Programming Principles in C Assignment

Link to gitlab repo

1. Problem solving

The following functions were implemented in the class functionClass.c:

The Class made use of the following libraries as can be seen below and the header class functionClass.h, as well as defining of constant size as 200 which will be used as the maximum number of elements which the user can enter in the array:

```
//functionClass file which includes methods to be used by runner file
//Libraries used in the program:
#include <stdio.h>
//Header file used in the program:
#include "functionClass.h"
//Defining of constant size
#define SIZE 200
```

a)i) init array()

Explanation

In this method the user is prompted to enter a list of numbers, the max being 200 numbers entered. Proper validation is done in case the user enters a decimal point the input is type casted into an integer. Negative integers are also allowed as a valid input. If the user enters a character, the user will need to re-enter the element as character is invalid in integer array. If the user wishes to stop entering numbers into the list, he may opt to enter 0 to break from the method. In this case the digit 0 was chosen as a stopping condition since 0 signifies an absence of a number. In case the user enters all 200 elements, the program would break from the method when the 200th element is entered.

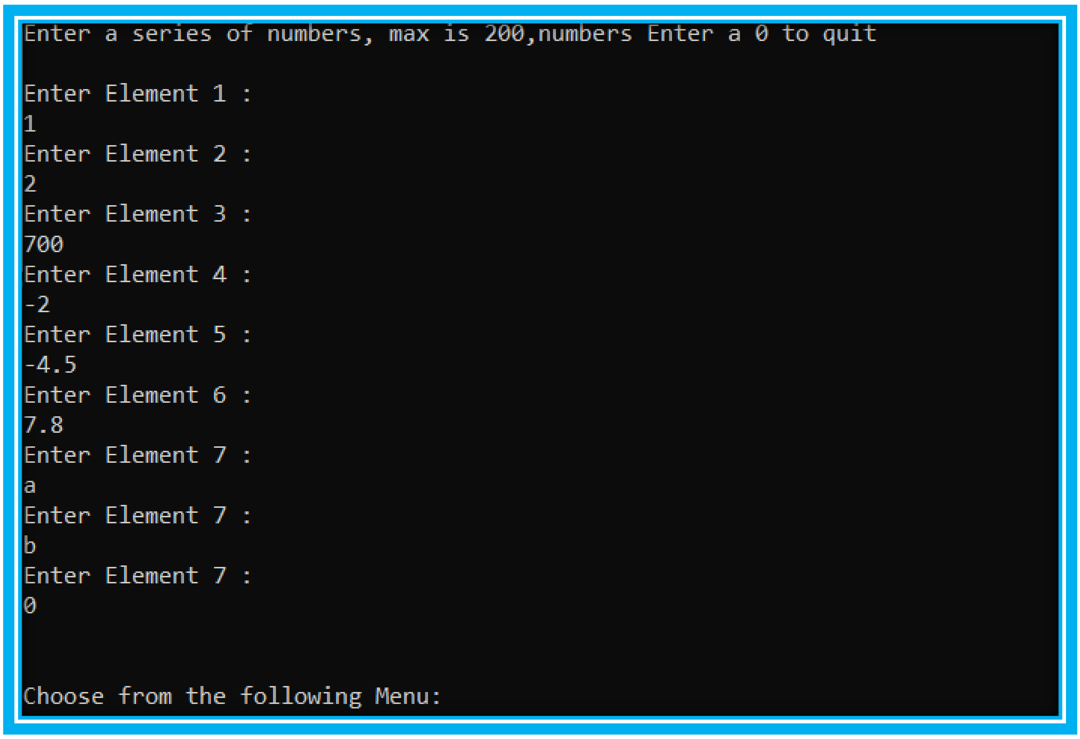
The init_array method in functionClass.c class:

```
//Method init_array with array parameter passing
int  init_array(int array[]) {
    int stop=1;//Declaring of variable stop which be used as a stopping condition for
             the loop
    int count=0;//Declaring variable count which will be used to find the size
    float temp;//Declaring of variable temp
    printf("Enter a series of numbers, max is 200,numbers Enter a 0 to quit\n\n");
    //Creating a variable c of type char in case user inputted a character instead of
       an int
    char c;
    do{
        do {
            //Inputting statements:
            printf("Enter Element %d :\n",count+1);
            scanf("%f", &temp);
            /*float is being used as an input in case user enter decimal point value,
             * typecasting it being done to transform this value as int and storing it
               in array[i]
             */
            array[count]=(int)(temp);
            //Variable c is being used in case user enters character
            c=getchar();
//If user types 0 which is the stopping condition or the array[count]=0, count is
decremented to not include last element and stop is initialised to 0, and the program
breaks from the inner loop
            if(array[count]==0){
                count--;
                stop=0;
                break;
            }
            //Breaking out of loop and initialising  stop to 0 when count is equal to
              200,since 200 is the max amount of numbers user can input
            if(count==SIZE-1){
                stop=0;
                break;
            }
        //do while loop being used to catch when a user inputted a character, and the
           user would have to re-enter element
        }while(c!='\n');
        //Incrementing count
        count++;
    //Do while loop quits when user enters q or max amount of numbers entered is
          reached
    }while(stop!=0);
    //If count is not equal max and the last element is not 0 already, then we set the
     last element as 0 to signify end of array

    if((count!=200 )&&( array[count]!=0)) {
        array[count + 1] = 0;
    }

    return count;
}
```

Testing

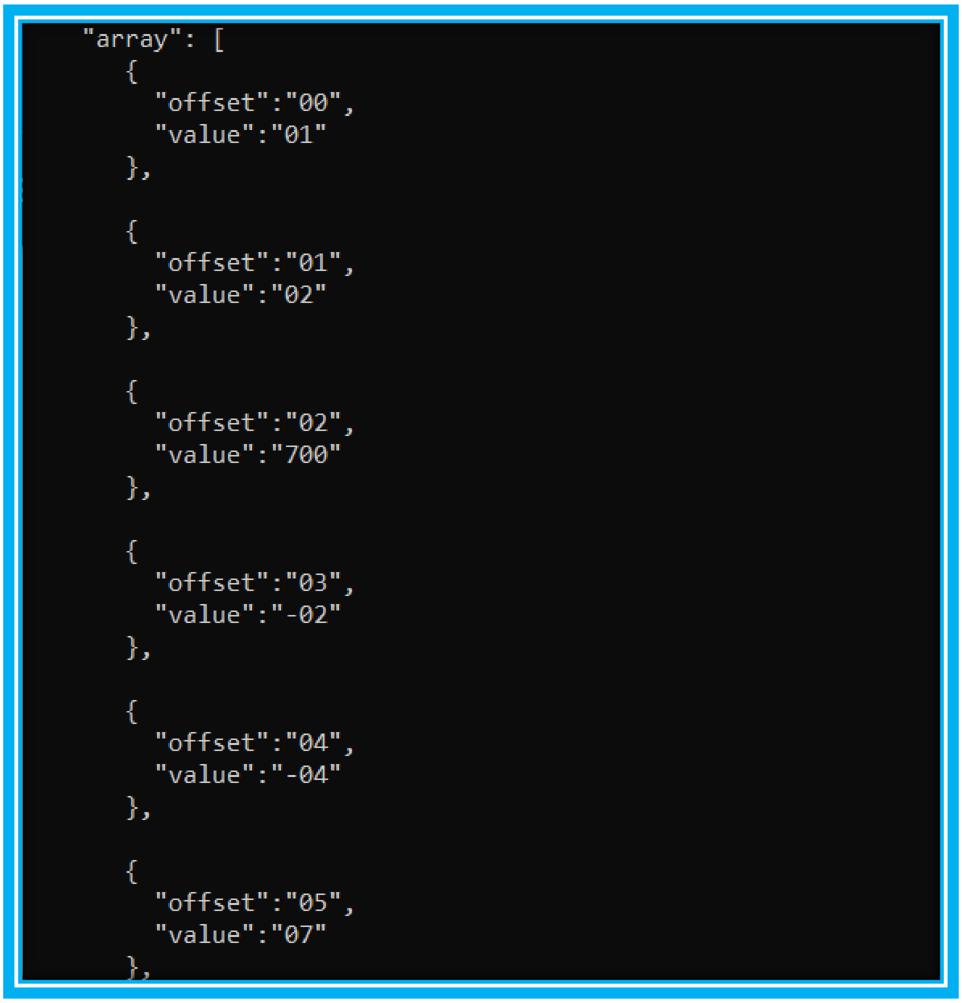
Input	Type of data	Output	Values stored in the array
1,2,700,-2	Valid data	Valid data	1,2,700,-2
-4.5,7.8	Invalid data but is accepted	Invalid data but is accepted	-4,7
a,b	Invalid data	Invalid data	NA
0	Breaking statement	Breaking statement	NA

Showing the values entered by the user:



```
Enter a series of numbers, max is 200,numbers Enter a 0 to quit

Enter Element 1 :
1
Enter Element 2 :
2
Enter Element 3 :
700
Enter Element 4 :
-2
Enter Element 5 :
-4.5
Enter Element 6 :
7.8
Enter Element 7 :
a
Enter Element 7 :
b
Enter Element 7 :
0


Choose from the following Menu:
```

Showing the values which are stored in the array through the display method for testing purposes:



```
    "array": [
       {
         "offset":"00",
         "value":"01"
       },

       {
         "offset":"01",
         "value":"02"
       },

       {
         "offset":"02",
         "value":"700"
       },

       {
         "offset":"03",
         "value":"-02"
       },

       {
         "offset":"04",
         "value":"-04"
       },

       {
         "offset":"05",
         "value":"07"
       },
```

a)ii) display()

Explanation

In this method the user is shown the values of the inputted array (passed as parameter) in the requested format. One can also display the features via shell redirection. To ensure requested output, proper indentation was used by utilising %s.

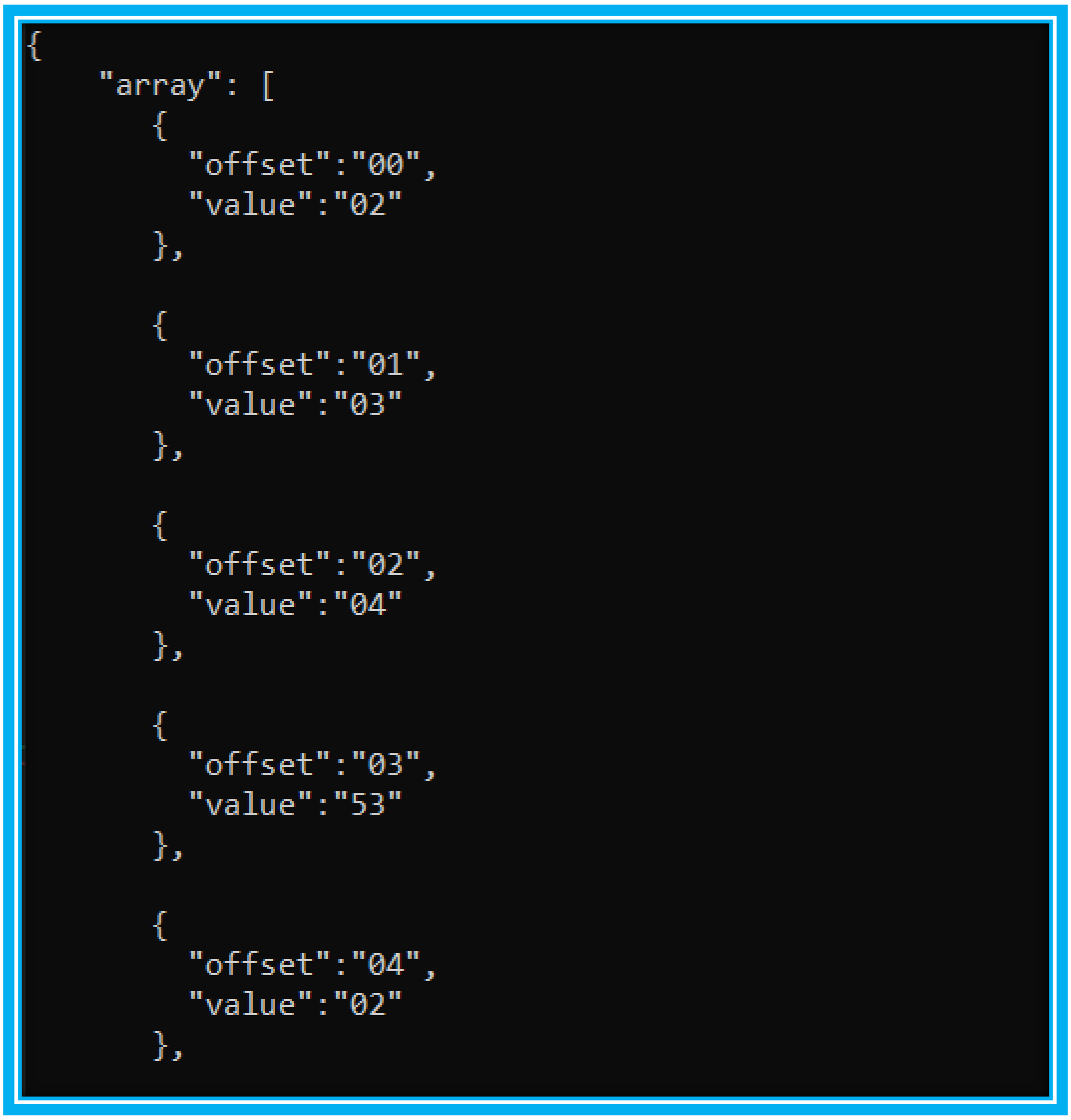
The display method in functionClass.c class:

```
//Method display with size and array parameter passing
void display(int size,int array[]){

            /*Utilised print statements to print the required format output
            *In some cases I had to use %(number of characters before output)s,
            *to reference strings in order to obtain the required output indentation*/
                printf("{\n");
                printf(" %10s: [\n","\"array\"");
                for (int i = 0; i < size; i++) {
                printf("%8s\n","{");
                printf("%17s:\"%.2d\",\n","\"offset\"", i);
                printf("%16s:\"%.2d\"\n","\"value\"", array[i]);
                if(i!=size-1)
                    printf(" %8s\n\n","},");
                else
                    printf(" %8s\n","},");
                }
                printf("%4s\n","]");
                printf("}");

}
```

