https://www.geeksforgeeks.org/sparse-matrix-representation/

1.

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

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

    printf("\nEnter the number of  integers that wiill be stored : ");
    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;

}
```

2.

```c


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    char *str;

    // initial memory allocation
    str = (char*)malloc(15);
    strcpy(str,"jason");
    printf("String = %s, Address = %u \n",str,str);

    //Reallocating memory
    str = (char*)realloc(str,25);
    strcat(str,".com");
    printf("String = %s, Address = %u \n",str,str);
```

```
    free(str);

    return 0;
}


/*


    Output : String = jason, Address = 26215072
String = jason.com, Address = 26216144



*/
```

3.

```
//Double pointer concept
#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("\n001 i = %d",*ip1);
    printf("\n002 i = %d",**ipp);



    return 0 ;

}
```

4.

```
//Double pointer concept
#include <stdio.h>
#include <stdlib.h>
```

```c
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("\n001 i = %d",*ip1);
    printf("\n002 i = %d",**ipp);


    ***ipp1 = 8;
    printf("\n003 i = %d",***ipp1);


    return 0 ;

}
```

5.

```c
//Double pointer concept
#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("\n001 i = %d",*ip1);
    printf("\n002 i = %d",**ipp);
```

```
    ***ipp1 = 8;
    printf("\n003 i = %d",***ipp1);


    return 0 ;

}
```

6.

```
//ASSIGNMENTS :
/*

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>
int main(){
    int *ptr;
    int n;

    printf("\nEnter the size of the array :");
    scanf("%d",&n);

    int arr[];

    ptr = ((int*)malloc(n+(sizeof(int))));

    if(ptr == NULL){
        printf("\nMemory not allocated\n");
        exit(0);
    }else{
        printf("\nMemory is allocated successfully\n");
    }
    //For population or generating the array
    for(i=0;i<num;i++){
        ptr[i] = i+1;
```

```
    }
    //For displaying the array
    for(i=0;i<num;i++){
        printf("%d,",ptr[i]);
    }
    //This is a standard practice to free the Dynamically allocated memory
    free(ptr);
    return 0 ;




}
```

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>

// Define a structure to store non-zero elements
struct SparseElement {
    int row;
    int column;
    int value;
};

// Function to create the sparse matrix representation
struct SparseElement* createSparseMatrix(int** matrix, int rows, int columns,
int* nonZeroCount) {
    // Count the number of non-zero elements
    *nonZeroCount = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            if (matrix[i][j] != 0) {
                (*nonZeroCount)++;
            }
        }
    }
```

```c
    // Allocate memory for the sparse matrix representation
    struct SparseElement* sparseMatrix = (struct
SparseElement*)malloc(*nonZeroCount * sizeof(struct SparseElement));
    if (sparseMatrix == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    // Store non-zero elements in the sparse matrix
    int k = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            if (matrix[i][j] != 0) {
                sparseMatrix[k].row = i;
                sparseMatrix[k].column = j;
                sparseMatrix[k].value = matrix[i][j];
                k++;
            }
        }
    }

    return sparseMatrix;
}

// Function to print the sparse matrix representation
void printSparseMatrix(struct SparseElement* sparseMatrix, int nonZeroCount) {
    printf("Sparse Matrix Representation:\n");
    printf("Row\tColumn\tValue\n");
    for (int i = 0; i < nonZeroCount; i++) {
        printf("%d\t%d\t%d\n", sparseMatrix[i].row, sparseMatrix[i].column,
sparseMatrix[i].value);
    }
}

int main() {
    int rows, columns;

    // Accept matrix dimensions from the user
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &columns);

    // Dynamically allocate memory for the matrix
    int** matrix = (int**)malloc(rows * sizeof(int*));
    if (matrix == NULL) {
        printf("Memory allocation failed.\n");
```

```
        exit(1);
    }
    for (int i = 0; i < rows; i++) {
        matrix[i] = (int*)malloc(columns * sizeof(int));
        if (matrix[i] == NULL) {
            printf("Memory allocation failed.\n");
            exit(1);
        }
    }

    // Accept the matrix elements from the user
    printf("Enter the elements of the matrix (%dx%d):\n", rows, columns);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    // Create the sparse matrix representation
    int nonZeroCount;
    struct SparseElement* sparseMatrix = createSparseMatrix(matrix, rows,
columns, &nonZeroCount);

    // Print the sparse matrix representation
    printSparseMatrix(sparseMatrix, nonZeroCount);

    // Free the allocated memory
    free(sparseMatrix);
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);

    return 0;
}
```

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.
```

```c
*/
#include <stdio.h>
#include <stdlib.h>

int main() {
    int rows, columns;

    // Step 1: Accept the number of rows and columns
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &columns);

    // Step 2: Dynamically allocate memory for the 2D array
    int** array = (int**)malloc(rows * sizeof(int*));
    if (array == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    for (int i = 0; i < rows; i++) {
        array[i] = (int*)malloc(columns * sizeof(int));
        if (array[i] == NULL) {
            printf("Memory allocation failed.\n");
            exit(1);
        }
    }

    // Step 3: Accept values from the user into the 2D array
    printf("Enter the elements of the matrix (%dx%d):\n", rows, columns);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            scanf("%d", &array[i][j]);
        }
    }

    // Step 4: Print the 2D array in matrix format
    printf("The matrix is:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            printf("%d ", array[i][j]);
        }
        printf("\n");
    }

    // Step 5: Free the allocated memory
    for (int i = 0; i < rows; i++) {
        free(array[i]); // Free each row
    }
```

```
    free(array); // Free the array of row pointers

    return 0;
}
```

9.

