Answer 1 [A] :
A ternary operator is a conditional operator that takes three operands. It is a compact version of an if-else statement.

The syntax for a ternary operator is:

```
variable = (condition) ? value_if_true : value_if_false;
```

If the condition is true, the operator returns the value of value_if_true. If the condition is false, it returns the value of value_if_false.

Here is a C program that uses a ternary operator to find the maximum of two numbers:

```
#include <stdio.h>
int main()
{
  int x = 5, y = 10;
  int max = (x > y) ? x : y;
  printf("Max Value is : %d\n", max);
  return 0;
}
```

Answer 1 [B] :
In C, there are two types of variables: local variables and global variables. Here are the main differences between the two:

1. Local variables are only accessible within the block/scope of code in which they are defined. On the other hand, global variables are accessible from any part of the program. Local variable is declared inside a function, whereas the global variable is declared outside the function.
2. Local variables are starts in a certain scope/function and destroyed when the function or block of code returns. But global variables are stored/created when the program starts and destroyed when the program ends.

3. The local variable doesn’t provide data sharing, whereas the Global variable provides data sharing.

Answer 2 :

To create a Fibonacci series without using an array the steps could be :

1. Initialize the first two terms of the series, a and b, as 0 and 1 respectively.
2. Print the first term of the series, which is 0.
3. Use a loop, which iterates from 0 to n-1, to calculate the next terms of the series.
4. In each iteration of the loop, calculate the next term of the series, c, as the sum of the previous two terms, a and b.
5. Print the next term, c.
6. Update the value of a and b, so that a becomes b and b becomes c.
7. Repeat steps 4 through 6, until the loop completes n-1 iterations.

Here's a a C program for the above steps:

```
#include <stdio.h>
int main() {
   int n, a = 0, b = 1, c, i;
   scanf("%d", &n);
   for (i = 0; i < n; i++) {
      if (i <= 1) {
         c = i;
      } else {
         c = a + b;
         a = b;
         b = c;
      }
      if(i<n-1)
         printf("%d, ", c);
      else
```

```
        printf("%d", c);
    }
    return 0;
}
```

Answer 3 :

A pointer is a variable that stores the memory address of another variable. Pointers are declared using the * operator, and they can be used to store the address of any type of variable, including arrays, structures, and functions.

when a function is called, the arguments are passed to the function either by value or by reference. Pass by value means that the function receives a copy of the argument, and any changes made to the argument within the function have no effect on the original variable.
Pass by reference means that the function receives the memory address of the argument, and any changes made to the argument within the function are reflected in the original variable.

An example of a C function that takes an integer argument and increments it by 1, using both pass by value and pass by reference:

```
#include <stdio.h>
void increment_by_value(int x) {
    x++;
}
void increment_by_reference(int *x) {
    (*x)++;
}

int main() {
    int a = 10;
    printf("a = %d\n", a);
    increment_by_value(a);  // Call increment_by_value with a as an argument
    printf("a = %d\n", a);
```

```
    increment_by_reference(&a);  // Call increment_by_reference with the address of a as an argument
    printf("a = %d\n", a);
    return 0;
}
```

Answer 4 :
Steps to solve this problem can be:

1. Create a frequency array of size 26.
2. Iterate through the input string and count the frequency of each character.
3. Sort the characters by their frequency.
4. Iterate through the frequency array, for each non-zero count, add the corresponding character to the input string.
5. Return the sorted string

The diagram for the input string "adsarbrro" would be:
input: "adsarbrro"
freq[26]: [1,1,1,1,0,0,0,2,0,0,0,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0]
output: "aabbdorrrrs"

As you can see, the frequency array stores the count of each character in the input string. In this example, the frequency of 'a' is 2, 'b' is 2, 'd' is 1, 'o' is 2, 'r' is 3, and 's' is 1. So, the characters will be ordered in the output string according to their frequency count and by their order of occurrence if frequency is same.
In this example, 'a' and 'b' have the same frequency so they will be ordered by the order of their occurrence in the input string which is 'a' then 'b'. Similarly, 'd','o','r' have same frequency so they will be ordered by their occurrence in the input string 'd','o','r' respectively. The rest of the characters will be arranged in the output string according to their frequency count.

