

```
\documentclass[14pt]{article}
```

```
\usepackage{listings}
```

```
\usepackage{color}
```

```
\usepackage{graphicx}
```

```
\usepackage{setspace}
```

```
\definecolor{dkgreen}{rgb}{0,0.6,0}
```

```
\definecolor{gray}{rgb}{0.5,0.5,0.5}
```

```
\definecolor{mauve}{rgb}{0.58,0,0.82}
```

```
\lstset{frame=tb,
```

```
language=C,
```

```
aboveskip=3mm,
```

```
belowskip=3mm,
```

```
showstringspaces=false,
```

```
columns=flexible,
```

```
basicstyle={\small\ttfamily},
```

```
numbers=none,
```

```
numberstyle=\tiny\color{gray},
```

```
keywordstyle=\color{blue},
```

```
commentstyle=\color{dkgreen},
```

```
stringstyle=\color{mauve},
```

```
breaklines=true,
```

```
breakatwhitespace=true
```

```
tabsize=3
```

}

\begin{document}

\title{\huge Sparse Matrices}

\date{\today}

\maketitle

\begin{center}

\vspace{30 mm}

\title{\huge Student: Preda Iulian Octavian}

\\\vspace{10 mm}

\title{\huge Computer Programming: Programming Techniques(English Language)}

\\\vspace{10 mm}

\title{\huge Group CEN1.2 B}

\\\vspace{10 mm}

\title{\huge 1st Year}

\date{}

\maketitle

\newpage

\section*{Introduction}

\vspace{20 mm}

In numerical analysis and computer science, a sparse matrix or sparse array is a matrix in which most of the elements are zero. Large sparse matrices often appear in scientific or engineering applications when solving partial differential equations.

\\\vspace{10 mm}

When storing and manipulating sparse matrices on a computer, it is beneficial and often necessary to use specialized algorithms and data structures that take advantage of the sparse structure of the matrix. Operations using standard dense-matrix structures and algorithms are slow and inefficient when applied to large sparse matrices as processing and memory are wasted on the zeroes. Sparse data is by nature more easily compressed and thus require significantly less storage. Some very large sparse matrices are hard to manipulate using standard dense-matrix algorithms.\\

\newpage

\end{center}

\section*{Problem statement}

Create a library for operations with sparse matrices. The library should provide at least addition and multiplication.

\begin{center}\includegraphics[height=3.5in, width = 3.5in]{sparse_matrix.png}

\end{center}

\newpage

\section*{Pseudocode}

For each operation required a function has been designed.

\\Here are the important functions:

\begin{lstlisting}

```
int readMatrix(int mat[1001000][3], int noRows, int noCols){  
    printf("Insert the elements of the matrix:\n");  
    int row,col,len=0;  
    for( row = 1; row <= noRows; row ++){  
        for( col = 1; col<= noCols; col ++){  
  
            int e;  
            scanf("%d",&e);  
            if( e != 0){  
                len ++;  
                mat[len][0] = row;  
                mat[len][1] = col;  
                mat[len][2] = e;  
            }  
        }  
    }  
    mat[len+1][0] = noRows + 1;
```

```

    mat[len+1][1] = noRows + 1;

    return len;
}

```

\end{lstlisting}

\begin{lstlisting}

```

int add(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]){

    if( noRows1 != noRows2 || noCols1 != noCols2 ){

        printf("Sorry, but you cannot add the matrices as they do not have the same sizes");

        isAdded = 0;

        return 0;

    }

    else{

        resLen = len1 + len2;

        int it1, it2; //the iterators for the matrices

        it1 = it2 = 1;

        while(it1 <= len1 || it2 <= len2){

            if(mat1[it1][0] < mat2[it2][0])

                it1 ++;

            else if(mat1[it1][0] > mat2[it2][0])

                it2 ++;

            else{

                if(mat1[it1][1] < mat2[it2][1])

                    it1 ++;