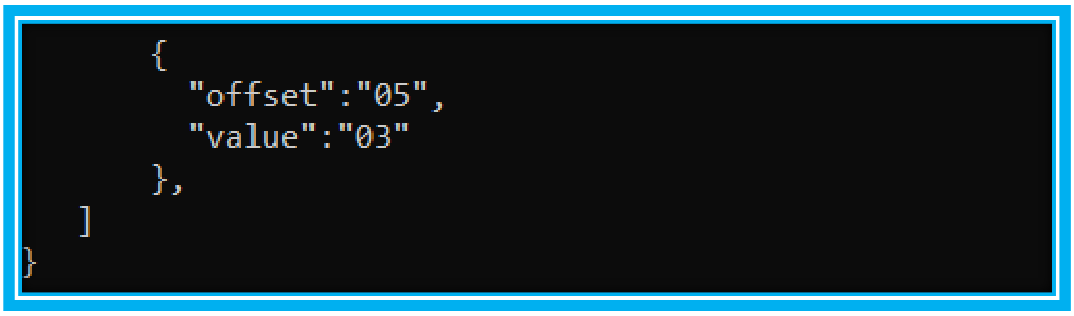
Testing

Array contents	Values Outputted
2,3,4,53,2,3	2,3,4,53,2,3

Showing the output to the user:



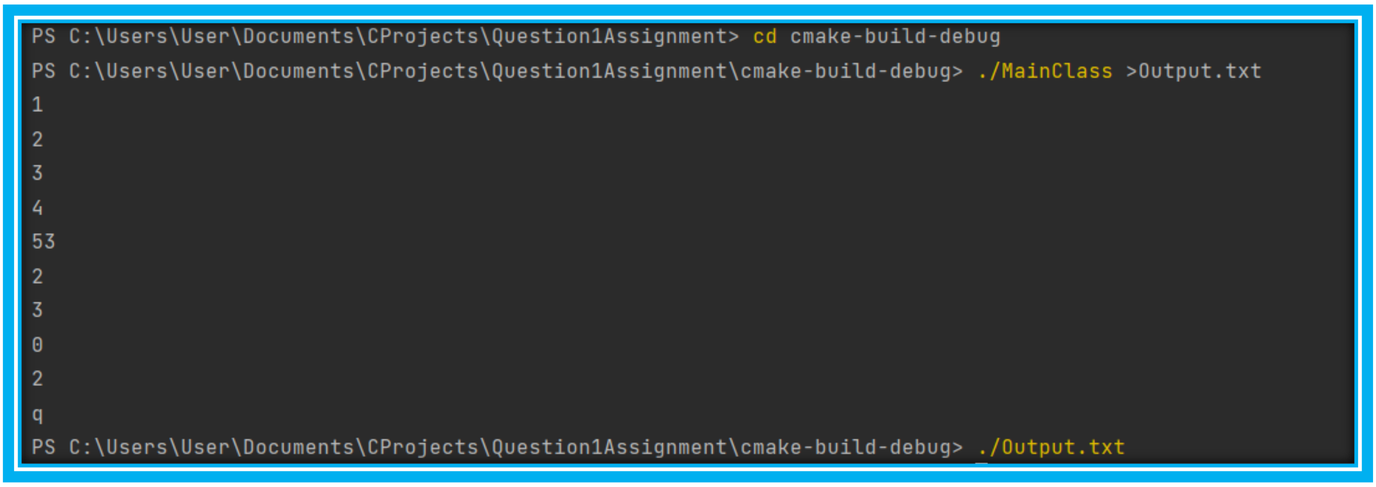
```
{
    "array": [
       {
         "offset":"00",
         "value":"02"
       },

       {
         "offset":"01",
         "value":"03"
       },

       {
         "offset":"02",
         "value":"04"
       },

       {
         "offset":"03",
         "value":"53"
       },

       {
         "offset":"04",
         "value":"02"
       },
```



```
       {
         "offset":"05",
         "value":"03"
       },
    ]
}
```

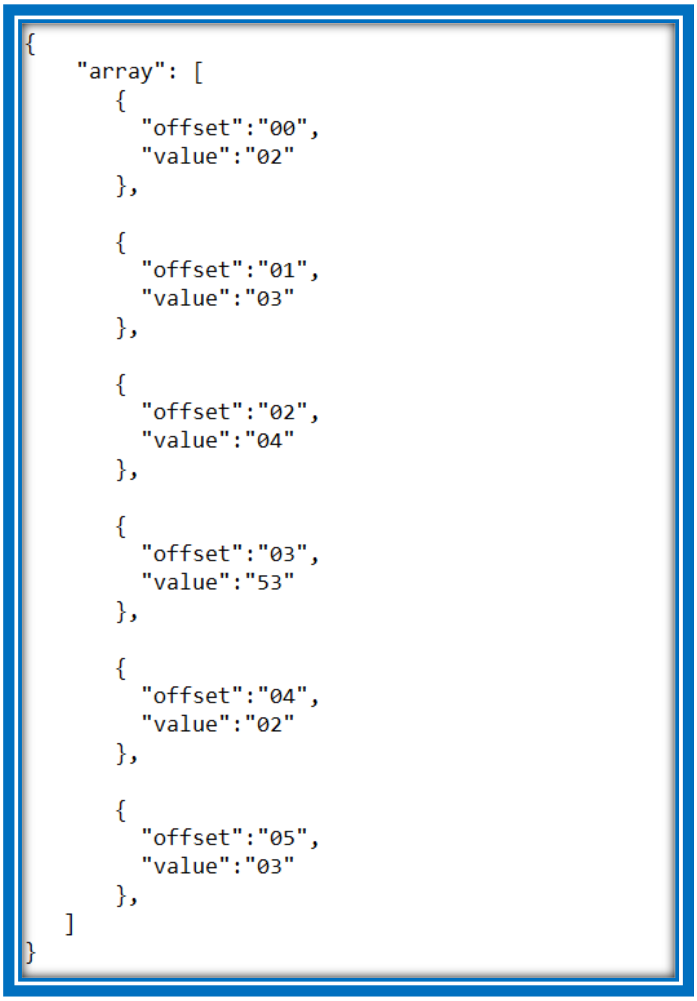
Showing the output via Shell Redirection:

Input from terminal:



```
PS C:\Users\User\Documents\CProjects\Question1Assignment> cd cmake-build-debug
PS C:\Users\User\Documents\CProjects\Question1Assignment\cmake-build-debug> ./MainClass >Output.txt
1
2
3
4
53
2
3
0
2
q
PS C:\Users\User\Documents\CProjects\Question1Assignment\cmake-build-debug> ./Output.txt
```

Output in File:



```
{
    "array": [
       {
         "offset":"00",
         "value":"02"
       },

       {
         "offset":"01",
         "value":"03"
       },

       {
         "offset":"02",
         "value":"04"
       },

       {
         "offset":"03",
         "value":"53"
       },

       {
         "offset":"04",
         "value":"02"
       },

       {
         "offset":"05",
         "value":"03"
       },
   ]
}
```

a)iii) reverse()

Explanation

In this method the contents of the first array are copied into the second array, both arrays are passed as a parameter as well as the size. The contents of the first array are placed in the second array, however in the reverse order. This was achieved by looping through all the elements in the array and utilising a counter to act as index for the values in the arrays.

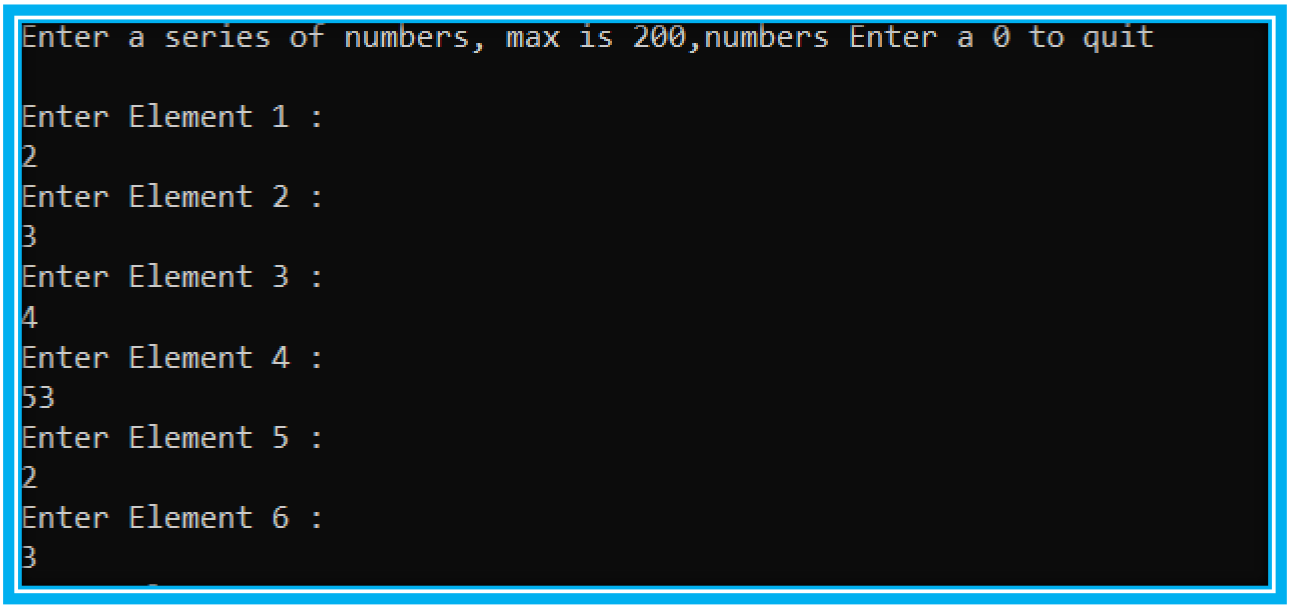
The reverse method in functionClass.c class:

```c
//Method reverse with size and 2 array parameter passing
int *reverse(int *arr1, int *arr2, int size) {
    //Declaring of variable count and setting it to size-1
    // since last element entered is escape element -1
    int count=size-1;
    for(int i=0;i<=size;i++) {
        //values of array 1 are placed in array 2
        arr2[count]=arr1[i];
        /*Count is decremented, since it is being used as an index for the
        *reverse array and the goal is for the first element in arr1 to become
        *the last element of arr2*/
        count--;
    }
    //Returning of arr2
    return arr2;
}
```

Testing

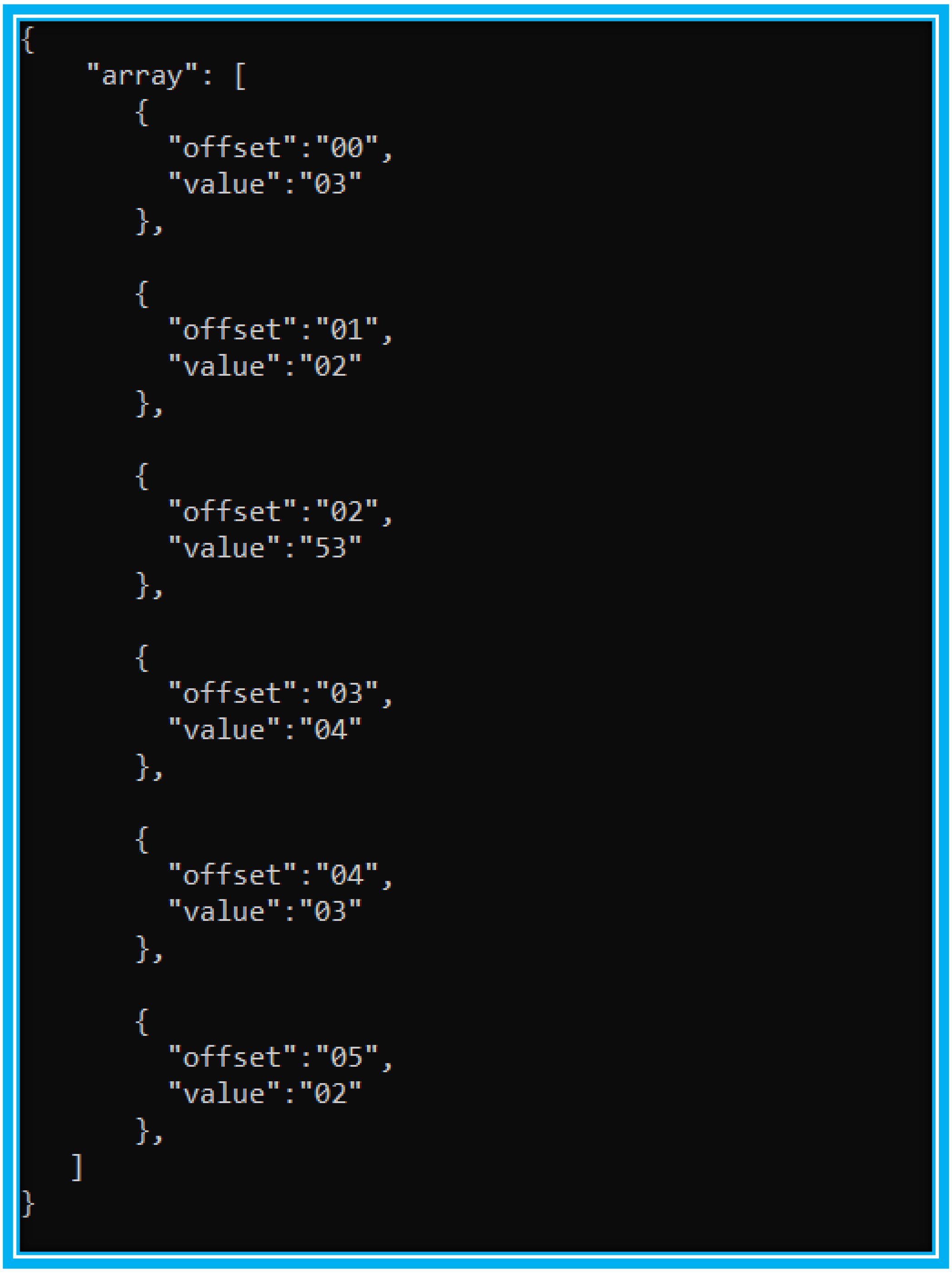
Array contents	Values Outputted
2,3,4,53,2,3	3,2,53,4,3,2

Showing the values which are inputted in the array through the init_array method for testing purposes:



```
Enter a series of numbers, max is 200,numbers Enter a 0 to quit

Enter Element 1 :
2
Enter Element 2 :
3
Enter Element 3 :
4
Enter Element 4 :
53
Enter Element 5 :
2
Enter Element 6 :
3
```

