

## Table of Contents

MODEL SET 1 .....	1
1. Define array. What is the drawbacks of array? Describe different types of array with suitable programs. ....	1
2. What is function? Describe types of functions with suitable practical examples. ....	2
3. What are the major differences between text file and binary file? Given a text file, create another text file deleting the following words “three”, “bad”, “time” .....	3
4. Draw the flowchart for finding largest of the three numbers and write an algorithm and explain it. ....	4
5. Write a program to add, subtract, multiply and divide two complex numbers (x+iy) and (c+id) by using function. ....	5
6. What is structure? Define a structure of employee having data members name, address, age and salary. Take data for n employee in an array dynamically and find the average salary. ....	5
7. What is string in C? How it is declared in C? Write a C program to count total number of alphabets, digits or special characters in a string using loop. ....	6
8. Write syntax to declare and initialize 2-dimensional array. Write a program to enter two 3x3 matrices and calculate the product of given matrices. ....	7
9. What is recursive function? Write a program in C to generate Fibonacci series up to n <sup>th</sup> term using recursion. ....	8
10. Justify that pointer is jewel of C language. Write a function that is passed an array of pointers to floats and returns a newly created array that contains those n float values in reverse order. Assume any necessary data. ....	9
11. Explain the use of graphical function, Make a program that contains basic graphic functions. ....	9
12. Write short notes on: .....	10
Dynamic Memory Allocation: .....	10
Delimiters: .....	11
MODEL SET 2 .....	12
1. Find the value of ‘a’ in each of the following statements. ....	12
2. Why flow chart is required? Explain different symbols used in the flow chart and explain with suitable example. ....	12
3. What is structure? How it is different from union? Define a structure of students having data members name, address, marks in C language, and marks in Information System. Take data for n students in an array dynamically and find the total marks obtained. ....	13
4. Explain switch statement with example. ....	14
5. Write a program to accept 2 numbers and sort them with using pointer. ....	15

6. What is recursion? Write a program to accept any number and print the sum of that single digit through recursive function.....	16
7. What are the basic four data types used in C programming? What are its size and range? Explain. ....	16
8. Explain the multidimensional array with example. Write a program to convert a lowercase character string to uppercase.....	17
9. Explain the array of structures and write a program to accept records of 25 persons which has name, age and address and also display them. ....	17
10. Explain the pointer arithmetic with examples. ....	18
11. Differentiate between break and exit statement with example. ....	20
12. Write short notes on: .....	21
a) Delimiters: .....	21
b) Graphics function: .....	21
MODEL SET 3 .....	21
1. What is dynamic memory allocation? List out possible functions used for DMA. Write a program in C to read and print the N student details using structure and dynamic memory allocation.....	21
2. What is array? What are the disadvantages of array? Write a program to enter two 3x3 matrices and calculate the product of given matrices. ....	22
3. What is file handling in C? What are the advantages of file handling? Create a text file and enter "to write a good program is very time consuming job." Create another text which contains reverse of above text.....	22
4. What is algorithm? How does it differ from flowchart?.....	23
5. What is type conversion? Discuss type caseing with suitable example.....	24
6. Write a program for the interest charged in installments for following case. A cassette player costs Rs. 2000. A shopkeeper sells it for Rs. 100 down payment and Rs. 100 for 21 more months. What is the monthly interest charged?.....	25
7. Write a program which will read a line and delete from it all occurrences of the word "that". ....	25
8. Define a structure of employee having data members name, address, age, and salary. Take data for n employee in an array dynamically and find the average salary.....	26
9. Determine which of the following are valid identifiers? If invalid, explain why? .....	26
10. Find the errors in the following program and explain it. Write the correct program. ....	26
11. Write and test the following power() function that returns x raised to the power n, where n can be any integer: double power(double x, int p).....	27
12. Explain the use of graphical function. Make a program that contains basic graphic functions.....	27

MODEL SET 4 .....	27
1. What is operator? Describe various operators used in C with suitable examples. ....	27
2. What is statement? Describe types of statements used in C with suitable example. ....	32
3. What is binary file? How is it different from text file? Create a text file and enter "I am the student of CSIT first semester". Create another text which contains only smaller letters of above text. ....	39
4. Draw a flow chart and write an algorithm to find out whether a given number is zero, +ve or -ve. ....	40
5. Differentiate between while and do while loop with example. ....	40
6. What is the function of a pointer variable? Explain the declaring and initializing pointers with example. ....	41
7. Differentiate between call by value and call by reference with example. ....	41
8. What is structure? How is it different from array and union? Discuss .....	43
9. Determine which of the following are valid identifiers? If invalid, explain why? .....	43
10. Write a program in C to sort the following data items using Bubble sort: .....	43
11. What do you mean by nesting of structure? Explain with suitable example. ....	44
12. Explain the use of graphical function. Make a program that displays a rectangle by using graphic handling function. ....	46
MODEL SET 5 .....	46
1. What is storage class in C? Describe various storage classes used in C with suitable example. ....	46
2. What is macro? How it is differ from function? 'Explain with suitable example. ....	48
3. What is file pointer? How it is differ from text file? Write a program in C to read all numbers from the input file "values.dat" and stores the average of these numbers in an output file named as "average.res" .....	49
4. What is identifier? Write naming conventions of defining identifiers in programming like C .....	50
5. What is the use of sizeof() operator? Find out size of various data types in bytes. ....	50
6. What is type casting? Define type casting with suitable example. ....	51
7. Write a program in C to check whether given entered year is leap year or not .....	51
8. What is switch statement? Write a program to read any operator and perform the respective operation on the operands. ....	51
9, program to find the greatest common divisor (GCD) and lowest common multiple (LCM) .....	53
10. Write a program in C to display following pattern: .....	53
11. What is Infinite Loop? Demonstrate infinite loop with suitable practical example. ....	54
12. Explain the use of graphical function. Make programs that display an arc by using graphic handling function. ....	54

MODEL SET 6 .....	55
1. What is structured programming? Describe advantages of structured programming.....	55
2. What is iterative statement? List out types of looping statements used in C and describe each of them with suitable example.....	55
3. How can you initialize two-dimensional array in C? Write a program in C to sort given array of numbers by using bubble sort technique. ....	57
4. Write a function to multiply two n x n matrices. ....	58
5. What is searching? Describe sequential searching with suitable example. ....	59
6. Write a program to reverse the given string without using the <code>strrev()</code> . ....	60
7. What do you mean by array of structures? Describe array of structures with suitable examples. ....	60
8. Write a program to input any two numbers and then find their product by using the concepts of pointers. ....	60
9. What is function? What are the advantages of using functions in C? Explain the use of functions with suitable example.....	60
10. Write a program to test if a user input string is a palindrome or not. ....	62
11. Write down the concept of typedef with suitable example.....	62
12. How graphics driver initializes in C? Write a program to draw a circle. ....	63
MODEL SET 7 .....	64
1. What is debugging? Explain various types of error occur in execution of C program....	64
2. What is dynamic memory allocation? What are the advantages of DMA over array? Show the concept of DMA with suitable example. ....	65
3. What is nesting of structure? Explain nesting of structure with suitable example. ....	65
4. Write a program to find the factorial of a given number by using recursion.....	66
5. What is a preprocessor directive in C? .....	66
6. Write a program to read name and roll number of 48 students and store them into file <code>stu.txt</code> by using <code>fprintf()</code> function.....	67
7. Write a program to convert the given string in to lower case letters without using string handling function <code>strlwr()</code> . ....	67
8. What is variable? Explain types of variables used in C with suitable example.....	68
9. What is function? What are the components of function in C? .....	69
10. What is the purpose of defining algorithm before writing the program? Write an algorithm to find roots of given quadratic equation.....	70
11. What is the concept behind ternary operator? Write a program in C to test whether given number is odd or even by using ternary (conditional) operator.....	71
12. Explain the use of graphical function. Make a program that contains basic graphic functions.....	71

MODEL SET 8 .....	72
1. How string declared in C? Explain two dimensional array of character. Also write a program to sort given n strings by using bubble sort.....	72
2. What is the concept behind pointers? Differentiate between malloc() and calloc() functions. Describe the DMA functions with suitable example. ....	73
3. What is array of structure? Explain array of structure with taking records of any 20 students and display them. ....	74
4. Write a program to generate Fibonacci series up to n terms using recursion function. (Hint Fibonacci sequence = (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ....) .....	75
5. What is if else if ladder? Explain it with their flow chart and complete C program.\.....	75
6. Write a program to read name of any n students from user then display only largest name and their length. ....	77
7. What is an algorithm? Write an algorithm to check given number is prime or composite. ....	77
8. What is escape sequence? Explain types and use of escape sequence with suitable example.....	78
9. What is the use of function declaration? Write a program in C to find the simple interest by using function.....	78
10. Write a program that computes the sum of digits of given integer number. ....	79
11. Write a program in C to read a text file that contains the word three, bad, time etc. and create another text file deleting the following words 'three", "bad", and "time". ....	79
12. Write short note on: .....	79
a. Graphic function.....	79
b. Formatted I/O.....	79

## MODEL SET 1

### 1. Define array. What is the drawbacks of array? Describe different types of array with suitable programs.

An array is a variable that can store multiple values of the same data type. Each item of an array is called an element of the array. Each element is distinguished from another with an index. All elements are stored contiguously in memory.

Array can be declared in the following way:

```
int nums[] = {1, 4, 6, 7, 4, 4};  
char name[] = "Joon";
```

An empty array can be declared in the beginning and elements can be assigned later on.

```
int marks[5];  
marks[0] = 100;  
marks[1] = 99;
```

Elements of an array can be accessed by its index.

```
int firstMark = marks[0];
```

The drawbacks of array are as follows:

- The size of array is fixed. The memory which is allocated it cannot be increased or decreased. So, while declaring we must know the size of array beforehand.
- There will be wastage of memory if elements inserted in an array are less than array size.
- If elements are inserted in the index that are out of range of the array size, it may affect the value of other variables in the memory.
- Inserting and deleting elements in an array is tedious.
- Array is homogeneous i.e.: it can only store data of one data type.
- Arrays can't be reassigned after initialization.
- Array of characters (strings) cannot be compared directly to check if the strings are same.

The different types of arrays are:

- One-Dimensional Array:

An array whose elements are variables like int, char, float, double, etc. but not an array is called one-dimensional array. It is used to store list of items.

Example:

```
#include <stdio.h>  
int main() {  
    int primeNums[] = {2, 3, 5, 7, 11, 13, 17, 19, 23}  
    printf("The fourth prime number is: %d", primeNumbers[3]);  
    return 0;  
}
```

- **Multidimensional Arrays:**

An array whose elements are also an array of elements storing values is called multidimensional arrays. The depth of nested arrays determines the dimension of multidimensional array.

Example:

```
#include <stdio.h>
int main() {
    int triangle[3][2] = {{0, 0}, {0, 2}, {2, 2}};
    printf("The co-ordinates of the triangles are:");
    int i;
    for (i = 0; i < 3; i++) {
        printf("\n(%d, %d)", triangle[i][0], triangle[i][1]);
    }
    return 0;
}
```

## 2. What is function? Describe types of functions with suitable practical examples.

A function is a routine or set of instructions or code that performs a specific task and can be processed independently. When the program passes the control to a function, the function performs the task and returns the control to the instruction following the calling instruction.

There are two types of functions:

- **Standard Library functions:**

The functions that are already predefined in built-in header files of C are called standard library functions. E.g.:

printf()	// included in stdio.h
sqrt()	// included in math.h

Example:

```
#include <stdio.h>
#include <math.h>
int main() {
    printf("The square root of 25 is %.0f", sqrt(25)).
    return 0;
}
```

- **User defined function:**

The functions which are defined by user are called user defined functions.

Example:

```
#include <stdio.h>
void sayHi() {
    printf("Hello!, World");
    return;
}
int main() {
    sayHi();
    return 0;
}
```

**3. What are the major differences between text file and binary file? Given a text file, create another text file deleting the following words “three”, “bad”, “time”.**

Text File	Binary File
It stores data using ASCII, Unicode, etc. format i.e.: human readable character	It stores data in binary format i.e.: 0, 1.
All characters are converted to ASCII of 1 byte size which takes more space.	Long numbers in length don't take up the space of the length of the characters, instead the numbers are converted into binary. So, all integers eat only 4 bytes
Error in text file can easily be recognized and emitted.	Errors in binary file corrupts the file and is not easily detected
In text file newline characters is first converted into a carriage return-line feed combination and then returned to the disk. Vice versa happens when a line is read from the hard disk.	In binary file, no such conversion from line feed combination is done.
ASCII code 26 is inserted at the end to signal EOF	No characters are inserted to signal EOF

Program:

```
#include <stdio.h>
#include <string.h>
void removeWords(char *line, char *removeStr) {
    int removeStrLen = strlen(removeStr), i;
    char *start = strstr(line, removeStr);
    while (start) {
        int pos = start - &line[0];
        for (i = pos + removeStrLen; line[i] != '\0'; i++) {
            line[i - removeStrLen] = line[i];
        }
        line[i - removeStrLen] = '\0';
        start = strstr(line, removeStr);
    }
}
int main() {
    FILE *ifp, *ofp;
    char line[200];
    ifp = fopen("input.txt", "r");
    ofp = fopen("output.txt", "w");
    if (!ifp || !ofp) {
        printf("File not found or in use.");
        return -1;
    }
    while (fgets(line, 200, ifp)) {
        removeWords(line, "three");
        removeWords(line, "bad");
        removeWords(line, "time");
        fputs(line, ofp);
    }
}
```



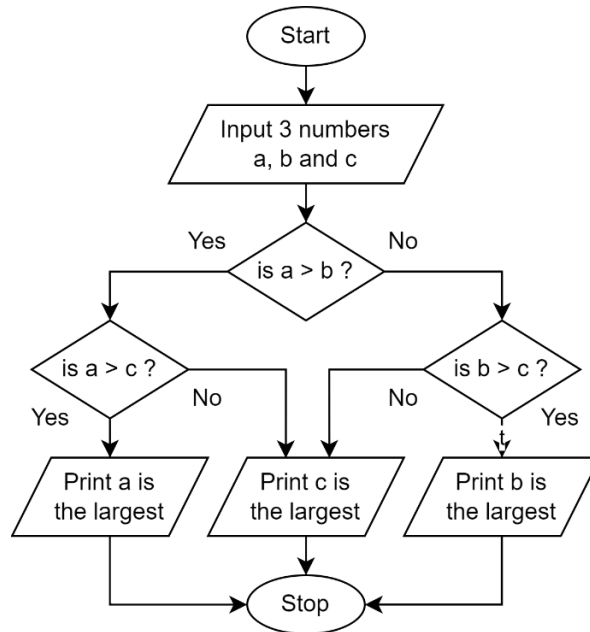
```

fclose(ifp);
fclose(ofp);
printf("Task completed successfully.");
return 0;
}

```

**4. Draw the flowchart for finding largest of the three numbers and write an algorithm and explain it.**

Flowchart:



Algorithm:

Step 1: Start

Step 2: Input three numbers a, b and c.

