```
#Hassan Badru

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

//Variable declaration
const int sizeOfMatrix = 3;
const int sizeOfInputString = 50;
int myIndex;
//create token string for conversion
char *myString;

//Create Matrix charateristics
int myRows;
int myColumns;



//Function Declaration *********************
//Initialize the matrix provided with zeros
int myMatrixInit(float myMatrix[][]);
//Declaration for operations to perform
int MultiplyByScalar(float *myScalar, float myMatrix[][], float
myResultMatrix[][]);
int Add2Matrices(float myMatrix1[][], float myMatrix2[][], float
myResultMatrix[][]);
int Substract2Matrices(float myMatrix1[][], float myMatrix2[][], float
myResultMatrix[][]);
int TrasnposeAMatrix(float myMatrix[][], float myResultMatrix[][]);

int initInput(char myInput[]);
int getScalarValue(char myInput[], float *myScalar);
int getAndFillMatrix(char myInput[], float myMatrix[][]);
int cleanMyInput(char myInput[sizeOfInputString]);
int checkMyInput(char myInput[sizeOfInputString]);

int main()
{
   //Create Input String
   char *myInput[sizeOfInputString];

   //Create Matrices and scalar
   float myScalarVariable;
   float myFirstMatrix[sizeOfMatrix][sizeOfMatrix];
   float mySecondMatrix[sizeOfMatrix][sizeOfMatrix];
   float myResultMatrix[sizeOfMatrix][sizeOfMatrix];
   float myScalarValue = 0;
   //Create pointers to my matrices and scalar
    float *pointerM1;
    float *pointerM2;
    float *pointerResultM;
    float *pointerScalar;

   //Set my pointers
    pointerM1 = &myFirstMatrix[0][0];
    pointerM2 = &mySecondMatrix[0][0];
    pointerResultM = &myResultMatrix[0][0];
    pointerScalar = &myScalarValue;

   int repeat = 0;

   //Tell the user about your program: What it does and what is
expected of user
  printf("Matrix Operations!\n\n");
   printf("This application performs operations on user provided
matrices.\n");
   printf("Operations available are: Mutiplication by scalar, addition
of two matrices,");
   printf("                          Substraction of two matricesf,
transpose of a matrice");
   printf("                          and other operations.\n\n");

   //Ask whether to proceed
   printf("Do you want to proceed? '1' for yes, '0' for no: ");
   scanf("%i",&repeat);

   //Proceed and continue processing as long as the user wants
  while(repeat == 1)
    {
        //Initialize input string
        initInput(myInput);
        //Initialize all matrices and scalar variable
        myMatrixInit(myFirstMatrix);
        printf("\n\n");
        myMatrixInit(mySecondMatrix);
        printf("\n\n");
        myMatrixInit(myResultMatrix);
        printf("\n\n");

       //Request the user to enter all values one line, each row
separated by commas
       //Choose how you want to word the request to the user (bound
your request)
       printf("\nYou will be asked to provide input for this
application.");
       printf("\nPlease provide all input in one line for a specific
structure:\n");
```

```
       printf("\nFor scalar values, please enter only a numeric value
and nothing else.");
       printf("\nFor a matrix, please enter all elements of the matrix
in one line.");
       printf("\nEnter each matrix by rows separated with hypens
instead of spaces.");
       printf("\n(i.e. '2.3-7.1-8,1.9,,' is a 3x3 matrix ([2.3 7.1 8],
[1.9 0 0],[0 0 0])");
       printf("\nFor negative elements, please enter 2 hyphens before
the number.");
       printf("\n(i.e. '2--7.1,-1.9,--,' is a 3x3 matrix ([2 -7.1 0],
[1.9 0 0],[0 0 0])");
       printf("\nYou may skip the remaining elements of a row, only if
they are all zeros.");
       printf("\n");

       //Call input function: use scanf("%s", myInput);
       //The 's' argument type after the percent character is for
string input
       //character string (not quoted); char * ; pointing to an array
of characters
       //large enough for the string and a terminating `\0' that will
be added
       //The name of the array is a pointer to the address of that
array

       //Prompt user for scalar input
       printf("\nInput a scalar value: ");
       getScalarValue(myInput, &myScalarValue);

       //Process execution: You must use the core code structure below
       //                   You may rearange it elsewhere in a
function
       //***********
       //Load input into the arrays
       printf("\nInput data for the first matrix:");
       getAndFillMatrix(myInput, myFirstMatrix);

       printf("\n\nInput data for the second matrix:");
       getAndFillMatrix(myInput, mySecondMatrix);

           printf("\n\n");

       //************
       //ask the user what operation he/she wants to do.