```
/*
Problem 2: String Concatenation Using Dynamic Memory
Objective: Create a program that concatenates two strings using dynamic memory
allocation.

**Description:**
Accept two strings from the user.
Use malloc to allocate memory for the first string.
Use realloc to resize the memory to accommodate the concatenated string.
Concatenate the strings and print the result.
Free the allocated memory.




*/


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

int main()
{
    char *str;

    // initial memory allocation
    str = (char*)malloc(15);
    strcpy(str,"jason");
    printf("String = %s, Address = %u \n",str,str);

    //Reallocating memory
    str = (char*)realloc(str,25);
    strcat(str,".com");
    printf("String = %s, Address = %u \n",str,str);

    free(str);

    return 0;
}
```

```
/*


    Output : String = jason, Address = 26215072
String = jason.com, Address = 26216144



*/
```

10.

```
//Structure Prblm stmnt scrnshot
/*
Objective
Create a program to manage student records using C structures.

**Requirements**
Define a Student structure with the following fields:

char name[50]
int rollNumber
float marks
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.
->Create a menu-driven interface with options:

Add Student
Display All Students
Find Student by Roll Number
Calculate Average Marks
Exit
Sample Input/Output

Input:


Enter your choice: 1
Enter name: John Doe
Enter roll number: 101
Enter marks: 85.5

Output:
```

```
Student added successfully!



*/

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

#define MAX_STUDENTS 100

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

// Declare global variables
struct Student students[MAX_STUDENTS];
int studentCount = 0;

// Function to add a student
void addStudent() {
    if (studentCount >= MAX_STUDENTS) {
        printf("Cannot add more students. Maximum limit reached.\n");
        return;
    }

    struct Student newStudent;
    printf("Enter name: ");
    scanf(" %s[^\n]", newStudent.name); // Read name with spaces
    printf("Enter roll number: ");
    scanf("%d", &newStudent.rollNumber);
    printf("Enter marks: ");
    scanf("%f", &newStudent.marks);

    students[studentCount] = newStudent;
    studentCount++;

    printf("Student added successfully!\n");
}

// Function to display all students
void displayStudents() {
    if (studentCount == 0) {
        printf("No students to display.\n");
        return;
    }
```

```c
    printf("\nStudent Records:\n");
    for (int i = 0; i < studentCount; i++) {
        printf("Name: %s, Roll Number: %d, Marks: %.2f\n",
               students[i].name, students[i].rollNumber, students[i].marks);
    }
}

// Function to find a student by roll number
void findStudentByRollNumber() {
    if (studentCount == 0) {
        printf("No students to search.\n");
        return;
    }

    int rollNumber;
    printf("Enter roll number to search: ");
    scanf("%d", &rollNumber);

    for (int i = 0; i < studentCount; i++) {
        if (students[i].rollNumber == rollNumber) {
            printf("Student Found: Name: %s, Marks: %.2f\n",
                   students[i].name, students[i].marks);
            return;
        }
    }

    printf("Student with roll number %d not found.\n", rollNumber);
}

// Function to calculate and display average marks
void calculateAverageMarks() {
    if (studentCount == 0) {
        printf("No students to calculate average.\n");
        return;
    }

    float totalMarks = 0;
    for (int i = 0; i < studentCount; i++) {
        totalMarks += students[i].marks;
    }

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

// Main function
int main() {
    int choice;
```

```
    while (1) {
        printf("\n--- Student Record Management System ---\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:
                displayStudents();
                break;
            case 3:
                findStudentByRollNumber();
                break;
            case 4:
                calculateAverageMarks();
                break;
            case 5:
                printf("Exiting program. Goodbye!\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

11.

```
#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;

}
```

12.

```
#include <stdio.h>
struct Coordinate{
    int x;
    int y;

};



void printCoordinate(struct Coordinate);

int main(){
    struct Coordinate pointA = {5,6};//Initialization
    printCoordinate(pointA);


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

13.

```
//Assignment using compounded literals


#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);
}
```

14.

```
//Array of structure

#include <stdio.h>

struct Coordinate{
    int x;
    int y;
};



int main(){
    struct Coordinate Pnt[5];

    for(int i=0;i<5;i++){
        printf("Initialize 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("Display the coordinates in the %d index \n",i);
        printf("%d %d",Pnt[i].x,Pnt[i].y);
        printf("\n");

    }
```

```
    return 0;
}
```

15.

```
//Initializing an array of structures
#include <stdio.h>

struct Date{
    int date;
    int day;
    int year;
};

int main(){
    struct Date myDates[5] = {{12,10,1975},{12,30,1980},{11,15,2005}};
    for(int i =0;i<3;i++)
    {
        printf(" %d/%d/%d \n",myDates[i].date,myDates[i].day,myDates[i].year);
    }


    return 0;

}
```

16.

```
#include <stdio.h>

struct Date{
    int date;
    int monthint year;
};

int main(){
    struct Date myDates ;

        printf(" %d/%d/%d \n",myDates[i].date,myDates[i].day,myDates[i].year);



    return 0;

}
```

17.

```c
#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;
}
```

18.

```c
#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);
    }
    return 0;
}
```