Step 3: If  $a > b$ :

    If  $a > c$ :

        Print a is the largest number.

    Else:

        Print c is the largest number.

Else:

    If  $b > c$ :

        Print b is the largest number.

    Else:

        Print c is the largest number.

Step 4: Stop

Explanation: This program inputs three numbers: a, b and c; and it check if  $a > b$ . If yes, it checks if  $a > c$ . If yes, it means  $a > b$  and c. so it'll print a is the largest if not then as  $a > b$  and  $c > a$ , so it'll print c is the largest. But if  $b > a$  then, it'll check if  $b > c$ , if yes, it'll print b is the largest, else it'll print c is the largest.

**5. Write a program to add, subtract, multiply and divide two complex numbers (x+iy) and (c+id) by using function.**

Program:

```
#include <stdio.h>
void add(int x, int y, int c, int d) {
    int re = x + c, im = y + d;
    printf("The sum is: %d + %di\n", re, im);
}
void subtract(int x, int y, int c, int d) {
    int re = x - c, im = y - d;
    printf("The difference is: %d + %di\n", re, im);
}
void multiply(int x, int y, int c, int d) {
    int re = x * c - y * d, im = x * d + y * c;
    printf("The product is: %d + %di\n", re, im);
}
void divide(int x, int y, int c, int d) {
    float re = 1.0 * (x * c + y * d) / (c * c + d * d);
    float im = 1.0 * (y * c - x * d) / (c * c + d * d);
    printf("The quotient is: %f + %fi\n", re, im);
}
int main() {
    int x, y, c, d;
    printf("Enter the real part of first complex number: ");
    scanf("%d", &x);
    printf("Enter the imaginary part of first complex number: ");
    scanf("%d", &y);
    printf("Enter the real part of second complex number: ");
    scanf("%d", &c);
    printf("Enter the imaginary part of second complex number: ");
    scanf("%d", &d);
    add(x, y, c, d);
    subtract(x, y, c, d);
    multiply(x, y, c, d);
    divide(x, y, c, d);
    return 0;
}
```

**6. What is structure? Define a structure of employee having data members name, address, age and salary. Take data for n employee in an array dynamically and find the average salary.**

A structure is a user defined data type in C. A structure creates a data type that can be used to group items of possibly different types into a single type. A structure is convenient way of grouping several pieces of related information together.

Syntax for declaring structure in C is as below:

```
struct Structure_name {
    data_type_1 member_1_name;
    data_type_2 member_2_name;
    .....
    data_type_n member_n_name;
}
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Employee {
    char name[20];
    char address[20];
    int age;
    int salary;
};
int main() {
    int n, i, sumSalary = 0;
    struct Employee *employees;
    printf("Enter the number of employee: ");
    scanf("%d", &n);
    employees = (struct Employee *)malloc(n * sizeof(struct Employee));
    for (i = 0; i < n; i++) {
        printf("\nEnter the name of employee %d: ", i + 1);
        scanf("%[^\n]s", employees[i].name);
        printf("Enter the address of employee %d: ", i + 1);
        scanf("%[^\n]s", employees[i].address);
        printf("Enter the age of employee %d: ", i + 1);
        scanf("%d", &employees[i].age);
        printf("Enter the salary of employee %d: ", i + 1);
        scanf("%d", &employees[i].salary);
    }
    for (i = 0; i < n; i++) {
        sumSalary += employees[i].salary;
    }
    printf("\nThe average salary is: %.2f", 1.0 * sumSalary / n);
    free(employees);
    return 0;
}
```

**7. What is string in C? How it is declared in C? Write a C program to count total number of alphabets, digits or special characters in a string using loop.**

Strings are one dimensional arrays of type char. By convention, a string in C is terminated by the end-of-string sentinel `\0` or null character. Its decimal value is 0. When declaring a character array to hold a string, we declare it to be one character longer than the largest string that it will hold.

A string can be declared in the following ways:

```
char str[] = "abcd";
char str[50] = "abcd";
char str[] = {'a', 'b', 'c', 'd', '\0'};
char str[5] = {'a', 'b', 'c', 'd', '\0'};
```

Program:

```
#include <stdio.h>
int main() {
    char string[100];
    int i, count = 0;
    printf("Enter a string: ");
    scanf("%[^\n]s", string);
    for (i = 0; string[i] != '\0'; i++) {
        if (string[i] != ' ') {
            count++;
        }
    }
    printf("The number of alphabets, digits or special characters are: %d",
        count);
    return 0;
}
```

**8. Write syntax to declare and initialize 2-dimensional array. Write a program to enter two 3x3 matrices and calculate the product of given matrices.**

In C programming, we can declare 2-dimensional array by using two subscripts as below:

```
int a[size of first index][size of second index];
```

e.g.: float x[3][4];

Here, x is a two-dimensional (2d) array. The array can hold 12 elements. We can think this array as table with 3 rows and each row has 4 columns.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Initialization of two-dimensional array:

We can initialize two-dimensional array by following three different ways:

- `int c[2][3] = {{4, 5, 7}, {7, 5, 3}};`
- `int c[][3] = {{4, 5, 7}, {7, 5, 3}};`
- `int c[2][3] = {4, 5, 7, 7, 5, 3};`

Program:

```
#include <stdio.h>
int main() {
    int m1 = 3, n1 = 3, m2 = 3, n2 = 3;
    int firstMatrix[m1][n1];
    int secondMatrix[m2][n2];
    int i, j, k;
    for (i = 0; i < m1; i++) {
        for (j = 0; j < n1; j++) {
            printf("Enter the number in row %d, column %d of first matrix: ",
                i + 1, j + 1);
            scanf("%d", &firstMatrix[i][j]);
        }
    }
}
```

```

for (i = 0; i < m2; i++) {
    for (j = 0; j < n2; j++) {
        printf("Enter the number in row %d, column %d of second matrix: ",
            i + 1, j + 1);
        scanf("%d", &secondMatrix[i][j]);
    }
}
int resultMatrix[m1][n2];
for (i = 0; i < m1; i++) {
    for (j = 0; j < n2; j++) {
        int sum = 0;
        for (k = 0; k < n1; k++) {
            sum += firstMatrix[i][k] * secondMatrix[k][j];
        }
        resultMatrix[i][j] = sum;
    }
}
printf("\nResult:\n");
for (i = 0; i < m1; i++) {
    for (j = 0; j < n2; j++) {
        printf("%d ", resultMatrix[i][j]);
    }
    printf("\n");
}
return 0;
}

```

**9. What is recursive function? Write a program in C to generate Fibonacci series up to  $n^{\text{th}}$  term using recursion.**

A recursive function is one that calls to itself to solve a smaller version of its task until a final call which does not require a self-call. Thus, a function is known as recursive function if it calls to itself.

Program:

```

#include <stdio.h>
int fibonacci(int n) {
    if (n == 1) return 0;
    if (n == 2) return 1;
    return (fibonacci(n - 1) + fibonacci(n - 2));
}
int main() {
    int terms, i;
    printf("Enter the number of terms: ");
    scanf("%d", &terms);
    for (i = 1; i <= terms; i++) {
        printf("%d ", fibonacci(i));
    }
    return 0;
}

```

**10. Justify that pointer is jewel of C language. Write a function that is passed an array of pointers to floats and returns a newly created array that contains those n float values in reverse order. Assume any necessary data.**

A pointer is a derived data type in C. Pointers are undoubtedly one of the most distinct and exciting features of C language. It has added power and flexibility to the language. Pointers are more efficient in handling arrays and tables. Pointer can be used to support dynamic memory management. Pointers reduce length and complexity of programs. Pointers increase the execution speed and thus reduce the program execution time. Hence by following characters real power of C lies in proper use of pointers. pointer is called the jewel of C language.

Program:

```
#include <stdio.h>
float* reverse(float* arr[], int n) {
    float* tempArray;
    int i;
    for (i = 0; i < n; i++) {
        tempArray[i] = *arr[n - i - 1];
    }
    return tempArray;
}
int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    float arrWithFloats[n], *arrWithPointers[n];
    for (i = 0; i < n; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%f", &arrWithFloats[i]);
    }
    for (i = 0; i < n; i++) {
        arrWithPointers[i] = &arrWithFloats[i];
    }
    printf("The array in reverse order is:\n");
    float* newArr = reverse(arrWithPointers, n);
    printf("Items after reversing: ");
    for (i = 0; i < n; i++) {
        printf("%f ", newArr[i]);
    }
    return 0;
}
```

**11. Explain the use of graphical function, Make a program that contains basic graphic functions.**

C graphics using graphics.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using graphics functions of graphics.h in Turbo C compiler can make graphics programs, animations projects and games. We can draw circles, lines, rectangles, bars and many other geometrical figures. We can change their colors using the available functions and fill them.

The commonly used graphic functions in C are listed below:

- `initgraph()`
- `closegraph()`
- `line()`
- `rectangle()`
- `arc()`
- `bar()`
- `ellipse()`
- `circle()`
- `setcolor()`
- `setbkcolor()`
- `getcolor()`
- `getmaxx()`
- `getmaxy()`
- `getx()`
- `gety()`
- `getpixel()`
- `putpixel()`

Program:

```
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
int main() {
    int gDriver = DETECT, gMode;
    initgraph(&gDriver, &gMode, (char *)"C:\\\\TURBOC3\\\\BGI");
    line(50, 40, 190, 40);
    rectangle(125, 115, 215, 165);
    arc(120, 200, 180, 0, 30);
    circle(120, 270, 30);
    ellipse(120, 350, 0, 360, 30, 20);
    getch();
    return 0;
}
```

## 12. Write short notes on:

### Dynamic Memory Allocation:

Dynamic memory allocation means to allocate the memory at runtime. Dynamic memory allocation is possible by 4 functions of `stdlib.h` header files:

1	<code>malloc()</code>	Allocate single block of requested memory.
2	<code>calloc()</code>	Allocate multiple blocks of requested memory.
3	<code>realloc()</code>	Reallocates the memory occupied by <code>malloc()</code> or <code>calloc()</code> functions.
4	<code>free()</code>	Free the dynamically allocated memory.

An array is a collection of fixed number of values a single type. That is, you need to declare the size of an array before you can use it. Sometimes the size of the array you declare may be insufficient. To solve this issue, you can allocate memory manually during runtime. This is known as dynamic memory allocation in C programming.

Example program for dynamic memory allocation:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n, i, *ptr, sum = 0;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    ptr = (int*)malloc(n * sizeof(int));
    if (!ptr) {
        printf("Error! Memory not allocated.");
        exit(-1);
    }
    for (i = 0; i < n; i++) {
        printf("Enter element %d: ", i);
        scanf("%d", ptr + i);
        sum += *(ptr + i);
    }
    printf("Sum: %d", sum);
    free(ptr);
    return 0;
}
```

### **Delimiters:**

A delimiter is any character or string that separates a sequence of characters or strings. It is used for readability and extracting chars or strings from the sequence. A delimiter pair is a pair of character or a string.

e.g.:

This is a text (here space is used as a delimiter.)

1,2,3,4,5,6 (here comma (,) is used as a delimiter.)

Some other delimiters are to separate code blocks, lines are:

- : Colon – used to create label
- ; Semicolon – used to terminate a statements
- # Hash – used in preprocessor directive
- , Comma – used as variable separator
- ( ) Parenthesis – used to enclose list of parameters in function definition and invocation, define precedence in an expression, and enclose type cast
- [ ] Brackets – used to declare an array and also used to dereference array values
- { } Braces – used to put values of automatically initialized array and to define the block of code



## MODEL SET 2

### 1. Find the value of 'a' in each of the following statements.

```
int i = 3, j = 4, k = 8;
```

```
float a = 4.5, b = 6.5, c = 3.5;
```

```
a) a = b - i / k + c / k  
6.937500
```

```
b) a = (b - k) / j + (j + c) / k  
0.562500
```

```
c) a = c - ((i + j) + (k + i)) * b  
-113.500000
```

```
d) a = c - i + j / k + i * k  
24.500000
```




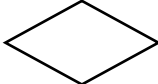
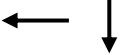

```
e) a = c + j % 2 + b  
10.000000
```

```
f) a = (b + 1) % (c + 1)  
Not valid
```

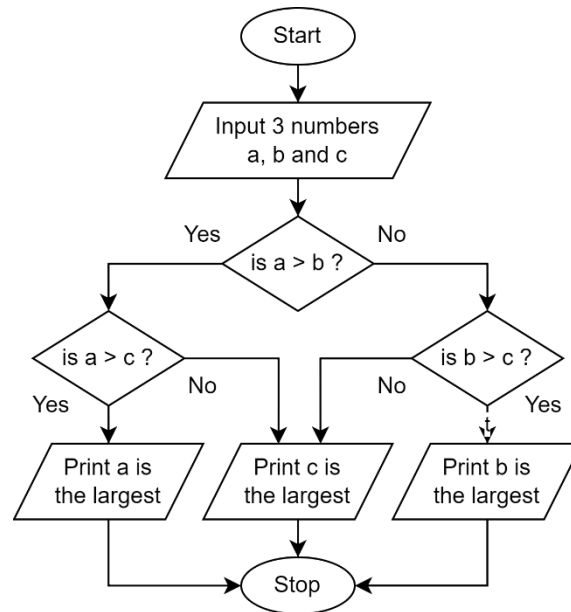
### 2. Why flow chart is required? Explain different symbols used in the flow chart and explain with suitable example.

A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes, and arrows to demonstrate a process or a program. It shows the steps as boxes of various kinds, and their order by connecting these with arrows. Flowcharts are required to analyze, design, document or manage a process or program in various fields in intuitive way.

Different symbols used in flowcharts are:

	Used to represent start and end of flowchart.
	Used for input and output operation.
	Used for arithmetic operation and data manipulation.
	Used in decision making.
	Used to indicate the flow of logic
	Used to connect pages.

Example of flowchart:



**3. What is structure? How it is different from union? Define a structure of students having data members name, address, marks in C language, and marks in Information System. Take data for n students in an array dynamically and find the total marks obtained.**

A structure is a user defined data type in C. A structure creates a data type that can be used to group items of possibly different types into a single type. A structure is convenient way of grouping several pieces of related information together.

The difference between structure and union are listed below:

Structure	Union
We can access all the members of structure at anytime	Only one member of union can be accessed at anytime
Memory is allocated for all variables	Allocate memory to fit the largest variables containing in the union.
All members of structure can be initialized	Only the first member of the union can be initialized
Struct keyword is used to declare structure	Union keyword is used to declare union
Syntax <pre> struct Structure_name {     data_type_1 member_1_name;     data_type_2 member_2_name;     .....     data_type_n member_n_name; } </pre>	Syntax <pre> union Union_name {     data_type_1 member_1_name;     data_type_2 member_2_name;     .....     data_type_n member_n_name; } </pre>
Example <pre> struct Student {     int rollNumber;     char name[20]; }; struct Student s; </pre>	Example <pre> union Employee {     int id;     char name[20]; }; union Employee e; </pre>

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Student {
    char name[20];
    char address[20];
    int marksC;
    int marksIS;
};
int main() {
    int n, i;
    struct Student *students;
    printf("Enter the number of students: ");
    scanf("%d", &n);
    students = (struct Student *)malloc(n * sizeof(struct Student));
    for (i = 0; i < n; i++) {
        printf("\nEnter the name of student %d: ", i + 1);
        scanf("%[^\n]s", students[i].name);
        printf("Enter the address of student %d: ", i + 1);
        scanf("%[^\n]s", students[i].address);
        printf("Enter the marks in C of student %d: ", i + 1);
        scanf("%d", &students[i].marksC);
        printf("Enter the marks in IS of student %d: ", i + 1);
        scanf("%d", &students[i].marksIS);
    }
    for (i = 0; i < n; i++) {
        printf("\nThe total marks of student %d is: %d", i,
            students[i].marksC + students[i].marksIS);
    }
    free(students);
    return 0;
}
```

#### **4. Explain switch statement with example.**

Switch statement is a multiple branch selection statement, which successively tests the value of an expression against a list of integer or character constants. When a match is found, the statement(s) associated with that constant are executed.