```

```

else if(mat1[it1][1] > mat2[it2][1])
    it2 ++;
else{
    if(mat1[it1][2] + mat2[it2][2] == 0)
        resLen -= 2;
    else
        resLen --;
    it1 ++;
    it2 ++;
}
}
}

it1 = it2 = 1;
int resIt = 1;
while(it1 <= len1 || it2 <= len2){
    if(mat1[it1][0] < mat2[it2][0]){
        resMat[resIt][0] = mat1[it1][0];
        resMat[resIt][1] = mat1[it1][1];
        resMat[resIt][2] = mat1[it1][2];
        it1 ++;
        resIt ++;
    }
    else if(mat1[it1][0] > mat2[it2][0]){
        resMat[resIt][0] = mat2[it2][0];
        resMat[resIt][1] = mat2[it2][1];

```

```
resMat[resIt][2] = mat2[it2][2];

it2 ++;

resIt ++;
}

else{

    if(mat1[it1][1] < mat2[it2][1]){

        resMat[resIt][0] = mat1[it1][0];

        resMat[resIt][1] = mat1[it1][1];

        resMat[resIt][2] = mat1[it1][2];

        it1 ++;

        resIt ++;

    }

    else if(mat1[it1][1] > mat2[it2][1]){

        resMat[resIt][0] = mat2[it2][0];

        resMat[resIt][1] = mat2[it2][1];

        resMat[resIt][2] = mat2[it2][2];

        it2 ++;

        resIt ++;

    }

    else{

        if(mat1[it1][2] + mat2[it2][2] == 0);

        else{

            resMat[resIt][0] = mat1[it1][0];

            resMat[resIt][1] = mat1[it1][1];

            resMat[resIt][2] = mat1[it1][2] + mat2[it2][2];
```



```

        reslt++;

    }

    it1++;

    it2++;

}

}

}

return resLen;

}

}

\end{lstlisting}

```

```

\begin{lstlisting}

```

```

int matMult(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]){

    int row1,row2,col,it1,it2;

    if( noCols1 != noRows2 ){

        printf("Sorry, but you cannot multiply the matrices as they do not have corresponding sizes");

        isMulted = 0;

        return 0;

    }

    else{

        for( row1 = 1; row1 <= noRows1; row1 ++){

            for( row2 = 1; row2 <= noRows2; row2 ++){

                int sum = 0;

                for( col = 1; col <= noCols1; col ++){

```

```

for( it1 = 1; it1 <= len1; it1 ++){

    if(row1 == mat1[it1][0] && col == mat1[it1][1]){

        for( it2 = 1; it2 <= len2; it2 ++){

            if(col == mat2[it2][0] && row2 == mat2[it2][1])

                sum += mat1[it1][2] * mat2[it2][2];

        }

    }

}

if(sum != 0){

    resLen ++;

    resMat[resLen][0] = row1;

    resMat[resLen][1] = row2;

    resMat[resLen][2] = sum;

}

}

return resLen;

}

```

\end{lstlisting}

\begin{lstlisting}

```

void intMult(int len, int mat[1001000][3], int fact){

    int it;

```

```

for( it = 1; it <= len; it ++){

    mat[it][2] *= fact;

}

}

```

\end{lstlisting}

\begin{lstlisting}

```

int subtr(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]){

    if (subtr1 == 1){

        intMult(len2, mat2, -1);

        isAdded = 1;

        resLen = add(len1, mat1, len2, mat2, resMat);

        intMult(len2, mat2, -1);

    }else{

        intMult(len1, mat1, -1);

        isAdded = 1;

        resLen = add(len1, mat1, len2, mat2, resMat);

        intMult(len1, mat1, -1);

    }

    return resLen;

}

