1) WAP for calloc()

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

int main()
{
   int *ptr = NULL;
   int n;

    printf("enter no of integers that will stores \n");
    scanf("%d",&n);

    ptr =(int *) calloc(n, sizeof(int));

    for(int i = 0; i < n; i++){
       printf("ptr[%d} = %d, ",i,ptr[i]);
    }
   return 0;
}
```

o/p

```
enter no of integers that will stores
5
ptr[0} = 0, ptr[1} = 0, ptr[2} = 0, ptr[3} = 0, ptr[4} = 0,
```

2) //WAP for realloc()

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

int main()
{
   char *str;

   //initial memory allocation
   str = (char *)malloc(15);
   strcpy(str ,"jason");
   printf("string before realloc: %s, address : %p\n",str,str);

   //reallocating memory
   str = (char *)realloc(str,25);
   strcat(str ,".com");
   printf("string after realloc: %s, address : %p\n",str,str);
   free(str);
```

```
    return 0;
}
```

o/p

```
string before realloc: jason, address : 0x6017b303e2a0
string after realloc: jason.com, address : 0x6017b303e6d0
```

3) concept of double pointer

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

int main()
{
    int **ipp;
    int i = 4, j = 5, k = 6;
    int *ip1, *ip2;

    ip1 = &i;
    ip2 = &j;

    ipp = &ip1;

    printf("001 i = %d\n",*ip1);
    printf("002 i = %d\n",**ipp);

    **ipp = 8;
    printf("003 i = %d\n",i);
    return 0;
}
```

o/p

```
001 i = 4
002 i = 4
003 i = 8
```

4) concept of triple pointer

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

int main()
{
```

```
    int **ipp;
    int ***ipp1;
    int i = 4, j = 5, k = 6;
    int *ip1, *ip2;

    ip1 = &i;
    ip2 = &j;

    ipp = &ip1;
    ipp1 = &ipp;

    printf("001 i = %d\n",*ip1);
    printf("002 i = %d\n",**ipp);
    printf("003 i = %d\n",***ipp1);

    ***ipp1 = 8;
    printf("003 i = %d\n",i);
    return 0;
}
```

o/p

```
001 i = 4
002 i = 4
003 i = 4
003 i = 8
```

5) **Problem 1: Dynamic Array Resizing**

Objective: Write a program to dynamically allocate an integer array and allow the user to resize it.

Description:

1. The program should ask the user to enter the initial size of the array.
2. Allocate memory using malloc.
3. Allow the user to enter elements into the array.
4. Provide an option to increase or decrease the size of the array. Use realloc to adjust the size.
5. Print the elements of the array after each resizing operation

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

```
void printArray(int *array, int size) {
    printf("Array elements: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
}

int main() {
    int *array = NULL;
    int size = 0, newSize = 0;

    // Step 1: Ask for the initial size of the array
    printf("Enter the initial size of the array: ");
    scanf("%d", &size);

    if (size <= 0) {
        printf("Invalid size. Exiting.\n");
        return 1;
    }

    // Step 2: Allocate memory dynamically using malloc
    array = (int *)malloc(size * sizeof(int));
    if (array == NULL) {
        printf("Memory allocation failed. Exiting.\n");
        return 1;
    }

    // Step 3: Input elements into the array
    printf("Enter %d elements for the array:\n", size);
    for (int i = 0; i < size; i++) {
        scanf("%d", &array[i]);
    }

    // Step 4: Provide resizing options
    int choice;
    do {
        printf("\nCurrent array size: %d\n", size);
        printArray(array, size);

        printf("\nOptions:\n");
        printf("1. Increase size\n");
        printf("2. Decrease size\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
        case 1:
            printf("Enter new size (greater than %d): ", size);
            scanf("%d", &newSize);
            if (newSize > size) {
                array = (int *)realloc(array, newSize * sizeof(int));
                if (array == NULL) {
                    printf("Memory allocation failed. Exiting.\n");
                    return 1;
                }
                printf("Enter %d new elements:\n", newSize - size);
                for (int i = size; i < newSize; i++) {
                    scanf("%d", &array[i]);
                }
                size = newSize;
            } else {
                printf("New size must be greater than current size.\n");
            }
            break;

        case 2:
            printf("Enter new size (less than %d): ", size);
            scanf("%d", &newSize);
            if (newSize < size && newSize > 0) {
                array = (int *)realloc(array, newSize * sizeof(int));
                if (array == NULL) {
                    printf("Memory allocation failed. Exiting.\n");
                    return 1;
                }
                size = newSize;
            } else {
                printf("New size must be less than current size and greater than 0.\n");
            }
            break;

        case 3:
            printf("Exiting program.\n");
            break;

        default:
            printf("Invalid choice. Please try again.\n");
    }

} while (choice != 3);

// Free allocated memory
free(array);

return 0;
```