The expression must evaluate to an integer type. The value of expression is tested against the constants present in the case labels. When a match is found, the statement sequence, if present, associated with that case is executed until the break statement or the end of the switch statement is reached. The statement following default is executed if no matches are found. The default is optional, and if it is not present, no action takes place if all matches fail.

Example of switch statement:

```
#include <stdio.h>
int main() {
    char ch;
    printf("Enter a character: ");
    scanf("%c", &ch);
    switch (ch) {
        case 'a':
        case 'A':
            printf("EXCELLENT");
            break;
        case 'b':
        case 'B':
            printf("VERY GOOD");
            break;
        case 'c':
        case 'C':
            printf("GOOD");
            break;
        case 'd':
        case 'D':
            printf("SATISFACTORY");
            break;
        case 'e':
        case 'E':
            printf("FAIL");
            break;
        default:
            printf("Wrong entry!");
    }
    return 0;
}
```

**5. Write a program to accept 2 numbers and sort them with using pointer.**

```
#include <stdio.h>
int main() {
    int num1, num2, *ptr1, *ptr2, temp;
    ptr1 = &num1;
    ptr2 = &num2;
    printf("Enter two numbers: ");
    scanf("%d%d", ptr1, ptr2);
    if (*ptr1 > *ptr2) {
        temp = *ptr1;
        *ptr1 = *ptr2;
        *ptr2 = temp;
    }
    printf("The sorted numbers are: %d %d", num1, num2);
    return 0;
}
```

**6. What is recursion? Write a program to accept any number and print the sum of that single digit through recursive function.**

Recursion is the process of a function calling itself to solve a smaller version of its task until a final call which does not require a self-call. The function that utilizes recursion is called recursive function.

Program:

```
#include <stdio.h>
int sumOfDigits(int num) {
    if (num / 10 == 0) {
        return num;
    }
    int sum = 0, remainder;
    while (num > 0) {
        remainder = num % 10;
        sum += remainder;
        num = num / 10;
    }
    return sumOfDigits(sum);
}
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("The sum of the digits till single digit is: %d", sumOfDigits(num));
    return 0;
}
```

**7. What are the basic four data types used in C programming? What are its size and range? Explain.**

C supports several different types of data, each of which may be represented differently within the computer's memory. A data type defines the amount of memory that will be used during program execution, valid range of values it can represent, and the operations that may be performed on it. C is strongly typed language so every variable in C must be declared of specific type explicitly. Data types can be either predefined or derived.

The C language supports four basic data types, each of which are represented differently within the computer memory. The basic types for a particular compiler are listed below:

Data Types	Description	Memory Requirement	Range
char	Stores a single character	1 byte	-128 to 127 ( $-2^7$ to $2^7 - 1$ )
int	Stores an integer value	2 or 4 bytes	-32,768 to 32,767 ( $-2^{15}$ to $2^{15} - 1$ ) or -2,147,483,648 to 2,147,483,647 ( $-2^{31}$ to $2^{31} - 1$ )
float	Store single-precision floating-point number.	4 bytes	-3.4E+38 to 3.4E+38 With six digits of precision
double	Double-precision floating-point number	8 bytes	-1.7E+308 to 1.7E+308 With ten digits of precision

**8. Explain the multidimensional array with example. Write a program to convert a lowercase character string to uppercase.**

An array inside of an array is called multidimensional array. The depth of the nested arrays determines the dimension of the array.

It is declared by put two big brackets after the variable name.

Example program using multidimensional array:

```
#include <stdio.h>
int main() {
    int triangle[3][2] = {{0, 0}, {0, 2}, {2, 2}};
    printf("The coordinates of the triangle are:");
    int i;
    for (i = 0; i < 3; i++) {
        printf("\n(%d, %d)", triangle[i][0], triangle[i][1]);
    };
    return 0;
}
```

Program to convert a lowercase character string to uppercase.

```
#include <stdio.h>
int main() {
    char string[100];
    printf("Enter a string: ");
    scanf("%[^\n]", string);
    int i;
    for (i = 0; string[i] != '\0'; i++) {
        if (string[i] >= 'a' && string[i] <= 'z') {
            string[i] = string[i] - 32;
        }
    }
    printf("The string in uppercase is: %s", string);
    return 0;
}
```

**9. Explain the array of structures and write a program to accept records of 25 persons which has name, age and address and also display them.**

A structure is a user defined data type in C. A structure creates a data type that can be used to group items of possibly different types into a single type. A structure is convenient way of grouping several pieces of related information together.

If we have to store the information of large number of people then declaring new variables for each structure would be tedious, hence we can store structures in an array which is called array of structures.

Syntax for declaring array of structure in C is as below:

```
struct Structure_name {
    data_type_1 member_1_name;
    data_type_2 member_2_name;
    .....
    .....
    data_type_n member_n_name;
}
struct Structure_name array_name[number_of_items];
```

Program to accept records of 25 persons which has name, age and address and also display them.

```
#include <stdio.h>
int main() {
    struct Person {
        char fullName[30];
        int age;
        char address[20];
    };
    int personLength = 25, i;
    printf("Enter the details of %d persons.\n", personLength);
    struct Person persons[personLength];
    for (i = 0; i < personLength; i++) {
        printf("\nEnter the name of person %d: ", i + 1);
        scanf(" %[^\\n]s", persons[i].fullName);
        printf("Enter the age of person %d: ", i + 1);
        scanf(" %d", &persons[i].age);
        printf("Enter the address of person %d: ", i + 1);
        scanf(" %[^\\n]s", persons[i].address);
    }
    for (i = 0; i < personLength; i++) {
        printf("\n\nDetails of person %d:", i + 1);
        printf("\nName:\\t\\t%s", persons[i].fullName);
        printf("\nAge:\\t\\t%d", persons[i].age);
        printf("\nAddress:\\t%s", persons[i].address);
    }
    return 0;
}
```

## 10. Explain the pointer arithmetic with examples.

Pointers in C language is a variable that stores/points the address of another variable. A pointer in C is used to allocate memory dynamically i.e.: at runtime. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

Pointer syntax: data\_type \*var\_name;

Example:               int \*p;  
                          char \*p;

Where, \* is used to denote that “p” is pointer variable and not a normal variable.

Example of using pointer:

```
#include <stdio.h>
int main() {
    int *ptr, q;
    q = 50;
    ptr = &q;
    printf("%d", *ptr);
    return 0;
}
```

We can perform arithmetic operations on the pointers like addition, subtraction, etc. However, as we know that pointer contains the address, the result of an arithmetic operation performed on the pointer will also be a pointer if the other operand is of type integer. In pointer-from-pointer subtraction, the result will be an integer value. Following arithmetic operations are possible on the pointer in C language:

- Increment
- Decrement
- Addition
- Subtraction
- Comparison

Let's see the example of incrementing pointer variable on 64-bit architecture.

```
#include <stdio.h>
int main() {
    int number = 50;
    int *p;      // pointer to int
    p = &number; // stores the address of number variable
    printf("Address of p variable is %p \n", p);
    p = p + 1;
    printf("After increment: Address of p variable is %p \n", p);
    return 0;
}
```

We can add a value to the pointer variable. The formula of adding value to pointer is given below:

$$\text{new\_address} = \text{current\_address} + (\text{number} * \text{sizeof}(\text{data\_type}));$$

Let's see the example of adding value to pointer variable on 64-bit architecture.

```
#include <stdio.h>
int main() {
    int number = 50;
    int *p;      // pointer to int
    p = &number; // stores the address of number variable
    printf("Address of p variable is %p \n", p);
    p = p + 3;    // adding 3 to pointer variable
    printf("After adding 3: Address of p variable is %p \n", p);
    return 0;
}
```



## 11. Differentiate between break and exit statement with example.

break	exit()
It is a keyword.	It is a pre-defined function.
It doesn't require any header file as it is pre-defined in stdio.h header file in C.	It requires header file stdlib.h for few C compilers.
It terminates the loop.	It terminates the program.
It is often used only within the loop and switch case statement.	It is often used anywhere within the program.
It cannot be used as a variable name as it is a reserved word in the C language.	It is not a reserved word so; it is often used as a variable name.
In a C program, more than one break statement can be executed.	In a C program, just one exit function will be executed.

Example program of break statement:

```
#include <stdio.h>
int main() {
    int a = 10;
    while (a < 20) {
        printf("value of a: %d\n", a);
        a++;
        if (a > 15) {
            break;
        }
    }
    printf("The break statement executed when the value became %d\n", a);
    return 0;
}
```

In this program, the loop is said to run 20 times but due to the break statement at  $a > 15$ , the loop terminates at 15<sup>th</sup> iteration.

Example program for exit() function:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    for (int i = 1; i < 5; i++) {
        if (i == 3) exit(0);
        printf("i = %d \t", i);
    }
    printf("\n");
    for (int j = 9; j > 0; j--) {
        if (j == 5) printf("j = %d \t", j);
    }
    return 0;
}
```

Here, as soon as i becomes 3 in first for loop the program terminates.

## 12. Write short notes on:

### a) Delimiters:

(Already done in MODEL SET 1 Q12)

### b) Graphics function:

(Already done in MODEL SET 1 Q11)

## MODEL SET 3

**1. What is dynamic memory allocation? List out possible functions used for DMA. Write a program in C to read and print the N student details using structure and dynamic memory allocation.**

(First part of the question already done in MODEL SET 1 Q12)

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    struct Student {
        char fullName[30];
        int rollNumber;
        char address[20];
        char section;
    };
    int studentLength, i;
    struct Student *students;
    printf("Enter the number of students: ");
    scanf("%d", &studentLength);
    students = (struct Student *)malloc(studentLength *
                                         sizeof(struct Student));
    for (i = 0; i < studentLength; i++) {
        printf("\nEnter the name of student %d: ", i + 1);
        scanf(" %[^\\n]s", students[i].fullName);
        printf("Enter the roll number of student %d: ", i + 1);
        scanf(" %d", &students[i].rollNumber);
        printf("Enter the address of student %d: ", i + 1);
        scanf(" %[^\\n]s", students[i].address);
        printf("Enter the section of student %d: ", i + 1);
        scanf(" %c", &students[i].section);
    }
    for (i = 0; i < studentLength; i++) {
        printf("\nDetails of Student %d:", i + 1);
        printf("\nName:\\t\\t%s", students[i].fullName);
        printf("\nRoll number:\\t%d", students[i].rollNumber);
        printf("\nAddress:\\t%s", students[i].address);
        printf("\nSection:\\t%c", students[i].section);
    }
    return 0;
}
```

**2. What is array? What are the disadvantages of array? Write a program to enter two 3x3 matrices and calculate the product of given matrices.**

(First and second part already done in MODEL SET 1 Q1)

(Program done in MODEL SET 1 Q8)

**3. What is file handling in C? What are the advantages of file handling? Create a text file and enter "to write a good program is very time consuming job." Create another text which contains reverse of above text.**

File handling in C refers to the task of storing data in the form of input or output produced by running C programs in data files, namely, a text file or a binary file for future reference and analysis. It involves various operations like creating a file, opening a file, reading a file or manipulating data inside a file.

The advantages of file handling are as follows:

- **Reusability:**  
It helps in preserving the data or information generated after running the program.
- **Large storage capacity:**  
Using files, you need not worry about the problem of storing data in bulk.
- **Saves time:**  
There are certain programs that require a lot of input from the user. You can easily access any part of the code with the help of certain commands.
- **Portability:**  
You can easily transfer the contents of a file from one computer system to another without having to worry about the loss of data.

Program:

```
#include <stdio.h>
int main() {
    FILE *fp1, *fp2;
    fp1 = fopen("file1.txt", "w");
    fp2 = fopen("file2.txt", "w");
    if (!fp1 || !fp2) {
        printf("Writing files failed.");
        return -1;
    }
    char string[] = "to write a good program is very time consuming job.";
    fputs(string, fp1);
    int length = 0, i;
    for (i = 0; string[i] != '\0'; i++) {
        length++;
    }
    for (i = length - 1; i >= 0; i--) {
        fputc(string[i], fp2);
    }
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

#### 4. What is algorithm? How does it differ from flowchart?

Algorithm:

An algorithm is a finite sequence of instructions for solving a stated problem. A stated problem is a well-defined task that has to be performed and must be solvable by an algorithm. The algorithm must eventually terminate, producing the required result.

Properties of a good algorithm:

- Input: Several quantities are provided to an algorithm initially before the algorithm begins. These quantities are inputs which are processed by the algorithm.
- Definiteness: The processing rules specified in the algorithms must be unambiguous and lead to specific action.
- Effectiveness: Each instruction must be sufficiently basic such that it can, in principle, be carried out in a finite time by person with paper and pencil.
- Finiteness: The total time to carry out all the steps in an algorithm must be finite.
- Output: An algorithm must have output.
- Correctness: Correct set of output must be produced from the set of inputs. For the same input, it must always produce the same output.

Example: Algorithm to find whether a number is odd or even:

Step 1: Start

Step 2: Input a number num.

Step 3: If num mod 2 is 0:

Print "The number is even."

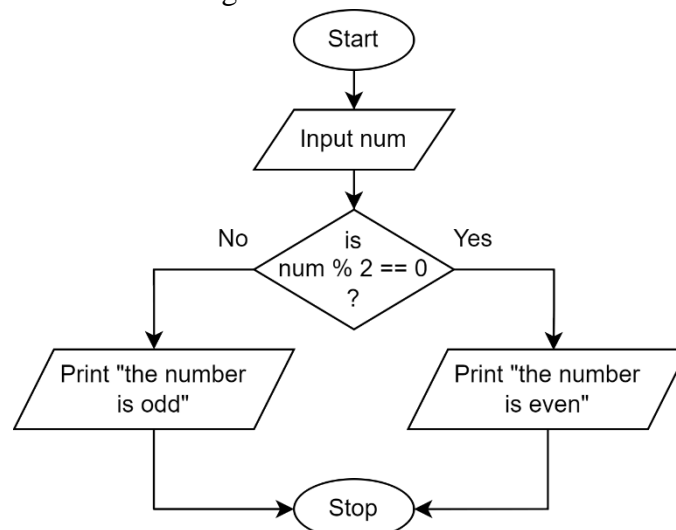
Step 4: Else:

Print "The number is odd."

Step 5: Stop

Flowchart: A flowchart is a common type of chart that represents an algorithm or a computer program showing the steps as boxes of various kinds, and their order by connecting these with arrows. Flowcharts are used in analyzing, designing, documenting, or managing a process or program in various fields.

Example: Flowchart to find whether given entered number is odd or even:



Key Differences between algorithm and flowchart are listed below:

- a. An algorithm involves a combination of sequential steps to interpret the logic of the solution. In contrast, a flowchart is the pictorial illustration of the algorithm.
- b. A flow chart is more understandable as compared to the algorithm.
- c. The algorithm is written in a language that can be perceived by humans. On the other hand, the flowchart is made up using different shapes and symbols.
- d. There are no stringent rules implemented in the algorithms while the flowchart is abode by predefined rules.
- e. Errors and bugs are easily detected in the algorithm as compared to the flowcharts.
- f. Flow charts are simple to create. On the contrary, the construction of the algorithm is complex.

## 5. What is type conversion? Discuss type caseing with suitable example.

Type conversion is converting one type of data to another type. It is also known as Type Casting. Type conversion in C can be classified into the following two types:

- a. Implicit Type Conversion:

When the type conversion is performed automatically by the compiler without programmer's intervention, such type of conversion is known as implicit type conversion or type promotion.

