

## Answer Script

### Question No. 01

- a) Define ternary operator. Write a C program to find the maximum of two numbers using ternary operators. (2+5)
- b) Write down the differences between local and global variables. (3)

### Answer No. 01

#### Ans. to the Q. No. (1-a)

##### **# Definition of Ternary Operator:**

Ternary Operator is an operator which takes three operands or variables, unlike the other operators which take one or two operands. The ternary operator in C is also known as the Conditional Operator. It is a way to shorten the simple if-else code of the block.

##### **# C Program to find the maximum of two numbers using ternary operators:**

```
#include <stdio.h>
```

```
int main() {
```

```
    int num1, num2;
```

```
    printf("Enter Two Numbers to find the Maximum of them: \n");
```

```
    scanf("%d%d", &num1, &num2);
```

```
    int equal;
```

```
    equal = (num1 == num2) ? 1 : 0; // Used Ternary Operator to check  
    the numbers are equal or not.
```

```
    // if the number is not equal,
```

```
    int maximum;
```

```
    maximum = (num1 > num2) ? num1 : num2; // Used Ternary Operator to  
    find the maximum number.
```

```
    // printing the result
```

```
    (equal) ? printf("The Numbers are Equal\n") : printf("The Maximum
```

```
Number is %d\n", maximum);
```

```
    return 0;  
}
```

**Ans. to the Q. No. (1-b)**

**# Differences between Global and Local variables:**

<b>Global Variables</b>	<b>Local Variables</b>
Global variables are declared outside all the function blocks.	Local Variables are declared within a function block.
Any change in global variable affects the whole program, wherever it is being used.	Any change in the local variable does not affect other functions of the program.
It can be accessed throughout the program by all the functions present in the program.	It can only be accessed by the function statements in which it is declared and not by the other functions.
Global variables are stored in the data segment of memory.	Local variables are stored in a stack in memory.
The scope remains throughout the program.	The scope is limited and remains within the function only in which they are declared.
A global variable exists in the program for the entire time the program is executed.	A local variable is created when the function is executed, and once the execution is finished, the variable is destroyed.
We cannot declare many variables with the same name.	We can declare various variables with the same name but in other functions.
If the global variable is not initialized, it takes zero by default.	If the local variable is not initialized, it takes the garbage value by default.

### Question No. 02

You want to build a Fibonacci series up to n terms. Now solve these problems without using an array and first write the steps to solve these problems and then write a C program for it.

**Note** - The Fibonacci sequence is a sequence where the next term is the sum of the previous two terms. The first two terms of the Fibonacci sequence are 0 followed by 1.

**Input Format:**

You are given a positive integer n.

**Output Format:**

Print the Fibonacci series up to n terms.

See the sample input-output for more clarification

Sample input	Sample output
10	0, 1, 1, 2, 3, 5, 8, 13, 21, 34

### Answer No. 02

**# Steps to solve the problem:**

In the case of Fibonacci series, next number is the sum of previous two numbers for example 0, 1, 1, 2, 3, 5, 8, 13, 21, etc. The first two numbers of the Fibonacci series are 0 and 1.

Let us suppose n = 10. First, we printed the first two terms of the Fibonacci sequence before using a for loop to print the next n terms.

Let us see how the for loop works:

Count	Previous Number	Present Number	Fibonacci Number
3	0	1	1
4	1	1	2
5	1	2	3
6	2	3	5
7	3	5	8
8	5	8	13
9	8	13	21
10	13	21	34

### **# C Program to Build a Fibonacci Series up to n terms:**

```
#include <stdio.h>
```

```
int main() {
```

```
    int n;
```

```
    scanf("%d", &n);
```

```
    int prev_num = 0, prsnt_num = 1;
```

```
    int fib_num = 0, count = 2;
```

```
    printf("0, 1, "); // printed the first and second fibonacci number
```

```
    while(1) {
```

```
        if (count == (n-1)) {
```

```
            break;
```

```
        }
```

```
        fib_num = prev_num + prsnt_num;
```

```
        printf("%d, ", fib_num);
```

```

        count++;

        int temp = prsnt_num;
        prsnt_num = fib_num;
        prev_num = temp;
    }

    printf("%d", prev_num + prsnt_num); // printed the last fibonacci
number

    return 0;
}

```

### Question No. 03

Define pointers of C. What are the differences between pass-by-value and pass-by-references, Explain it with C program using function.