}

o/p

```
Current array size: 3
Array elements: 1 2 3

Options:
1. Increase size
2. Decrease size
3. Exit
Enter your choice: 1
Enter new size (greater than 3): 4
Enter 1 new elements:
3

Current array size: 4
Array elements: 1 2 3 3

Options:
1. Increase size
2. Decrease size
3. Exit
Enter your choice: 2
Enter new size (less than 4): 2

Current array size: 2
Array elements: 1 2

Options:
1. Increase size
2. Decrease size
3. Exit
Enter your choice: 3
Exiting program.
```

6) Problem 2: String Concatenation Using Dynamic Memory

Objective: Create a program that concatenates two strings using dynamic memory allocation.

Description:

1. Accept two strings from the user.
2. Use malloc to allocate memory for the first string.

3. Use realloc to resize the memory to accommodate the concatenated string.
4. Concatenate the strings and print the result.
5. Free the allocated memory.

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

int main() {
    char *str1, *str2;
    int len1, len2;

    // Input the first string
    printf("Enter the first string: ");
    str1 = (char *)malloc(100 * sizeof(char));
    scanf("%s", str1);
    len1 = strlen(str1);

    // Input the second string
    printf("Enter the second string: ");
    str2 = (char *)malloc(100 * sizeof(char));
    scanf("%s", str2);
    len2 = strlen(str2);

    // Allocate memory for concatenated string
    str1 = (char *)realloc(str1, (len1 + len2 + 1) * sizeof(char));

    // Concatenate the strings
    strcat(str1, str2);

    // Print the result
    printf("Concatenated string: %s\n", str1);

    // Free allocated memory
    free(str1);
    free(str2);

    return 0;
}
```

o/p

```
Enter the first string: hello
Enter the second string: world
Concatenated string: helloworld
```

7) Problem 3: Sparse Matrix Representation

Objective: Represent a sparse matrix using dynamic memory allocation.

Description:

1. Accept a matrix of size m×nm \times nm×n from the user.
2. Store only the non-zero elements in a dynamically allocated array of structures (with fields for row, column, and value).
3. Print the sparse matrix representation.
4. Free the allocated memory at the end.

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

int main() {
    int m, n, nonZeroCount = 0;
    int *rows, *cols, *values;

    // Input matrix dimensions
    printf("Enter the number of rows (m) and columns (n): ");
    scanf("%d %d", &m, &n);

    // Input matrix elements
    int matrix[m][n];
    printf("Enter the elements of the %dx%d matrix:\n", m, n);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &matrix[i][j]);
            if (matrix[i][j] != 0) {
                nonZeroCount++;
            }
        }
    }

    // Allocate memory for sparse matrix representation
    rows = (int *)malloc(nonZeroCount * sizeof(int));
    cols = (int *)malloc(nonZeroCount * sizeof(int));
    values = (int *)malloc(nonZeroCount * sizeof(int));
    if (!rows || !cols || !values) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    // Fill sparse matrix representation
    int index = 0;
```

```
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (matrix[i][j] != 0) {
                rows[index] = i;
                cols[index] = j;
                values[index] = matrix[i][j];
                index++;
            }
        }
    }

    // Print sparse matrix representation
    printf("\nSparse Matrix Representation:\n");
    printf("Row  Column  Value\n");
    for (int i = 0; i < nonZeroCount; i++) {
        printf("%d     %d      %d\n", rows[i], cols[i], values[i]);
    }

    // Free allocated memory
    free(rows);
    free(cols);
    free(values);

    return 0;
}
```

o/p

```
Enter the number of rows (m) and columns (n): 2 3
Enter the elements of the 2x3 matrix:
1 2 3
2 2 5

Sparse Matrix Representation:
Row  Column  Value
0    0     1
0    1     2
0    2     3
1    0     2
1    1     2
1    2     5
```

8)Problem 5: Dynamic 2D Array Allocation

Objective: Write a program to dynamically allocate a 2D array.