```

\end{lstlisting}

```
\begin{lstlisting}

void printMat(int mat[1001000][3]){

    int row,col;

    int reslt = 1;

    for( row = 1; row <= noRows1; row ++){

        for( col = 1; col <= noCols1; col ++){

            if( mat[reslt][0] == row && mat[reslt][1] == col ){

                printf("%d ", mat[reslt][2]);

                reslt ++;

            }

            else

                printf("0 ");

        }

        printf("\n");

    }

}
```

```
\end{lstlisting}
```

\newpage

\section*{Application Design}

\begin{quote}

The library contains the header functions.h contains the function prototypes to compute the required operations. These are all of them:\\

\$-void int readMatrix(int mat[1001000][3], int noRows, int noCols)\\

\$-int add(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3])\\

\$-int matMult(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3])\\

\$-void intMult(int len, int mat[1001000][3], int fact)\\

\$-void int subtr(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3])\\

\$-void void printMat(int Mat[1001000][3])\\

\end{quote}

\begin{quote}

I used multiple files for for the source code for the purpose of modularity. Every file contains a function linked to the library functions.h\\

Function readMatrix(int mat[1001000][3], int noRows, int noCols)) includes some steps, like\\

```
\begin{enumerate}
```

```
\item read the number of rows and columns of the current matrix
```

```
\item go through the matrix
```

```
\item read the current element e and if it is not 0, we store it
```

```
\end{enumerate}
```

```
\end{quote}
```

```
\begin{quote}
```

Function `add(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3])` includes some steps, like:\\

```
\begin{enumerate}
```

```
\item check that the 2 matrices have the same size, as otherwise we cannot add them
```

```
\item finding the resulting matrix size, as it is the sum of the sizes of the other 2 matrices
```

\$\$ number of common elements

```
\item check if two elements have the same position in the two matrices, and if we do, we check if their sum is 0 in order to decrease the number of elements by 2 (as we need to exclude this
```

element from the resulting matrix)

```
\item we check if the elements of the original matrices are on the same row and if not we just add the current element to the result matrix
```

```
\item check if they are on the same column
```

```
\item if they are located at the same position, we add to the result matrix their sum
```

```
\end{enumerate}
```

```
\end{quote}
```

```
\begin{quote}
```

Function `int matMult(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3])` includes some steps, like:\\

```
\begin{enumerate}
```

```
\item check that the 2 matrices have sizes that allow multiplication(i.e. the number of collumns from the first matrix  $=$  the number of rown from the second matrix)
```

```
\item determine the size of the resulting matrix
```

```
\item check if the current element ( given by row1 and col) is in the first matrix
```

```
\item we add to the current sum the product of the values of the 2 elements
```

```
\item we add the current element to the resulting matrix
```

```
\end{enumerate}
```

```
\end{quote}
```

```
\begin{quote}
```

```
Function intMult(int len, int mat[1001000][3], int fact) multiplies the value of the current element by a given integer
```

```
\end{quote}
```

```
\begin{quote}
```

```
void int subtr(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]) is used to subtract a matrix form another
```

```
\end{quote}
```

```
\begin{quote}
```

```
void printMat(int Mat[1001000][3]) is used to print to the screen the matrix
```

```
\end{quote}
```

```
\begin{quote}
```

```
The most important file, main.c , has one purpose, to call all the other function in a designed order.\\
```

```
First, the user introduces the two matrices the programe will perform operation on.
```

```
Then the program sums the matrices up, subrtact them and multiply them.\\
```

```
\end{quote}
```

\cleardoublepage

\section*{Source Code}

\begin{lstlisting}

//-----functions.h-----

int noRows1,noCols1,noRows2,noCols2; //the sizes of the 2 matrices

int mat1[1001000][3],mat2[1001000][3]; //the 2 matrices after they are

//compressed into the following pattern: row, column, value

int resMat[1001000][3]; //the resulting matrix

int resLen; //the length of the resulting matrix

int len1,len2; //the lengths of the 2 matrices after compression

int isAdded,isMulted; //to check that matrix add or mult has taken place

int fact; //the integer by which a matrix is multiplied

int subtr1; //the integer used for choosing how to subtract matrices


```

int readMatrix(int mat[1001000][3], int noRows, int noCols);

int add(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]);

int matMult(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]);

void intMult(int len, int mat[1001000][3], int fact);

int subtr(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]);

void printMat(int Mat[1001000][3]);

