CCA-101: Fundamentals of IT & Programming

Assignment – 2

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

The only language computer hardware can understand is binary code consisting of 1s and 0s. Learn how compilers and interpreters are used to translate a computer program into binary code in this video lesson.

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

A **compiler** is a computer program that translates a program written in a high-level language to the machine language of a computer. The high-level program is referred to as 'the source code.' A typical computer program processes some type of input data to produce output data. The compiler is used to translate source code into machine code or compiled code. This does not yet use any of the input data. When the compiled code is executed, referred to as 'running the program,' the program processes the input data to produce the desired output.

When using a compiler, the entire source code needs to be compiled before the program can be executed. The resulting machine code is typically a compiled file, such as a file with an .exe extension. Once you have a compiled file, you can run the program over and over again without having to compile it again. If you have multiple inputs that require processing, you run the compiled code as many times as needed.

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

Variables in C are associated with data type. Each data type requires an amount of memory and performs specific operations.

There are some common data types in C −

- **int** − Used to store an integer value.
- **char** − Used to store a single character.
- **float** − Used to store decimal numbers with single precision.
- **double** − Used to store decimal numbers with double precision.

The following table displays data types in C language −

Data Types	Bytes	Range
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	1.2E-38 to 3.4E+38
double	8	2.3E-308 to 1.7E+308

Here is the syntax of datatypes in C language,

```
data_type variable_name;
```

Here is an example of datatypes in C language,

Example

```
#include >stdio.h>

int main() {

   // datatypes

   int a = 10;

   char b = 'S';

   float c = 2.88;

   double d = 28.888;

   printf("Integer datatype : %d\n",a);

   printf("Character datatype : %c\n",b);

   printf("Float datatype : %f\n",c);

   printf("Double Float datatype : %lf\n",d);

   return 0;

}
```

Here is the output,

Output

```
Integer datatype : 10
Character datatype : S
Float datatype : 2.880000
Double Float datatype : 28.888000
```

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

a) X=20/5*2+30-5

b) The order of algebraic operations we can follow is as below

c) B =Brackets should be operated first(inside the brackets we may also have more than one operations ,so repeat them according to BODMAS rule)

d) O=of ,means bracket wise multiplication or division like 2(3)=6 and 4(1/4)=4

e) D =division symbol i.e. ÷ ,a totally indicative symbol of division,let us do an example like,16÷2 =8

f) M=multiplication symbol such as x or × or * that means a product of two or more numbers or terms.Example wise,2*3=6 or 2×3=6,or 2x3=6

g) A =addition symbol like + ,simply adding two or more numbers or terms like as 2+ 3=5.

h) S= subtraction symbol -,we can take an example as 4–4=0,which could also be spelt as four minus four is zero.

i) This is the order what we should follow for a mathematical calculation.

j) Your question is 10+5×2, here we go with the above rules.

k) 10+5×2=10+10=20(here we have two operations multiplication and addition,we do this according to priorities assigned by BODMAS rule i.e.first multiply and the add.)

Q4. Describe the syntax of the following statements

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

In the previous tutorial we learned while loop in C. A do while loop is similar to while loop with one exception that it executes the statements inside the body of do-while before checking the condition. On the other hand in the while loop, first the condition is checked and then the statements in while loop are executed. So you can say that if a condition is false at the first place then the do while would run once, however the while loop would not run at all.

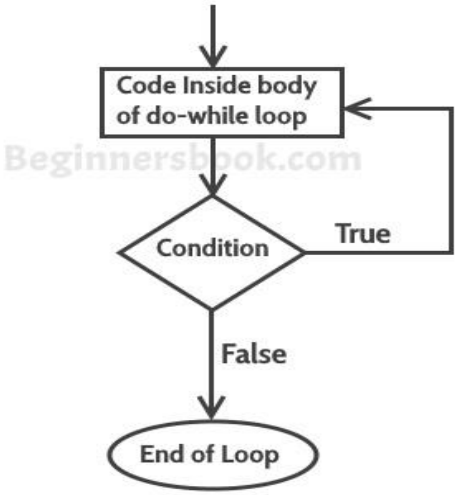
C - do..while loop

Syntax of do-while loop

```
do
{
    //Statements
```

```
}while(condition test);
```

Flow diagram of do while loop



Example of do while loop

```
#include <stdio.h>
int main()
{
        int j=0;
        do
        {
                printf("Value of variable j is: %d\n", j);
                j++;
        }while (j<=3);
        return 0;
}
```

Output:

```
Value of variable j is: 0
Value of variable j is: 1
Value of variable j is: 2
Value of variable j is: 3
```

While vs do..while loop in C

Using while loop:

```
#include <stdio.h>
int main()
{
    int i=0;
    while(i==1)
```

```
    {
        printf("while vs do-while");
    }
    printf("Out of loop");
}
```

Output:

```
Out of loop
```

Same example using do-while loop

```
#include <stdio.h>
int main()
{
   int i=0;
   do
   {
        printf("while vs do-while\n");
   }while(i==1);
   printf("Out of loop");
}
```

Output:

```
while vs do-while
Out of loop
```

Explanation: As I mentioned in the beginning of this guide that do-while runs at least once even if the condition is false because the condition is evaluated, after the execution of the body of loop.

Q5. Find the output of the following program segments

a) #include <stdio.h>

int main()

{

int i;

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

{

printf("IMS Ghaziabad\n");

}

}

b) #include <stdio.h>

int main()

{

int i = 1;

while (i <= 2)

{

printf("IMS Ghaziabad\n");

i = i + 1;

}

}

c)| #include <stdio.h>

void main()

{

int a = 10, b=100;

if(a > b)

printf("Largest number is %d\n", a);

else

printf("Largest number is %d\n", b);

}