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.

Compiler

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

1. Ans**Primary data types**:

   These are fundamental data types in C namely integer(`int`), floating point(`float`), character(`char`) and `void`.

2. **Derived data types**:

   Derived data types are nothing but primary datatypes but a little twisted or grouped together like **array**, **stucture**, **union** and **pointers**. These are discussed in details later.

Data type determines the type of data a variable will hold. If a variable `x` is declared as `int`. it means x can hold only integer values. Every variable which is used in the program must be declared as what data-type it is.

Integer type

Integers are used to store whole numbers.

Size and range of Integer type on 16-bit machine:

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Floating point type

Floating types are used to store real numbers.

Size and range of Integer type on 16-bit machine

Type	Size(bytes)	Range

Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

Character type

Character types are used to store characters value.

Size and range of Integer type on 16-bit machine

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

Q3. Find the output of the following expressions

a) X=20/5*2+30-5 (b) Y=30 – (40/10+6) +10 c) Z= 40*2/10-2+10

ans

Q4.Describe the syntax of the following statements

a) If – else statement b) for loop c) while loop d) do-while loop

Ans Types of Loops in C

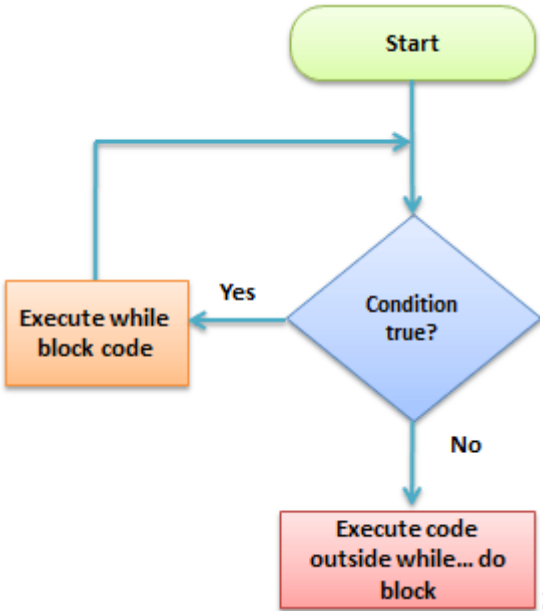
Depending upon the position of a control statement in a program, looping in C is classified into two types:

1. Entry controlled loop

2. Exit controlled loop

In an **entry controlled loop,** a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.



Sample Loop

The control conditions must be well defined and specified otherwise the loop will execute an infinite number of times. The loop that does not stop executing and processes the statements number of times is called as an **infinite loop**. An infinite loop is also called as an "**Endless loop**." Following are some characteristics of an infinite loop:

1. No termination condition is specified.

2. The specified conditions never meet.

The specified condition determines whether to execute the loop body or not.

'C' programming language provides us with three types of loop constructs:

1. The while loop

2. The do-while loop

3. The for loop

While Loop in C

A while loop is the most straightforward looping structure. Syntax of while loop in C programming language is as follows:

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

It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false. Once the condition becomes false, the control goes out of the loop.

After exiting the loop, the control goes to the statements which are immediately after the loop. The body of a loop can contain more than one statement. If it contains only one statement, then the curly braces are not compulsory. It is a good practice though to use the curly braces even we have a single statement in the body.

shortly.

Following program illustrates while loop in C programming example:

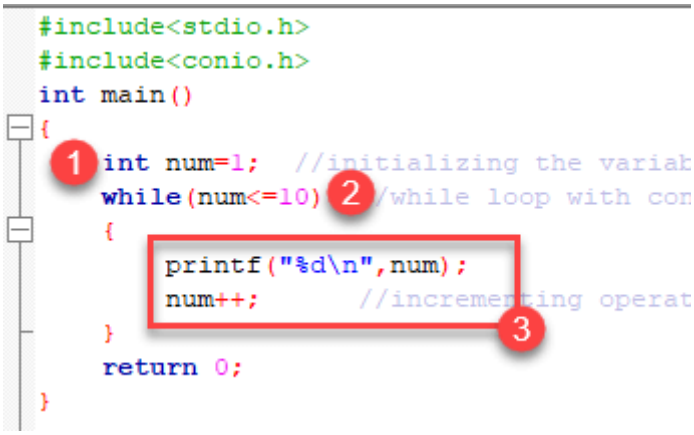
```
#include<stdio.h>
#include<conio.h>
int main()
{
        int num=1;       //initializing the variable
        while(num<=10)   //while loop with condition
        {
                printf("%d\n",num);
                num++;           //incrementing operation
        }
        return 0;
}
```

Output:

```
1
2
3
4
5
```

```
6
7
8
9
10
```

The above program illustrates the use of while loop. In the above program, we have printed series of numbers from 1 to 10 using a while loop.



1. We have initialized a variable called num with value 1. We are going to print from 1 to 10 hence the variable is initialized with value 1. If you want to print from 0, then assign the value 0 during initialization.
2. In a while loop, we have provided a condition (num<=10), which means the loop will execute the body until the value of num becomes 10. After that, the loop will be terminated, and control will fall outside the loop.
3. In the body of a loop, we have a print function to print our number and an increment operation to increment the value per execution of a loop. An initial value of num is 1, after the execution, it will become 2, and during the next execution, it will become 3. This process will continue until the value becomes 10 and then it will print the series on console and terminate the loop.

\n is used for formatting purposes which means the value will be printed on a new line.

Do-While loop in C

A do...while loop in C is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

Syntax of do...while loop in C programming language is as follows:

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

Similar to the while loop, once the control goes out of the loop the statements which are immediately after the loop is executed.

The critical difference between the while and do-while loop is that in while loop the while is written at the beginning. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)

The following loop program in C illustrates the working of a do-while loop:

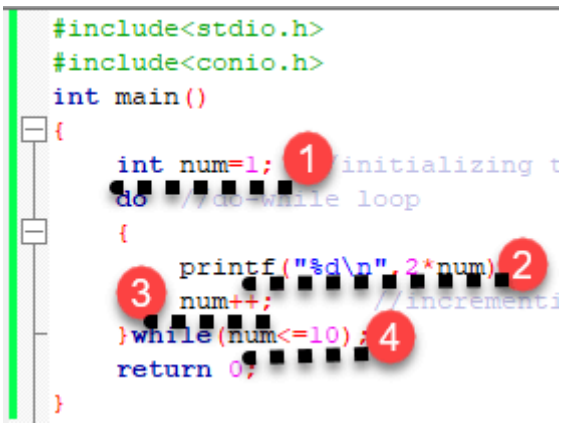
Below is a do-while loop in C example to print a table of number 2:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int num=1;       //initializing the variable
        do       //do-while loop
        {
                printf("%d\n",2*num);
                num++;           //incrementing operation
        }while(num<=10);
        return 0;
}
```

Output:

```
2
4
6
8
10
12
14
16
18
20
```

In the above example, we have printed multiplication table of 2 using a do-while loop. Let's see how the program was able to print the series.



1. First, we have initialized a variable 'num' with value 1. Then we have written a do-while loop.
2. In a loop, we have a print function that will print the series by multiplying the value of num with 2.
3. After each increment, the value of num will increase by 1, and it will be printed on the screen.
4. Initially, the value of num is 1. In a body of a loop, the print function will be executed in this way: 2*num where num=1, then 2*1=2 hence the value two will be printed. This will go on until the value of num becomes 10. After that loop will be terminated and a statement which is immediately after the loop will be executed. In this case return 0.

For loop in C

A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop syntax in C is as follows:

```
for (initial value; condition; incrementation or decrementation )
{
  statements;
}
```

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.

Following program illustrates the for loop in C programming example:

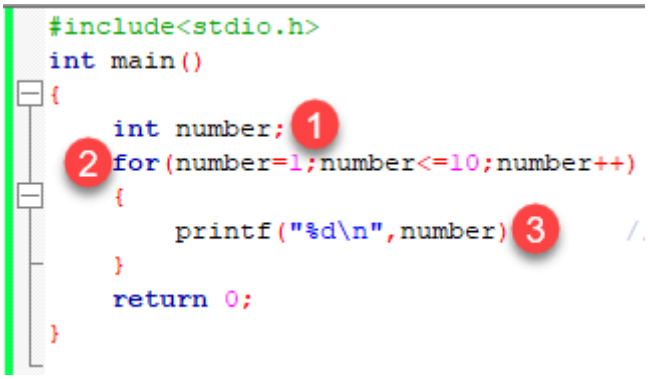
```
#include<stdio.h>
int main()
{
```

```
        int number;
        for(number=1;number<=10;number++)//for loop to print 1-10 numbers
        {
                printf("%d\n",number);            //to print the number
        }
        return 0;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

The above program prints the number series from 1-10 using for loop.



1. We have declared a variable of an int data type to store values.
2. In for loop, in the initialization part, we have assigned value 1 to the variable number. In the condition part, we have specified our condition and then the increment part.

In C, the for loop can have multiple expressions separated by commas in each part.

For example:

```
for (x = 0, y = num; x < y; i++, y--) {
  statements;
}
```

Also, we can skip the initial value expression, condition and/or increment by adding a semicolon.

For example:

```
int i=0;
int max = 10;
for (; i < max; i++) {
  printf("%d\n", i);
}
```

Notice that loops can also be nested where there is an outer loop and an inner loop. For each iteration of the outer loop, the inner loop repeats its entire cycle.

Consider the following example, that uses nested for loop in C programming to output a multiplication table:

```
#include <stdio.h>
int main() {
int i, j;
int table = 2;
int max = 5;
for (i = 1; i <= table; i++) { // outer loop
  for (j = 0; j <= max; j++) { // inner loop
    printf("%d x %d = %d\n", i, j, i*j);
  }
  printf("\n"); /* blank line between tables */
}}
```

Output:

```
1 x 0 = 0
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5

2 x 0 = 0
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
```

The nesting of for loops can be done up-to any level. The nested loops should be adequately indented to make code readable. In some versions of 'C,' the nesting is limited up to 15 loops, but some provide more.

The nested loops are mostly used in array applications which we will see in further tutorials.

Break Statement in C

The break statement is used mainly in in the switch statement. It is also useful for immediately stopping a loop.

We consider the following program which introduces a break to exit a while loop:

```
#include <stdio.h>
int main() {
int num = 5;
while (num > 0) {
  if (num == 3)
    break;
  printf("%d\n", num);
  num--;
}}
```

Q5. Find the output of the following program segments

a) b) c)

#include <stdio.h>

int main()

{

int i;

for (i=1; i<2; i++)

{

printf("IMS Ghaziabad\n");

}

}

#include <stdio.h>

```
int main()
{
int i = 1;
while ( i <= 2 )
{
printf( "IMS Ghaziabad\n");
i = i + 1;
}
}
```

```
#include <stdio.h>
void main()
{
int a = 10, b=100;
if( a > b )
printf( "Largest number is %d\n”, a);
else
printf( "Largest number
```

Q5. Find the output of the following program segments

a) b) c)

```
#include <stdio.h>
int main()
{
int i;
for (i=1; i<2; i++)
{
```

```
printf( "IMS Ghaziabad\n");
}
}
```

```
#include <stdio.h>
int main()
{
int i = 1;
while ( i <= 2 )
{
printf( "IMS Ghaziabad\n");
i = i + 1;
}
}
```

```
#include <stdio.h>
void main()
{
int a = 10, b=100;
if( a > b )
printf( "Largest number is %d\n”, a);
else
printf(
```

Ans