       //Choices are: Multiply by scalar, add, substract, transpose
       //Perform only one operation at a time, then prompt user again
for another choice
       //If at any time user would like to enter another set of
matrices and/or scalar value
```

```
       //then prompt for only what needs to be changed
       //At any time, ask the user he/she would like to continue
       //(i.e. continue to prompt user whether to repeat exercise
again)
       //************


      printf("\n\nPlease select what you would like to do from the
following options:\n");
      printf("\nEnter the corresponding number for your choice\n");
      printf(" To multiply Matrix 1 with the scalar value: 1 \n");
      printf(" To multiply Matrix 2 with the scalar value: 2 \n");
      printf(" To add Matrix 1 to Matrix 2 together: 3 \n");
      printf(" To substract Matrix 2 from Matrix 1: 4 \n");
      printf(" To substract Matrix 1 from Matrix 2: 5 \n");
      printf(" To get the transpose of Matrix 1: 6 \n");
      printf(" To get the transpose of Matrix 2: 7 \n");
      printf(" To re-enter Matrix 1: 8 \n");
      printf(" To re-enter Matrix 2: 9 \n");
      printf(" To re-enter the scalar value: 10 \n");
      printf(" To clear all variables and start over: 11 \n");

      int operationChoice;
      printf("\nWhat operation would you like to do?: ");
      scanf("%i",&operationChoice);

      //OPERATIONS
      switch(operationChoice)
      {
        case 1: MultiplyByScalar(pointerScalar, myFirstMatrix,
myResultMatrix);
        break;
        case 2: MultiplyByScalar(pointerScalar, mySecondMatrix,
myResultMatrix);
        break;
                case 3: Add2Matrices(myFirstMatrix, mySecondMatrix,
myResultMatrix);
                break;
                case 4: Substract2Matrices(myFirstMatrix,
mySecondMatrix, myResultMatrix);
                break;
                case 5: Substract2Matrices(mySecondMatrix,
myFirstMatrix, myResultMatrix);
                break;
                case 6: TrasnposeAMatrix(myFirstMatrix,
myResultMatrix);
                break;
                case 7: TrasnposeAMatrix(mySecondMatrix,
myResultMatrix);
                break;
                case 8: getAndFillMatrix(myInput, myFirstMatrix);
                break;
                case 9: getAndFillMatrix(myInput, mySecondMatrix);
                break;
                case 10: getScalarValue(myInput, &myScalarValue);
                break;
                case 11: cleanMyInput(myInput);
                break;
          }
     printf("Do you want to start over (Press 1 for yes)");
     scanf("%i", &repeat);
   }

   return 0;
}

//Function Definitions ****************
//Passing by address
int myMatrixInit(float myMatrix[sizeOfMatrix][sizeOfMatrix])
{
   //Complete with code
   for(myRows = 0; myRows < sizeOfMatrix; myRows++){
      for(myColumns = 0; myColumns < sizeOfMatrix; myColumns++){
           myMatrix[myRows][myColumns] = 0;
      }
  }
     printMyMatrix(myMatrix);
  return 1;
}

//*****************
//insert other function definition here

int MultiplyByScalar(float *myScalar, float myMatrix[sizeOfMatrix]
[sizeOfMatrix], float myResultMatrix[sizeOfMatrix][sizeOfMatrix]){
  //Multiply a matrix by a scalar
  for(myRows = 0; myRows < sizeOfMatrix; myRows++){
      for(myColumns = 0; myColumns < sizeOfMatrix; myColumns++){
           myResultMatrix[myRows][myColumns] = (*myScalar) *
(myMatrix[myRows][myColumns]);
      }
  }

  //Print Operation
  //iterate to print
  /*for(myRows = 0; myRows < sizeOfMatrix; myRows++){
      if(myRows == 0)
          printf("%f \t * \t", myScalar);
      else
          printf("\t\t\t");
      for(myColumns = 0; myColumns < sizeOfMatrix; myColumns++){
           printf("%f \t", myResultMatrix[myRows][myColumns]);
      }
      if(myRows == ceil(sizeOfMatrix/2))
          printf(" = ");

      printf("\n");
  } */

  //Print the result at each stage
  printMyMatrix(myResultMatrix);

  //return myResultMatrix;
  return 1;
}

int Add2Matrices(float myMatrix1[sizeOfMatrix][sizeOfMatrix], float
myMatrix2[sizeOfMatrix][sizeOfMatrix], float
myResultMatrix[sizeOfMatrix][sizeOfMatrix]){
  //Add 2 matrices together
  for(myRows = 0; myRows < sizeOfMatrix; myRows++){
      for(myColumns = 0; myColumns < sizeOfMatrix; myColumns++){
           myResultMatrix[myRows][myColumns] = myMatrix1[myRows]
[myColumns] + myMatrix2[myRows][myColumns];
      }
  }

  //Print the result at each stage
  printMyMatrix(myResultMatrix);

  //return myResultMatrix;
  return 1;
}

int Substract2Matrices(float myMatrix1[sizeOfMatrix][sizeOfMatrix],
float myMatrix2[sizeOfMatrix][sizeOfMatrix], float