```

\end{lstlisting}

\begin{lstlisting}

//-----readMat.c-----

#include "functions.h"

```

int readMatrix(int mat[1001000][3], int noRows, int noCols){

```

```

    ///first, we read the number of rows and columns of the current matrix

```

```

    ///now we go through the matrix

```

```

    printf("Insert the elements of the matrix:\n");

```

```

    int row,col,len=0;

```

```

    for( row = 1; row <= noRows; row ++){

```

```

        for( col = 1; col<= noCols; col ++){

```

```

            ///we read the current element \var e and if it is not 0, we store it

```

```

            int e;

```

```

            scanf("%d",&e);

```

```

            if( e != 0){

```

```

                len ++;

```

```

                mat[len][0] = row;

```

```

        mat[len][1] = col;

        mat[len][2] = e;
    }

}

}

mat[len+1][0] = noRows + 1;
mat[len+1][1] = noRows + 1;

return len;
}

```

//-----readMat.c-----

```
#include "functions.h"
```

```
int add(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]){
```

```
    ///the first thing we need to do is to check that the 2 matrices have the
```

```
    ///same size, as otherwise we cannot add them
```

```
    if( noRows1 != noRows2 || noCols1 != noCols2 ){
```

```
        printf("Sorry, but you cannot add the matrices as they do not have the same sizes");
```

```
        isAdded = 0;
```

```
        return 0;
```

```
    }
```

```
    ///now we need to calculate the resulting matrix
```

```
    ///first we need to know its size, as it is the sum of the sizes of
```

```
    ///the other 2 matrices - number of common elements
```

```
    else{
```

```
        resLen = len1 + len2;
```

```

int it1, it2; //the iterators for the matrices

it1 = it2 = 1;

while(it1 <= len1 || it2 <= len2){

    ///first we check if they are on the same row

    if(mat1[it1][0] < mat2[it2][0])

        it1 ++;

    else if(mat1[it1][0] > mat2[it2][0])

        it2 ++;

    else{

        ///now we check if they are on the same column

        if(mat1[it1][1] < mat2[it2][1])

            it1 ++;

        else if(mat1[it1][1] > mat2[it2][1])

            it2 ++;

        else{

            ///we know that we have 2 elements located at the same position

            ///now we have to check if their sum is 0 in order to decrease

            ///the number of elements by 2(as we need to exclude this

            ///element from the new matrix), or not, to decrease it by 1

            if(mat1[it1][2] + mat2[it2][2] == 0)

                resLen -= 2;

            else

                resLen --;

            it1 ++;

            it2 ++;

```

```

    }

}

}

///now we do the addition itself

it1 = it2 = 1;

int resIt = 1; //the iterator tfor the resulting matrix

while(it1 <= len1 || it2 <= len2){

    ///first we check if the elements of the original matrices are on

    ///the same row and if not we just add the current element to the

    ///result matrix

    if(mat1[it1][0] < mat2[it2][0]){

        resMat[resIt][0] = mat1[it1][0];

        resMat[resIt][1] = mat1[it1][1];

        resMat[resIt][2] = mat1[it1][2];

        it1 ++;

        resIt ++;

    }

    else if(mat1[it1][0] > mat2[it2][0]){

        resMat[resIt][0] = mat2[it2][0];

        resMat[resIt][1] = mat2[it2][1];

        resMat[resIt][2] = mat2[it2][2];

        it2 ++;

        resIt ++;

    }

    else{