### Answer No. 03

#### **Definition of Pointer of C:**

Pointers in C are used to store the address of variables or a memory location. This variable can be of any data type i.e, int, char, function, array, or any other pointer. The size of the pointer depends on the architecture.

#### Syntax:

```
datatype *var_name;
```

Let pointer "ptr" holds the address of an integer variable or hold the address of memory whose value(s) can be accessed as integer values through "ptr". So to define "ptr" we have to write the below syntax:

```
int *ptr;
```

### **# Differences between Pass-By-Value and Pass-By-Reference:**

<b>Pass-By-Value</b>	<b>Pass-By-Reference</b>
While calling a function, we pass values of variables to it. Such functions are known as "Pass By Values".	While calling a function, instead of passing the values of variables, we pass the address of variables(location of variables) to the function known as "Call By References.
In this method, the value of each variable in the calling function is copied into corresponding dummy variables of the called function.	In this method, the address of actual variables in the calling function is copied into the dummy variables of the called function.
With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function.	With this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them.
Thus actual values of a and b remain unchanged even after exchanging the values of x and y.	Thus actual values of a and b get changed after exchanging values of x and y.
Values of variables are passed by the Simple technique.	Pointer variables are necessary to define to store the address values of variables.

### **# Explanation of Pass-By-Value with Function:**

In this parameter passing method, values of actual parameters are copied to the function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in the actual parameters of the caller.

**Example:**

```
// pass by value

#include <stdio.h>

void swap(int x, int y); // function prototype

int main()
{
    int a = 10, b = 20;

    swap(a, b);

    printf("a=%d b=%d\n", a, b);

    return 0;
}

void swap(int x, int y)
{
    int t;

    t = x;
    x = y;
    y = t;

    printf("x=%d y=%d\n", x, y);
}
```

**# Explanation of Pass-By-Reference with Function:**

Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in the actual parameters of the caller.

**Example:**

```
#include <stdio.h>
```

```

void swap(int*, int*); // function prototype

int main()
{
    int a = 7, b = 9;

    // pass reference
    swap(&a, &b);

    printf("a=%d b=%d\n", a, b);

    return 0;
}

// function to swap two variables
void swap(int* x, int* y)
{
    int t;

    t = *x;
    *x = *y;
    *y = t;

    printf("x=%d y=%d\n", *x, *y);
}

```

#### Question No. 04

You are given a string *S* of small letters. Now sort the string in ascending order using the frequency array. Write the steps to solve this problem with proper diagram. Lastly, write a C program for it and use function in this program. **(10)**

**Note** - The given input may contain duplicate characters



Sample input	Sample output
adsarbrro	aabdorrrs

#### Answer No. 04

#### # Steps to sort the string in ascending order by frequency array:

- > The frequency of a character is the number of times it appears in the string.
- > In the output, characters with higher frequency come first.
- > If two characters have the same frequency, whichever occurs earliest in S, must come first. In other words, the sorting must be stable.
- > Assume string consists of uppercase and lowercase English letters. We need to consider the uppercase and lowercase of the same character as a different one.

#### # C program of this code:

```
#include <stdio.h>
```

```
void sort(char s[]); // prototype of the function
```

```
// driver code
```

```
int main () {
    char s[100];
    scanf("%s", &s); // input statements of string
    sort(s);
}
```

```
// function to sort the string by frequency array
```

```
void sort(char s[]) {

    int n = 0;
    for (int i = 0; s[i] != '\0'; i++) { // finding the length of the
string
        n++;
    }
}
```

```

char ans[n]; // declared the answer string
int freq[27]; // declared the frequency array

for (int i = 1; i <= 26; i++) {
    freq[i] = 0;
}

for (int i = 0; s[i] != '\0'; i++) { // count the frequency of
every letter
    int temp = s[i] - 96;
    freq[temp]++;
}

int k = 0;

for (int i = 1; i <= 26; i++) {
    if (freq[i] != 0) {
        char ch = 96 + i;
        for (int j = 1; j <= freq[i]; j++) {
            ans[k] = ch; // putting the character on the answer
string
            k++;
        }
    }
}

printf("%s", ans); // printing the answer
}

```

### Question No. 05

What is the difference between malloc and calloc? Write a c program to take **N** elements in an array. You must use malloc to declare this array and show the output of the array. (4+6)

### Answer No. 05

The name "malloc" stands for memory allocation.