Description:

1. Accept the number of rows and columns from the user.
2. Use malloc (or calloc) to allocate memory for the rows and columns dynamically.
3. Allow the user to input values into the 2D array.
4. Print the array in matrix format.
5. Free all allocated memory at the end.

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

int main() {
    int rows, cols, i, j;

    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &cols);

    int **array = (int **)malloc(rows * sizeof(int *));
    if (array == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    for (i = 0; i < rows; i++) {
        array[i] = (int *)malloc(cols * sizeof(int));
        if (array[i] == NULL) {
            printf("Memory allocation failed for row %d.\n", i);


                for (int k = 0; k < i; k++) {
                free(array[k]);
            }
            free(array);
            return 1;
        }
    }

       printf("Enter the values for the 2D array:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &array[i][j]);
        }
    }
```

```
    printf("The 2D array in matrix format:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("%d ", array[i][j]);
        }
        printf("\n");
    }

    for (i = 0; i < rows; i++) {
        free(array[i]);
    }
    free(array);

    printf("Memory successfully freed.\n");

    return 0;
}
```

9) //structure

```
#include<stdio.h>

struct date

{

int day;

int month;

int year;

}today;//can declare and define like this also

int main()

{

// struct date today;//memory allocated

printf("size of today=%d \n",sizeof(today));

//accessing members in struct

today.day=21;
```

```
today.month=11;

today.year=2024;

printf("today's date is %d-%d-%d", today.day,today.month,today.year);

return 0;

}
```

10. //un named structures

```
//structure

#include<stdio.h>

struct

{

int day;

int month;

int year;

}today;

int main()

{

printf("size of today=%d \n",sizeof(today));

today.day=21;

today.month=11;

today.year=2024;

printf("today's date is %d-%d-%d", today.day, today. month,today.year);

return 0;
}
```

11)

Problem 1: Student Record Management System

Objective

Create a program to manage student records using structures.

Requirements

1. Define a Student structure with the following fields:
   - char name[50]
   - int rollNumber
   - float marks
2. Implement functions to:
   - Add a new student record.
   - Display all student records.
   - Find and display a student record by roll number.
   - Calculate and display the average marks of all students.
3. Implement a menu-driven interface to perform the above operations.

Output

1. Add Student
2. Display All Students
3. Find Student by Roll Number
4. Calculate Average Marks
5. Exit

Enter your choice: 1

Enter name: John Doe

Enter roll number: 101

Enter marks: 85.5

Student added successfully!

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

// Define the Student structure
struct student {
    char name[50];
    int rollno;
    float marks;
};

// Declare an array of students
struct student students[100];
int studentCount = 0;

// Function to add a new student
void addStudent() {
    if (studentCount >= 100) {
        printf("Student list is full!\n");
        return;
    }
    struct student s;
    printf("Enter name: ");
    scanf(" %[^\n]", s.name);  // Read name with spaces
    printf("Enter roll number: ");
```

```
    scanf("%d", &s.rollno);
    printf("Enter marks: ");
    scanf("%f", &s.marks);

    students[studentCount++] = s;
    printf("Student added successfully!\n");
}

// Function to display all students
void displayAllStudents() {
    if (studentCount == 0) {
        printf("No students found!\n");
        return;
    }
    for (int i = 0; i < studentCount; i++) {
        printf("Student %d:\n", i + 1);
        printf("  Name: %s\n", students[i].name);
        printf("  Roll Number: %d\n", students[i].rollno);
        printf("  Marks: %.2f\n", students[i].marks);
    }
}

// Function to find a student by roll number
void findStudentByRollNo() {
    int rollno;
    printf("Enter roll number to search: ");
    scanf("%d", &rollno);

    for (int i = 0; i < studentCount; i++) {
        if (students[i].rollno == rollno) {
            printf("Student found:\n");
            printf("  Name: %s\n", students[i].name);
            printf("  Marks: %.2f\n", students[i].marks);
            return;
        }
    }
    printf("Student with roll number %d not found!\n", rollno);
}

// Function to calculate and display average marks
void calculateAverageMarks() {
    if (studentCount == 0) {
        printf("No students found to calculate average marks!\n");
        return;
    }
    float totalMarks = 0.0;
    for (int i = 0; i < studentCount; i++) {
        totalMarks += students[i].marks;
```

```
    }
    printf("Average Marks: %.2f\n", totalMarks / studentCount);
}

// Main function with menu-driven interface
int main() {
    int choice;

    do {
        printf("\nMenu:\n");
        printf("1. Add Student\n");
        printf("2. Display All Students\n");
        printf("3. Find Student by Roll Number\n");
        printf("4. Calculate Average Marks\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            addStudent();
            break;
        case 2:
            displayAllStudents();
            break;
        case 3:
            findStudentByRollNo();
            break;
        case 4:
            calculateAverageMarks();
            break;
        case 5:
            printf("Exiting the program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

12) The program shows how to use a structure to group related information about a student (name, roll number, and marks) and how to initialize and display it

