DS LAB 2

Date: 5/08/24

Name: Utkarsh Yadav

Roll: 23053172

2.1 WAP to create a 1-D array of n elements and perform the following menu based operations using

function.

a. insert a given element at specific position.

b. delete an element from a specific position of the array.

c. linear search to search an element

d. traversal of the array

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

void transverse(int* p, int size){
  printf("The elements in the array are: \n");
  for(int i=0; i<size; i++, p++){
    printf("array[%d]: %d \n", i, *p);
  }
  free(p);
}

void insertElement(int *p, int size){

  int element, index;
  printf("Enter element to insert:  \n");
  scanf("%d", &element);

  printf("Enter index to insert it at:  \n");
  scanf("%d", &index);

  p = realloc(p, sizeof(int)*(size+1));
  int dupSize = size;
  for(int i = size+1; i>index; i--, dupSize--){
        *(p+i) = *(p+dupSize);
  }
  *(p+index) = element;
  printf("Updated Array: \n");
  transverse(p, size+1);
}

void linearSearch(int* p, int size){
    int element;
    printf("Element to search for: ");
    scanf("%d", &element);
    for(int i=0; i<size; i++, p++){
        if(*p == element){
          printf("Element found at position %d", i+1);
          free(p);
          return;
        }
    }
    printf("Element not found");
    free(p);
}

void deleteElementbyIndex(int* p, int size){
  int index;
  printf("Index at which element is to be deleted: ");
  scanf("%d", &index);

  for(int i=size; i>index; index++){
    *(p+index) = *(p+index+1);
  }
  printf("Updated array: \n");
  transverse(p, size-1);

}

int main(){
  int choice;
  int size;
  printf("Enter number of elements: ");
  scanf("%d", &size);

  int* array = (int*)malloc(sizeof(int)*size);

  printf("Enter Elements: \n");
  for(int i = 0; i<size; i++){
    printf("array[%d] = ", i);
    scanf("%d", &array[i]);
  }


  printf("***MENU***\n");
  printf("1. INSERT\n");
  printf("2. DELETE\n");
  printf("3. LINEAR SEARCH\n");
  printf("4. TRANSVERSE\n");
  printf("5. EXIT\n");
  printf("Enter a choice: ");
  scanf("%d", &choice);

  switch(choice){

    case 1:
      insertElement(array, size);
      break;

    case 2:
      deleteElementbyIndex(array, size);
      break;

    case 3:
      linearSearch(array, size);
      break;

    case 4:
      transverse(array, size);
      break;

    case 5:
    free(array);
    printf("exited the menu");
    return 0;
    break;

  }


}
```

```
$ ./menu
Enter number of elements: 5
Enter Elements:
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
***MENU****
1. INSERT
2. DELETE
3. LINEAR SEARCH
4. TRANSVERSE
5. EXIT
Enter a choice: 1
Enter element to insert:
50
Enter index to insert it at:
2
Updated Array:
The elements in the array are:
array[0]: 1
array[1]: 2
array[2]: 50
array[3]: 3
array[4]: 4
array[5]: 5
```

2.2 Write a program to perform the following operations on a given square matrix using functions:

i. Find the no.of nonzero elements

ii. Display upper triangular matrix

iii. Display the elements of just above and below the main diagonal

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

int** allocateMatrix(int rows, int cols);
void fillMatrix(int** array, int rows, int cols);
void printUpperTriangular(int** array, int rows, int cols);
int countNonZeroElements(int** array, int rows, int cols);
void printNonZeroAboveBelowDiagonal(int** array, int rows, int cols);
void freeMatrix(int** array, int rows);

int main() {
    int rows, cols;

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

    int** array = allocateMatrix(rows, cols);

    fillMatrix(array, rows, cols);

    printUpperTriangular(array, rows, cols);

    int nonZeroCount = countNonZeroElements(array, rows, cols);
    printf("Number of non-zero elements: %d\n", nonZeroCount);

    printNonZeroAboveBelowDiagonal(array, rows, cols);
    freeMatrix(array, rows);

    return 0;
}

int** allocateMatrix(int rows, int cols) {
    int** array = (int **)malloc(rows * sizeof(int*));
    for (int i = 0; i < rows; i++) {
        array[i] = (int *)malloc(cols * sizeof(int));
    }
    return array;
}

void fillMatrix(int** array, int rows, int cols) {
    printf("Enter elements: \n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("array[%d][%d] = ", i, j);
            scanf("%d", &array[i][j]);
        }
    }
}

void printUpperTriangular(int** array, int rows, int cols) {
    printf("Upper Triangular Matrix: \n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (j >= i) {
                printf("%d ", array[i][j]);
            } else {
                printf("  ");
            }
        }
        printf("\n");
    }
}

int countNonZeroElements(int** array, int rows, int cols) {
    int count = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (array[i][j] != 0) {
                count++;
            }
        }
    }
    return count;
}

void printNonZeroAboveBelowDiagonal(int** array, int rows, int cols) {
    printf("Non-zero elements above the diagonal:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = i + 1; j < cols; j++) {
            if (array[i][j] != 0) {
                printf("array[%d][%d] = %d\n", i, j, array[i][j]);
            }
        }
    }

    printf("Non-zero elements below the diagonal:\n");
    for (int i = 1; i < rows; i++) {
        for (int j = 0; j < i; j++) {
            if (array[i][j] != 0) {
                printf("array[%d][%d] = %d\n", i, j, array[i][j]);
            }
        }
    }
}


void freeMatrix(int** array, int rows) {
    for (int i = 0; i < rows; i++) {
        free(array[i]);
    }
    free(array);
}

```

```
$ ./sm
Enter rows: 3
Enter columns: 3
Enter elements:
array[0][0] = 1
array[0][1] = 2
array[0][2] = 3
array[1][0] = 4
array[1][1] = 5
array[1][2] = 6
array[2][0] = 7
array[2][1] = 8
array[2][2] = 9
Upper Triangular Matrix:
1 2 3
  5 6
    9
Number of non-zero elements: 9
Non-zero elements above the diagonal:
array[0][1] = 2
array[0][2] = 3
array[1][2] = 6
Non-zero elements below the diagonal:
array[1][0] = 4
array[2][0] = 7
array[2][1] = 8
```

2.3 WAP to represent a given sparse matrix in 3-tuple format using 2-D array.

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

int main(){
  int rows, cols;

  printf("Number of rows & columns: ");
  scanf("%d %d", &rows, &cols);

    int **array = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        array[i] = (int *)malloc(cols * sizeof(int));
    }

    printf("Enter the elements of the Sparse matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &array[i][j]);
        }
    }

    printf("Original Matrix: \n");

     for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf(" %d", array[i][j]);
        }
        printf("\n");
     }


    printf("3 tuple representation of Sparse Matrix: \n");

    printf("\n r c v");

    for(int i=0; i<rows; i++){
      for(int j=0; j<cols; j++){
        if(array[i][j] != 0){
          printf("\n %d %d %d", i, j, array[i][j]);
        }
      }
    }

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

    return 0;
}
```

```
Number of rows & columns: 3 3
Enter the elements of the Sparse matrix:
Element [0][0]: 1
Element [0][1]: 0
Element [0][2]: 0
Element [1][0]: 0
Element [1][1]: 2
Element [1][2]: 0
Element [2][0]: 0
Element [2][1]: 0
Element [2][2]: 3
Original Matrix:
 1 0 0
 0 2 0
 0 0 3
3 tuple representation of Sparse Matrix:

 r c v
 0 0 1
 1 1 2
 2 2 3
```