Showing the values in the array which are outputted through the display method for testing purposes:



```
{
    "array": [
       {
         "offset":"00",
         "value":"03"
       },

       {
         "offset":"01",
         "value":"02"
       },

       {
         "offset":"02",
         "value":"53"
       },

       {
         "offset":"03",
         "value":"04"
       },

       {
         "offset":"04",
         "value":"03"
       },

       {
         "offset":"05",
         "value":"02"
       },
   ]
}
```

a)iv) frequency()

Explanation

This method accepts an integer array and a freq array as a parameter, the method first calculates the size of the int array as it wasn't passed as a parameter. In continuation the method loops through all the elements in the int array, then utilising a second for loop it loops through the freq array to check whether the element appears in the freq array. If the element isn't found in the freq array, the program will resort to add the item to the freq array. Succeeding this, if the same value is found whilst searching through the freq array, the frequency of said value is increased by 1, this was done using 2 nested loops by looping through the int array and freq array. Finally, the last element of freq array is filled with "value" as -1 and "frequency" as 0 to signify end of freq array.

functionClass.h header class which includes the declaration of the struct freq which will be used in functionClass.c and main.c classes:

```
//Header file which allows main class to access methods of functionClass.c files
//Declaration of struct freq and assigning 2 int values, value and frequency
typedef struct {
    int value;
    int frequency;
}freq;
```

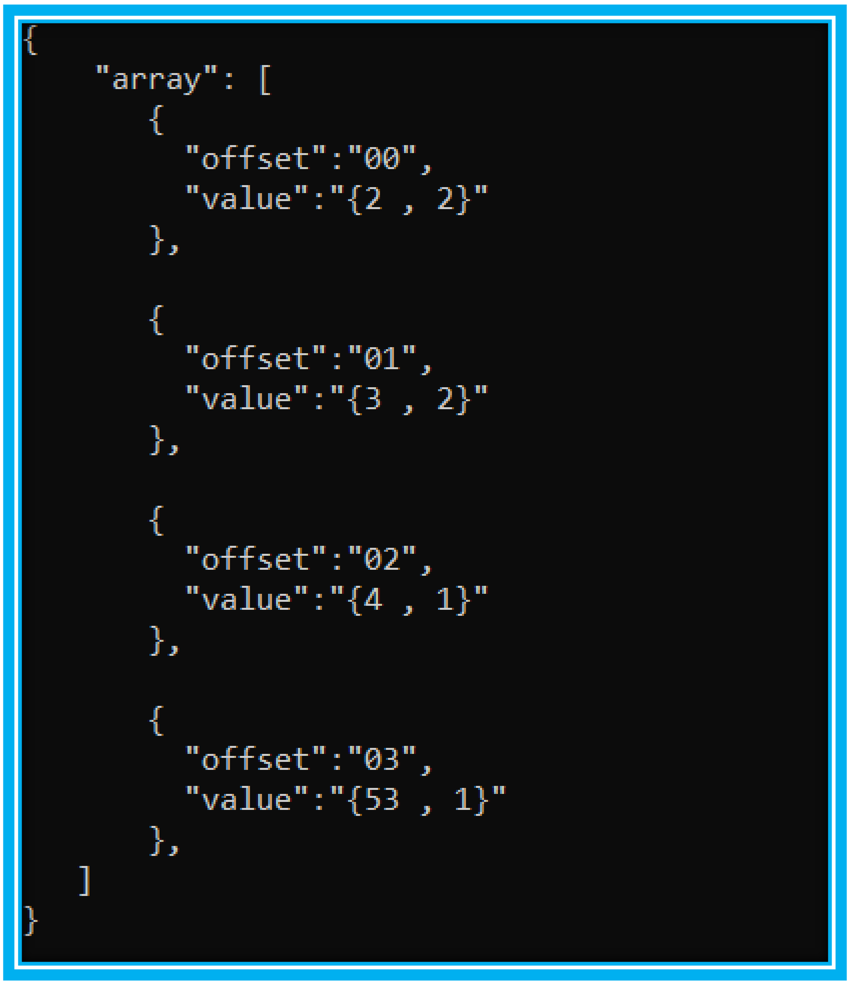
functionClass.c class which includes frequency method:

```
//Method frequency with  array and struct freq array parameter
void frequency(int arr[],freq pairs[]){
    int size=0;//Declaring of variable size and setting it to 0

        for (int i = 0; i <= size; i++) {//Using a for loop to find out the size of the Array

            if (arr[i] !=0)//if element in array is 0 this is the stopping condition and variable size stops incrementing
                size += 1;
            else{//When the element is 0, breaks from loop
                size+=1;
                break;
            }
            if(size==200)//To check if size is max, if it is break
                break;
        }


    int psize=0;//Declaring of variable psize to measure size of pairs array and setting it to 0

    int check;//Declaring of variable check
    for(int i=0;i<size;i++) {//Using a for loop to go through all the elements in the array arr

        check=0;//Setting variable check to 0
        for(int j=0;j<psize;j++) {//Using a for loop to go through all the elements in the array pairs

            if(arr[i]==(pairs[j].value)) {//Checking whether the value of the array arr is inside the pairs array

                check=1;//Setting check = 1 when the element in array arr was found in pairs array
            }
        }

/*Whilst going through all the elements in pairs array, if the elements wasn't found
therefore check will remain 0 signifying that we need to add the element in the pairs
array*/
        if(check==0){
            pairs[psize].value = arr[i];//Adding element to pairs array
            pairs[psize].frequency=0;//Setting the frequency of that element to 0 which
                                       will later be incremented
            psize += 1;//Incrementing psize since a new element was added to pairs
                         array
        }
    }

    for(int i=0;i<size-1;i++) {//Using a for loop to go through all the elements in the
                                  array arr
        int temp = arr[i];
        for (int j = 0; j < psize; j++) {//Using a for loop to go through all the
                                             elements in the array pairs
            if (temp == (pairs[j].value)) {//Checking whether the value of the array
                                                 arr is inside the pairs array

                pairs[j].frequency += 1;//Incrementing frequency when the value of
                                                 array arr appears in the pairs array
            }
        }
    }

    //Setting value as -1 and frequency as 0 to signify end of array
    pairs[psize].value=-1;
    pairs[psize].frequency=0;

}
```

Testing

Array contents	Values Outputted
2,3,4,53,2,3	{2,2},{3,2},{4,1},{53,1}

Showing the values which are stored in the array through the display_freq method for testing purposes:



```
{
    "array": [
       {
         "offset":"00",
         "value":"{2 , 2}"
       },

       {
         "offset":"01",
         "value":"{3 , 2}"
       },

       {
         "offset":"02",
         "value":"{4 , 1}"
       },

       {
         "offset":"03",
         "value":"{53 , 1}"
       },
   ]
}
```

a)v)display_freq()

Explanation

This method accepts a freq array as a parameter, the method first calculates the size of the freq array as it wasn't passed as a parameter. In continuation the method utilises the selection sort to sort the values in the freq array in ascending order. The values are outputted in the requested format in ascending order. One can also display the features via shell redirection. To ensure requested output, proper indentation was used by utilising %s.

The display_freq method in functionClass.c class:

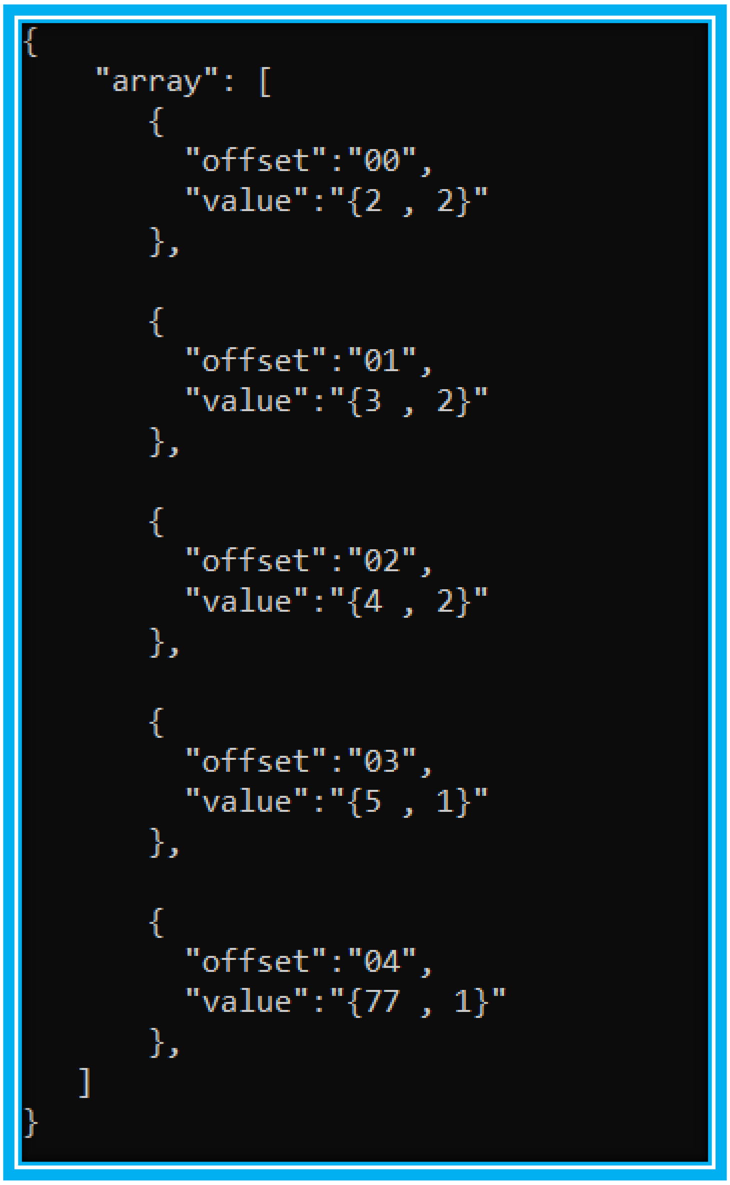
```
//Declaration of method display_freq with struct freq array parameter
void display_freq(freq pairs[]){
    int size=1;//Declaring of variable size and initialising it to 1
    //For loop and if statement used to find size of array, if a null is encountered
        the variable size will stop incrementing

    //Using a for loop to find out the size of the pairs array
    for(int i=0;i<size;i++) {
        //If frequency!=0 and value!=-1,end of array has not been reached and thus size
           is incremented

        if ((pairs[i].frequency != 0) && (pairs[i].value != -1)) {
            size++;
        }//If frequency=-1 and value=-1, end of array is reached and thus size is
             decrementing to not count this instance
        else
            size--;
    }

    //Sorting the array in ascending order for value
    freq temp;//Creating a variable called temp of type freq which will be used to swap
                the elements
    for (int top = 0; top < size-1; top++) {//Outer loop used to loop through all the
                                         elements in the pairs array with index top

        for (int seek = top + 1; seek < size; seek++) {
     //Inner loop used to loop through the elements which have an index of top+1

            //Comparing elements from the first loop with the second loop
            if (pairs[top].value > pairs[seek].value) {
                //Swapping the elements
                temp = pairs[top];
                pairs[top] = pairs[seek];
                pairs[seek] = temp;
            }
        }

    }
                //Utilised print statements to print the required format output
                //In some cases had to use %(number of characters before output)s, to
                  reference strings
                printf("{\n");
                printf(" %10s: [\n","\"array\"");
                for (int i = 0; i < size; i++) {
                    printf("%8s\n","{");
                    printf("%17s:\"%.2d\",\n","\"offset\"", i);
                    printf("%16s:\"{%d , %d}\"\n","\"value\"", pairs[i].value,
                                                                pairs[i].frequency);
                    if(i!=size-1)
                        printf(" %8s\n\n","},");
                    else
```

```
                        printf(" %8s\n","},");
                }
                printf("%4s\n","]");
                printf("}");

}
```

Testing

Array contents	Values Outputted
2,3,4,77,2,3,5,4	{2,2},{3,2},{4,2},{5,1},{77,1}

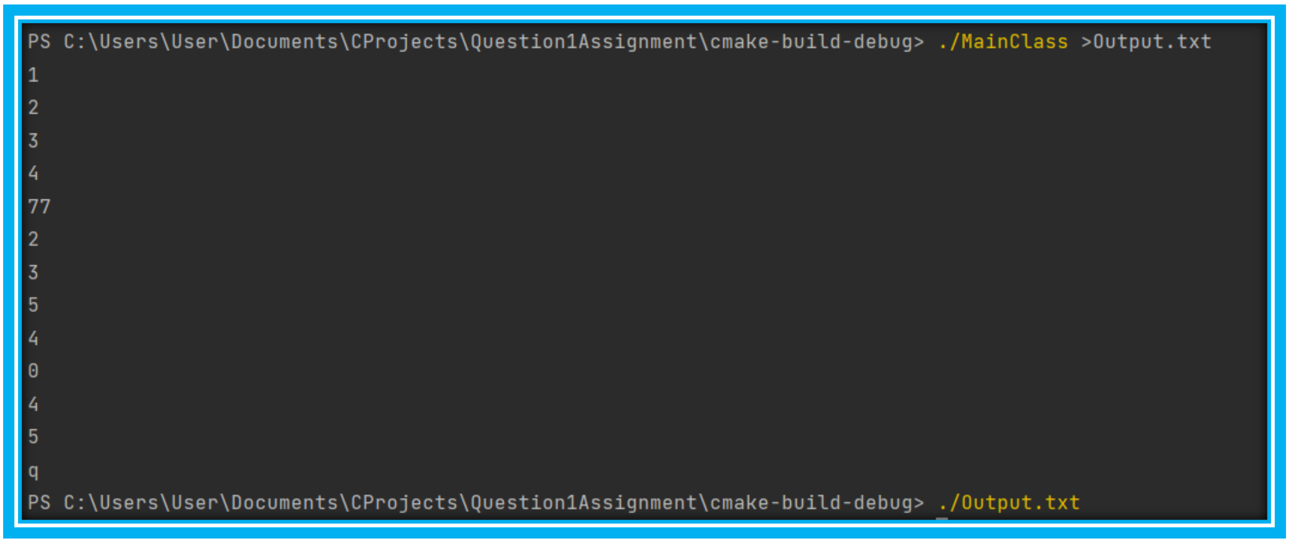
Showing the output to the user:



```
{
    "array": [
       {
         "offset":"00",
         "value":"{2 , 2}"
       },

       {
         "offset":"01",
         "value":"{3 , 2}"
       },

       {
         "offset":"02",
         "value":"{4 , 2}"
       },

       {
         "offset":"03",
         "value":"{5 , 1}"
       },

       {
         "offset":"04",
         "value":"{77 , 1}"
       },
   ]
}
```