```
int x;
for (x = 97; x <= 122; x++) {
    printf("%c", x);      /* Implicit casting from int to
                           char thanks to %c */
}
```

- b. Explicit Type Conversion:

The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion. The explicit type conversion is also known as type casting.

Type casting in C is done in the following form:

```
(data_type) expression;
```

Where, data\_type is any valid C data type, and expression may be constant, variable or expression. For example,

```
int x = 7, y = 5;
float z;
z = (float) x / (float) y; /* Here the value of z is 1.4 */
```

**6. Write a program for the interest charged in installments for following case. A cassette player costs Rs. 2000. A shopkeeper sells it for Rs. 100 down payment and Rs. 100 for 21 more months. What is the monthly interest charged?**

Program:

```
#include <stdio.h>
int main() {
    float costPrice, totalPrice, totalInterest, monthlyInterest;
    costPrice = 2000;
    totalPrice = 100 + (100 * 21);
    totalInterest = (totalPrice - costPrice) / costPrice * 100;
    monthlyInterest = totalInterest / 22; // payment is for 22 months
    printf("The monthly interest charged is %.2f%%", monthlyInterest);
    return 0;
}
```

Output:

The monthly interest charged is 0.45%

**7. Write a program which will read a line and delete from it all occurrences of the word "that".**

```
#include <stdio.h>
#include <string.h>
int main() {
    char line[200];
    printf("Enter a string: ");
    scanf("%[^\\n]s", line);
    char removeStr[] = "that";
    int removeStrLen = strlen(removeStr), i;
    char *found = strstr(line, removeStr);
    while (found) {
        int pos = found - &line[0];
        if (line[pos - 1] == ' ') {
            pos--;
            removeStrLen++;
        } else if (line[pos + removeStrLen] == ' ') {
            removeStrLen++;
        }
        for (i = pos + removeStrLen; line[i] != '\\0'; i++) {
            line[i - removeStrLen] = line[i];
        }
        line[i - removeStrLen] = '\\0';
        found = strstr(line, removeStr);
    }
    printf("The string without \"that\" is:\\n%s", line);
    return 0;
}
```

**8. Define a structure of employee having data members name, address, age, and salary. Take data for n employee in an array dynamically and find the average salary.**

(Already done in MODEL SET 1 Q6)

**9. Determine which of the following are valid identifiers? If invalid, explain why?**

(Question incomplete)

**10. Find the errors in the following program and explain it. Write the correct program.**

Wrong Program:

```
1  #include <math.h>
2  #include <stdio.h>
3- main()
4  {
5-   float p, r, n;
6   printf("Please enter a value for the principal(P):");
7-   scanf(%f, &p);
8-   printf("Please enter a value for the Interest rate(R));
9   scanf(" %f", &r);
10  printf("Please enter a value for number of years (n):");
11-   scanf("%f, n);
12-   f = p * pow + r / 100), n);
13  printf("\n The final value (F) is; %.2f\n", f);
```

Errors:

- Return type of main() function missing in line 3.
- Double quotes missing in the first argument of scanf() function in line 7.
- Ending double quote missing in the argument of printf() function in line 8.
- Ending double quote missing in the first argument of scanf() function in line 11.
- Incorrect formula in line 12.
- Variable “F” used in line 12 is not defined previously.
- Return statement missing in main() function.
- Closing curly bracket of main() function missing.

Correct Program:

```
1  #include <math.h>
2  #include <stdio.h>
3+ int main()
4  {
5+   float p, r, n, f;
6   printf("Please enter a value for the principal(P):");
7+   scanf("%f", &p);
8+   printf("Please enter a value for the Interest rate(R)");
9   scanf(" %f", &r);
10  printf("Please enter a value for number of years (n):");
11+   scanf("%f", n);
12+   f = p * pow(1 + (r / 100), n);
13  printf("\n The final value (F) is; %.2f\n", f);
14+   return 0;
15+ }
```

**11. Write and test the following power() function that returns x raised to the power n, where n can be any integer: double power(double x, int p)**

```
#include <math.h>
#include <stdio.h>
double power(double x, int p) {
    int i;
    double result = 1.0;
    for (i = 0; i < p; i++) {
        result *= x;
    }
    return result;
}
int main() {
    double x = 2.5;
    int n = 2;
    printf("%lf raised to the power %d is %lf", x, n, power(x, n));
    return 0;
}
```

**12. Explain the use of graphical function. Make a program that contains basic graphic functions.**

(Already done in MODEL SET 1 Q11)

## MODEL SET 4

**1. What is operator? Describe various operators used in C with suitable examples.**

An operator is a symbol which helps the user to command the computer to do a certain mathematical or logical manipulation. Operators are used in C language program to operate on data and variables.

C has a rich set of operators which can be classified as:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Unary Operators
6. Conditional Operators
7. Bitwise Operators
8. Special Operators

1. Arithmetic Operators:

All the basic arithmetic operations can be carried out in C. All the operators have almost the same meaning as in other languages. Both unary and binary operations are available in C language.



Operator	Meaning
+	Addition or Unary Plus
-	Subtraction or Unary Minus
*	Multiplication
/	Division
%	Modulus Operator

Examples of arithmetic operators are:

$x + y$ ,  $x - y$ ,  $-x + y$ ,  $a * b + c$ ,  $-a * b$  etc.

a) Integer Arithmetic:

When an arithmetic operation is performed on two whole numbers or integers than such an operation is called as integer arithmetic. It always gives an integer as the result. Let  $x = 27$  and  $y = 5$  be 2 integer numbers. Then the integer operation leads to the following results.

$$x + y = 32$$

$$x - y = 22$$

$$x * y = 115$$

$$x \% y = 2$$

$$x / y = 5$$

In integer division the fractional part is truncated.

b) Floating point arithmetic:

When an arithmetic operation is performed on two real numbers or fraction numbers such an operation is called floating point arithmetic.

Let  $x = 14.0$  and  $y = 4.0$  then

$$x + y = 18.0, x - y = 10.0, x * y = 56.0, x / y = 3.50$$

c) Mixed mode arithmetic:

When one of the operands is real and other is an integer and if the arithmetic operation is carried out on these 2 operands, then it is called as mixed mode arithmetic. If any one operand is of real type, then the result will always be real  
 $15 / 10.0 = 1.5$

## 2. Relational Operators

Often it is required to compare the relationship between operands and bring out a decision and program accordingly. This is when the relational operator comes into picture. C supports the following relational operators.

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

A simple relational expression contains only one relational operator and takes the following form.

exp1 relational\_operator exp2

Where exp1 and exp2 are expressions, which may be simple constants, variables, or combination of them.

6.5 <= 25 TRUE, -65 > 0 FALSE, 10 < 7 + 5 TRUE

Relational expressions are used in decision making statements of C language such as if, while and for statements to decide the course of action of a running program.

### 3. Logical Operators:

C has the following logical operators; they compare or evaluate logical and relational expressions.

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

#### a) Logical AND (&&)

This operator is used to evaluate 2 conditions or expressions with relational operators simultaneously. If both the expressions to the left and to the right of the logical operator is true then the whole compound expression is true.

Example:  $a > b \ \&\& \ x == 10$

The expression to the left is  $a > b$  and that on the right is  $x == 10$  the whole expression is true only if both expressions are true i.e., if “a” is greater than “b” and “x” is equal to 10.

#### b) Logical OR (||)

The logical OR is used to combine 2 expressions or the condition evaluates to true if any one of the 2 expressions is true.

Example:  $a < m \ || \ a < n$

The expression evaluates to true if any one of them is true or if both of them are true. It evaluates to true if “a” is less than either “m” or “n” and when “a” is less than both “m” and “n”.

#### c) Logical NOT (!)

The logical not operator takes single expression and evaluates to true if the expression is false and evaluates to false if the expression is true. In other words it just reverses the value of the expression.

Example:  $! (x \geq y)$

The NOT expression evaluates to true only if the value of “x” is neither greater than or equal to “y”.

#### 4. Assignment Operators:

The Assignment Operator evaluates an expression on the right of the expression and substitutes it to the value or variable on the left of the expression.

Example  $x = a + b$

Here the value of  $a + b$  is evaluated and substituted to the variable  $x$ . In addition, C has a set of shorthand assignment operators of the form.

$\text{var oper} = \text{exp};$

Here  $\text{var}$  is a variable,  $\text{exp}$  is an expression and  $\text{oper}$  is a C binary arithmetic operator.

The operator  $\text{oper} =$  is known as shorthand assignment operator.

Example  $x += 1$  is same as  $x = x + 1$

The commonly used shorthand assignment operators are as follows

Shorthand assignment operators

Statement with simple assignment operator	Statement with shorthand operator
---	-----------------------------------

$a = a + 1$	$a += 1$
-------------	----------

$a = a - 1$	$a -= 1$
-------------	----------

$a = a * (n+1)$	$a *= (n+1)$
-----------------	--------------

$a = a / (n+1)$	$a /= (n+1)$
-----------------	--------------

$a = a \% b$	$a \% = b$
--------------	------------

#### 5. Unary Operators:

The increment and decrement operators are one of the unary operators which are very useful in C language. They are extensively used in for and while loops. The syntax of the operators is given below:

i.  $++\text{variable name}$

ii.  $\text{variable name}++$

iii.  $--\text{variable name}$

iv.  $\text{variable name}--$

The increment operator  $++$  adds the value 1 to the current value of operand and the decrement operator  $--$  subtracts the value 1 from the current value of operand.  $++\text{variable name}$  and  $\text{variable name}++$  mean the same thing when they form statements independently, they behave differently when they are used in expression on the right-hand side of an assignment statement. Consider the following:

$m = 5;$

$y = ++m;$  (prefix)

In this case the value of  $y$  and  $m$  would be 6

Suppose if we rewrite the above statement as

$m = 5;$

$y = m++;$  (post fix)

#### 6. Conditional or Ternary Operator:

The conditional operator consists of 2 symbols the question mark (?) and the colon (:). The syntax for a ternary operator is as follows:

`exp1 ? exp2 : exp3`

The ternary operator works as follows:

`exp1` is evaluated first. If the expression is true then `exp2` is evaluated & its value becomes the value of the expression. If `exp1` is false, `exp3` is evaluated and its value becomes the value of the expression.

#### 7. Bitwise Operators:

C has a distinction of supporting special operators known as bitwise operators for manipulation data at bit level. A bitwise operator operates on each bit of data. Those operators are used for testing, complementing or shifting bits to the right or left. Bitwise operators may not be applied to a float or double.

Operator	Meaning
<code>&amp;</code>	Bitwise AND
<code> </code>	Bitwise OR
<code>^</code>	Bitwise Exclusive
<code>&lt;&lt;</code>	Shift left
<code>&gt;&gt;</code>	Shift right

#### 8. Special operators:

Special operator includes: `&` (reference operator that is used to get the memory address of the variable), `*` (dereference operator which is used to get the value of the variable stored in the given pointer) and `sizeof()` operator (which returns the size of the datatype or the variable).

Example Program to demonstrate operators in C:

```
#include <stdio.h>
const float PI = 3.1415926;
int main() {
    float radius, volume;
    printf("Enter the radius: ");
    scanf("%f", &radius);
    volume = 4 / 3.0 * PI * radius * radius * radius;
    printf("The volume of the sphere is: %f", volume);
    return 0;
}
```

## 2. What is statement? Describe types of statements used in C with suitable example.

Statement is an executable part of the program it will do some action. In general, all arithmetic actions and logical actions are falls under Statements Categories.

there are few Statements categories

- Expression Statements.
- Compound Statements.
- Selection Statements.
- Iterative Statements.
- Jump Statements.

Expression Statements:

It is combination of variables, constants, operators, function calls and followed by a semicolon. Expression can be any operation like Arithmetic operation or Logical Operation.

Few Examples for expression Statements

- `x = y + 10;`
- `20 > 90;`
- `a ? b : c ;`
- `a = 10 + 20 * 30;`
- `;` (This is NULL Statement).

Compound Statement:

Compound statement is combination of several expression statements. Compound Statement is Enclosed within the Braces { }.

Compound statement is also called as Block Statement.

There is no need of any semicolon at the end of Compound Statement.

Example for Compound Statement

```
{
    int a = 10, b = 20, c;
    c = a + b;
    printf("value of c is: %d\n", c);
}
```

Selection Statements:

Selection Statements are used in decisions making situations. Few examples of selection statements are:

- if statement
- if-else statement
- Nested if statement
- else-if Ladder
- switch statement

if-else statement:

A conditional statement is written using an if-else statement. If the specified condition is true, the statements will be executed. Otherwise, the else block statement will be executed.

Syntax of if-else statement

```
if (condition) {  
    -----  
    -----  
} else {  
    -----  
    -----  
}
```

Example program:

```
#include <stdio.h>  
void main() {  
    int a = 12;  
    if (a % 2 == 0) {  
        printf("Number is Even");  
    } else {  
        printf("Number is Odd");  
    }  
}
```

Nested if statement

Nested if is an if statement within an if statement.

Syntax of nested if statement

```
if (outer-condition) {  
    if (inner-condition) {  
        -----  
        -----  
    }  
}
```

Example program:

```
#include <stdio.h>  
void main()  
{  
    int a = 10;  
    if(a % 2 == 0)  
    {  
        if(a % 5 == 0)  
        {  
            printf("Number is Divisible by 5");  
        }  
        printf("Number is Divisible by 2");  
    }  
}
```

else-if ladder

Syntax of nested else-if ladder statement

```
if (condition 1) {  
    -----  
} else if (condition 2) {  
    -----  
} else if (condition 3) {  
    -----  
} else {  
    -----  
}
```

Example program:

```
#include <stdio.h>  
void main() {  
    int a = 15;  
    if (a % 2 == 0) {  
        printf("Number is Divisible by 2");  
    } else if (a % 5 == 0) {  
        printf("Number is Divisible by 5");  
    } else if (a % 7 == 0) {  
        printf("Number is Divisible by 7");  
    } else {  
        printf("Number is not Divisible by 2, 5 or 7");  
    }  
}
```

switch statement in C language

The switch statement is a multiway decision maker that compares an expression to one of a set of constant values and braces them accordingly. Because several if-statements are time expensive, a switch case statement is a better solution.

Syntax of switch statement

```
switch (expression)  
{  
    case constant-1:  
        statement;  
        break;  
    case constant-2:  
        statement;  
        break;  
    -----  
    -----  
    case constant-n:  
        statement;  
        break;  
    default:  
        statement;  
}
```

Example program:

```
#include <stdio.h>
#include <conio.h>
void main() {
    int a, b, c, choice;
    while (choice != 3) {
        printf("\n 1. Press 1 for Addition");
        printf("\n 2. Press 2 for Subtraction");
        printf("\n Enter your choice: ");
        scanf("%d",&choice);
        switch (choice) {
            case 1:
                printf("Enter two numbers: ");
                scanf("%d%d",&a,&b);
                c = a + b;
                printf("%d",c);
                break;;
            case 2:
                printf("Enter two numbers: ");
                scanf("%d%d",&a,&b);
                c = a - b;
                printf("%d",c);
                break;;
            default:
                printf("You have passed a wrong key.");
                printf("\n Press any key to continue");
        }
    }
}
```

Iterative Statements:

These are also called loops. If we want to execute a part of program many times we use loops. List of basic loops in C language are:

- while loop
- do-while loop
- for loop

while loop:

When we aren't sure if the loop will be run or the number of times the loop will run, we use a while loop. The condition of a while loop is verified first, and then the statement is executed.

Syntax of while Loop

```
while (condition) {
    statement 1;
    statement 2;
    ...
}
```



Example program:

```
#include <stdio.h>
int main() {
    int i;
    i = 1;
    while (i <= 10) {
        printf("%d\t", i);
        i++;
    }
    return 0;
}
```

Output:

1 2 3 4 5 6 7 8 9 10

do...while loop:

When we are certain about the condition, we use a do-while loop. It enters the loop at least once before determining if the provided condition is true or false. It is run at least once to see if any of the conditions are true or false.

Syntax of do...while loop

```
Do {
    statement 1;
    statement 2;
    ...
} while (condition);
```

Example program:

```
#include <stdio.h>
int main() {
    int i;
    i = 1;
    do {
        printf("%d\t", i);
        i++;
    } while (i <= 10);
    return 0;
}
```

Output:

1 2 3 4 5 6 7 8 9 10

for loop in C:

In a for loop, there are three expressions. The first expression sets the index value, the second checks if the loop should be iterated, and the third sets the index value for future iteration.

Syntax of for loop:

```
for (Initialization; condition; Increment/Decrement) {
    statement 1;
    statement 2;
    ...
}
```

Example program:

```
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i <=10; i++) {
        printf("%d\t", i);
    }
    return 0;
}
```

Output:

1 2 3 4 5 6 7 8 9 10

Jump Statements:

These are Unconditional statements Jump statements are useful for Transfer the Control one part of program to other part of Program there are few Jump Statements in C

- break.
- continue.
- goto.
- return.

break Statement:

Loop: The break statement is used to stop the loop statement from running.

Switch: When using the break statement in a switch-case, it must be used in each instance. If this is not the case, control will be passed at a next case.

The break statement will end control of the innermost loop in a nested loop, and execution will begin on the line after the innermost loop block.

Example of break statement

```
#include <stdio.h>
int main() {
    int i;
    i = 1;
    while (i <= 10) {
        printf("%d\t", i);
        i++;
        if (i == 5) {
            break; // Terminates the loop using break statement.
        }
    }
    return 0;
}
```

Output:

1 2 3 4

continue Statement:

The continue statement skips the current iteration but keeps the loop running. It is used just for a condition, allowing us to bypass code for that condition.

Example of continue Statement

```
#include <stdio.h>
int main() {
    int i;
    i = 1;
    while(i <= 10) {
        if (i == 5) {
            i++;
            continue; // Skip the current iteration based on condition.
        }
        printf("%d\t", i);
        i++;
    }
    return 0;
}
```

Output:

1 2 3 4 6 7 8 9 10

goto statement:

By transferring control to another part of the program, the goto statement can modify the program's execution sequence:

Syntax of goto statement:

```
label:
.....
goto label;
```

Example program:

```
#include <stdio.h>
int main() {
    start: printf("Hello! World!");
    goto start;
    return 0;
}
```

return statement:

return statement returns the value from the function to the place where it was called from and terminates the function giving control back to the block where the function was called from.

Syntax of return statement:

```
int func() {
    statement;
    if (condition) {
        return;
    }
}
```

Example program:

```
#include <stdio.h>
void checkEven(int num) {
    if (num % 2 == 0) {
        printf("Number is even.");
        return;
    }
    printf("Number is odd.");
    return;
}
int main() {
    checkEven(5);
    return 0;
}
```

**3. What is binary file? How is it different from text file? Create a text file and enter “I am the student of CSIT first semester”. Create another text which contains only smaller letters of above text.**

A binary file is the one in which data is stored in binary format instead of ASCII characters. Data is stored in the file in the same way as it is stored in the main memory for processing. It is normally used for storing numeric information (int, float, double). Normally a binary file can be created only from within a program and its contents can be read only by a program.

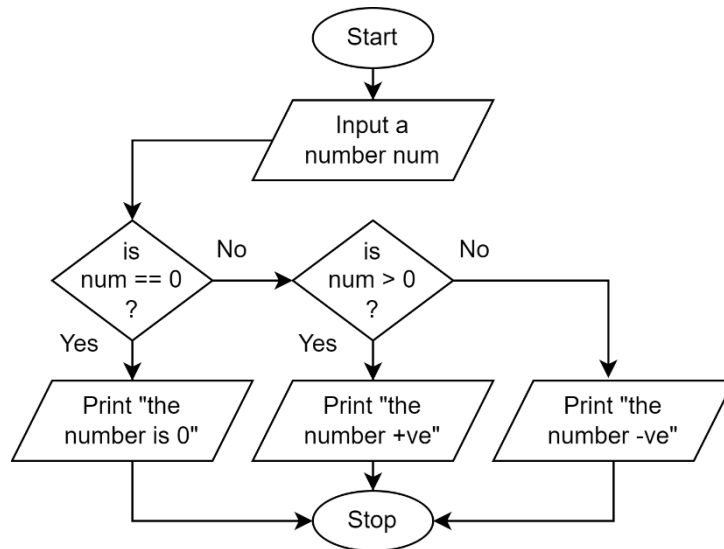
(Differences between binary and text files already done in MODEL SET 1 Q3)

Program:

```
#include <stdio.h>
int main() {
    FILE *fp1, *fp2;
    fp1 = fopen("file1.txt", "w");
    fp2 = fopen("file2.txt", "w");
    if (!fp1 || !fp2) {
        printf("Writing files failed.");
        return -1;
    }
    char string[] = "I am the student of CSIT first semester";
    fputs(string, fp1);
    int length = 0, i;
    for (i = 0; string[i] != '\0'; i++) {
        length++;
    }
    for (i = 0; i < length; i++) {
        if (string[i] >= 'a' && string[i] <= 'z' || string[i] == ' ') {
            fputc(string[i], fp2);
        }
    }
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

**4. Draw a flow chart and write an algorithm to find out whether a given number is zero, +ve or -ve.**

Flowchart:



Algorithm:

Step 1: Start.

Step 2: Input a number num.

Step 3: if num == 0:

Print "the number is 0"

Step 4: else if num > 0:

Print "the number is +ve"

Step 5: else:

Print "the number is -ve"

Step 6: Stop.

**5. Differentiate between while and do while loop with example.**

while loop	do while loop
Condition is checked first then the statement is executed.	Statement is executed first, and thereafter condition is checked.
If the condition is false to begin with, the statement never runs.	Statement will run once even if the condition is false to begin with
If there is a single statement, brackets are not required.	Brackets are always required.
Variable in condition is initialized before the execution of loop.	Variable may be initialized before or within the loop.
while loop is entry controlled loop.	do-while loop is exit controlled loop.
Syntax: while(condition) { statement(s); }	Syntax: do { statement(s); } while(condition);

Example of while loop:

```
#include <stdio.h>
int main() {
    int i = 5;
    while (i < 10) {
        printf("Hello! World\n");
        i++;
    }
    return 0;
}
```

Example of do while loop:

```
#include <stdio.h>
int main() {
    int i = 5;
    do {
        printf("GFG\n");
        i++;
    } while (i < 10);
    return 0;
}
```

## 6. What is the function of a pointer variable? Explain the declaring and initializing pointers with example.

A pointer variable in C language is a variable that stores/points the address of another variable. A pointer in C is used to pass the value to a function by reference, allocate memory dynamically, etc. The pointer variable might be belonging to any other data types such as int, float, char, double, short, etc.

A pointer can be declared by the following syntax:

```
int *p;
```

A pointer can be initialized as follows:

```
int a = 10;
int *ptr = &a;
```

## 7. Differentiate between call by value and call by reference with example.

Call by value	Call by reference
In call by value, a copy of actual arguments is passed to formal arguments of the called function and any change made to the formal argument in the called function have no effect on the value of actual arguments in the calling function.	In call by reference, the location (address) of the actual arguments is passed to formal arguments of the called function. This means by accessing the address of actual arguments we can alter them within from the call function.
In call by value, actual arguments will remain safe, they cannot be modified accidentally.	In call by reference, alteration to actual arguments is possible within from called function; Therefore, the code must handle arguments carefully else you get unexpected results.

Call by value example in C:

Program:

```
#include <stdio.h>
void interchange(int num1, int num2) {
    int temp;
    temp = num1;
    num1 = num2;
    num2 = temp;
}
int main() {
    int a = 100;
    int b = 200;
    printf("Before swapping: (a, b) = (%d, %d)\n", a, b);
    interchange(a, b);
    printf("After swapping: (a, b) = (%d, %d)\n", a, b);
    return 0;
}
```

Output:

Before swapping: (a, b) = (100, 200)

After swapping: (a, b) = (100, 200)

Call by reference example in C:

Program:

```
#include <stdio.h>
void interchange(int *num1, int *num2) {
    int temp;
    temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
int main() {
    int a = 100;
    int b = 200;
    printf("Before swapping: (a, b) = (%d, %d)\n", a, b);
    interchange(&a, &b);
    printf("After swapping: (a, b) = (%d, %d)\n", a, b);
    return 0;
}
```

Output:

Before swapping: (a, b) = (100, 200)

After swapping: (a, b) = (200, 100)

## 8. What is structure? How is it different from array and union? Discuss

(What is structure? How is it different from union? Done in MODEL SET 2 Q3)

Structures differ from arrays in the following ways:

Array	Structure
An array is a collection of variables of same data type.	An structure is a collection of variables of difference data type.
Syntax: data_type array_name[size];	Syntax: struct Structure_name { data_type_1 member_1_name; data_type_2 member_2_name; ..... data_type_n member_n_name; }
Array elements are stored in contiguous memory location.	Structure elements may not be stored in a contiguous memory location.
Array elements are accessed by their index number.	Structure elements are accessed by their name.
Array declaration and element accessing operator is '['] (square bracket).	Structure element accessing operator is '.' (dot operator).
Array name points to the first element of that array. So, array name is a pointer.	Structure name is not a pointer and doesn't point to the first element of the structure.
Objects (instances) of an array cannot be created.	Structure objects (instances or structure variables) can be created.
Every element of an array is of same size.	Elements of structures will have the dize of their respected datatype.

## 9. Determine which of the following are valid identifiers? If invalid, explain why?

(????????????????????????????????)

## 10. Write a program in C to sort the following data items using Bubble sort:

a[ ] = {25, 57, 48, 37, 12, 92, 86, 33}

Program:

```
#include <stdio.h>
int main() {
    int i, j, temp, num;
    int a[] = {25, 57, 48, 37, 12, 92, 86, 33};
    num = sizeof(a) / sizeof(int);
    for (i = 0; i < num - 1; i++) {
        for (j = 0; j < num - i - 1; j++) {
            if (a[j] > a[j + 1]) {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}
```



```

printf("Array elements after bubble sort:\n");
for (i = 0; i < num; i++) {
    printf("%d ", a[i]);
}
return 0;
}

```

Output:

Array elements after bubble sort:  
12 25 33 37 48 57 86 92

## 11. What do you mean by nesting of structure? Explain with suitable example.

Nested Structures:

A nested structure in C is a structure within structure. One structure can be declared inside another structure in the same way structure members are declared inside a structure. The members of nested structures can be accessed by chaining the dot operator.

Use of nested structures:

Consider there are two structures Employee (dependent on structure) and another structure called Organisation (Outer structure). The structure Organisation has the data members like organisation\_name, organisation\_number. The Employee structure is nested inside the structure Organisation and it has the data members like employee\_id, name, salary.

The structure can be nested in the following different ways:

1. By separate nested structure
2. By embedded nested structure

1. By separate nested structure:

In this method, the two structures are created, but the dependent structure (Employee) should be used inside the main structure (Organisation) as a member.

```

struct Employee {
    int employee_id;
    char name[20];
    int salary;
};

struct Organisation {
    char organisation_name[20];
    char org_number[20];

    // Dependent structure is used
    // as a member inside the main
    // structure for implementing
    // nested structure
    struct Employee emp;
};

```

2. By Embedded nested structure:

Using this method, allows to declare structure inside a structure and it requires fewer lines of code.

Note: Whenever an embedded nested structure is created, the variable declaration is compulsory at the end of the inner structure, which acts as a member of the outer structure.

```
// Declaration of the main
// structure
struct Organisation {
    char organisation_name[20];
    char org_number[20];

    // Declaration of the dependent
    // structure
    struct Employee {
        int employee_id;
        char name[20];
        int salary;

        // variable is created which acts
        // as member to Organisation structure.
    } emp;
};
```

Example program using nested structure:

```
#include <stdio.h>
#include <string.h>
struct Organisation {
    char organisation_name[20];
    char org_number[20];
    struct Employee {
        int employee_id;
        char name[20];
        int salary;
    } emp;
};
int main() {
    struct Organisation chainForChange;
    strcpy(chainForChange.organisation_name, "Chain For Change");
    strcpy(chainForChange.emp.name, "Joon Shakya");
    printf("%s from %s is a programmer.", chainForChange.emp.name,
        chainForChange.organisation_name);
    return 0;
}
```

**12. Explain the use of graphical function. Make a program that displays a rectangle by using graphic handling function.**

(First part of the question already done in MODEL SET 1 Q11)

Program:

```
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
int main() {
    int gDriver = DETECT, gMode;
    initgraph(&gDriver, &gMode, (char*)"C:\\TURBOC3\\BGI");
    rectangle(100, 100, 240, 200);
    getch();
    closegraph();
    return 0;
}
```

## MODEL SET 5

**1. What is storage class in C? Describe various storage classes used in C with suitable example.**

Storage class defines the scope (visibility) and lifetime of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program:

- auto
- register
- static
- extern

auto Storage Class:

The auto storage class is the default storage class for all local variables.

```
{
    int mount;
    auto int month;
}
```

The example above defines two variables within the same storage class. 'auto' can only be used within functions, i.e., local variables.

register Storage Class:

The register storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

```

{
    register int miles;
}

```

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

#### static Storage Class:

The static storage class instructs the compiler to keep a local variable in existence during the lifetime of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

In C programming, when static is used on a global variable, it causes only one copy of that member to be shared by all the objects of its class.

```

#include <stdio.h>
/* function declaration */
void func(void);
static int count = 5; /* global variable */
main() {
    while(count-->0) {
        func();
    }
    return 0;
}
/* function definition */
void func(void) {
    static int i = 5; /* local static variable */
    i++;
    printf("i is %d and count is %d\n", i, count);
}

```

When the above code is compiled and executed, it produces the following result –

```

i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0

```

#### extern Storage Class:

The extern storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function, which will also be used in other files, then `extern` will be used in another file to provide the reference of defined variable or function. Just for understanding, `extern` is used to declare a global variable or function in another file.