```

```

///now we check if they are on the same column

if(mat1[it1][1] < mat2[it2][1]){

    resMat[resIt][0] = mat1[it1][0];

    resMat[resIt][1] = mat1[it1][1];

    resMat[resIt][2] = mat1[it1][2];

    it1 ++;

    resIt ++;

}

else if(mat1[it1][1] > mat2[it2][1]){

    resMat[resIt][0] = mat2[it2][0];

    resMat[resIt][1] = mat2[it2][1];

    resMat[resIt][2] = mat2[it2][2];

    it2 ++;

    resIt ++;

}

else{

    ///as they are located as the same position, we add to the

    ///resulting matrix their sum

    if(mat1[it1][2] + mat2[it2][2] == 0);

        ///if it is 0 we do not add anything

    else{

        resMat[resIt][0] = mat1[it1][0];

        resMat[resIt][1] = mat1[it1][1];

        resMat[resIt][2] = mat1[it1][2] + mat2[it2][2];

        resIt ++;
    }
}

```

```

        }

        it1++;

        it2++;

    }

}

}

return resLen;

}

}

```

//-----matMult.c-----

```
#include "functions.h"
```

```
int matMult(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]){
```

```
    ///the first thing we need to do is to check that the 2 matrices have
```

```
    ///sizes that allow multiplication(noCols1 = noRows2)
```

```
    int row1,row2,col,it1,it2;
```

```
    if( noCols1 != noRows2 ){
```

```
        printf("Sorry, but you cannot multiply the matrices as they do not have corresponding sizes");
```

```
        isMulted = 0;
```

```
        return 0;
```

```
    }
```

```
    else{
```

```
        ///now we need to determine the size of the resulting matrix
```

```
        for( row1 = 1; row1 <= noRows1; row1 ++){
```

```

for( row2 = 1; row2 <= noRows2; row2 ++){

    int sum = 0;

    ///we compute the value of the current element

    for( col = 1; col <= noCols1; col ++){

        for( it1 = 1; it1 <= len1; it1 ++){

            ///now we need to check if the current element, given by

            ///row1 and col is in the first matrix

            if(row1 == mat1[it1][0] && col == mat1[it1][1]){

                for( it2 = 1; it2 <= len2; it2 ++){

                    ///now we check if it is in the second matrix

                    ///given by row2 and col

                    if(col == mat2[it2][0] && row2 == mat2[it2][1])

                        ///we add to the current sum the product of

                        ///the values of the 2 elements

                        sum += mat1[it1][2] * mat2[it2][2];

                }

            }

        }

    }

    ///we add 1 to the size of the result matrix if the sum is != 0

    if(sum != 0){

        resLen ++;

        resMat[resLen][0] = row1;

        resMat[resLen][1] = row2;

        resMat[resLen][2] = sum;

```

```
    }  
    }  
}
```

```
    ///finally, we compute the actual multiplication  
    return resLen;  
}  
}
```

```
//-----scalarMult.c-----  
  
#include "functions.h"  
  
void intMult(int len, int mat[1001000][3], int fact){  
    ///the only thing we need to do here is to multiply the value of the  
    ///current element by fact  
  
    int it;  
  
    for( it = 1; it <= len; it ++){  
        mat[it][2] *= fact;  
    }  
}
```

```
//-----subtract.c-----  
  
#include "functions.h"  
  
int subtr(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]){
```



```

///this is basically multiplying one matrix with -1 and then adding them
if (subtr1 == 1){
    intMult(len2, mat2, -1);

    isAdded = 1;

    resLen = add(len1, mat1, len2, mat2, resMat);

    intMult(len2, mat2, -1);
}else{

    intMult(len1, mat1, -1);

    isAdded = 1;

    resLen = add(len1, mat1, len2, mat2, resMat);

    intMult(len1, mat1, -1);
}

return resLen;
}

```

//-----printMat.c-----

```

#include "functions.h"

void printMat(int mat[1001000][3]){

    int row,col;

    int resIt = 1;

    for( row = 1; row <= noRows1; row ++){

        for( col = 1; col <= noCols1; col ++){

            if( mat[resIt][0] == row && mat[resIt][1] == col ){

                printf("%d ", mat[resIt][2]);

                resIt ++;
            }
        }
    }
}