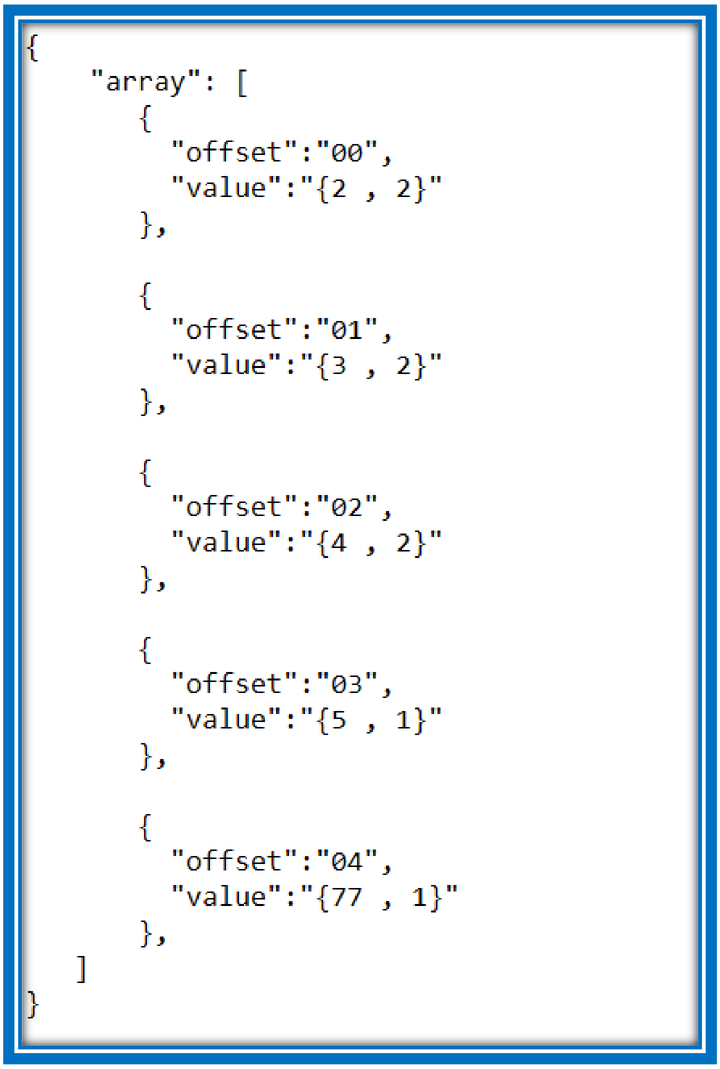
Showing the output via Shell Redirection:

Input from terminal:



```
PS C:\Users\User\Documents\CProjects\Question1Assignment\cmake-build-debug> ./MainClass >Output.txt
1
2
3
4
77
2
3
5
4
0
4
5
q
PS C:\Users\User\Documents\CProjects\Question1Assignment\cmake-build-debug> ./Output.txt
```

Output in File:



```
{
    "array": [
       {
         "offset":"00",
         "value":"{2 , 2}"
       },

       {
         "offset":"01",
         "value":"{3 , 2}"
       },

       {
         "offset":"02",
         "value":"{4 , 2}"
       },

       {
         "offset":"03",
         "value":"{5 , 1}"
       },

       {
         "offset":"04",
         "value":"{77 , 1}"
       },
   ]
}
```

b)Command-line Menu

Explanation

The user is presented with a command-line method showing all the options from task 1)a) .

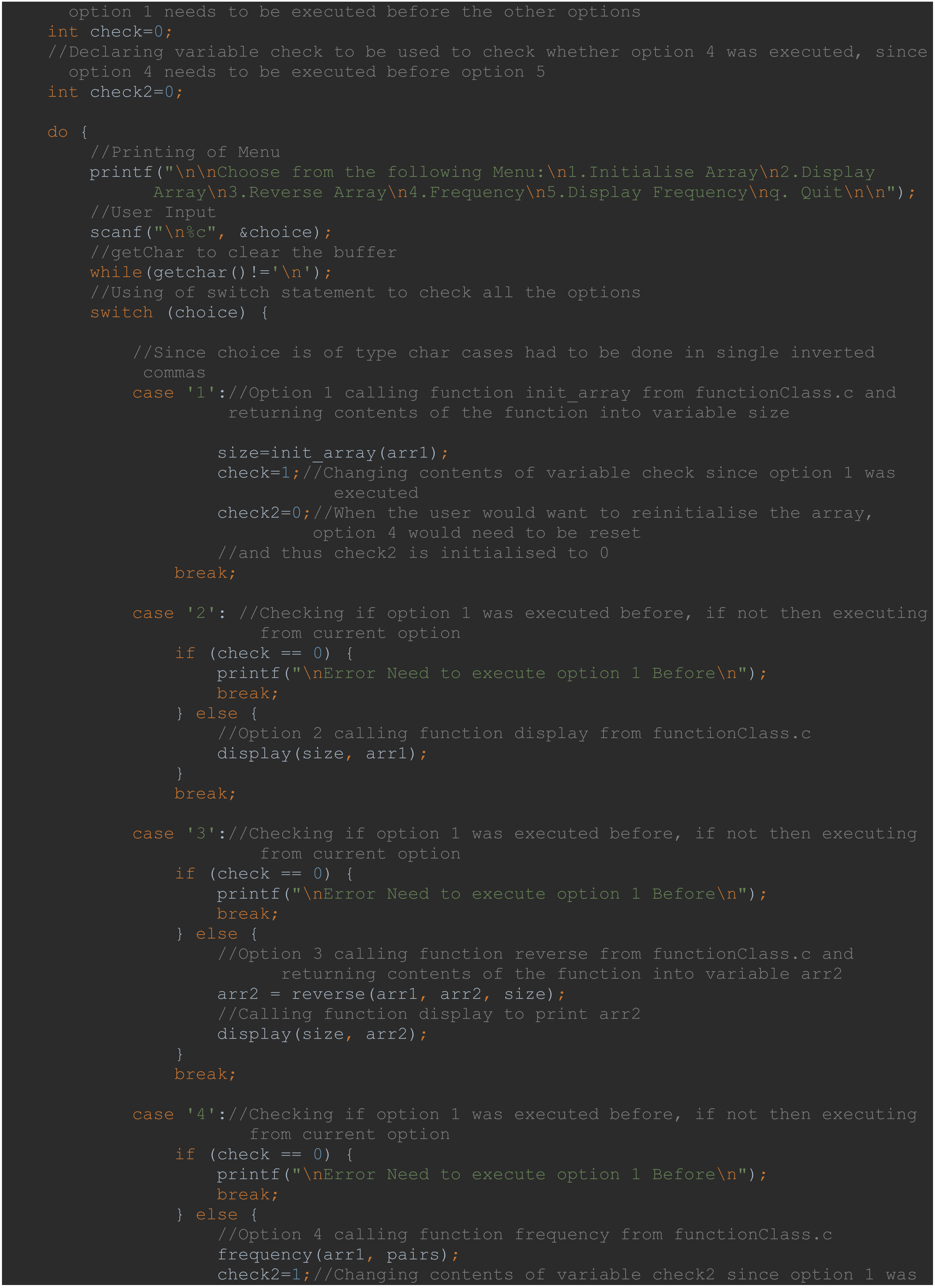
Proper validation is ensured in case the user enters an option which is not valid. Furthermore, validation has also been ensured that the user would access option 1 before attempting any other option. The same validation method was applied in case the user opted to access option 5 before accessing option 4.

The functionclass.h header files which contain methods which will be utilised by the main.c class:

```
//Header file which allows main class to access methods of functionClass.c files
//Declaration of struct freq and assigning 2 int values, value and frequency
typedef struct {
    int value;
    int frequency;
}freq;
//Declaration of method init_array with array parameter
int init_array(int array[]);
//Declaration of method display with integer and array parameter
void display(int size,int array[]);
//Declaration of method reverse with integer and 2 array parameter
int *reverse(int *arr1,int *arr2 ,int size);
//Declaration of method frequency with integer array and struct freq array parameter
void frequency(int arr[],freq pairs[]);
//Declaration of method display_freq with struct freq array parameter
void display_freq(freq pairs[]);
```

The main.c class:

```
//Runner file
//Libraries used in the program:
#include <stdio.h>
//Header file used in the program:
#include "functionClass.h"
//Defining of constant size
#define SIZE 200


//Declaration of Main Menu function with parameters
void MainMenu(int size, int arr1[],int arr2[],freq pairs[]);

int main(void) {
    //Declaring array variables which will be used throughout the menu options
    int arr1[SIZE];
    int arr2[SIZE];
    int size=SIZE;
    freq pairs[SIZE];

    //Calling MainMenu method and passing the values by reference in case of arrays and
      passing by values in case of size
    MainMenu(size,arr1,arr2,pairs);
    return 0;
}


//MainMenu method
void MainMenu(int size, int *arr1, int *arr2,freq pairs[]) {
    //Declaring the user input which will be used as an input for the menu, declaring
      it as type char to accept 'q' as a termination of the menu
    char choice;
    //Declaring variable check to be used to check whether option 1 was executed, since
```



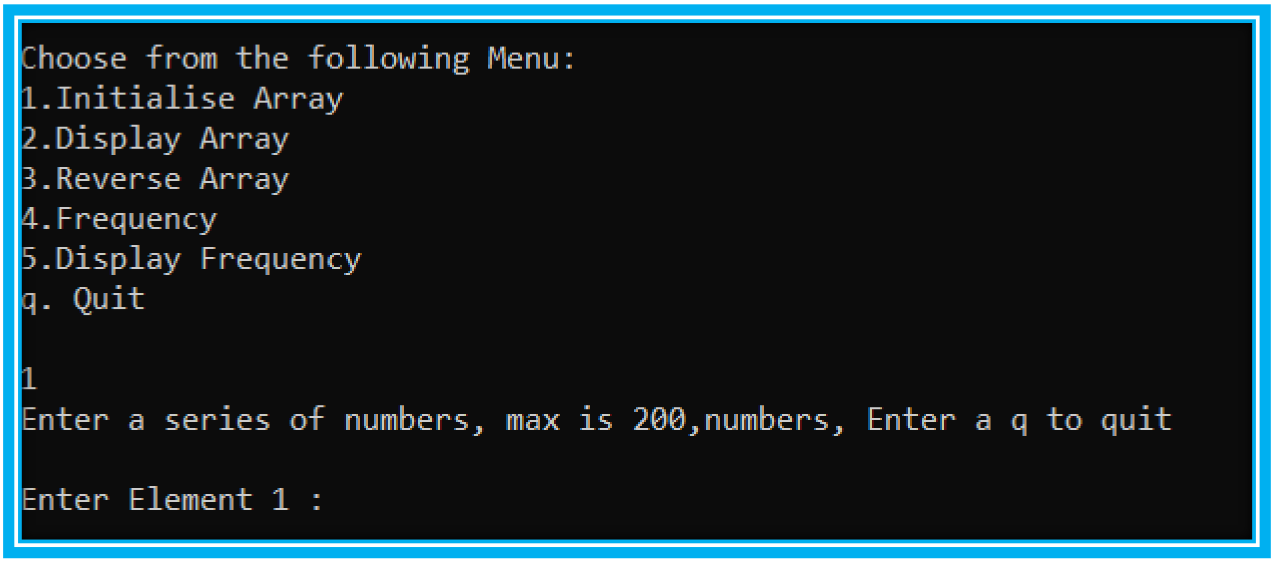
```
      option 1 needs to be executed before the other options
    int check=0;
    //Declaring variable check to be used to check whether option 4 was executed, since
      option 4 needs to be executed before option 5
    int check2=0;

    do {
        //Printing of Menu
        printf("\n\nChoose from the following Menu:\n1.Initialise Array\n2.Display
              Array\n3.Reverse Array\n4.Frequency\n5.Display Frequency\nq. Quit\n\n");
        //User Input
        scanf("\n%c", &choice);
        //getChar to clear the buffer
        while(getchar()!='\n');
        //Using of switch statement to check all the options
        switch (choice) {

            //Since choice is of type char cases had to be done in single inverted
             commas
            case '1'://Option 1 calling function init_array from functionClass.c and
                     returning contents of the function into variable size

                        size=init_array(arr1);
                        check=1;//Changing contents of variable check since option 1 was
                                   executed
                        check2=0;//When the user would want to reinitialise the array,
                                 option 4 would need to be reset
                        //and thus check2 is initialised to 0
                break;

            case '2': //Checking if option 1 was executed before, if not then executing
                        from current option
                if (check == 0) {
                    printf("\nError Need to execute option 1 Before\n");
                    break;
                } else {
                    //Option 2 calling function display from functionClass.c
                    display(size, arr1);
                }
                break;

            case '3'://Checking if option 1 was executed before, if not then executing
                        from current option
                if (check == 0) {
                    printf("\nError Need to execute option 1 Before\n");
                    break;
                } else {
                    //Option 3 calling function reverse from functionClass.c and
                          returning contents of the function into variable arr2
                    arr2 = reverse(arr1, arr2, size);
                    //Calling function display to print arr2
                    display(size, arr2);
                }
                break;

            case '4'://Checking if option 1 was executed before, if not then executing
                       from current option
                if (check == 0) {
                    printf("\nError Need to execute option 1 Before\n");
                    break;
                } else {
                    //Option 4 calling function frequency from functionClass.c
                    frequency(arr1, pairs);
                    check2=1;//Changing contents of variable check2 since option 1 was
```

```
                               executed
                }
                break;

            case '5'://Checking if option 4 was executed before, if not then executing
                      from current option
                if (check2 == 0) {
                    printf("\nError Need to execute option 4 Before\n");
                    break;
                } else {
                    //Option 5 calling function display_freq from functionClass.c
                    display_freq(pairs);
                }
                break;

            case 'q'://Option q Quitting from the Menu and displaying Goodbye message
                    printf("\n\nGoodbye\n\n");
                break;

                //In case an invalid option is entered an invalid message is displayed
            default:printf("\n\nRe-enter option\n\n");
                break;

        }
        //Do while loop used to loop the Menu until the user decides to quit
    } while (choice != 'q');
}
```