The `extern` modifier is most used when there are two or more files sharing the same global variables or functions as explained below.

First File: `main.c`

```
#include <stdio.h>
int count;
extern void write_extern();
main() {
    count = 5;
    write_extern();
}
```

Second File: `support.c`

```
#include <stdio.h>
extern int count;
void write_extern(void) {
    printf("count is %d\n", count);
}
```

Here, `extern` is being used to declare `count` in the second file, whereas it has its definition in the first file, `main.c`. Now, compile these two files as follows –

```
$gcc main.c support.c
```

It will produce the executable program `a.out`. When this program is executed, it produces the following result:

```
count is 5
```

## **2. What is macro? How it is differ from function? 'Explain with suitable example.**

A macro is a segment of code which is replaced by the value of macro. Macro is defined by `#define` directive. There are two types of macros:

- Object-like Macros
- Function-like Macros

Object-like Macros:

The object-like macro is an identifier that is replaced by value. It is widely used to represent numeric constants. For example:

```
#define PI 3.14
```

Here, `PI` is the macro name which will be replaced by the value `3.14`.

Function-like Macros

The function-like macro looks like function call. For example:

```
#define MIN(a,b) ((a)<(b)?(a):(b))
```

Here, `MIN` is the macro name.

Macros differ from function as follows:

Macro	Function
Macros are preprocessed.	Functions are compiled.
No type-checking is done in macro.	Type-checking is done in function.
Using macro increases the code length.	Using function keeps the code length unaffected.
Use of macro can lead to side effects at later stages.	Functions do not lead to any side effects in any case.
Speed of execution using macro is faster.	Speed of execution using function is slower.
Before compilation, macro name is replaced by macro value.	During function call, transfer of control takes place.
Macros are useful when small code is repeated many times.	Functions are useful when large code is to be written.
Macro does not check any compile-time errors.	Function checks compile-time errors.

Example program using macros:

```
#include<stdio.h>
#define CUBE(b) (b) * (b) * (b)
int main()
{
    printf("%d", CUBE(1 + 2));
    return 0;
}
```

Example program using macros

```
#include <stdio.h>
int cube(int b) {
    return b * b * b;
}
int main() {
    printf("%d", cube(1 + 2));
    return 0;
}
```

**3. What is file pointer? How it is differ from text file? Write a program in C to read all numbers from the input file "values.dat" and stores the average of these numbers in an output file named as "average.res".**

A file pointer stores the current position of a read or write within a file. All operations within the file are made with reference to the pointer. The data type of this pointer is defined in stdio.h and is named FILE. It is different from text file as file pointer stores the current position of the read or write within a file whereas a text file stores the actual content.

Program:

```
#include <stdio.h>
int main() {
    FILE *ifp, *ofp;
    ifp = fopen("values.dat", "r");
    ofp = fopen("average.res", "w");
    if (!ifp || !ofp) {
        printf("File not found or in use.");
        return -1;
    }
    int sum = 0, num, length;
    for (length = 0; !feof(ifp); length++) {
        fscanf(ifp, "%d ", &num);
        sum += num;
    }
    fprintf(ofp, "%.3f", 1.0 * sum / length);
    fclose(ifp);
    fclose(ofp);
    return 0;
}
```

#### **4. What is identifier? Write naming conventions of defining identifiers in programming like C**

In C language, an identifier is a combination of alphanumeric characters, i.e. first begin with a letter of the alphabet or an underline, and the remaining are letter of an alphabet, any numeric digit, or the underline.

The naming convention for defining identifiers in C are:

- The case of alphabetic characters is significant. For example, using "TUTORIAL" for a variable is not the same as using "tutorial" and neither of them is the same as using "TutoRial" for a variable. All three refer to different variables.
- There is no rule on how long an identifier can be. We can run into problems in some compilers if an identifier is longer than 31 characters. This is different for the different compilers.
- A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
- The first letter of an identifier should be either a letter or an underscore.
- You cannot use keywords like int, while etc. as identifiers.

#### **5. What is the use of sizeof() operator? Find out size of various data types in bytes.**

The sizeof() operator is used to return the size of the datatype or the variable.

Syntax:

```
sizeof(variable_name); // or
sizeof(data_type);
```

The size of the various datatypes are:

Type	Size (in bytes)
unsigned char	1
signed char or char	1
unsigned int	2
signed int or int	2
unsigned short int	2
signed short int or short int	2
unsigned long int	4
signed long int or long int	4
double	8
float	4
long double	10

**6. What is type casting? Define type casting with suitable example.**

(Already done in MODEL SET 3 Q5)

**7. Write a program in C to check whether given entered year is leap year or not**

```
#include <stdio.h>
int main() {
    int year;
    printf("Enter a year: ");
    scanf("%d", &year);
    if (year % 400 == 0)
        printf("It is a leap year");
    else if (year % 100 == 0)
        printf("It is not a leap year");
    else if (year % 4 == 0)
        printf("It is a leap year");
    else
        printf("It is not a leap year");
    return 0;
}
```

**8. What is switch statement? Write a program to read any operator and perform the respective operation on the operands.**

(First part already done in MODEL SET 2 Q4)



Program:

```
#include <stdio.h>

float add(int a, int b);
float subtract(int a, int b);
float multiply(int a, int b);
float divide(int a, int b);

int main() {
    int a, b, choice;
    float result;
    printf("What operation do you want to do?\n");
    printf("1. Add\n");
    printf("2. Subtract\n");
    printf("3. Multiply\n");
    printf("4. Divide\n\nEnter your choice <1-4>: ");
    scanf("%d", &choice);
    if (choice < 1 || choice > 4) {
        printf("The choice is incalid.");
        return 1;
    }
    printf("\nEnter first number: ");
    scanf("%d", &a);
    printf("Enter second number: ");
    scanf("%d", &b);
    switch (choice) {
        case 1:
            result = add(a, b);
            break;
        case 2:
            result = subtract(a, b);
            break;
        case 3:
            result = multiply(a, b);
            break;
        case 4:
            result = divide(a, b);
            break;
    }
    printf("\nThe result is: %f", result);
    return 0;
}

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

float subtract(int a, int b) { return a - b; }

float multiply(int a, int b) { return a * b; }

float divide(int a, int b) { return 1.0 * a / b; }
```

**9, program to find the greatest common divisor (GCD) and lowest common multiple (LCM)**

```
#include <stdio.h>
int main() {
    int num1, num2, gcd, lcm, remainder, bigger, smaller;
    printf("Enter first number: ");
    scanf("%d", &num1);
    printf("Enter second number: ");
    scanf("%d", &num2);
    bigger = (num1 > num2) ? num1 : num2;
    smaller = (num1 < num2) ? num1 : num2;
    remainder = bigger % smaller;
    while (remainder != 0) {
        bigger = smaller;
        smaller = remainder;
        remainder = bigger % smaller;
    }
    gcd = smaller;
    lcm = num1 * num2 / gcd;
    printf("GCD of %d and %d = %d\n", num1, num2, gcd);
    printf("LCM of %d and %d = %d\n", num1, num2, lcm);
    return 0;
}
```

**10. Write a program in C to display following pattern:**

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

Program:

```
#include <stdio.h>
int main() {
    int i, j;
    for (i = 1; i <= 5; i++) {
        for (j = 1; j <= i; j++) {
            printf("%d\t", i + j - 1);
        }
        printf("\n");
    }
}
```

## 11. What is Infinite Loop? Demonstrate infinite loop with suitable practical example.

An infinite loop is a looping construct that does not terminate the loop and executes the loop forever. It is also called an indefinite loop or an endless loop. It either produces a continuous output or no output.

Program:

```
#include <stdio.h>
int main() {
    FILE *fp;
    fp = fopen("input.txt", "w");
    if (!fp) {
        printf("Error occurred");
        return -1;
    }
    while (!feof(fp)) {
        printf("File handling\n");
    }
    return 0;
}
```

## 12. Explain the use of graphical function. Make programs that display an arc by using graphic handling function.

(First part already done in MODEL SET 1 Q11)

Program:

```
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
int main() {
    int gDriver = DETECT, gMode;
    initgraph(&gDriver, &gMode, (char*)"C:\\TURBOC3\\BGI");
    arc(120, 120, 180, 0, 30);
    getch();
    closegraph();
    return 0;
}
```

## MODEL SET 6

### 1. What is structured programming? Describe advantages of structured programming.

Structured Programming can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn't support the possibility of jumping from one instruction to some other with the help of any statement. Therefore, the instructions in this approach will be executed in a serial and structured manner.

The structured program mainly consists of three types of elements:

- Selection Statements
- Sequence Statements
- Iteration Statements

Advantages of Structured Programming Approach:

- It is easier to read and understand.
- It is user friendly.
- It requires less time to write
- It is easier to Maintain.
- It is mainly problem based instead of being machine based.
- Development is easier as it requires less effort and time.
- It is easier to debug.
- It is machine-independent, mostly. i.e., programs developed in high level languages can be run on any computer.

### 2. What is iterative statement? List out types of looping statements used in C and describe each of them with suitable example.

Iterative statements are also called loops. If we want to execute a part of program many times, we use loops. List of looping statements in C language are:

- while loop
- do-while loop
- for loop

while loop:

When we aren't sure if the loop will be run or the number of times the loop will run, we use a while loop. The condition of a while loop is verified first, and then the statement is executed.

Syntax of while Loop

```
while (condition) {  
    statement 1;  
    statement 2;  
    ...  
}
```

Example program:

```
#include <stdio.h>
int main() {
    int i;
    i = 1;
    while (i <= 10) {
        printf("%d\t", i);
        i++;
    }
    return 0;
}
```

Output:

1 2 3 4 5 6 7 8 9 10

do...while loop:

When we are certain about the condition, we use a do-while loop. It enters the loop at least once before determining if the provided condition is true or false. It is run at least once to see if any of the conditions are true or false.

Syntax of do...while loop

```
Do {
    statement 1;
    statement 2;
    ...
} while (condition);
```

Example program:

```
#include <stdio.h>
int main() {
    int i;
    i = 1;
    do {
        printf("%d\t", i);
        i++;
    } while (i <= 10);
    return 0;
}
```

Output:

1 2 3 4 5 6 7 8 9 10

for loop in C:

In a for loop, there are three expressions. The first expression sets the index value, the second checks if the loop should be iterated, and the third sets the index value for future iteration.

Syntax of for loop:

```
for (Initialization; condition; Increment/Decrement) {
    statement 1;
    statement 2;
    ...
}
```

Example program:

```
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i <=10; i++) {
        printf("%d\t", i);
    }
    return 0;
}
```

**3. How can you initialize two-dimensional array in C? Write a program in C to sort given array of numbers by using bubble sort technique.**

(First part done in MODELSET 1 Q8)

Program:

```
#include <stdio.h>
int main() {
    int i, j, temp, num;
    printf("Enter the number of items: ");
    scanf("%d", &num);
    int arr[num];
    for (i = 0; i < num; i++) {
        printf("Enter the item #%d: ", i + 1);
        scanf("%d", &arr[i]);
    }
    for (i = 0; i < num - 1; i++) {
        for (j = 0; j < num - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    printf("\nItems after bubble sort:\n");
    for (i = 0; i < num; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

#### 4. Write a function to multiply two $n \times n$ matrices.

```
#include <stdio.h>
void multiplyMatrices(int n, int firstMatrix[n][n], int secondMatrix[n][n])
{
    int i, j, k;
    int resultMatrix[n][n];
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            int sum = 0;
            for (k = 0; k < n; k++) {
                sum += firstMatrix[i][k] * secondMatrix[k][j];
            }
            resultMatrix[i][j] = sum;
        }
    }
    printf("\nThe product of the two matrices are:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("%d\t", resultMatrix[i][j]);
        }
        printf("\n");
    }
    return;
}

int main() {
    int i, j, n;
    printf("Enter the order of the square matrix (n): ");
    scanf("%d", &n);
    int firstMatrix[n][n];
    int secondMatrix[n][n];
    printf("\nFor first matrix: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("Enter the element in row %d, column %d: ", i + 1, j + 1);
            scanf("%d", &firstMatrix[i][j]);
        }
    }
    printf("\nFor second matrix: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("Enter the element in row %d, column %d: ", i + 1, j + 1);
            scanf("%d", &secondMatrix[i][j]);
        }
    }
    multiplyMatrices(n, firstMatrix, secondMatrix);
    return 0;
}
```

## 5. What is searching? Describe sequential searching with suitable example.

Searching technique refers to finding a key element among the list of elements.

If the given element is present in the list, then the searching process is said to be successful.

If the given element is not present in the list, then the searching process is said to be unsuccessful.

Based on the type of search operation, these algorithms are generally classified into two categories:

- **Sequential Search:**  
In this, the list or array is traversed sequentially, and every element is checked.  
For example: Linear Search.
- **Interval Search:**  
These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half.  
For Example: Binary Search.

Example program for sequential search:

```
#include <stdio.h>
int main() {
    int i, n, toSearch;
    printf("Enter the number of items: ");
    scanf("%d", &n);
    int arr[n];
    for (i = 0; i < n; i++) {
        printf("Enter the item #%d: ", i + 1);
        scanf("%d", &arr[i]);
    }
    printf("Enter a number to search: ");
    scanf("%d", &toSearch);
    for (i = 0; i < n; i++) {
        if (arr[i] == toSearch) {
            printf("The number was found at index: %d", i);
            return 0;
        }
    }
    printf("The number was not found in the list.");
    return 0;
}
```



**6. Write a program to reverse the given string without using the `strrev()`.**

```
#include <stdio.h>
int main() {
    char str[100], reverse[100];
    int i, length = 0;
    printf("Enter a string: ");
    scanf("%[^\n]s", str);
    for (i = 0; str[length] != '\0'; i++) {
        length++;
    }
    for (i = 0; i < length; i++) {
        reverse[i] = str[length - 1 - i];
    }
    reverse[i] = '\0';
    printf("The reversed string is: %s", reverse);
    return 0;
}
```

**7. What do you mean by array of structures? Describe array of structures with suitable examples.**

(Already done in MODEL SET 2 Q9)

**8. Write a program to input any two numbers and then find their product by using the concepts of pointers.**