```
#include <stdio.h>
struct student{
	char name[50];
	int rollNumber;
	float marks;
};


int main(){
	//struct date today;

	struct student s1 = {.rollNumber = 1234, .name = "Abhinav", .marks = 95.5};

	printf("S1's Name roll number and marks is %s %d & %f \n", s1.name,
s1.rollNumber,s1.marks);

	return 0;
}
```

o/p

S1's Name roll number and marks is Abhinav 1234 & 95.500000

13) This program demonstrates the use of a structure in C to pass data to a function, showcasing structure initialization and passing structures as arguments by value

```
#include <stdio.h>


struct Coordinate{
	int x;
	int y;
};


void printCoordinate(struct Coordinate);


int main(){

	printCoordinate((struct Coordinate){5, 6});
	/*struct Coordinate pointA = {5,6};
	printCoordinate(pointA);*/
```

```
	return 0;
}
 void printCoordinate(struct Coordinate temp){
	printf("x = %d  y = %d \n",temp.x, temp.y);
}
```

o/p

x = 5 y = 6

14) This program demonstrates the use of arrays of structures in C, showcasing how to initialize, store, and retrieve structured data for multiple entities

```
#include <stdio.h>


struct Coordinate{
	int x;
	int y;
};



int main(){

	struct Coordinate Pnt[5];

	for(int i = 0; i < 5; i++){
	printf("Intilize the struct present in the %d index \n",i);
	scanf("%d %d",&Pnt[i].x,&Pnt[i].y);
	printf("\n");
	}

	for(int i = 0; i < 5; i++){
	printf("Diaply the Coordinates at index %d is (%d,%d) \n",i,Pnt[i].x,Pnt[i].y);
	printf("\n");
	}

	return 0;
}
```

o/p

```
Intilize the struct present in the 0 index
1 2

Intilize the struct present in the 1 index
3 4

Intilize the struct present in the 2 index
5 6

Intilize the struct present in the 3 index
7 8

Intilize the struct present in the 4 index
9 10

Display the Coordinates at index 0 is (1,2)

Display the Coordinates at index 1 is (3,4)

Display the Coordinates at index 2 is (5,6)

Display the Coordinates at index 3 is (7,8)

Display the Coordinates at index 4 is (9,10)
```

15) This program demonstrates how to use arrays of structures to store and retrieve information about the months of the year, including their names and the number of days they have.

```
#include <stdio.h>


struct Month{
        int noOfDays;
        char name[3];
 };


int main(){
        struct Month allMonths[12];

        for(int i = 0; i < 12; i++){
```

```
        printf("Enter The Month Name and the no. of days associated with that month");
        scanf("%s  %d", allMonths[i].name, &allMonths[i].noOfDays);
        printf("\n");
        }

        for(int j = 0; j < 12; j++){
        printf("Name of the Month = %s having %d
\n",allMonths[j].name,allMonths[j].noOfDays);
        }
```

o/p

```
Enter The Month Name and the no. of days associated with that month
Jan 31

Enter The Month Name and the no. of days associated with that month
Feb 28

Enter The Month Name and the no. of days associated with that month
Mar 31
...

Name of the Month = Jan having 31
Name of the Month = Feb having 28
Name of the Month = Mar having 31
...
```