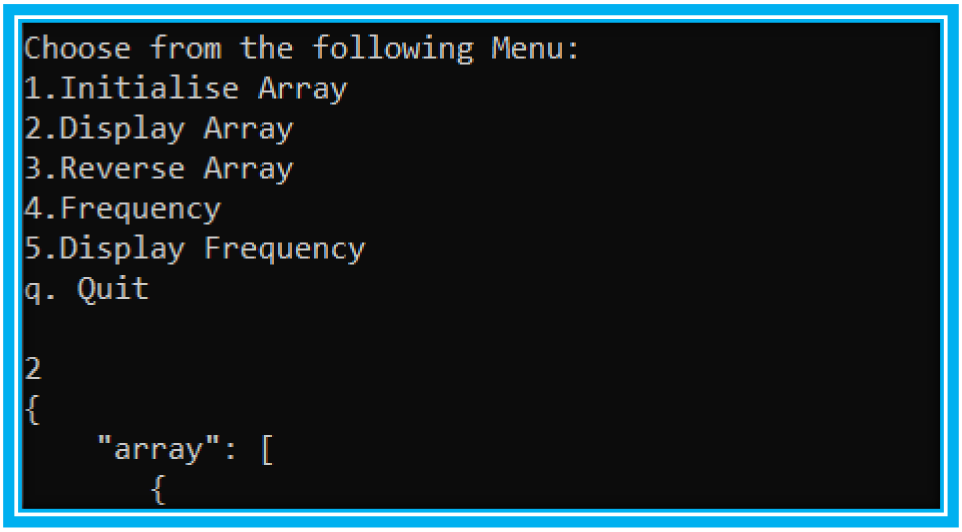
Testing

Array contents	Values Outputted
1,2,4	Valid data
a,b,-2,7	Invalid data
Accessing option 4 before 5, Accessing option 3 before 1	Invalid data

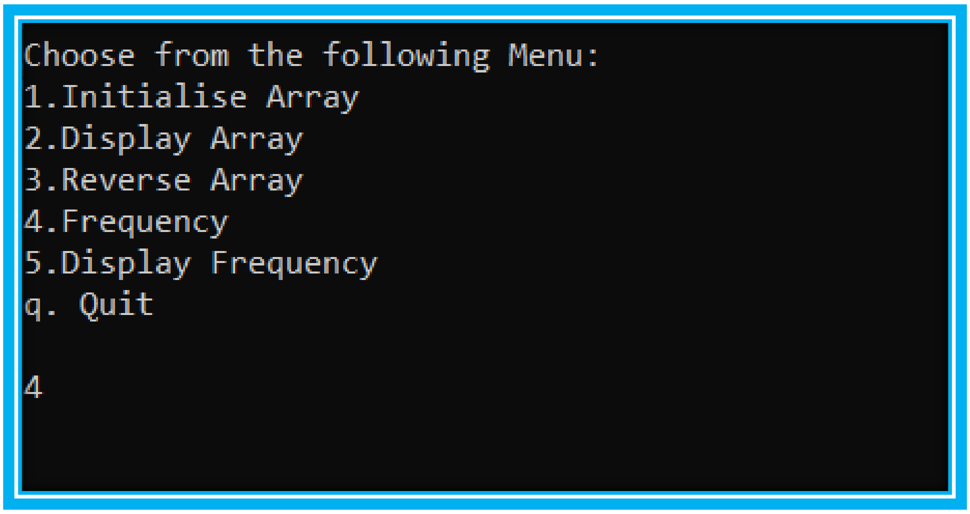
Valid data:



```
Choose from the following Menu:
1.Initialise Array
2.Display Array
3.Reverse Array
4.Frequency
5.Display Frequency
q. Quit

1
Enter a series of numbers, max is 200,numbers, Enter a q to quit

Enter Element 1 :
```

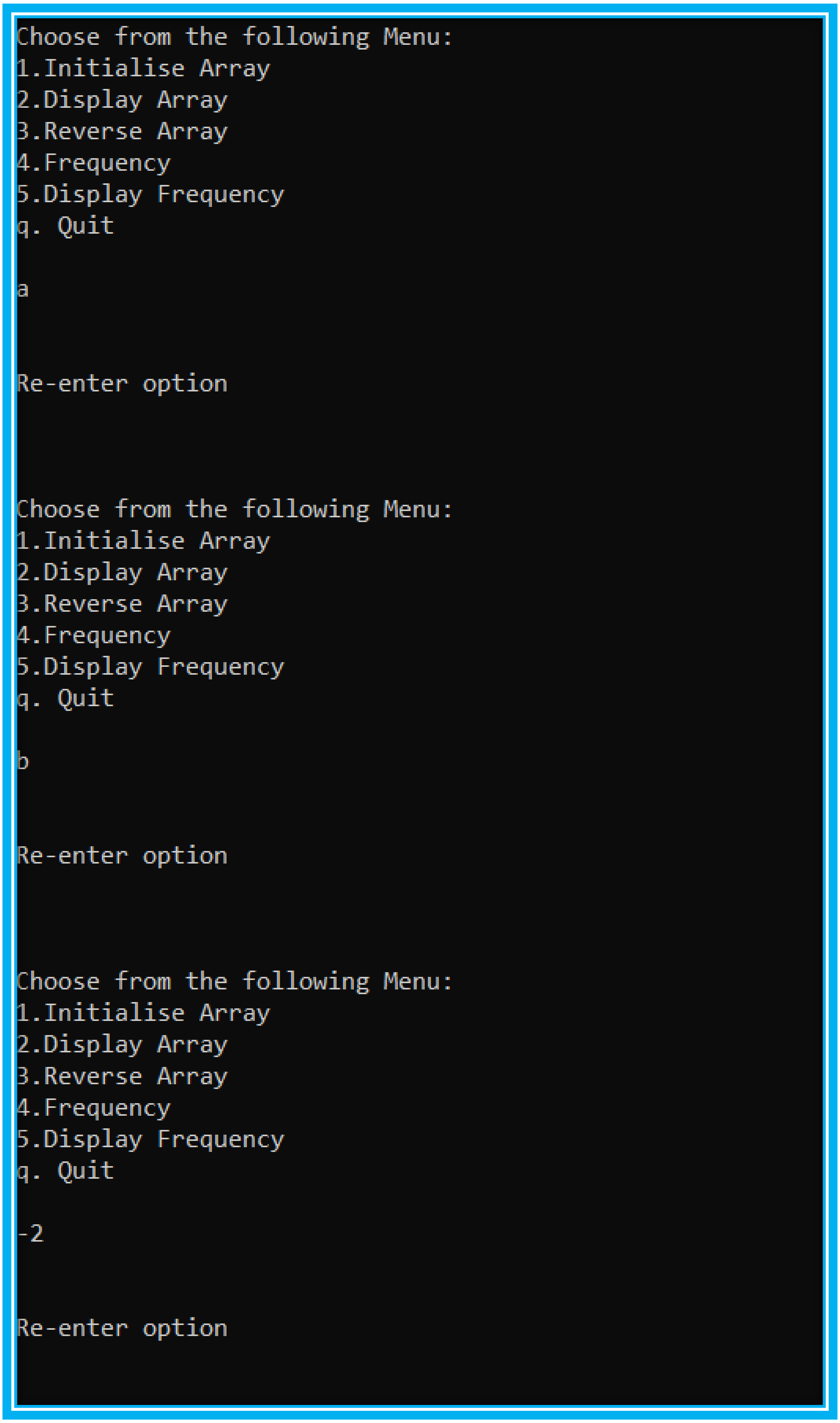


```
Choose from the following Menu:
1.Initialise Array
2.Display Array
3.Reverse Array
4.Frequency
5.Display Frequency
q. Quit

2
{
    "array": [
        {
```

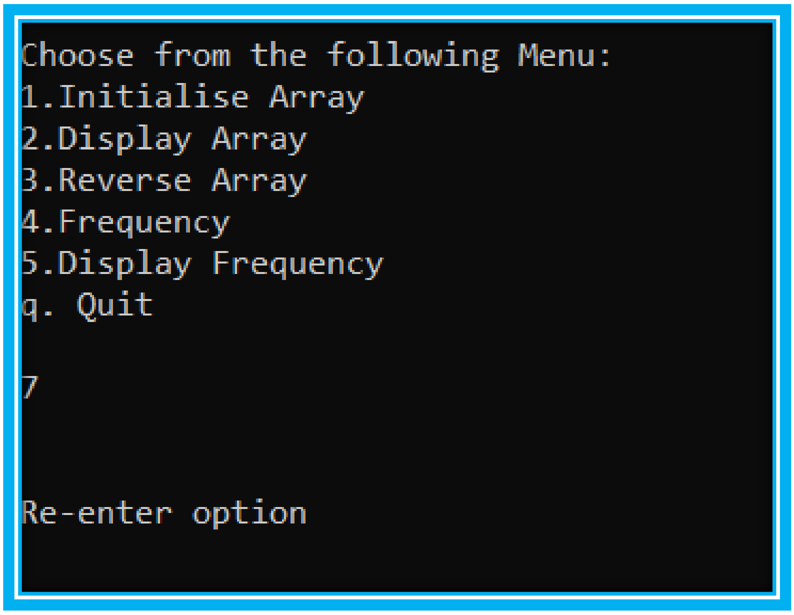


```
Choose from the following Menu:
1.Initialise Array
2.Display Array
3.Reverse Array
4.Frequency
5.Display Frequency
q. Quit

4
```

Invalid data:

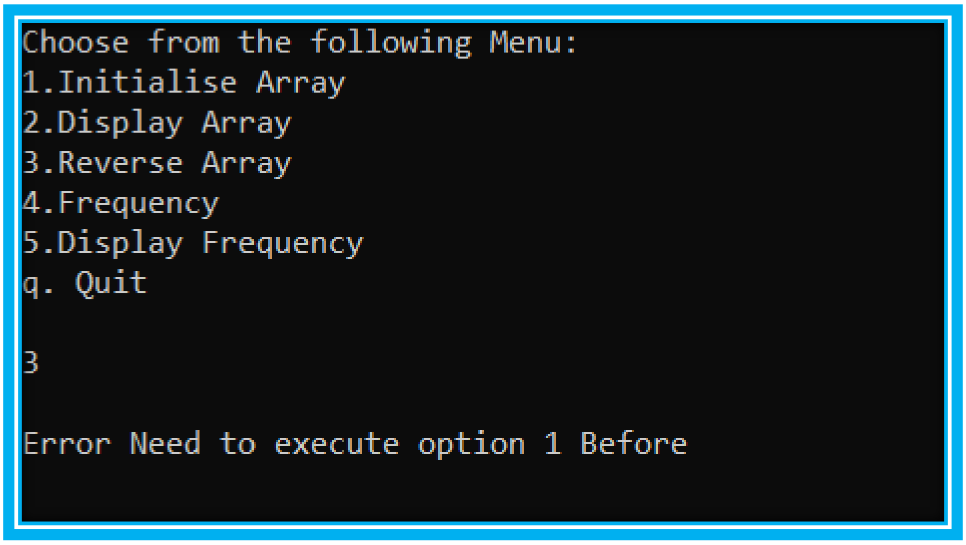


```
Choose from the following Menu:
1.Initialise Array
2.Display Array
3.Reverse Array
4.Frequency
5.Display Frequency
q. Quit

a


Re-enter option



Choose from the following Menu:
1.Initialise Array
2.Display Array
3.Reverse Array
4.Frequency
5.Display Frequency
q. Quit

b


Re-enter option



Choose from the following Menu:
1.Initialise Array
2.Display Array
3.Reverse Array
4.Frequency
5.Display Frequency
q. Quit

-2


Re-enter option
```



```
Choose from the following Menu:
1.Initialise Array
2.Display Array
3.Reverse Array
4.Frequency
5.Display Frequency
q. Quit

7


Re-enter option
```

Invalid data when user tries to access option 3 before 1:



```
Choose from the following Menu:
1.Initialise Array
2.Display Array
3.Reverse Array
4.Frequency
5.Display Frequency
q. Quit

3

Error Need to execute option 1 Before
```

2. A DataTable library a)

The following functions were implemented in the class functionClass2a.c and utilised the following structs from the functionClass2a.h header file:

```
//Declaration of union location
typedef union{
    //Assigning an array of float
    float floatrows[ROWS];
    //Assigning a 2d array of char
    char stringrows[ROWS][SSIZE];
}location;

//Declaration of struct DataTable_t
typedef struct{
//An array containing pointers setting it to type union location to have both float and
character string pointers
    location *columns[COLS];
    //An integer to hold the number of populated rows
    int nrows;
    //2 integers to hold the start of the string columns and end of the columns
    int colstart,colend;
    //Assigning a 2d array of char to hold the labels
    char labels[COLS][SSIZE];
}DataTable_t;
```

The Class made use of the following libraries as can be seen below and the header class functionClass2a.h, as well as defining of constants:

```
//functionClass file which includes methods to be used by runner file
//Libraries used in the program:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
//Header file used in the program
#include "functionClass2a.h"
//Defining of constants
#define COLS 6
#define SSIZE 64
#define ROWS 1000
```

For testing purposes, a Testing file containing 998 rows and 6 columns was used for Task2a), the first three columns were of type float and the other three were of type string.

2a)initDT()

Explanation

In this method a new DataTable_t structure is initialised, by allocating memory for said DataTable on the heap. The function allows an integer parameter to be passed. If the parameter is 0, this implies an initial commitment of the DataTable_t with a static size according to the requirements in the assignment brief, but if the parameter is 1 then a new DataTable_t is created upon specific specifications (i.e., specific rows and columns. When the initial commitment is chosen, the user is prompted to enter the labels for the columns.