```
#include <stdio.h>
int main() {
    int num1, num2, result, *ptr1, *ptr2;
    ptr1 = &num1;
    ptr2 = &num2;
    printf("Enter the first number: ");
    scanf("%d", ptr1);
    printf("Enter the second number: ");
    scanf("%d", ptr2);
    result = *ptr1 * *ptr2;
    printf("The product is: %d", result);
    return 0;
}
```

**9. What is function? What are the advantages of using functions in C? Explain the use of functions with suitable example.**

In C, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by `{ }`. A function can be called multiple times to provide reusability and modularity to the C program. In other words, we can say that the collection of functions creates a program. The function is also known as procedure or subroutine in other programming languages.

Advantage of functions in C are as follows:

- Functions allow the divide and conquer strategy to be used for the development of programs.

When developing even a moderately sized program, it is very difficult if not impossible, to write the entire program as a single large main function. Such programs are very difficult to test, debug and maintain. The task to be performed is normally divided into several independent sub tasks, thereby reducing the overall complexity; a separate function is written for each sub task. In fact, we can further divide each sub task into smaller sub tasks, further reducing the complexity.

- Functions help avoid duplication of effort and code in programs.

During the development of a program, the same or similar activity may be required to be performed more than once. The use of functions in such situations avoids duplication of effort and code in programs. This reduces the size of the source program as well as the executable program. It also reduces the time required to write, test, debug and maintain such programs, thus reducing program development and maintenance cost.

- Functions enable us to hide the implementation details of a program.

E. g., we have used library functions such as sqrt, log, sin, etc. without ever knowing how they are implemented. However, although we need to know the implementation details for user-defined functions, once a function is developed and tested, we can continue to use it without going into its implementation details.

- Members of team can work on the program in parallel.

The divide and conquer approach also allows the parts of a program to be developed, tested and debugged independently and possibly concurrently by members of a programming team. The involvement of several programmers, which is the norm in the development of a software project, reduces the overall development time.

- Reusability.

The functions developed for one program can be used, if required, in another with little or no modification. This further reduces program development time and cost.

Example of function:

```
#include <stdio.h>
void search(int arr[], int length, int toFind) {
    int i;
    for (i = 0; i < length; i++) {
        if (arr[i] == toFind) {
            printf("The number was found at index: %d", i);
            return;
        }
    }
    printf("The number was not found in the list.");
}
int main() {
    int arr[] = {4, 5, 6, 34, 6};
    int i, toSearch, n = 5, toFind = 34;
    search(arr, n, toFind);
    return 0;
}
```

### 10. Write a program to test if a user input string is a palindrome or not.

```
#include <stdio.h>
int main() {
    char str[100];
    int i, length = 0;
    printf("Enter a string: ");
    scanf("%s", str);
    for (i = 0; str[i] != '\0'; i++) {
        length++;
    }
    for (i = 0; i < length / 2 + 1; i++) {
        if (str[i] != str[length - 1 - i]) {
            printf("The string you entered is not palindrome.");
            return 0;
        }
    }
    printf("The string you entered is palindrome.");
    return 0;
}
```

### 11. Write down the concept of typedef with suitable example.

The typedef is a keyword used in C programming to provide some meaningful names to the already existing data type names in the C program. It behaves similarly as we define the alias for the commands. In short, we can say that this keyword is used to redefine the name of an already existing data type.

Syntax of typedef

```
typedef <existing_name> <alias_name>
```

In the above syntax, 'existing\_name' is the name of an already existing data type while 'alias name' is another name given to the existing variable.

For example, suppose we want to create a variable of type unsigned int, then it becomes a tedious task if we want to declare multiple variables of this type. To overcome the problem, we use a typedef keyword.

```
typedef unsigned int unit;
```

In the above statements, we have declared the unit variable of type unsigned int by using a typedef keyword.

Now, we can create the variables of type unsigned int by writing the following statement:

```
unit a, b;
```

instead of writing:

```
unsigned int a, b;
```

typedef keyword is useful when we are dealing with the long data type especially, structure declarations.

Example program of typedef keyword:

```
#include <stdio.h>
int main() {
    typedef unsigned int unit;
    unit i, j;
    i = 10;
    j = 20;
    printf("Value of i is: %d", i);
    printf("\nValue of j is: %d", j);
    return 0;
}
```

## 12. How graphics driver initializes in C? Write a program to draw a circle.

Initialization of the graphics drivers is done using the `initgraph()` function provided in `graphics.h` library.

First, we call the `initgraph()` function that will initialize the graphics mode on the computer.

The method `initgraph()` has the following prototype.

```
void initgraph(int far *graphdriver,
               int far *graphmode,
               char far *pathtodriver);
```

The method initializes the graphics system by loading the graphics driver from disk (or validating a registered driver) then putting the system into graphics mode. It also resets all graphics settings (color, palette, current position, viewport, etc.) to their defaults.

The parameters are described as follows:

- `*graphdriver`  
This is an integer value that specifies the graphics driver to be used. You can give a value using a constant of the `graphics_drivers` enumeration type or set `graphdriver = DETECT` which requests autodetect of the graphics driver.
- `*graphmode`  
This is an integer value that specifies the initial graphics mode (unless `*graphdriver = DETECT`). If `*graphdriver` is `DETECT`, then `initgraph()` function sets `*graphmode` to the highest resolution available for the detected graphics driver.
- `*pathtodriver`  
Specifies the directory path where `initgraph()` looks for graphics drivers (\*.BGI) first. If they're not there, `initgraph()` function looks in the current directory.

This is how the graphics driver is initialized in C.

Program to draw a circle:

```
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
int main() {
    int gDriver = DETECT, gMode;
    initgraph(&gDriver, &gMode, (char*)"C:\\TURBOC3\\BGI");
    circle(100, 100, 60);
    getch();
    closegraph();
    return 0;
}
```

## MODEL SET 7

### 1. What is debugging? Explain various types of error occur in execution of C program.

A software bug is an error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result or causes it to behave in unintended ways. Most bugs arise from mistakes and errors made by people in either a program's source code or its design, and a few are caused by compilers producing incorrect code.

Debugging is a methodical process of finding and reducing the number of bugs (or defects) in a computer program, thus making it behave as originally expected.

There are two main types of errors that need debugging:

- **Syntax Error:**

Errors occur when you violate the rules of writing C syntax is said to be “Syntax errors”. This compiler error indicates that this must be fixed before the code will be compiled. These errors are identified by the compiler, so these errors are called “compile-time errors”.

```
void main() {
    int a // here semi colon(;)missed
}
void main() {
    int a;
    // here parenthesis{}) missed
```

- **Run-time error:**

Errors which are occurred after a successful compilation of program is said to be “run-time errors”. Number divisible by zero, array index out of bounds, string index out of bounds, etc. are most frequent run-time errors. These errors can't be very hard to detect at the compile time.

```
void main() {
    int a = 10;
    int c = a / 0; // Here number divisible zero error occurs
}
void main() {
    int a[3] = {1, 2, 3};
    int out = a[4]; // Here array out of bounds error occurs
}
```

- **Linker Errors**

These errors are generated after compilation we link the different object files with the main's object. These errors occurred when the executable program cannot be generated. This may be due occurred because of wrong function declaration, importing incorrect header files, etc. Most frequent linker error is writing Main() instead of a main() method.

```
void Main() { // Here Main() method used instead of main() method
    ...
}
```

## **2. What is dynamic memory allocation? What are the advantages of DMA over array? Show the concept of DMA with suitable example.**

C language requires the number of elements in an array to be specified at compile time. But we may not be able to do so always. Our initial judgement of size, if it is wrong, may cause failure of the program or wastage of memory space. The process of allocating memory at run time is known as dynamic memory allocation.

The advantages of using dynamic memory allocation over array are:

- Data structures can grow and shrink according to the requirement.
- We can allocate (create) additional storage whenever we need them.
- We can de-allocate (free/delete) dynamic space whenever we are done with them.
- Dynamic memory allocation is done at run time.

Example program using Dynamic memory allocation

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *ptr;
    ptr = (int *)calloc(10, sizeof(int));
    ptr[0] = 3;
    ptr[1] = 234;
    ptr[2] = 35;
    ptr = (int *)realloc(ptr, 3 * sizeof(int));
    return 0;
}
```

## **3. What is nesting of structure? Explain nesting of structure with suitable example.**

(Already done in MODEL SET 4 Q11)

#### 4. Write a program to find the factorial of a given number by using recursion.

```
#include <stdio.h>
int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    return factorial(n - 1) * n;
}
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("The value of %d! is: %d", num, factorial(num));
    return 0;
}
```

#### 5. What is a preprocessor directive in C?

Preprocessor directives are the special instructions which are executed before code passes through the compilation process. All C programs are checked for the preprocessor statements before compiling and if any preprocessor statements are used in the program, then they are processed first and then preprocessed program is handed over to the compiler for further compilation. Since all preprocessor statements are processed first and this is the reason for calling them preprocessor directives. It begins with hash (#) symbol and do not require semicolon at the end.

One of the most important preprocessor directives is `#include` which is used for including header file.

For examples:

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

Similarly, `#define` is another commonly used preprocessor directive which is used for creating symbolic constants and for defining macros.

For examples:

```
#define PI 3.141592
#define SIZE 10
```

Other examples of preprocessor directives are:

- `#if`, `#elif`, `#else`, and `#endif`: It tests the program using a certain condition; these directives can be nested too.
- `#undef`: It undefines a certain preprocessor macro.
- `#ifdef`: It returns true if a certain macro is defined.
- `#ifndef`: It returns true if a certain macro is not defined.
- `#error` can be performed to stop compilation.
- `#warning` is performed to continue compilation with messages in the console window.

**6. Write a program to read name and roll number of 48 students and store them into file stu.txt by using fprintf() function.**

```
#include <stdio.h>
int main() {
    FILE *fp;
    fp = fopen("stu.txt", "w");
    if (!fp) {
        printf("Some error occurred while writing the file.");
        return -1;
    }
    int i, rollNumber;
    char name[30];
    for (i = 0; i < 48; i++) {
        printf("\nEnter the name of student %d: ", i + 1);
        scanf("%[^\n]s", name);
        printf("Enter the roll number of student %d: ", i + 1);
        scanf("%d", &rollNumber);
        fprintf(fp, "%s, %d\n", name, rollNumber);
    }
    fclose(fp);
    return 0;
}
```

**7. Write a program to convert the given string in to lower case letters without using string handling function strlwr().**

```
#include <stdio.h>
int main() {
    char string[100];
    printf("Enter a string: ");
    scanf("%[^\n]", string);
    int i;
    for (i = 0; string[i] != '\0'; i++) {
        if (string[i] >= 'A' && string[i] <= 'Z') {
            string[i] = string[i] + 32;
        }
    }
    printf("The string in lowercase is: %s", string);
    return 0;
}
```



## 8. What is variable? Explain types of variables used in C with suitable example.

A variable in simple terms is a storage place that has some memory allocated to it. A variable is used to store some form of data. Different types of variables require different amounts of memory, different type of memory locations, and some specific set of operations that can be applied to them.

A typical variable declaration is of the form:

```
Data_type variable_name;  
// for multiple variables:  
Data_type variable1_name, variable2_name, variable3_name;
```

The types of variables in C are:

- local variable
- global variable
- static variable
- automatic variable
- external variable

### Local Variable

A variable that is declared inside the function or block is called a local variable.

Example:

```
void function1() {  
    int x = 10; // local variable  
}
```

You must have to initialize the local variable before it is used.

### Global Variable

A variable that is declared outside the function or block is called a global variable. It is available to all the functions and any function can change the value of the global variable.

Example:

```
int value = 20; // global variable  
void function1() {  
    int x = value + 10;  
}
```

### Static Variable

A variable that is declared with the static keyword is called static variable.

It retains its value between multiple function calls.

Example:

```
void function1() {  
    int x = 10; // local variable  
    static int y = 10; // static variable  
    x = x + 1;  
    y = y + 1;  
    printf("%d, %d", x, y);  
}
```

If you call this function many times, the local variable will print the same value for each function call, e.g. 11, 11, 11 and so on. But the static variable will print the incremented value in each function call, e.g. 11, 12, 13 and so on.

### Automatic Variable

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using auto keyword.

Example:

```
void main() {
    int x = 10; // local variable (also automatic)
    auto int y = 20; // automatic variable
}
```

### External Variable

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use extern keyword.

Example file: myfile.h

```
extern int x = 10; // external variable (also global)
```

Example file: program1.c

```
#include "myfile.h"
#include <stdio.h>
void printValue(){
    printf("Global variable: %d", global_variable);
}
```

## 9. What is function? What are the components of function in C?

A function is a routine or set of instructions are code that performs a specific task and can be processed independently. When the program passes the control to a function, the function performs the task and returns the control to the instruction following the calling instruction.

A function usually has three components. They are:

- Function Prototype/Declaration
- Function Definition
- Function Call

Function Prototype/Declaration:

Function declaration is a statement that informs the compiler about

- Name of the function
- Type of arguments
- Number of arguments
- Type of Return value

Syntax for function declaration

```
Return_type function_name([arguments type]);
```

For example,

```
void sort(int []); /* function name = sort, receives an array
                    of integer as argument and returns nothing */
int sum(int, int); /* function name = sum, receives two integers
                    as argument and returns an integer */
```

A function declaration doesn't require name of arguments to be provided, only type of the arguments can be specified.

Function Definition:

Function definition consists of the body of function. The body consists of block of statements that specify what task is to be performed. When a function is called, the control is transferred to the function definition.

Syntax for function definition

```
returntype function_name ([arguments])
{
    statement(s);
    ... ..
}
```

return Statement:

A return statement is used to return values to the invoking function by the invoked function. The data type of value a function can return is specified during function declaration. A function with void as return type don't return any value. A return statement is usually place at the end of function definition or inside a branching statement.

Syntax of return statement

```
return value;
```

For example,

```
int product (int x, int y) {
    int p = x * y;
    return p;
}
```

In this function, the return type of product() is int. So it returns an integer value p to the invoking function.

Function call:

A function call can be made by using a call statement. A function call statement consists of function name and required argument enclosed in round brackets.

Syntax for function call

```
function_name([actual arguments]);
```

For example,

```
sort(a);
p = product(x, y);
```

**10. What is the purpose of defining algorithm before writing the program? Write an algorithm to find roots of given quadratic equation.**

An algorithm is a well-defined step by step procedure that describes how to solve a problem. The purpose of defining algorithm before writing the program is to get more freedom to think about the possible solution and ways to solve a program. It helps users to perform hit and trial on various logics, plan out and optimize the logic, so that it will be easier during the coding stage as the instructions and logics have already become clear with the help of the algorithm.

Algorithm to find the roots of a quadratic equation:

Step 1: Start

Step 2: Get the coefficients a, b and c

Step 3: Calculate discriminant =  $b^2 - 4 * a * c$ .

