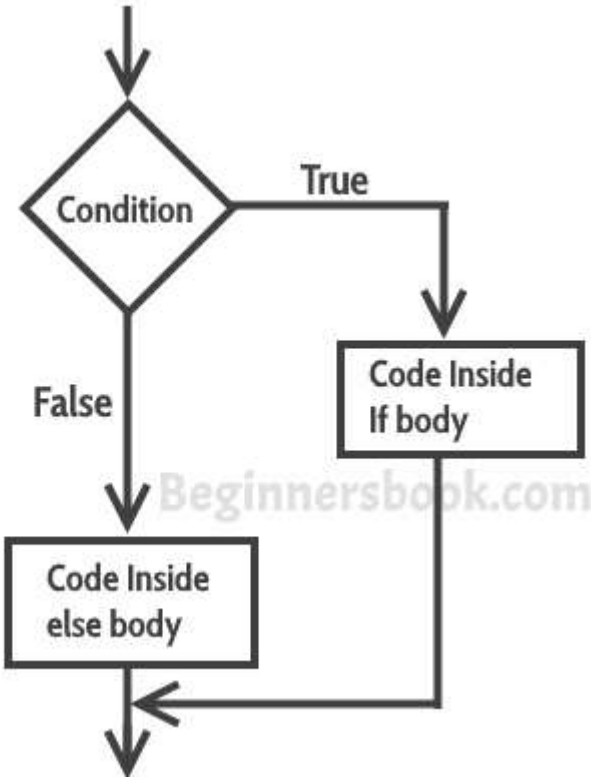
C If else statement

Syntax of if else statement:

If condition returns true then the statements inside the body of "if" are executed and the statements inside body of "else" are skipped.

If condition returns false then the statements inside the body of "if" are skipped and the statements in "else" are executed.

```
if(condition) {
  // Statements inside body of if
}
else {
  //Statements inside body of else
```



Example of if else statement

In this program user is asked to enter the age and based on the input, the if..else statement checks whether the entered age is greater than or equal to 18. If this condition meet then display message “You are eligible for voting”, however if the condition doesn’t meet then display a different message “You are not eligible for voting”.

b) for loop

Ans. What Is a For Loop?

A **for loop** enables a particular set of conditions to be executed repeatedly until a condition is satisfied. Imagine a situation where you would have to print numbers from 1 to 100. What would you do? Will you type in the printf command a hundred times or try to copy/paste it? This simple task would take an eternity. Using a for loop you can perform this action in three statements. This is the most basic example of the for loop. It can also be used in many advanced scenarios depending on the problem statement.

Check out the flowchart of a for loop to get a better idea of how it looks:

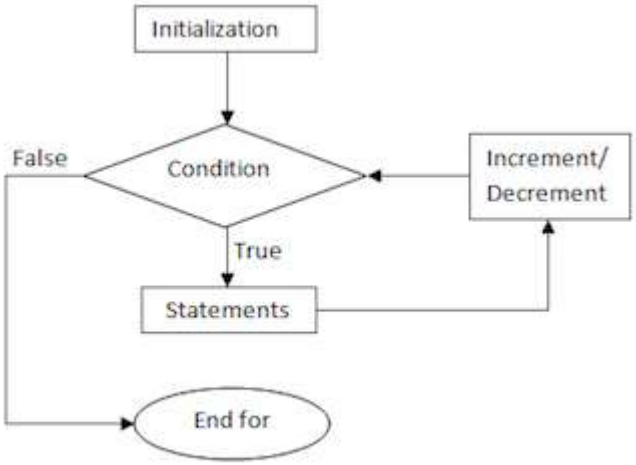


fig: Flowchart for for loop

Syntax of a For Loop

```
1. for (initialization statement; test expression; updat
   e statement) {
2.  // statements
3. }
```

c) while loop

Ans. A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax

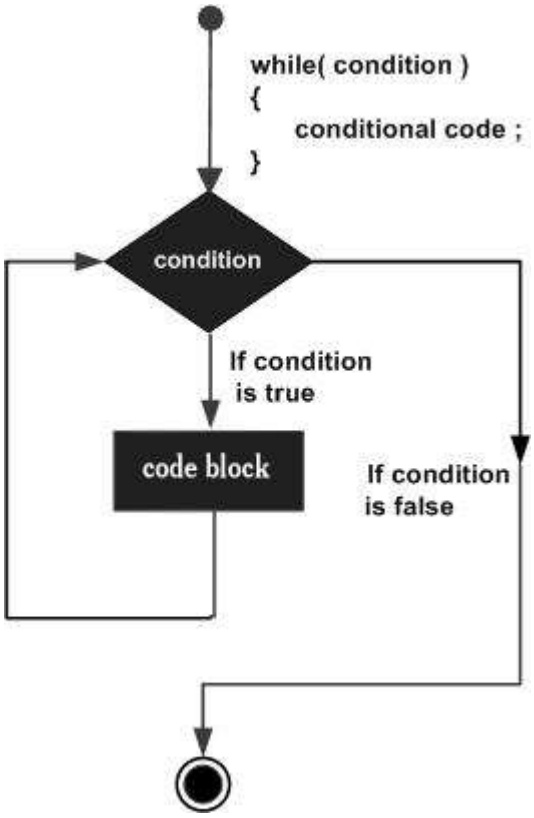
The syntax of a **while** loop in C programming language is −

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

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, the program control passes to the line immediately following the loop.

Flow Diagram



Here, the key point to note is that a while loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```

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

```
value of a: 10
value of a: 11
```

```
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

d) do-while loop

Ans.

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax

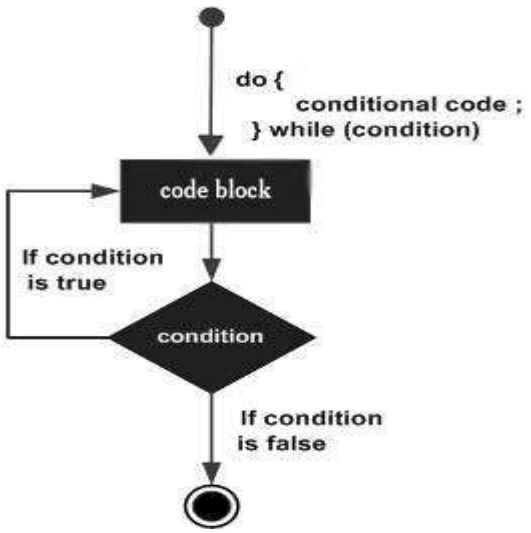
The syntax of a **do...while** loop in C programming language is −

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

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

Flow Diagram



Example

```
#include <stdio.h>

int main () {

  /* local variable definition */
  int a = 10;

  /* do loop execution */
  do {
    printf("value of a: %d\n", a);
    a = a + 1;
  }while( a < 20 );

  return 0;
}
```

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

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```