```

```

    }

    else

        printf("0 ");

    }

    printf("\n");

}

}

```

\end{lstlisting}

\begin{lstlisting}

//-----main.c-----

//-----readMat.c-----

#include "functions.h"

int readMatrix(int mat[1001000][3], int noRows, int noCols){

///first, we read the number of rows and columns of the current matrix

///now we go through the matrix

printf("Insert the elements of the matrix:\n");

int row,col,len=0;

for(row = 1; row <= noRows; row ++){

for(col = 1; col<= noCols; col ++){

///we read the current element \var e and if it is not 0, we store it

int e;

scanf("%d",&e);

```

        if( e != 0){

            len ++;

            mat[len][0] = row;

            mat[len][1] = col;

            mat[len][2] = e;

        }

    }

}

mat[len+1][0] = noRows + 1;

mat[len+1][1] = noRows + 1;

return len;

}

```

```

//-----readMat.c-----

```

```

#include "functions.h"

```

```

int add(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]){

```

```

    ///the first thing we need to do is to check that the 2 matrices have the

```

```

    ///same size, as otherwise we cannot add them

```

```

    if( noRows1 != noRows2 || noCols1 != noCols2 ){

```

```

        printf("Sorry, but you cannot add the matrices as they do not have the same sizes");

```

```

        isAdded = 0;

```

```

        return 0;

```

```

    }

```

```

    ///now we need to calculate the resulting matrix

```

```

    ///first we need to know its size, as it is the sum of the sizes of

```



```

        resLen--;

        it1++;

        it2++;

    }

}

}

///now we do the addition itself

it1 = it2 = 1;

int resIt = 1; //the iterator tfor the resulting matrix

while(it1 <= len1 || it2 <= len2){

    ///first we check if the elements of the original matrices are on

    ///the same row and if not we just add the current element to the

    ///result matrix

    if(mat1[it1][0] < mat2[it2][0]){

        resMat[resIt][0] = mat1[it1][0];

        resMat[resIt][1] = mat1[it1][1];

        resMat[resIt][2] = mat1[it1][2];

        it1++;

        resIt++;

    }

    else if(mat1[it1][0] > mat2[it2][0]){

        resMat[resIt][0] = mat2[it2][0];

        resMat[resIt][1] = mat2[it2][1];

        resMat[resIt][2] = mat2[it2][2];

        it2++;

```

```

    reslt++;
}
else{
    ///now we check if they are on the same column
    if(mat1[it1][1] < mat2[it2][1]){
        resMat[reslt][0] = mat1[it1][0];
        resMat[reslt][1] = mat1[it1][1];
        resMat[reslt][2] = mat1[it1][2];

        it1++;
        reslt++;
    }
    else if(mat1[it1][1] > mat2[it2][1]){
        resMat[reslt][0] = mat2[it2][0];
        resMat[reslt][1] = mat2[it2][1];
        resMat[reslt][2] = mat2[it2][2];

        it2++;
        reslt++;
    }
    else{
        ///as they are located as the same position, we add to the
        ///resulting matrix their sum
        if(mat1[it1][2] + mat2[it2][2] == 0);

        ///if it is 0 we do not add anything
        else{
            resMat[reslt][0] = mat1[it1][0];

```

```

        resMat[resIt][1] = mat1[it1][1];

        resMat[resIt][2] = mat1[it1][2] + mat2[it2][2];

        resIt++;

    }

    it1++;

    it2++;

}

}

}

return resLen;

}

}

```

```

//-----matMult.c-----

```

```

#include "functions.h"