The C program for this problem can be as follows : -

```
#include <stdio.h>
#include <string.h>
void sortByFrequency(char* s) {
```

```
    int freq[26] = {0};
    int n = strlen(s);
    for (int i = 0; i < n; i++) {
        freq[s[i] - 'a']++;
    }
    int k = 0;
    for (int i = 0; i < 26; i++) {
        while (freq[i]--) {
            s[k++] = i + 'a';
        }
    }
}
int main() {
    char s[26];
    scanf("%s", &s);
    sortByFrequency(s);
    printf("%s", s);
    return 0;
}
```

Answer 5 :

malloc() and calloc() are both C library functions that are used to dynamically allocate memory. The main difference between the two is how they initialize the memory that they allocate.

malloc() simply allocates a block of memory of a specified size, and returns a pointer to the first byte of that block. The contents of the memory are undefined, meaning that they could contain any value.

calloc() also allocates a block of memory of a specified size, but it also initializes the memory to all zeroes before returning a pointer to the first byte.

Here is an example of how you might use malloc() to dynamically allocate an array of integers in C:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int n;
   printf("Number of elements : ");
   scanf("%d", &n);
   int* arr = (int*) malloc(n * sizeof(int));
   printf("Enter the elements : ");
   for(int i=0; i<n; i++)
      scanf("%d", &arr[i]);
   printf("The elements are : ");
   for(int j=0; j<n; j++)
      printf("%d ", arr[j]);
  return 0;
}
```

Answer 6 :

A function is a block of code that performs a specific task. It can be reused multiple times throughout a program, which helps to make the code more organized. Functions can also accept inputs, called parameters, and can return outputs, called return values.

There are two types of functions :

1. Built-in Function / Predefined functions : hese are functions that are already defined in the C library, such as printf(), scanf(), strcpy(), etc.
2. User-defined functions: These are functions that you write yourself in your program, according to the requirements of your problem.

Types of user-defined functions :

1. **Function without return value and without parameter** : A function which does not take any parameter and does not return any value.

Example :

```
void print_hello() {
   printf("Hello!\n");
}
```

2. **Function without return value but with parameter** : A function which takes some parameters but does not return any value.
   Example :

```
void print_multiple_time(char* name, int n) {
   for(int i=0;i<n;i++)
      printf("Hello, %s!\n", name);
}
```

3. **Function with return value but without parameter** : A function which does not take any parameter but returns a value.
   Example :

```
int get_current_year() {
   return 2023;
}
```

4. **Function with return value and with parameter** : A function which takes some parameters and returns a value.
   Example :

```
int add(int a, int b) {
   return a + b;
}
```

Using user-defined functions has many benefits such as:

1. Code Reusability:
   A function can be called multiple times from different parts of the program, which makes it easier to reuse the code and write more efficient programs.

2. Modularity:
   A program can be divided into small, self-contained functions, which makes it easier to understand, modify and debug the code.

3. Reduced complexity:
   Breaking down a complex program into smaller, more manageable functions can make it easier to understand the overall logic of the program.

4. Easy Testing :
   Because a function is a self-contained piece of code, it is easy to test individual functions to ensure they are working correctly, which can make it easier to find and fix bugs in the program.

Answer 7 :