The initDT method in functionClass2a.c class:

```
//Declaration of method initDT with interger parameters choice
DataTable_t *initDT(int choice) {
    //Declaration of DataTable_t pointer
    DataTable_t *data;
    //If choice is 0 , then the DataTable is initialised from a file
    if(choice==0) {
        //Declaration of array of pointers of type location
        location *assignmemory[COLS];

        //Creating size in heap for DataTable_t
        data = (DataTable_t *) malloc(sizeof(DataTable_t));

        //Creating size in heap for data->columns through assignmemory array
        for (int i = 0; i < COLS; i++) {
            assignmemory[i] = (location *) malloc(sizeof(location));
            data->columns[i] = assignmemory[i];
        }

//Utilising a for loop for the user to input labels and storing the user input in data-
>labels array
        for (int i = 0; i < COLS; i++) {
            char tmp[SSIZE];
            printf("Enter label %d", i + 1);
            scanf("%s", tmp);
            //Utilising the strcpy to copy the contents of tmp into data->labels[i]
            strcpy(data->labels[i], tmp);
        }
        data->colstart=3;
        data->colend=COLS;
    }
//If choice is 1 , then the DataTable is initialised from an already existing Datatable
to have specific rows and columns
    else if(choice==1) {
        //Declaration of array of pointers of type location
        location *assignmemory[COLS];

        //Creating size in heap for DataTable_t
        data = (DataTable_t *) malloc(sizeof(DataTable_t));
        //Creating size in heap for data->columns through assignmemory array
        for (int i = 0; i < COLS; i++) {
            assignmemory[i] = (location *) malloc(sizeof(location));
            data->columns[i] = assignmemory[i];
        }
    }
    //Returning DataTable_t pointer
    return data;
}
```

2a)deinitDT()

Explanation

In this method all sources pertaining to the DataTable_t structure are relinquished through the free function.

The deinitDT method in functionClass2a.c class:

```
//Declaration of method deinitDT with DataTable_T pointer parameter
void deinitDT(DataTable_t *data){

    //Freeing the data->columns from memory
    for( int i=data->colstart;i<data->colend;i++){
        free(data->columns[i]);
    }
    //Freeing the DataTable_t from memory
    free(data);

}
```

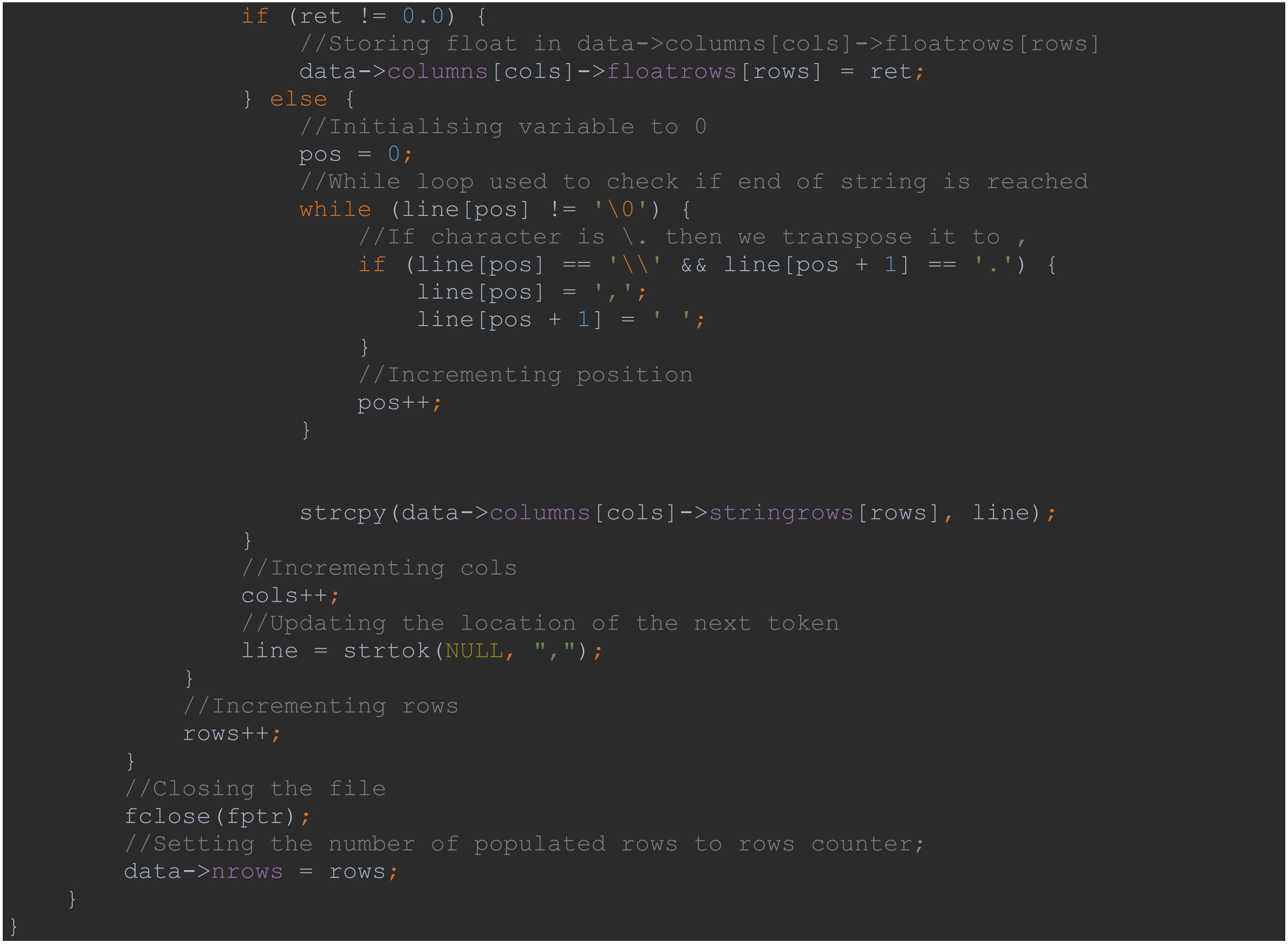
2a)loadDT()

Explanation

In this method contents of a csv file are loaded into a newly initialised DataTable_t structure. This was executed by first opening the file in read mode (if file is not found error message is displayed) and reading a line at a time from the file and placing it in a string called line. The string was then parsed to check whether a \, occurs, if so, it was replaced with a \., to not be recognised as a column separator. Afterwards, the string line was split into tokens, each token being a cell in the DataTable. Atof was used to check if the column was a float or string, and based on the respective data types, the cells were added to the DataTable_t. In case the cell is string \. Is parsed into a ' , ' followed by a space. Following this, the csv file was closed by utilising fclose() and the meta data for the number of populated rows was set to the counter rows.

The loadDT method in functionClass2a.c class:

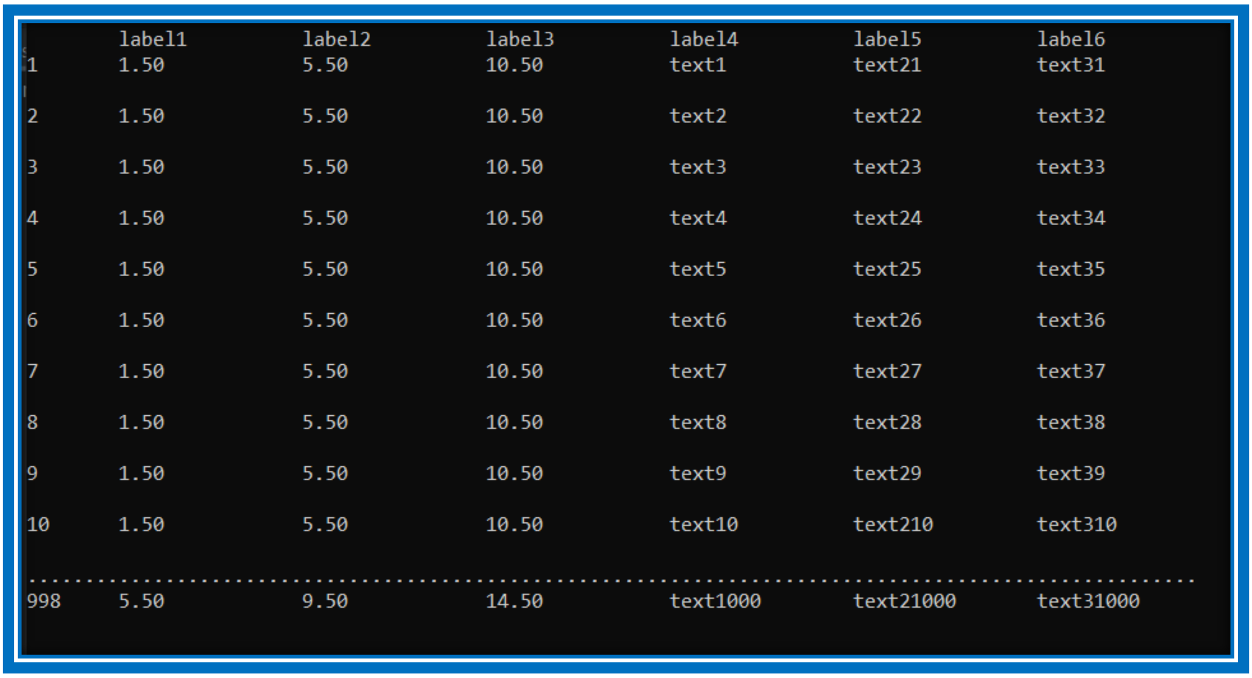
```
//Declaration of method loadDT with DataTable_T pointer parameter
void loadDT(DataTable_t *data){
    //Declaration of file pointer fptr
    FILE *fptr;
    //Opening the file in reading mode
    if((fptr=fopen("TestingFILE.csv","r"))==NULL)
    {
        //If file is not found an Error message is presented
        printf("Error file not found");
    }
    else {
        //Declaring of variable tmp to act as a temporary string
        char tmp[SSIZE];
        //Declaring variable rows
        int rows = 0;
        //Reading one line from file
        while (fgets(tmp, SSIZE, fptr) != NULL) {
            //Declaring of variable line to hold the current line read from file
            char *line;
            //Declaring variable to hold the position of string tmp
            int pos = 0;
            //While loop used to check if end of string is reached
            while (tmp[pos] != '\0') {
//If character is \, then we transpose it to \. to not be influenced by strtok later on
                if (tmp[pos] == '\\' && tmp[pos + 1] == ',') {
                    tmp[pos] = '\\';
                    tmp[pos + 1] = '.';
                }
                //Incrementing position
                pos++;
            }

//Separating the text read form file into tokens, each token is separated by a , since
it is a CSV file
            line = strtok(tmp, ",");
//Declaring variable cols to act as a counter for the columns in the DataTable_t
            int cols = 0;
            //While loop being used to check if end of line was not reached
            while (line != NULL) {
                //Utilising function atof to transpose token line into float
                float ret = atof(line);
//Checking if variable ret !=0, since an atof of a string token would be 0 else token
is a string
```



```
                if (ret != 0.0) {
                    //Storing float in data->columns[cols]->floatrows[rows]
                    data->columns[cols]->floatrows[rows] = ret;
                } else {
                    //Initialising variable to 0
                    pos = 0;
                    //While loop used to check if end of string is reached
                    while (line[pos] != '\0') {
                        //If character is \. then we transpose it to ,
                        if (line[pos] == '\\' && line[pos + 1] == '.') {
                            line[pos] = ',';
                            line[pos + 1] = ' ';
                        }
                        //Incrementing position
                        pos++;
                    }


                    strcpy(data->columns[cols]->stringrows[rows], line);
                }
                //Incrementing cols
                cols++;
                //Updating the location of the next token
                line = strtok(NULL, ",");
            }
            //Incrementing rows
            rows++;
        }
        //Closing the file
        fclose(fptr);
        //Setting the number of populated rows to rows counter;
        data->nrows = rows;
    }
}
```

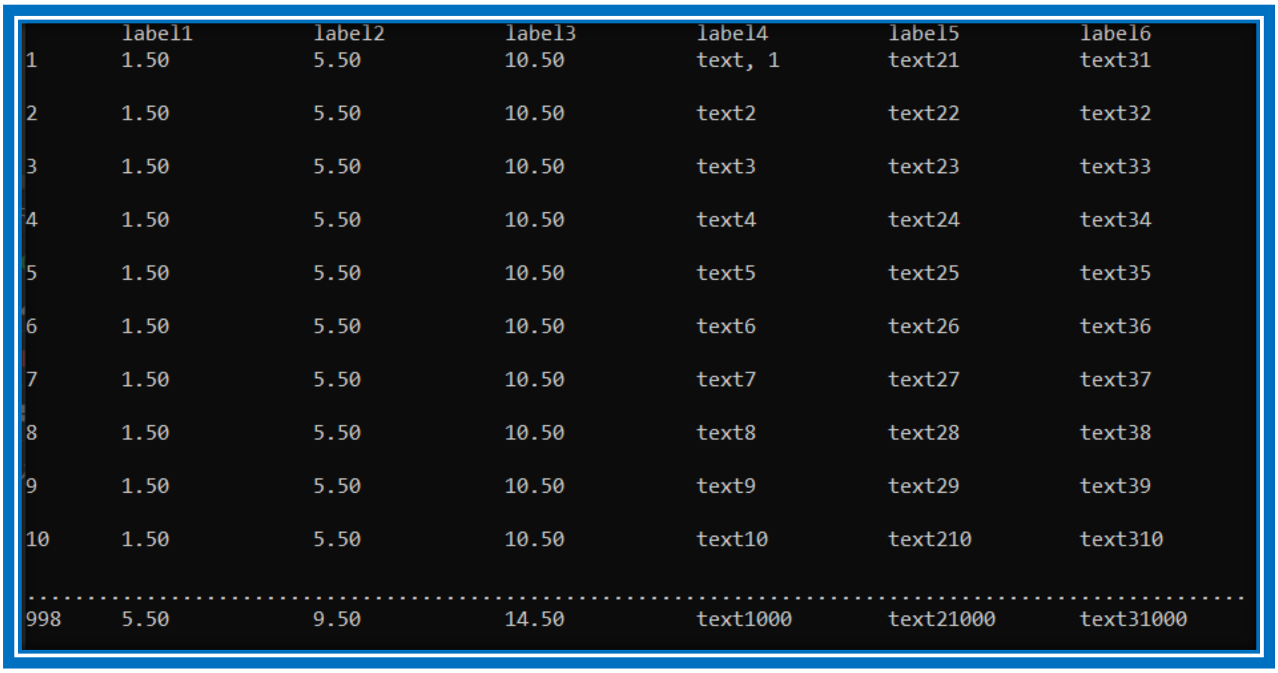
Testing

Utilising the showDT method for testing purposes to show the data loaded from TestingFILE.csv into the DataTable_t:



```
        label1          label2          label3          label4          label5          label6
1       1.50            5.50            10.50           text1           text21          text31

2       1.50            5.50            10.50           text2           text22          text32

3       1.50            5.50            10.50           text3           text23          text33

4       1.50            5.50            10.50           text4           text24          text34

5       1.50            5.50            10.50           text5           text25          text35

6       1.50            5.50            10.50           text6           text26          text36

7       1.50            5.50            10.50           text7           text27          text37

8       1.50            5.50            10.50           text8           text28          text38

9       1.50            5.50            10.50           text9           text29          text39

10      1.50            5.50            10.50           text10          text210         text310

.....................................................................................................
998     5.50            9.50            14.50           text1000        text21000       text31000
```