The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer of void which can be cast into pointers of any form.

The name "calloc" stands for contiguous allocation.

The malloc() function allocates memory and leaves the memory uninitialized, whereas the calloc() function allocates memory and initializes all bits to zero.

#### # Differences between malloc() and calloc():

malloc()	calloc()
malloc() function creates a single block of memory of a specific size.	calloc() function assigns multiple blocks of memory to a single variable.
malloc() is faster.	calloc() is slower.
malloc() indicates memory allocation.	calloc() indicates contiguous allocation.
The number of arguments in malloc() is 1.	The number of arguments in calloc() is 2.
malloc() has high time efficiency.	calloc() has low time efficiency.
The memory block allocated by malloc() has a garbage value.	The memory block allocated by calloc() is initialized by zero.

#### # C program to take N elements in an array by using malloc():

```
#include <stdio.h>
```

```
int main () {  
    int *p, N;  
    printf("Enter the value of N: ");  
    scanf("%d", &N);  
  
    p = (int*)malloc(N * sizeof(int)); // Used malloc to declare this
```

array

```
printf("\nEnter %d elements: \n", N);
for (int i = 1; i <= N; i++) {
    scanf("%d", &p[i]);
}

printf("\nThe entered elements are: \n");
for (int i = 1; i <= N; i++) {
    printf("%d ", p[i]);
}

return 0;
}
```

#### Question No. 06

What is a function? Types of user-defined functions explain with proper examples. What are the benefits of using user-defined functions? (2+5+3)

#### Answer No. 06

##### **# Function:**

Functions are "self-contained" modules of code that accomplish a specific task. Functions usually "take in" data, process it, and "return" a result. Once a function is written, it can be used over and over and over again. Functions can be "called" from the inside of other functions.

##### **# Types of User-Defined functions with proper examples:**

A user-defined function is one that is defined by the user when writing any program, as we do not have library functions that have predefined definitions. To meet the specific requirements of the user, the user has to develop his or her own functions. Such functions must be defined properly by the user. There is no such kind of requirement to add any particular library to the program.

There are 4 types of user-defined functions in C:

1. Function with no arguments and no return value
2. Function with no arguments and a return value
3. Function with arguments and no return value
4. Function with arguments and with return value

**Explanations:**

**1) Function with no arguments and no return value:**

Functions that have no arguments and no return values. Such functions can either be used to display information or to perform any task on global variables.

**Example:**

```
#include <stdio.h>

void sum()
{
int x, y;
printf("Enter x and y\n");
scanf("%d %d", &x, &y);
printf("Sum of %d and %d is: %d", x, y, x + y);
}

// Driver code
int main()
{
// function call
sum();

return 0;
}
```

**2) Function with no arguments and a return value:**

Functions that have no arguments but have some return values. Such functions are used to perform specific operations and return their value.

**Example:**

```
#include <stdio.h>
```

```

int sum()
{
int x, y, s = 0;
printf("Enter x and y\n");

scanf("%d %d", &x, &y);
s = x + y;
return s;
}

// Driver code
int main()
{
// function call
printf("Sum of x and y is %d",
      sum());
return 0;
}

```

### **3) Function with arguments and no return value:**

Functions that have arguments but no return values. Such functions are used to display or perform some operations on given arguments.

#### Example:

```

#include <stdio.h>

void sum(int x, int y)
{
printf("Sum of %d and %d is: %d",
      x, y, x + y);
}

// Driver code
int main()
{
int x, y;
printf("Enter x and y\n");

```

```
scanf("%d %d", &x, &y);
```

```
// function call
```

```
sum(x, y);
```

```
return 0;
```

```
}
```

#### **4) Function with arguments and with return value:**

Functions that have arguments and some return value. These functions are used to perform specific operations on the given arguments and return their values to the user.

