CCA-101: Fundamentals of IT & Programming

Assignment - 2

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

Programs written in the machine language of a given type of computer can be directly executed by the CPU of that type of computer. High-level language programs must be translated into machine language before they can be executed. (Machine language instructions are encoded as binary numbers that are meant to be used by a machine, not read or written by people. High-level languages use a syntax that is closer to human language.)

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

Ans. Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows −

Sr.No.	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Enumerated types** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void**

	The type specifier *void* indicates that no value is available.
4	**Derived types** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

The array types and structure types are referred collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, where as other types will be covered in the upcoming chapters.

Integer Types

The following table provides the details of standard integer types with their storage sizes and value ranges −

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision −

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The void Type

The void type specifies that no value is available. It is used in three kinds of situations −

Sr.No.	Types & Description
1	**Function returns as void** There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, **void exit (int status);**
2	**Function arguments as void** There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, **int rand(void);**
3	**Pointers to void** A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

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. a**) If – else statement**

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:

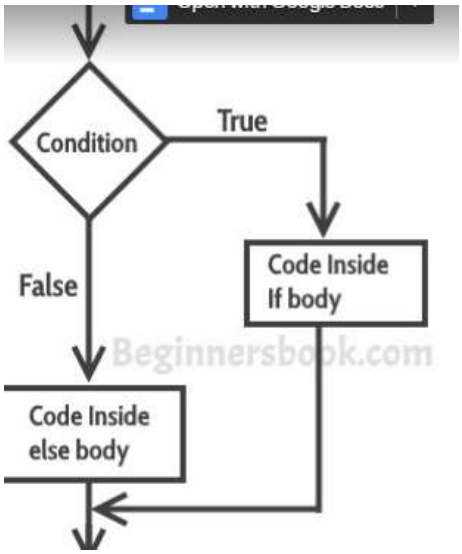
If condition returns true then the statements inside the body of “if” are executed and the statements inside body of “else” are skipped.

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:

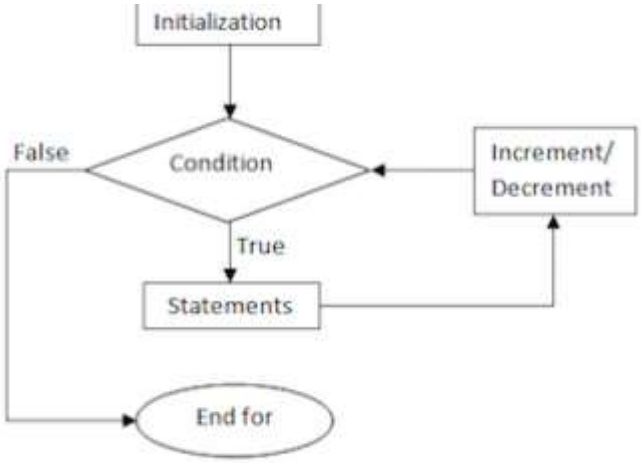


fig: Flowchart for for loop

Syntax of a For Loop

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

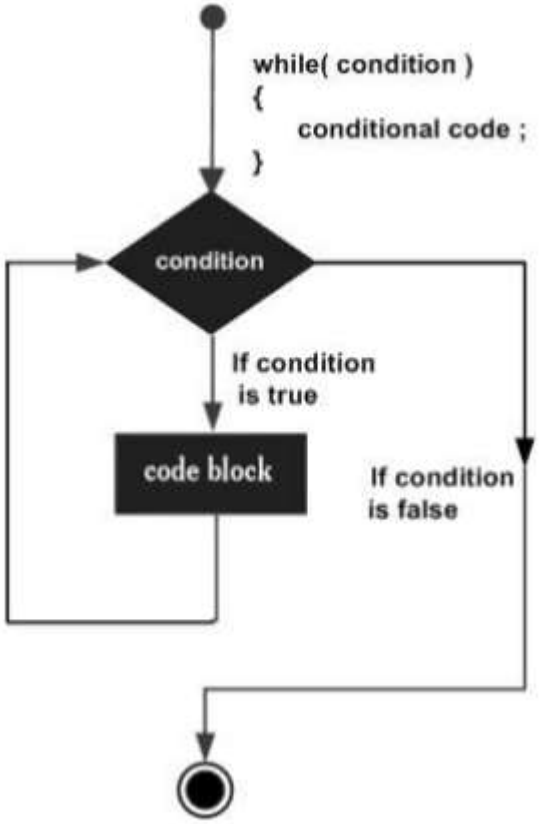
c) while loop

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.



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.
Example

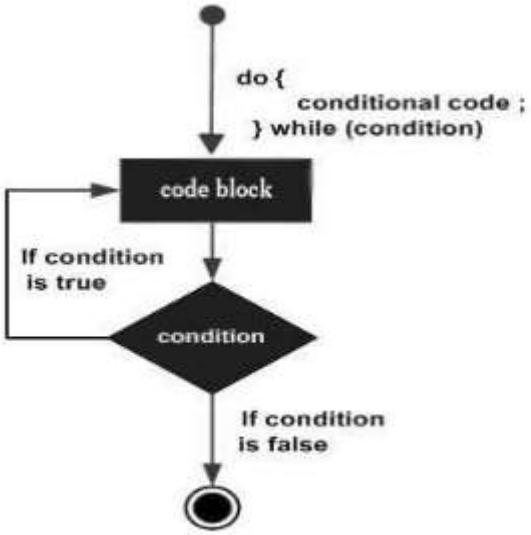
```
#include &lt;stdio.h&gt;
int main () {
/* local variable definition */
```

```
int a = 10;
/* do loop execution */
do {
printf(&quot;value of a: %d\n&quot;, a);
a = a + 1;
}while( a &lt; 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
```

Flow Diagram



value of a: 19