```

```

int matMult(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]){

    ///the first thing we need to do is to check that the 2 matrices have

    ///sizes that allow multiplication(noCols1 = noRows2)

    int row1,row2,col,it1,it2;

    if( noCols1 != noRows2 ){

        printf("Sorry, but you cannot multiply the matrices as they do not have corresponding sizes");

        isMulted = 0;

        return 0;

    }

```

```

else{

    ///now we need to determine the size of the resulting matrix

    for( row1 = 1; row1 <= noRows1; row1 ++){

        for( row2 = 1; row2 <= noRows2; row2 ++){

            int sum = 0;

            ///we compute the value of the current element

            for( col = 1; col <= noCols1; col ++){

                for( it1 = 1; it1 <= len1; it1 ++){

                    ///now we need to check if the current element, given by

                    ///row1 and col is in the first matrix

                    if(row1 == mat1[it1][0] && col == mat1[it1][1]){

                        for( it2 = 1; it2 <= len2; it2 ++){

                            ///now we check if it is in the second matrix

                            ///given by row2 and col

                            if(col == mat2[it2][0] && row2 == mat2[it2][1])

                                ///we add to the current sum the product of

                                ///the values of the 2 elements

                                sum += mat1[it1][2] * mat2[it2][2];

                        }

                    }

                }

            }

        }

        ///we add 1 to the size of the result matrix if the sum is != 0

        if(sum != 0){

            resLen ++;

```



```

        resMat[resLen][0] = row1;

        resMat[resLen][1] = row2;

        resMat[resLen][2] = sum;
    }
}
}

```

```

    ///finally, we compute the actual multiplication

    return resLen;

}

}

```

```

//-----scalarMult.c-----

```

```

#include "functions.h"

```

```

void intMult(int len, int mat[1001000][3], int fact){

```

```

    ///the only thing we need to do here is to multiply the value of the

```

```

    ///current element by fact

```

```

    int it;

```

```

    for( it = 1; it <= len; it++){

```

```

        mat[it][2] *= fact;

```

```

    }

```

```

}

```

```
//-----subtract.c-----

#include "functions.h"

int subtr(int len1, int mat1[1001000][3], int len2, int mat2[1001000][3], int resMat[1001000][3]){

    ///this is basically multiplying one matrix with -1 and then adding them

    if (subtr1 == 1){

        intMult(len2, mat2, -1);

        isAdded = 1;

        resLen = add(len1, mat1, len2, mat2, resMat);

        intMult(len2, mat2, -1);

    }else{

        intMult(len1, mat1, -1);

        isAdded = 1;

        resLen = add(len1, mat1, len2, mat2, resMat);

        intMult(len1, mat1, -1);

    }

    return resLen;

}
```

```
//-----printMat.c-----

#include "functions.h"

void printMat(int mat[1001000][3]){

    int row,col;

    int reslt = 1;

    for( row = 1; row <= noRows1; row ++){

        for( col = 1; col <= noCols1; col ++){
```

```
    if( mat[reslt][0] == row && mat[reslt][1] == col ){  
        printf("%d ", mat[reslt][2]);  
        reslt++;  
    }  
    else  
        printf("0 ");  
}  
printf("\n");  
}  
}
```

\end{lstlisting}

\newpage

\section*{Experiments and results}

\includegraphics[height=3.5in, width = 3.5in]{1.png}

\includegraphics[height=3.5in, width = 3.5in]{2.png}

\newpage

\section*{Conclusions}

\vspace{20 mm}

Working on this project, I have gained a new appreciation for these relatively simple matrices and their many uses in the information world. The applications for sparse matrices are many and varied. They can be used in combinatorics and application areas such as network theory, which have a low density of significant data or connections. Large sparse matrices often appear in scientific or engineering applications when solving partial differential equations.

\\vspace{20mm}

\section*{References}

\large 1)www.stackoverflow.com

\\

\\vspace{6mm}

\\

2)https://en.wikipedia.org/wiki/Sparse_matrix

\\

\\vspace{6mm}

\\

3)http://www.sharelatex.com

\end{document}