##### Example:

```
#include <stdio.h>
```

```
int sum(int x, int y)
```

```
{
```

```
return x + y;
```

```
}
```

```
// Driver code
```

```
int main()
```

```
{
```

```
int x, y;
```

```
printf("Enter x and y\n");
```

```
scanf("%d %d", &x, &y);
```

```
// function call
```

```
printf("Sum of %d and %d is: %d",  
      x, y, sum(x, y));
```

```
return 0;
```

```
}
```

#### **# Benefits of using User-Defined functions:**

i) Reduction in Program Size:

Since any sequence of statements which are repeatedly used in a program can be combined together to form a user-defined function. And these functions can be called as many times as required. This avoids writing of same code again and again reducing program size.

ii) Reducing Complexity of Program:

Complex programs can be decomposed into small sub-programs or user-defined functions.

iii) Easy to Debug and Maintain:

During debugging it is very easy to locate and isolate faulty functions. It is also easy to maintain a program that uses user-defined functions.

iv) Readability of Program:

While using the user-defined function, a complex problem is divided into different sub-programs with clear objectives and interfaces which makes it easy to understand the logic behind the program.

v) Code Reusability:

Once the user-defined function is implemented it can be called or used as many times as required which reduces code repeatability and increases code reusability.

Question No. 07

You have been given **an NxM** matrix, Now your task is to take two matrix as input and find the sum of this two matrix. (10)

Sample Input	Sample Output
3 3	2 3 4



1 2 3  
4 5 6  
7 8 9

1 1 1  
3 2 4  
2 1 2

7 7 10  
9 9 11

### Answer No. 07

#### **C Program to solve the above problem:**

```
#include <stdio.h>
```

```
int main () {
```

```
    int N, M;    scanf("%d%d", &N, &M); // taking the size of row and  
column of the matrixes
```

```
    int matrix1[200][200], matrix2[200][200], sum[200][200]; //  
declared two matrix and one for finding sum
```

```
    // taking the elements of the 1st matrix
```

```
    for (int i = 0; i < N; i++) {  
        for (int j = 0; j < N; j++) {  
            scanf("%d", &matrix1[i][j]);  
        }  
    }
```

```
    // taking the elements of the 2nd matrix
```

```
    for (int i = 0; i < N; i++) {  
        for (int j = 0; j < N; j++) {  
            scanf("%d", &matrix2[i][j]);  
        }  
    }
```

```
    // adding the elements of 1st & 2nd matrix
```

```
    for (int i = 0; i < N; i++) {
```

```

        for (int j = 0; j < N; j++) {
            sum[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }

    // printing the summation matrix
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", sum[i][j]);
        }
        printf("\n");
    }

    return 0;
}

```

#### Question No. 08

- a) What is structure? Explain Why we use it? (2)
- b) How many ways to access structure members? Explain with example. (3)
- c) Write a C program to store the information of a student using structure. (5)

Sample Input	Sample Output
Enter information: Enter name: Vanya Enter roll number: 25 Enter marks: 80	Displaying Information: Name: Vanya Roll number: 25 Marks: 80

#### Answer No. 08

### **Ans. to the Q. No. (8-a)**

#### **# Structure:**

A struct (or structure) is a collection of variables (can be of different types) under a single name.

#### **Syntax of Structure:**

```
struct structureName {  
    dataType member1;  
    dataType member2;  
    ...  
};
```

#### **Example:**

```
struct Person {  
    char name[50];  
    int citNo;  
    float salary;  
};
```

#### **# The Reason why we use structure:**

It is very easy to maintain as we can represent the whole record by using a single name. In structure, we can pass a complete set of records to any function using a single parameter. You can use an array of structures to store more records with similar types.

### **Ans. to the Q. No. (8-b)**

#### **# The ways to access the structure members:**

There is only one way to access structure members.

Array elements are accessed using the Subscript variable, Similarly Structure members are accessed using dot [.] operator. (.) is called as "Structure member Operator". By Using this Operator in between "Structure name" & "member name", we can access a structure member.

#### **Example:**

```
#include<stdio.h>  
#include<conio.h>
```

```

struct Student
{
    int class;
    int age;
} v1 = {4,10};
void main ()
{
printf (" %d\n",v1.class); // accessing the structure member class.
printf(" %d",v1.age); // accessing the structure member age.
}

```

**Ans. to the Q. No. (8-c)**

**# C program to store the information of a student using structure:**

```

#include <stdio.h>

// declaring structure to store the information of a student
struct student {
    char name[45];
    int roll;
    double marks;
};

int main () {

    struct student s1; // declared a structure variable for one
    student

    // storing the information of the student
    printf("Enter information:\n");
    printf("Enter name: ");
    scanf("%s", &s1.name); // storing name
    printf("Enter roll number: ");
    scanf("%d", &s1.roll); // storing roll
    printf("Enter marks: ");
    scanf("%lf", &s1.marks); // storing marks

    // displaying the information of the student