```
#include <stdio.h>
int main()
{
   int N, M;
   scanf("%d %d", &N, &M);
   int mat1[N][M], mat2[N][M], i, j, sum[N][M];
   for (i = 0; i < N; i++) {
      for (j = 0; j < M; j++) {
         scanf("%d", &mat1[i][j]);
      }
   }
   printf("\n");
   for (i = 0; i < N; i++) {
      for (j = 0; j < M; j++) {
         scanf("%d", &mat2[i][j]);
      }
   }
   for (i = 0; i < N; i++) {
      for (j = 0; j < M; j++) {
         sum[i][j] = mat1[i][j] + mat2[i][j];
      }
   }
   printf("\n");
   for (i = 0; i < N; i++) {
      for (j = 0; j < M; j++) {
         printf("%d ", sum[i][j]);
      }
```

```
        printf("\n");
    }
    return 0;
}
```

Answer 8 [A] :

In C, a structure (often called a struct) is a user-defined data type that groups together a collection of variables of different types under a single name. The variables inside a struct are called fields, and they can be of any type, including other structs.

Structures allow you to create complex data structures that can represent real-world objects or concepts. For example, you could use a struct to represent a point in a two-dimensional space, with fields for the x and y coordinates. Or you could use a struct to represent a student, with fields for the student's name, ID, and grades. One of the main advantages of using structures is that they make it easy to organize and manipulate data. By grouping related data together into a single struct, you can pass around, return, or store the struct as a single unit, rather than having to pass around or store each individual piece of data separately. Another advantage of using struct is it makes it easy to read and understand the code, it's more readable when the data items are packaged in meaningful structure.
In short, structures are used to group together a collection of variables of different types under a single name, making it easy to organize, manipulate, and reason about data and functionality.

Answer 8 [B] :
There are two ways to access the members of a structure in C: the dot notation and the arrow notation.

1. **The dot notation**: This notation is used when a structure variable is passed directly. This notation is similar to the way we access the members of an array. The dot operator(.) is used to access the members of the structure.
   Example,
   struct student
   {

```
    int roll_no;
    float percentage;
};
int main() {
    struct student s1;
    s1.roll_no = 101;
}
```

2. **The Arrow notation**: This notation is used when a pointer variable to the structure is passed. This notation is useful when we pass a pointer to the structure. The arrow operator (->) is used to access the members of the structure.
   Example,

```
struct student
{
    int roll;
    float percent;
};

int main() {
    struct student s1;
    struct student *s;
    s = &s1;
    s->roll = 101;
    s->percent = 87.5;
}
```

Answer 8 [C] :

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

struct student {
    int roll;
    char name[20];
    int marks;
```

```
};

int main() {
   struct student s1;
   printf("Enter information:\n");
   printf("Enter name: ");
   scanf("%s", &s1.name);
   printf("Enter roll number: ");
   scanf("%d", &s1.roll);
   printf("Enter marks: ");
   scanf("%d", &s1.marks);
   printf("\nDisplaying Information:\n");
   printf("Name: %s\n", s1.name);
   printf("Roll number: %d\n", s1.roll);
   printf("Marks: %d\n", s1.marks);
   return 0;
}
```

Answer 9 :

1. **Syntax errors:** These are errors that occur when the C compiler encounters code that does not conform to the C language syntax. For example, the following program contains a syntax error because the semicolon is missing at the end of the second line
   Example,
   ```
   int main()
   {
      printf("Hello World)
      return 0;
   }
   ```

2. **Compile-time errors:** These are errors that are detected by the compiler when it attempts to generate machine code from the source code. For example, the following program contains a compile-time error because the variable x is used before it is defined
   Example,

```
int main()
{
   int y = x;
   int x = 10;
   printf("y = %d", y);
   return 0;
}
```

3. **Run-time errors:** These are errors that occur when the program is executed and cannot be detected by the compiler. For example, the following program contains a run-time error because the array arr is indexed out of bounds
   Example,

```
#include <stdio.h>

int main() {
   int arr[5] = {1, 2, 3, 4, 5};
   printf("%d", arr[5]);
   return 0;
}
```

4. **Arithmetic errors:** These errors occur when there is a mistake in the arithmetic calculations like dividing by zero
   Example,

```
#include <stdio.h>

int main()
{
   int a = 5, b = 0;
   int c = a/b;
   printf("%d", c);
   return 0;
}
```

5. **Uninitialized variable errors:** These occur when you use a variable that has not been initialized.