Step 4: if discriminant > 0:  $x1 = (-b + \sqrt{\text{discriminant}}) / 2 * a$ ,  
 $x2 = (-b - \sqrt{\text{discriminant}}) / 2 * a$ ,  
Print the roots are x1, x2

Step 5: if discriminant = 0: Print the root is:  $(-b / 2 * a)$ .

Step 6: if discriminant < 0:  $\text{realPart} = -b / (2 * a)$ ,  
 $\text{imagPart} = \sqrt{-\text{discriminant}} / (2 * a)$ ,  
Print the roots are:  $(\text{realPart}) + (\text{imagPart})i$  and  
 $(\text{realPart}) - (\text{imagPart})i$

Step 7: Stop

**11. What is the concept behind ternary operator? Write a program in C to test whether given number is odd or even by using ternary (conditional) operator.**

The operators, which require three operands to act upon, are known as ternary operators. It can be represented by “ ? : ”. It is also known as conditional operator. The operator improves the performance and reduces the line of code.

The syntax of ternary operator in C language is:

`expression1 ? expression2 : expression3;`

Program:

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    num % 2 == 0 ? printf("The number is even.") :
                printf("The number is odd.");
    return 0;
}
```

**12. Explain the use of graphical function. Make a program that contains basic graphic functions.**

(Already done in MODEL SET 1 Q11)

## MODEL SET 8

### 1. How string declared in C? Explain two dimensional array of character. Also write a program to sort given n strings by using bubble sort.

A string is an array of characters concluded with a NULL character '\0'. C does not directly support string as a data type. Hence, character arrays must be used to display a string in C. The general syntax of declaring a string in C is as follows:

```
char variable[array_size];
```

Strings can also be initialized during declaration as follows:

```
char name[20] = "Joon Shakya";
```

One dimensional array of characters are strings. So, two-dimensional array of characters is array of strings.

Two-dimensional array of characters can be declared as follows:

```
char string_array_name[number_of_strings][length_of_longest_string];
```

Example:

```
char language[5][10];
```

Two-dimensional array of characters can also be initialized during declaration as follows:

```
char language[5][10] = {"Java", "Python", "C++", "HTML", "SQL"};
```

Program to sort array of strings in alphabetical order using bubble sort:

```
#include <stdio.h>
#include <string.h>
int main() {
    int i, j, num;
    printf("Enter the number of strings: ");
    scanf("%d", &num);
    char strings[num][20], temp[20];
    for (i = 0; i < num; i++) {
        printf("Enter the string #%d: ", i + 1);
        scanf("%[^\n]s", strings[i]);
    }
    for (i = 0; i < num - 1; i++) {
        for (j = 0; j < num - i - 1; j++) {
            if (strcmp(strings[j], strings[j + 1]) > 0) {
                strcpy(temp, strings[j]);
                strcpy(strings[j], strings[j + 1]);
                strcpy(strings[j + 1], temp);
            }
        }
    }
    printf("\nStrings after bubble sort:\n");
    for (i = 0; i < num; i++) {
        printf("%s\n", strings[i]);
    }
    return 0;
}
```

**2. What is the concept behind pointers? Differentiate between malloc() and calloc() functions. Describe the DMA functions with suitable example.**

Pointers in C language is a variable that stores/points the address of another variable. A pointer in C is used to allocate memory dynamically i.e.: at runtime. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

Pointer syntax: `data_type *var_name;`

Example:

```
int *p;  
char *p;
```

Where, \* is used to denote that “p” is pointer variable and not a normal variable.

Differences between malloc() and calloc() functions are:

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.
The number of arguments in malloc() is 1.	The number of arguments in calloc() is 2.
malloc() is faster.	calloc() is slower.
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.
malloc() stands for memory allocation.	calloc() stands for contiguous allocation.

The various functions for Dynamic memory allocation are:

- **malloc():**  
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 casted into pointers of any form.  
Syntax of malloc():  

```
ptr = (castType*)malloc(size);
```

  
Example:  

```
ptr = (float*)malloc(100 * sizeof(float));
```

  
The above statement allocates 400 bytes of memory. It's because the size of float is 4 bytes. And the pointer ptr holds the address of the first byte in the allocated memory. The expression results in a NULL pointer if the memory cannot be allocated.
- **calloc():**  
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.  
Syntax of calloc():  

```
ptr = (castType*)calloc(n, size);
```

  
Example:  

```
ptr = (float*)calloc(25, sizeof(float));
```

  
The above statement allocates contiguous space in memory for 25 floats.

- `free()`:  
Dynamically allocated memory created with either `calloc()` or `malloc()` doesn't get freed on their own. We must explicitly use `free()` to release the space.  
Syntax of `free()`  
`free(ptr);`  
This statement frees the space allocated in the memory pointed by `ptr`.
- `realloc()`:  
If the dynamically allocated memory is insufficient or more than required, you can change the size of previously allocated memory using the `realloc()` function.  
Syntax of `realloc()`:  
`existing_ptr = realloc(existing_ptr, new_size);`  
Example:  
`ptr = realloc(ptr, 25);`  
Here, `ptr` is reallocated with a new size 25.

Example program:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *ptr, i, n1, n2;
    printf("Enter size: ");
    scanf("%d", &n1);
    ptr = (int *)malloc(n1 * sizeof(int));
    printf("Addresses of previously allocated memory:\n");
    for (i = 0; i < n1; i++) {
        printf("%p\n", ptr + i);
    }
    printf("\nEnter the new size: ");
    scanf("%d", &n2);
    ptr = realloc(ptr, n2 * sizeof(int));
    printf("Addresses of newly allocated memory:\n");
    for (i = 0; i < n2; i++) {
        printf("%p\n", ptr + i);
    }
    free(ptr);
    return 0;
}
```

### **3. What is array of structure? Explain array of structure with taking records of any 20 students and display them.**

(First part already done in MODEL SET 2 Q9)

Program:

```
#include <stdio.h>
int main() {
    struct Student {
        char fullName[30];
        int rollNumber;
        char address[20];
        char section;
    };
    int studentLength = 20, i;
    printf("Enter the details of %d students.\n", studentLength);
    struct Student students[studentLength];
    for (i = 0; i < studentLength; i++) {
        printf("\nEnter the name of student %d: ", i + 1);
        scanf(" %[^\\n]s", students[i].fullName);
        printf("Enter the roll number of student %d: ", i + 1);
        scanf(" %d", &students[i].rollNumber);
        printf("Enter the address of student %d: ", i + 1);
        scanf(" %[^\\n]s", students[i].address);
        printf("Enter the section of student %d: ", i + 1);
        scanf(" %c", &students[i].section);
    }
    for (i = 0; i < studentLength; i++) {
        printf("\n\nDetails of student %d:", i + 1);
        printf("\nName:\\t\\t%s", students[i].fullName);
        printf("\nRoll Number:\\t%d", students[i].rollNumber);
        printf("\nAddress:\\t%s", students[i].address);
        printf("\nSection:\\t%c", students[i].section);
    }
    return 0;
}
```

**4. Write a program to generate Fibonacci series up to n terms using recursion function.**  
(Hint Fibonacci sequence = (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ....))

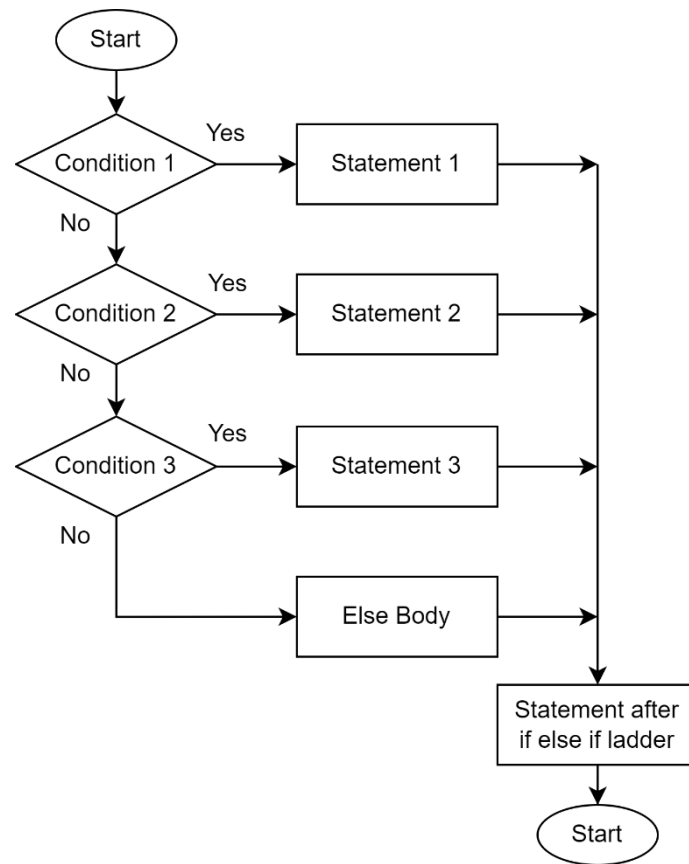
(Already done in MODEL SET 1 Q9)

**5. What is if else if ladder? Explain it with their flow chart and complete C program.\\**

if-else-if ladder is the use of if, else if conditions multiple times to decide among multiple condition if one fails. if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.



Flowchart to show if else if ladder:



Example program:

```
#include <stdio.h>
int main() {
    float percentage;
    printf("Enter your percentage: ");
    scanf("%f", &percentage);
    if (percentage > 80) {
        printf("Distinction");
    } else if (percentage > 70) {
        printf("First Division");
    } else if (percentage > 60) {
        printf("Second Division");
    } else if (percentage > 40) {
        printf("Third Division");
    } else {
        printf("Fail");
    }
}
```

**6. Write a program to read name of any n students from user then display only largest name and their length.**

```
#include <stdio.h>
#include <string.h>
int main() {
    int i, j, num, maxLen = 0, len;
    char maxName[50], tempName[50];
    printf("Enter the number of students: ");
    scanf("%d", &num);
    for (i = 0; i < num; i++) {
        printf("Enter the name of student %d: ", i + 1);
        scanf(" %[^\\n]s", tempName);
        len = strlen(tempName);
        if (strlen(tempName) > maxLen) {
            maxLen = len;
            strcpy(maxName, tempName);
        }
    }
    printf("Longest Name:\\t %s\\nLength:\\t\\t %d", maxName, maxLen);
    return 0;
}
```

**7. What is an algorithm? Write an algorithm to check given number is prime or composite.**

Writing a logical step-by-step method to solve the problem is called the algorithm. In other words, an algorithm is a procedure for solving problems. To solve a mathematical or computer problem, this is the first step in the process. An algorithm includes calculations, reasoning, and data processing. Algorithms can be presented by natural languages, pseudocode, and flowcharts, etc.

Algorithm to check if the given number is prime or composite:

Step 1: Start.

Step 2: Get a number from the user.

Step 3: Set count = 0.

Step 4: Loop from i = 2, while i <= num / 2, incrementing i in every iteration.

    If (num % i == 0) then

        Set count = 1 and break out of the loop.

    Else, continue the loop.

Step 5: If (count == 1) then

    Print the number is a prime number.

    Else if num > 1:

        Print the number is composite number.

    Else

        Print the input is invalid

Step 6: Stop.

## 8. What is escape sequence? Explain types and use of escape sequence with suitable example.

An escape sequence in C language is a sequence of characters that don't represent itself when used inside string literal or character instead, symbolizes some different character. It is composed of two or more characters starting with backslash \.

The various escape sequences in C are:

- \a Alarm or Beep
- \b Backspace
- \f Form Feed
- \n New Line
- \r Carriage Return
- \t Tab (Horizontal)
- \v Vertical Tab
- \\ Backslash
- \' Single Quote
- \" Double Quote
- \? Question Mark
- \nnn octal number
- \xhh hexadecimal number
- \0 Null

## 9. What is the use of function declaration? Write a program in C to find the simple interest by using function.

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts –

```
return_type function_name(parameter_list);
```

Example:

```
int max(int num1, int num2);
```

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration:

```
int max(int, int);
```

Program to find simple interest:

```
#include <stdio.h>
int main() {
    float p, t, r, i;
    printf("Enter the principal (P): ");
    scanf("%f", &p);
    printf("Enter the time (T): ");
    scanf("%f", &t);
    printf("Enter the interest rate (R): ");
    scanf("%f", &r);
    i = 0.01 * p * t * r;
    printf("Interest is: Rs. %.2f", i);
    return 0;
}
```

**10. Write a program that computes the sum of digits of given integer number.**

```
#include <stdio.h>
int main() {
    int num, remainder, sum = 0;
    printf("Enter a number: ");
    scanf("%d", &num);
    while (num > 0) {
        remainder = num % 10;
        sum = sum + remainder;
        num = num / 10;
    }
    printf("The sum of digits is: %d", sum);
    return 0;
}
```

**11. Write a program in C to read a text file that contains the word three, bad, time etc. and create another text file deleting the following words 'three', 'bad', and 'time'.**

(Already done in MODEL SET 3 Q3)

**12. Write short note on:**

**a. Graphic function**

(Already done in MODEL SET 1 Q11)

**b. Formatted I/O**

C language comes with standard formatted I/O functions so that a programmer can perform formatted output and input in a program. The formatted functions accept various format specification strings along with a list of variables in the form of parameters. This format specification string refers to a character string used for specifying the data type of every variable present as the output or input along with the width and size of the I/O.

The formatted I/O functions in C are:

- scanf()

We use the scanf() function for getting the formatted inputs or standard inputs so that the printf() function can provide the program with numerous options of conversion. It takes first argument as string with format specifiers and rest of the arguments are the memory locations of the variables to store the data in order of the format specifiers.

Syntax for scanf()

```
scanf(format_specifier, &data_a, &data_b, ....., &data_n);
```

Here, & refers to the address operator

The purpose of the scanf() function is to read the characters that we get from the standard input, convert them according to the string of format specification, and then store the available inputs in the memory slots that the other arguments represent.

Example of scanf()

```
scanf("%d %c", &info_a, &info_b);
```

When we consider the string data names in a program, the '&' character does not prefix the data name.

- `printf()`

We use the `printf()` function for generating the formatted outputs or standard outputs in accordance with a format specification. The output data and the format specification string act as the parameters of the function `printf()`. It takes first argument as string with format specifiers and rest of the arguments are the values/variables to print the data in order of the format specifiers.

Syntax for `printf()`

```
printf(format_specifiers, info_a, info_b, ..., info_n);
```

Example of `printf()`

```
printf("%d %c", info_a, info_b);
```

The character that we specify after the ‘%’ character refers to the conversion character. It is because ‘%’ allows the conversion of any given data type into another data type for further printing.

Conversion Character Description and Meaning of Character

Conversion Character	Description and Meaning of Character
c	The program takes the data in the form of characters.
d	The program performs the conversion of the data into integers (decimals).
f	The program generates data output in the form of a double or a float with the default of a Precision 6.
s	The program identifies the data as a string, and a single character from this string gets printed unless and until the program reaches a NULL character.