```

```
printf("\nDisplaying Information:\n");  
printf("Name: %s\n", s1.name); // displaying name  
printf("Roll: %d\n", s1.roll); // displaying roll  
printf("Marks: %.2lf\n", s1.marks); // displaying marks  
  
return 0;  
}
```

### Question No. 09

a) Explain 5 types of Errors in C with Example (You have to write program) (10)

### Answer No. 09

In any programming language errors are common. If we miss any syntax like parenthesis or semicolon then we get syntax errors. Apart from this we also get run time errors during the execution of code.

There are 5 types of error in C:

1. Syntax Errors
2. Runtime Errors
3. Logical Errors
4. Linked Errors
5. Semantic Errors

#### **Explanation of each type with program:**

##### **1) Syntax Errors:**

These are also referred to as compile-time errors. These errors have occurred when the rule of C writing techniques or syntaxes has been broken. These types of errors are typically flagged by the compiler prior to compilation.

Example:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      // missing semicolon
6      printf("Phitron")
7      return 0;
8  }
```

Build results window:

Line	Message
	=== Build file: "no target" in "no project"
B ...	In function 'main':
B ... 6	error: expected ';' before 'return'
	=== Build failed: 1 error(s), 0 warning(s)

## 2) Runtime Errors:

This type of error occurs while the program is running. Because this is not a compilation error, the compilation will be completed successfully. These errors occur due to segmentation faults when a number is divided by a division operator or modulo division operator.

Example:

```

#include <stdio.h>

int main()
{
    int array[5];
    printf("%d", array[10]);
    return 0;
}

/*
an array of length 5 i.e. array[5],
but during runtime, if we try to access
10 elements i.e array[10] then we get
segmentation fault errors called runtime errors.
Giving only an array length of 5 |
*/

```

### 3) Logical Errors:

Even if the syntax and other factors are correct, we may not get the desired results due to logical issues. These are referred to as logical errors. We sometimes put a semicolon after a loop, which is syntactically correct but results in one blank loop. In that case, it will display the desired output.

#### Example:

```

#include <stdio.h>

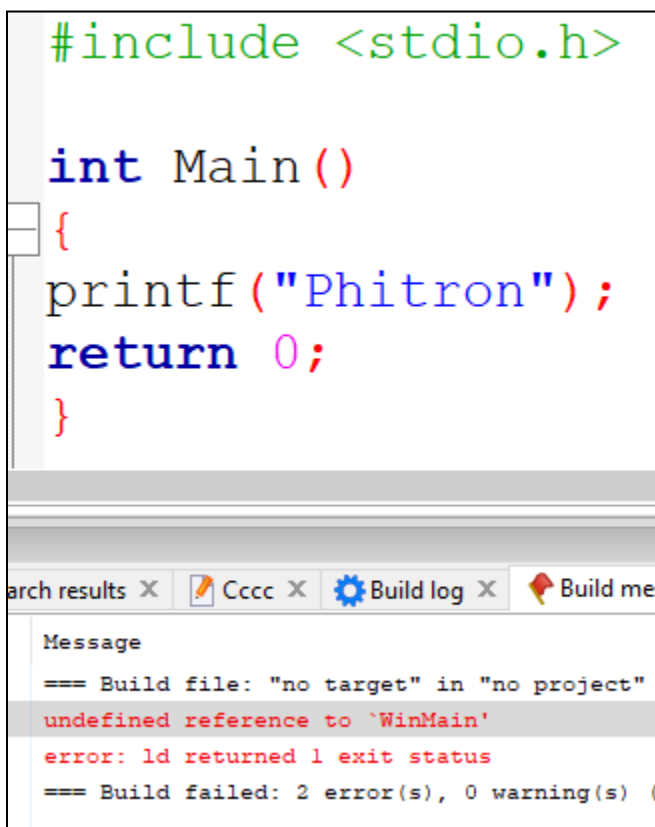
int main()
{
    int i;
    for(i = 0; i <= 5; i++);
    /* the for loop iterates 5 times
    but the output will be displayed
    only one time due to the semicolon
    at the end of for loop. */
    {
        printf("Phitron");
    }
    return 0;
}

```

#### 4) Linked Errors:

When the program is successfully compiled and attempts to link the different object files with the main object file, errors will occur. When this error occurs, the executable is not generated. This could be due to incorrect function prototyping, an incorrect header file, or other factors. If `main()` is written as `Main()`, a linked error will be generated.

Example:



```
#include <stdio.h>

int Main()
{
    printf("Phitron");
    return 0;
}
```

arch results X Cccc X Build log X Build me

Message

=== Build file: "no target" in "no project"

undefined reference to `WinMain'

error: ld returned 1 exit status

=== Build failed: 2 error(s), 0 warning(s) (

#### 5) Semantic Errors:

When a sentence is syntactically correct but has no meaning, semantic errors occur. This is similar to grammatical errors. If an expression is entered on the left side of the assignment operator, a semantic error may occur.

Example:



```
1  #include <stdio.h>
2
3  int main()
4  {
5      int x = 10;
6      b = 20, c;
7      x + y = c;
8      printf("%d", c);
9      return 0;
10 }
```

Build log X Build messages X

Line	Message
	=== Build file: "no target" in "no project" (compiler: u
B ...	In function 'main':
B ... 6	error: 'b' undeclared (first use in this function)
B ... 6	note: each undeclared identifier is reported only once
B ... 6	error: 'c' undeclared (first use in this function)
B ... 7	error: 'y' undeclared (first use in this function)
	=== Build failed: 3 error(s), 0 warning(s) (0 minute(s),

### Question No. 10

- Why is C called the Mother of all Languages?
- What are source code & .obj file and .exe file?
- When you run a C program How many files are created on your device?  
What are they?

- d) What is the base condition in the recursion function? Describe a real-life example of recursion.
- e) Why should we declare large data types first in structure? ( 5x2 = 10)

Answer No. 10

**Ans. to the Q. No. (10-a):**

**C is called the mother of all languages because of the following reasons:**

- › Most of the compilers and JVMs are written in C language.
- › Most of the languages which are developed after C language has borrowed heavily from it like C++, Python, Rust, Javascript, etc.
- › C language had provided many new concepts for the programming language like variables, data types, loops, array, function, file handling, dynamic memory allocation, pointers and most of these concepts are used in many modern languages like C++, Java, C#.

**Ans. to the Q. No. (10-b):**

**# Source Code:**

Source code is a group of instructions written by a programmer using programming languages in any text editor like CodeBlocks, Visual Studio, etc, and then saved in a file.

**# Object File:**

An object file is a computer file generated by a program called a compiler and contains data and instructions.

In common practice, an object file is denoted by a ".obj" file extension. There are several different formats for these files, however.

**# Exe. File:**

The .exe file extension is short for "executable." An executable file (EXE file) is a computer file that contains an encoded sequence of instructions that the system can execute directly when the user clicks the file icon.

**Ans. to the Q. No. (10-c):**

It depends. Depending on the compiler type used and flags passed, lots of

conditions might happen. Consider these for example:

- > Various temporary files are frequently created by the C compiler (and subsequently removed).
- > One or more object files are produced by the C compiler.
- > Temporary files may be created (and deleted) by the linker in addition to producing an executable.
- > Some C compilers (like GCC, for example) also serve as front-end drivers for the linker and the compiler. Any unnecessary temporary files, such as the object files produced during the compilation stage, may be deleted in this scenario by the compiler.

So depending on the system and compiler being used, there will be variations.

#### **Ans. to the Q. No. (d):**

##### **# Base Condition:**

The condition that is applied to stop a recursive function from executing endlessly – once a pre-specified base condition is met, the function knows it's time to exit. A base condition is a must in every recursive function otherwise it will continue to execute like an infinite loop.

##### **# Real life example of recursion:**

Finding factorial of a number is a real life example of recursion.

##### **Algorithm to find the factorial of a number:**

```
int factorial(int n) //factorial with recursion
{
    if(n == 0) // here is the base condition
    {
        return 1;
    }
    return n * factorial(n-1);
}
```

#### **Ans to the Q. No. (e):**

##### **# The reason why we should declare large data type first in structure:**

In structure, if we declare a small variable first, then we need more memory in our device. But if we want to minimize the storage then we have to declare the large variable first.

So, we should declare large data type first in structure.

**THE END**