Example,

```
#include <stdio.h>
int main()
{
   int a;
   printf("%d", a);
   return 0;
}
```

Answer 10 [A] :

C is often referred to as the "mother of all languages" because it is the predecessor of many other programming languages and has influenced the development of many others. C was one of the first high-level programming languages, which means that it was designed to be more human-readable than machine code (the raw instructions that computers can execute directly). This made it much easier for programmers to write and understand complex programs.

C was developed in the early 1970s by Dennis Ritchie at Bell Labs, and it was designed to be used in the development of the UNIX operating system. Because of its simplicity, portability, and efficiency, C quickly became popular for a wide range of programming tasks, and it was widely adopted in the industry.

In addition to its direct descendants like C++ and C#, many other programming languages have been influenced by C, including Java, JavaScript, Perl, PHP, and Python. The structure, syntax, and features of C have been adopted and adapted in many other languages, making it one of the most widely used and influential programming languages in history. Also the pointer concept and being able to access the underlying hardware makes it a powerful language.

Answer 10 [B] :

Source code is the set of instructions that a programmer writes in a programming language. It is the human-readable form of a program, and it can be edited and modified by a programmer to add new features or fix bugs. The source code is typically saved in a plain-text file with a specific file extension that corresponds to the programming language being used

When a program is ready to be run on a computer, the source code must be translated into machine code, which is the set of instructions that a computer's processor can understand and execute directly. This translation process is called compiling.

When a program is compiled, the source code is transformed into an object file, which contains machine code along with other information, such as symbols and data. Object files have a .obj file extension. Object files are not executable on their own.

A linker then used to link the multiple object files and combine them into a single executable file, this combined file is called as executable file and it have a .exe file extension. The executable file contains all of the machine code and other information needed to run the program, in a format that the computer's operating system can understand.

Answer 10 [C] :
When you run a C program, the following files may be created on your device:

1. **The source code file:** This is the file that you, the programmer, write and edit. It contains the C code in a human-readable format, and typically has a .c or .h file extension.
2. **The object file:** This is a file that is created when you compile the C program. It contains machine code that corresponds to the C source code, along with other information like symbols and data. Object files typically have a .o or .obj file extension.
3. **The executable file:** This is a file that is created when you link the object file (or multiple object files) together. It contains all of the machine code and other information needed to run the program, in a format that the operating system can understand. Executable files typically have a .exe file extension on Windows and no extension on Linux and macOS.

It is worth noting that this process of creating all these files is platform dependent and might vary based on the compiler or the build system. Also, when running a program that is interpreted like python, no executable file is generated.

Answer 10 [D] :
The base condition in a recursive function is a special case that the function checks for before making a recursive call. It is the stopping point for the recursion, and the function will return a value (or perform some other action) once it reaches the base condition. Without a base condition, a recursive function would continue to call itself indefinitely, leading to an infinite loop.

A real-life example of recursion is solving a mathematical function. for example, the factorial function which is defined as the product of all the integers from 1 to n, where n is a positive integer, is often computed recursively, with the base condition n = 0 or n = 1 .
In such a case the function will return 1.

Answer 10 [E] :
When a struct is defined in C, the order of the elements in the struct determines the layout of the data in memory. If a struct contains a large data type, such as a double or an array, and that data type is placed at the end of the struct, any padding that is added to align the other members of the struct will be added after the large data type. This can cause the struct to take up more memory than is necessary.

On the other hand, if the large data type is declared first in the struct, it will be aligned on the appropriate boundary, and any padding will be added before the large data type. This can help to minimize the amount of memory that is used by the struct.
Therefore it's a good practice to declare large data types first in the structure to keep the memory efficient.