Utilising the showDT method for testing purposes to show the how the method handles \, as can be seen in row 1 in label4:



```
        label1          label2          label3          label4          label5          label6
1       1.50            5.50            10.50           text, 1         text21          text31

2       1.50            5.50            10.50           text2           text22          text32

3       1.50            5.50            10.50           text3           text23          text33

4       1.50            5.50            10.50           text4           text24          text34

5       1.50            5.50            10.50           text5           text25          text35

6       1.50            5.50            10.50           text6           text26          text36

7       1.50            5.50            10.50           text7           text27          text37

8       1.50            5.50            10.50           text8           text28          text38

9       1.50            5.50            10.50           text9           text29          text39

10      1.50            5.50            10.50           text10          text210         text310

......................................................................................................
998     5.50            9.50            14.50           text1000        text21000       text31000
```

2a)exportDT()

Explanation

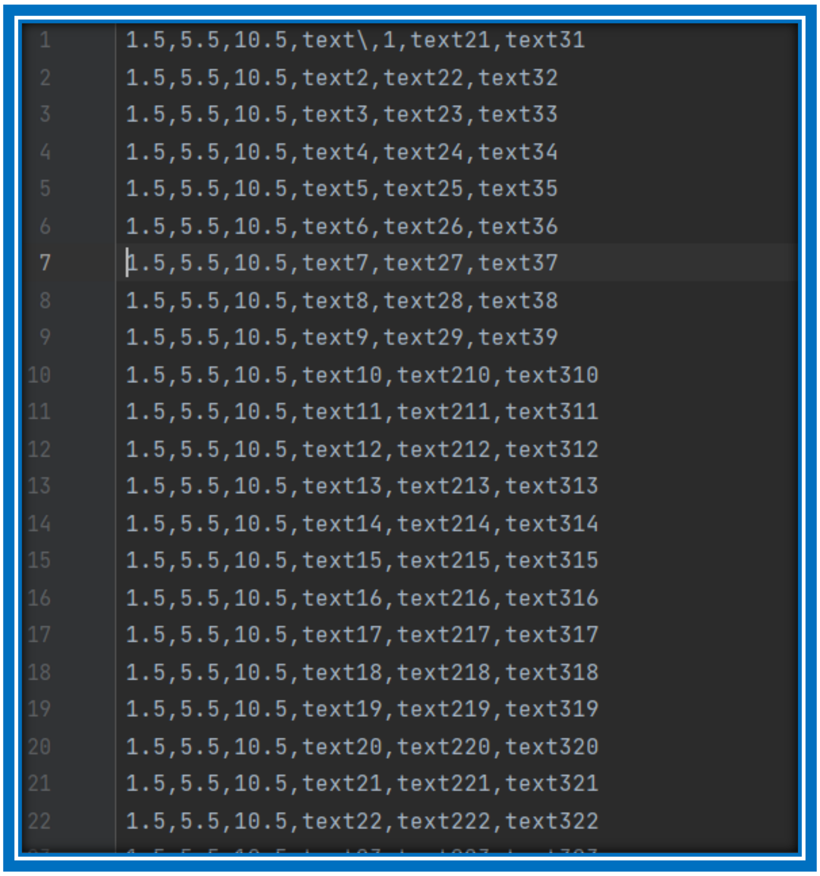
In this method contents of DataTable_t structure are exported into a csv file. This was executed by first opening the OutputFile.csv in writing mode. The method loops through all the cells in the DataTable_t structure and appends the cells to a string, separating each cell with a ‘,’. After the method loops through a whole row, it resorts to print the string onto the OutputFile.csv. If the last character of the string is not a \n, the method will print a \n to the file to signify end of row. The string cells which had a ’ ,’ followed by a space were parsed to store a \, in the csv file. The file was closed by the fclose() function.

The exportDT method in functionClass2a.c class:

```
//Declaration of method exportDT with DataTable_T pointer parameter
void exportDT(DataTable_t *data){
    //Declaration of file pointer fptr
    FILE *fptr;
    //Opening the file in writing mode
    fptr=fopen("OutputFile.csv","w");
    //Declaration of char array temp and initialising it all to null
    char temp[ROWS]={'\0'};


    //Looping through all the populated rows
    for(int i=0;i<data->nrows;i++){
        //Looping through all the populated columns
        for(int j=0;j<data->colend;j++) {
            if(j<data->colstart){
                //Declaration of char array text
                char text[ROWS];
                //Transforming the float into string using gcvt
                gcvt(data->columns[j]->floatrows[i],4,text);
                //Appending the float to the string text
                strcat(temp,text);
                //Appending a , to the string to signify end of entity
                strcat(temp,",");
            }
            else{
                //Creating a string words to hold data->columns[j]->stringrows[i]
                char *words=data->columns[j]->stringrows[i];
                //Declaring variable to hold the position of string tmp
                int pos=0;
                //While loop used to check if end of string is reached
                while(words[pos]!='\0'){
                    //If character is , then we transpose it to \,
                    if(words[pos]==','&&words[pos+1]==' '){
                        words[pos]='\\';
                        words[pos+1]=',';
                    }
                    //Incrementing position
                    pos++;
                }
                //Appending the words to the string temp
                strcat(temp,words);
//If statement being used to append a , to the string temp to signify end of entity,
besides when final column is reached
                if(j!=data->colend-1) {
                    strcat(temp, ",");
                }
            }
        }

        //Printing to file the string temp
        fprintf(fptr, "%s", temp);
        //Printing \n to file to indicate a new row in case there wasn't
        if(temp[strlen(temp)-1]!='\n'){
            //Printing to file the string temp
            fprintf(fptr, "%s", "\n");
        }
        //Clearing contents of string temp
        memset(temp,'\0',sizeof(temp));
    }

    //Closing the file
    fclose(fptr);
}
```

Testing

Output in OutputFile.csv:



```
1.5,5.5,10.5,text\,1,text21,text31
1.5,5.5,10.5,text2,text22,text32
1.5,5.5,10.5,text3,text23,text33
1.5,5.5,10.5,text4,text24,text34
1.5,5.5,10.5,text5,text25,text35
1.5,5.5,10.5,text6,text26,text36
1.5,5.5,10.5,text7,text27,text37
1.5,5.5,10.5,text8,text28,text38
1.5,5.5,10.5,text9,text29,text39
1.5,5.5,10.5,text10,text210,text310
1.5,5.5,10.5,text11,text211,text311
1.5,5.5,10.5,text12,text212,text312
1.5,5.5,10.5,text13,text213,text313
1.5,5.5,10.5,text14,text214,text314
1.5,5.5,10.5,text15,text215,text315
1.5,5.5,10.5,text16,text216,text316
1.5,5.5,10.5,text17,text217,text317
1.5,5.5,10.5,text18,text218,text318
1.5,5.5,10.5,text19,text219,text319
1.5,5.5,10.5,text20,text220,text320
1.5,5.5,10.5,text21,text221,text321
1.5,5.5,10.5,text22,text222,text322
```

2a)showDT()

Explanation

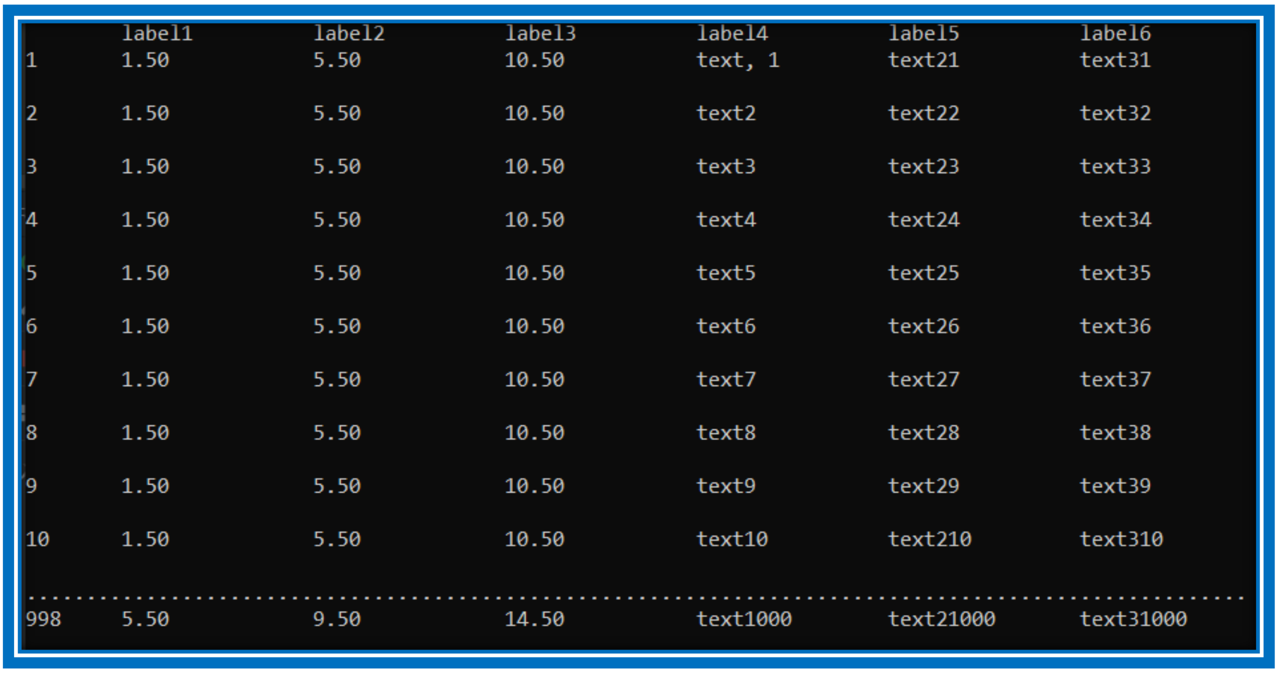
In this method contents of DataTable_t structure are displayed, showing the labels followed by the first 10 rows, and the last row. This was achieved by utilising a nested for loop to loop through all the columns and the first 10 rows, printing the relevant cells respectively. If there are less than 10 rows, all the rows are printed. After the first 10 rows are printed a line of ' …' is printed which is followed by printing all the columns in the last row. Row numbers are also clearly displayed at the beginning of each row. If each cell has more than 10 characters, only the first 10 characters are displayed.

The showDT method in functionClass2a.c class:

```
//Declaration of method showDT with DataTable_T pointer parameter
void showDT(DataTable_t *data){
    //For loop being used to show the labels
    printf("\t");
    for(int i=0;i<data->colend;i++) {
        printf("%-10.10s\t", data->labels[i]);
    }
    printf("\n");
    //If number of rows is smaller than 10 than ten than temp holds data->nrows
    //Else temp is set to hold 10
    int temp;
    if(data->nrows>10){
        temp=10;
    }else {
        temp = data->nrows;
    }
    //For loop being used to print the first 10 rows and all columns
    for(int j=0;j<=temp;j++) {
        //Displaying the row numbers
        if(j!=temp) {
            printf("%d\t", j + 1);
        }
        for(int i=0;i<data->colend;i++) {
//if the final row is reached .... are printed else data->columns[i]->floatrows[j] is printed
            if(j==temp){
                printf(".................");
            }
            else if(i<data->colstart){//Displaying the float rows
                printf("%-10.2f\t", data->columns[i]->floatrows[j]);
            }
            else{//Displaying the string rows
                printf("%-10.10s\t", data->columns[i]->stringrows[j]);
            }
        }
        printf("\n");
    }
    //Printing the number of the last row
    printf("%d\t",(int)data->nrows);
    //Looping through all the columns in the last row
    for(int i=0;i<data->colend;i++) {
//If the column is float data->columns[i]->floatrows[(int)data->nrows-1] is printed
        if(i<data->colstart) {
            printf("%-10.2f\t", data->columns[i]->floatrows[(int)data->nrows-1]);
        }
//Else the column is string data->columns[i]->stringrows[(int)data->nrows-1] is printed
        else{
            printf("%-10.10s\t", data->columns[i]->stringrows[(int)data->nrows-1]);
        }
    }

}
```

Testing

Utilising the showDT method to print the DataTable_t structure:



```
        label1          label2          label3          label4          label5          label6
1       1.50            5.50            10.50           text, 1         text21          text31

2       1.50            5.50            10.50           text2           text22          text32

3       1.50            5.50            10.50           text3           text23          text33

4       1.50            5.50            10.50           text4           text24          text34

5       1.50            5.50            10.50           text5           text25          text35

6       1.50            5.50            10.50           text6           text26          text36

7       1.50            5.50            10.50           text7           text27          text37

8       1.50            5.50            10.50           text8           text28          text38

9       1.50            5.50            10.50           text9           text29          text39

10      1.50            5.50            10.50           text10          text210         text310

.........................................................................................................
998     5.50            9.50            14.50           text1000        text21000       text31000
```