myResultMatrix[sizeOfMatrix][sizeOfMatrix]){
  //Add 2 matrices together
  for(myRows = 0; myRows < sizeOfMatrix; myRows++){
      for(myColumns = 0; myColumns < sizeOfMatrix; myColumns++){
           myResultMatrix[myRows][myColumns] = myMatrix1[myRows]
[myColumns] - myMatrix2[myRows][myColumns];
      }
  }

  //Print the result at each stage
  printMyMatrix(myResultMatrix);

  //return myResultMatrix;
  return 1;
```

```
}

int TrasnposeAMatrix(float myMatrix[sizeOfMatrix][sizeOfMatrix], float
myResultMatrix[sizeOfMatrix][sizeOfMatrix]){
  //Add 2 matrices together
  for(myRows = 0; myRows < sizeOfMatrix; myRows++){
      for(myColumns = 0; myColumns < sizeOfMatrix; myColumns++){
           myResultMatrix[myColumns][myRows] = myMatrix[myRows]
[myColumns];
      }
  }
   printMyMatrix(myResultMatrix);
  //return myResultMatrix;
  return 1;
}

//Print a single matrix
int printMyMatrix(float myMatrix[sizeOfMatrix][sizeOfMatrix]){
  //Print the result at each stage
  //iterate to print
  for(myRows = 0; myRows < sizeOfMatrix; myRows++){
      for(myColumns = 0; myColumns < sizeOfMatrix; myColumns++){
           printf("%.2f \t", myMatrix[myRows][myColumns]);
      }
      printf("\n");
  }
}

int initInput(char myInput[sizeOfInputString]){
   for(myIndex = 0; myIndex < sizeOfInputString; myIndex++){
       myInput[myIndex] = '\0';
       char check = myInput[myIndex];
   }
   return 1;
}


int getScalarValue(char myInput[sizeOfInputString], float *myScalar){
      //Collect input from user: initialize input array (load all
element with \0)\
      if(initInput(myInput))
          printf("\nInput is initialized\n\n");

      //Collect Scalar value
      printf("\nPlease enter a scalar value: ");
      scanf("%s", myInput);
        //Replace all hyphens with spaces
        while((myString = strpbrk(myInput, "-")) != NULL)
                *myString = ' ';
      //Check input: must be all numbers, hyphen for space and commas, 
No other character
      //             if not, then print error and prompt user again
      while(checkMyInput(myInput) == 0){
           printf("\nSorry, but your input does not follow the 
prescribed format. Please try again.");
           //init input string
           initInput(myInput);

          //Prompt user for input
          //Collect Scalar value
          printf("\nPlease enter a scalar value\n\n");
          scanf("%s", myInput);
          //Replace all hyphens with spaces
            while((myString = strpbrk(myInput, "-")) != NULL)
                    *myString = ' ';
      }

      //pin onto the element location, then convert to float

       double atof(const char *str);
       //The string pointed to by the argument str is converted to a 
floating-point number (type double). Any
       //initial whitespace characters are skipped (space, tab, 
carriage return, new line, vertical tab, or formfeed).
       //The number may consist of an optional sign, a string of 
digits with an optional decimal character, and an
       //optional e or E followed by a optionally signed exponent. 
Conversion stops when the first unrecognized
       //character is reached.

      myString = &myInput[0];
      *myScalar = atof(myString);

}

int getAndFillMatrix(char myInput[sizeOfInputString], float 
myMatrix[sizeOfMatrix][sizeOfMatrix]){
       //Collect input from user: initialize input array (load all 
element with \0)\
       initInput(myInput);

       //Collect Scalar value
       printf("\nPlease provide a matrix according to instructions 
provided: ");
       //scanf("%s49[^\n]", myInput);
       scanf("%s", myInput);
       cleanMyInput(myInput);
        printf("\n");
       //Check input: must be all numbers, hiphen for space and 
```

```
commas, No other character
       //             if not, then print error and prompt user again
       while(checkMyInput(myInput) == 0){
            printf("\nSorry, but your input does not follow the
prescribed format. Please try again.");
            //init input string
            initInput(myInput);

           //Prompt user for input
           //Collect Scalar value
           printf("\nPlease provide a matrix according to instructions
provided: ");
           //scanf("%s49[^\n]", myInput);
           scanf("%s", myInput);
           cleanMyInput(myInput);
           printf("\n");
       }

       //pin onto the element location, determine the number of
character that make up the element, then convert to float

        //char *strpbrk(const char *str1, const char *str2);
        //Finds the first character in the string str1 that matches
any character specified in str2.
        //A pointer to the location of this character is returned. A
null pointer is returned if no character in str2
        //exists in str1.
       myString = &myInput[0];

       for(myRows = 0; myRows < sizeOfMatrix; myRows++){
           //if no pointer then continue
           if(myString == NULL)
               break;
           if(myString[0] == ',') //End of this row (i.e. no other
data for the row or remaining elements are all zeroes)
                   myString++;

           for(myColumns = 0; myColumns < sizeOfMatrix; myColumns++){
               while(myString[0] == ' ') //iterate through string
until you get to a number or end of row
                          myString++;
               //Get next data value
               //Check input string until you get a number
               if(myString[0] == ',') //End of this row (i.e. no other
data for the row or remaining elements are all zeroes)
                   break;

               myMatrix[myRows][myColumns] = (float) atof(myString);

               if((myString = strpbrk(myString, " ,")) == NULL)
```

```
                   break;
           }
           if(myString == NULL)
               break;
           else{//Cover the case where user enters too many matrix
columns
               while(myString[0] == ' ') //iterate through string
until you get to a number or end of row
                    myString++;
                    char temp; //ADDED
               if(myString[0] != ',' && ((temp = strpbrk(myString,
" ,")) != NULL)){//End of this row (i.e. if other data for this row
then report an error)
                   printf("\nThere is an error in your matrix entry.
You entered too many columns!\n");
                   if(myString[0] != ',')
                       if((myString = strpbrk(myString, " ,")) ==
NULL)
                                break;
               }
           }
       }
       if(myString != NULL)
       {//Cover the case where user enters too many matrix rows
           while(myString[0] == ' ') //iterate through string until
you get to a number or end of row
                myString++;
           if(myString[0] == ',') //End of this row (i.e. no other
data for the row)
                myString++;
           if((myString = strpbrk(myString, " ,")) != NULL) //Suppose
to be end of this row (i.e. if other data for this/other row(s) then
report an error)
                   printf("\nThere is an error in your matrix entry.
You entered too many rows\n");
       }


       printf("\nThe Matrix entered is: \n");
       //Print the matrix from user input
       printMyMatrix(myMatrix);

       return 1;
}

int cleanMyInput(char myInput[sizeOfInputString]){
       myString = &myInput[0];
       char *temp;
       //remove hypen and replace with spaces
       while((myString = strpbrk(myString,"-"))!=NULL){
            temp = myString++;
            if(temp[0] == '-' && myString[0] == '-'){
                //spare the negative signs for negative numbers
                while(temp[0] == '-' && myString[0] == '-'){
                    temp[0] = ' ';
                    temp = myString++;
                }
            }
            else temp[0] = ' ';
            char tempSingleChar;
            tempSingleChar = myString[0];
            if(!isdigit(tempSingleChar))//this is to account for 
spaces put between numbers and the comma
                 if(myString[0] != '.')
                      temp[0] = ' ';
       }
           return 1;
}


int checkMyInput(char myInput[sizeOfInputString]){
    int Success = 0;
    for(myIndex = 0; myIndex < sizeOfInputString; myIndex++){
        if(myInput[myIndex] == '\0') break;
        //isdigit for a digit (0 to 9)
        //isspace for a whitespace character (space, tab, carriage 
return, new line, vertical tab, or formfeed)
        if(isspace((int)myInput[myIndex]) || myInput[myIndex] == '-' 
|| myInput[myIndex] == ',' || myInput[myIndex] == '.' || 
isdigit(myInput[myIndex]))
             Success = 1;
    }
    if(myInput[0] == ',')
        Success = 0;

    int rowCount = 0, colCount = 0;
    myString = &myInput[0];
    while(myString[0] == ' ') //iterate through string until you get 
to a number or end of row
         myString++;
    //iterate to check number of rows and columns provided start with 
r = 0 and c = 0
    while((myString = strpbrk(myString," ,"))!=NULL && Success == 1)
{ //if test fails, then no need to continue.
            while(myString[0] == ' ') //iterate through string until 
you get to a number or end of row
                 myString++;
            if(myString[0] == ','){ //if end of row, then increment 
row count and reset col count
                rowCount++;
                colCount = 0;
                myString++;
                while(myString[0] == ' ') //iterate through string
until you get to a number or end of row
                     myString++;
            } else colCount++; //increment col count
            if(rowCount >= sizeOfMatrix || colCount >=
sizeOfMatrix) //if input breaks required matrix size, then return Fail
                if(rowCount >= sizeOfMatrix && colCount > 0)
                     Success = 0;
                else if(colCount >= sizeOfMatrix)
                          Success = 0;
            else myString++;
    }
    return Success;

    //return 1; // if success
    //return 0; //if fails
}
```