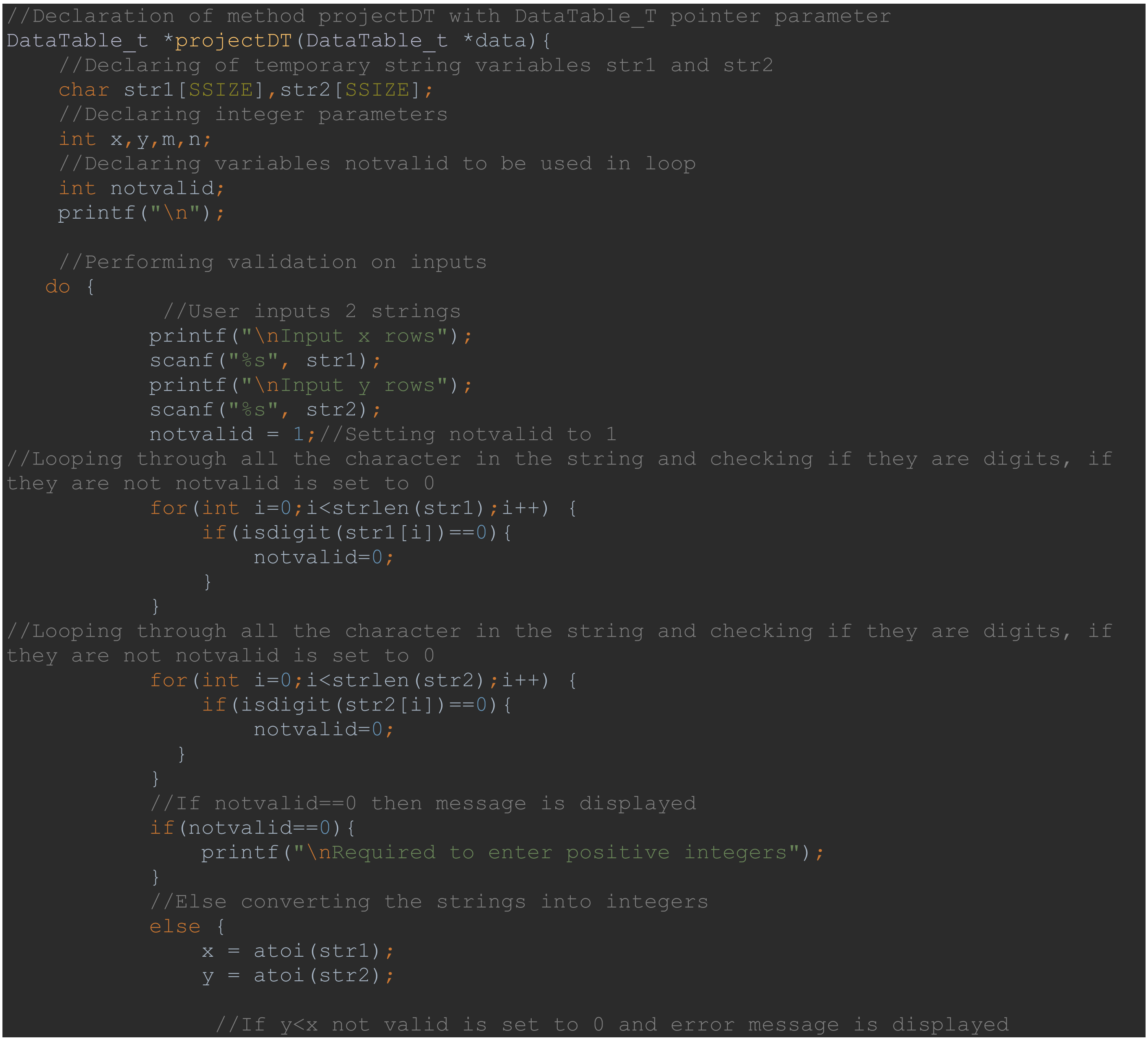
2a)projectDT()

Explanation

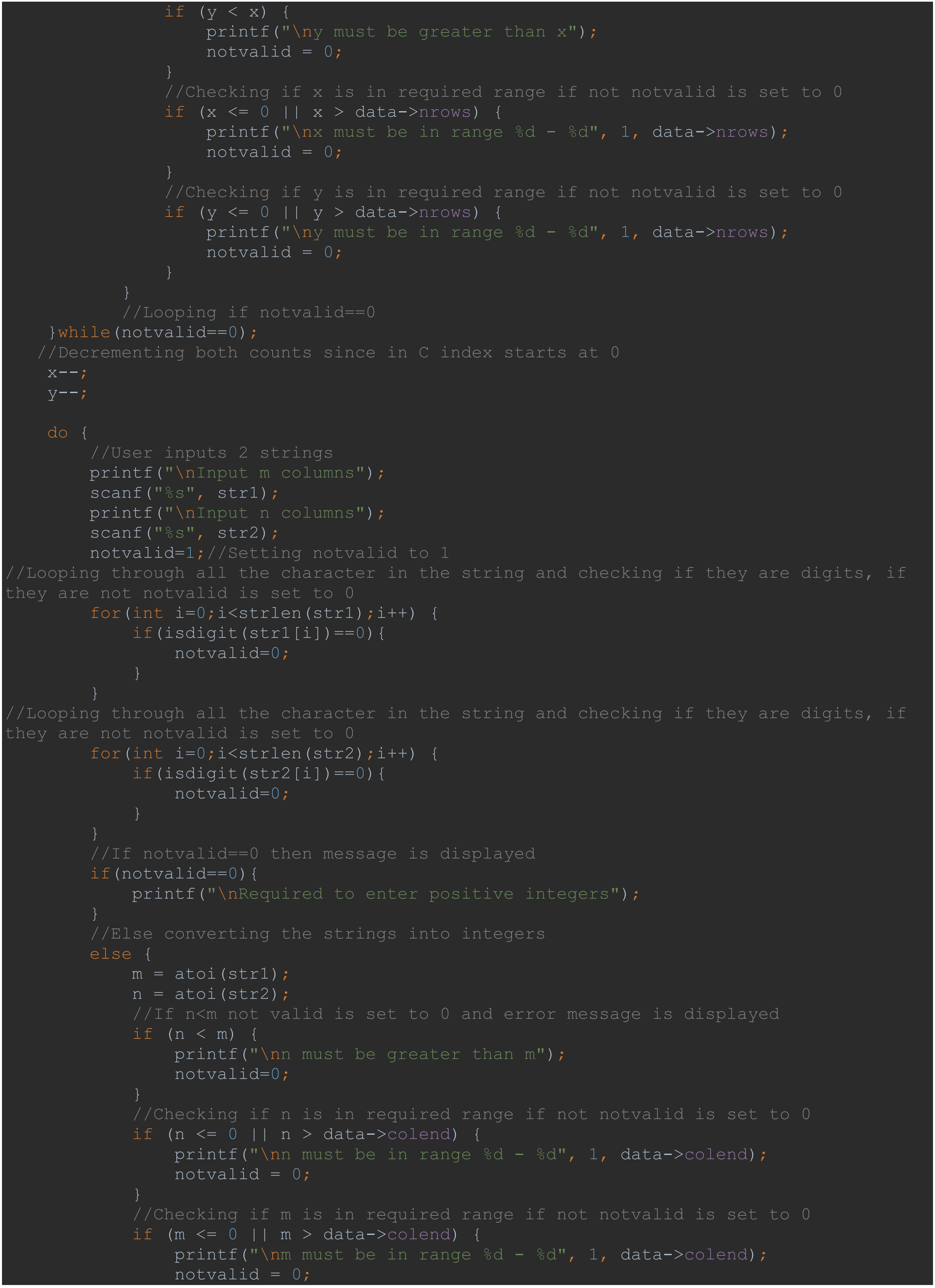
This method returns a pointer to a new DataTable_t structure but containing the rows from x to y and columns from m to n. This method requires the user to input x, y, m and n and proper validation is ensured for all inputs, in case the user input; is not in the required range, is not of an integer data type or if the second input is smaller than the first one. The method then declares and initialises a new DataTable_t, reusing the initDT function but passing 1 as a parameter, as a new DataTable is created upon specific specifications. Meta data for the new DataTable structure is set to have the column start (first column which is a string) and column end at (n-m)+1.

The creation of this meta data was required for the new DataTable to utilise the other methods. The relevant cells are copied from the initial DataTable to the new one.

The projectDT method in functionClass2a.c class:



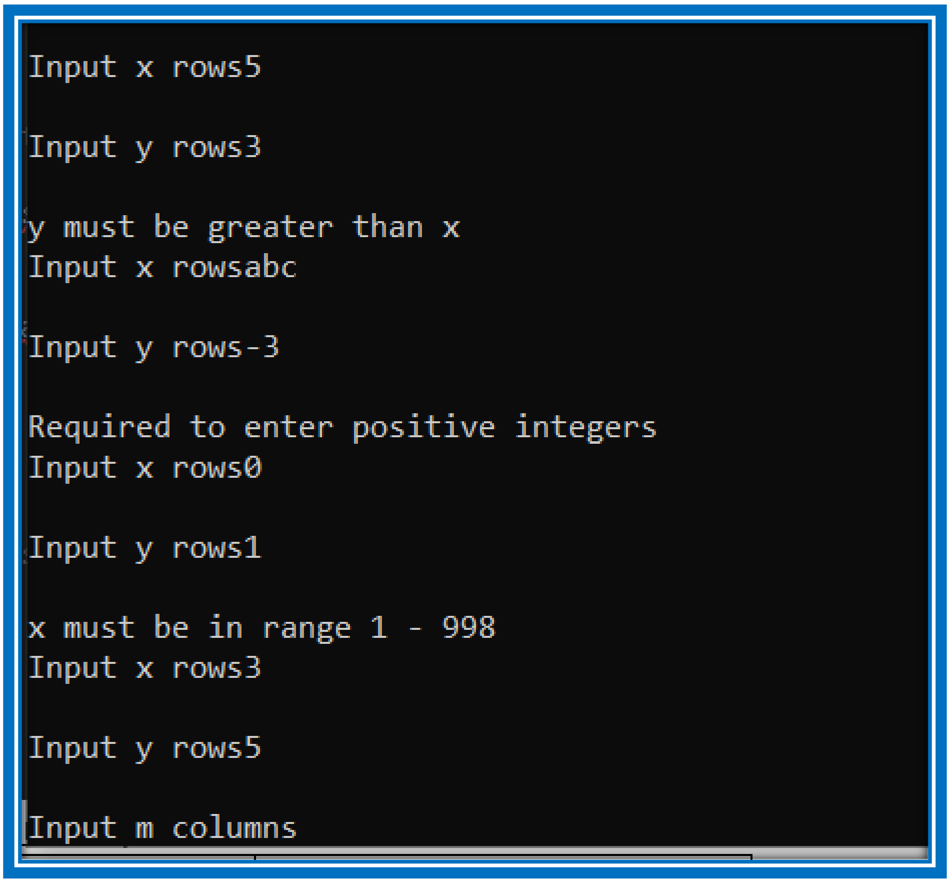
```
//Declaration of method projectDT with DataTable_T pointer parameter
DataTable_t *projectDT(DataTable_t *data){
    //Declaring of temporary string variables str1 and str2
    char str1[SSIZE],str2[SSIZE];
    //Declaring integer parameters
    int x,y,m,n;
    //Declaring variables notvalid to be used in loop
    int notvalid;
    printf("\n");

    //Performing validation on inputs
   do {
            //User inputs 2 strings
           printf("\nInput x rows");
           scanf("%s", str1);
           printf("\nInput y rows");
           scanf("%s", str2);
           notvalid = 1;//Setting notvalid to 1
//Looping through all the character in the string and checking if they are digits, if
they are not notvalid is set to 0
           for(int i=0;i<strlen(str1);i++) {
               if(isdigit(str1[i])==0){
                   notvalid=0;
               }
           }
//Looping through all the character in the string and checking if they are digits, if
they are not notvalid is set to 0
           for(int i=0;i<strlen(str2);i++) {
               if(isdigit(str2[i])==0){
                   notvalid=0;
             }
           }
           //If notvalid==0 then message is displayed
           if(notvalid==0){
               printf("\nRequired to enter positive integers");
           }
           //Else converting the strings into integers
           else {
               x = atoi(str1);
               y = atoi(str2);

                //If y<x not valid is set to 0 and error message is displayed
```



```
               if (y < x) {
                   printf("\ny must be greater than x");
                   notvalid = 0;
               }
               //Checking if x is in required range if not notvalid is set to 0
               if (x <= 0 || x > data->nrows) {
                   printf("\nx must be in range %d - %d", 1, data->nrows);
                   notvalid = 0;
               }
               //Checking if y is in required range if not notvalid is set to 0
               if (y <= 0 || y > data->nrows) {
                   printf("\ny must be in range %d - %d", 1, data->nrows);
                   notvalid = 0;
               }
           }
           //Looping if notvalid==0
    }while(notvalid==0);
   //Decrementing both counts since in C index starts at 0
    x--;
    y--;

    do {
        //User inputs 2 strings
        printf("\nInput m columns");
        scanf("%s", str1);
        printf("\nInput n columns");
        scanf("%s", str2);
        notvalid=1;//Setting notvalid to 1
//Looping through all the character in the string and checking if they are digits, if
they are not notvalid is set to 0
        for(int i=0;i<strlen(str1);i++) {
            if(isdigit(str1[i])==0){
                notvalid=0;
            }
        }
//Looping through all the character in the string and checking if they are digits, if
they are not notvalid is set to 0
        for(int i=0;i<strlen(str2);i++) {
            if(isdigit(str2[i])==0){
                notvalid=0;
            }
        }
        //If notvalid==0 then message is displayed
        if(notvalid==0){
            printf("\nRequired to enter positive integers");
        }
        //Else converting the strings into integers
        else {
            m = atoi(str1);
            n = atoi(str2);
            //If n<m not valid is set to 0 and error message is displayed
            if (n < m) {
                printf("\nn must be greater than m");
                notvalid=0;
            }
            //Checking if n is in required range if not notvalid is set to 0
            if (n <= 0 || n > data->colend) {
                printf("\nn must be in range %d - %d", 1, data->colend);
                notvalid = 0;
            }
            //Checking if m is in required range if not notvalid is set to 0
            if (m <= 0 || m > data->colend) {
                printf("\nm must be in range %d - %d", 1, data->colend);
                notvalid = 0;
            }
        }
        //Looping if notvalid==0
    }while(notvalid==0);
    //Decrementing both counts since in C index starts at 0
    m--;
    n--;

    //Calculating the amount of rows and columns the new Datatable will have:
    int tablerows=(y-x)+1;
    int tablecols=0;
    //Declaring and initialising new DataTable_t
    DataTable_t *data2= initDT(1);
    data2->nrows=tablerows;
        data2->colend =(n-m) +1;
    //Resetting tablecols to 0

    tablecols=0;
    //Looping through respective columns in initial DataTable
    for(int i=m;i<=n;i++) {
        //Copying data->labels[i] into data2->labels[tablecols]
        strcpy(data2->labels[tablecols],data->labels[i]);
        //Incrementing tablecols
        tablecols++;
    }
    //Temporary variable check declared and initialised to 0
    int check=0;
    //Resetting tablecols and tablerows
    tablerows=0;
    tablecols=0;
    //Looping through the rows
    for(int j=x;j<=y;j++) {
        //Looping through the columns
        for (int i = m; i <= n; i++) {
      //Copying the relevant cells based on the whether the row is a float or string
            if (i<data->colstart) {//float
                data2->columns[tablecols]->floatrows[tablerows]= data->columns[i]-
                                  >floatrows[j];
            } else {//String
                if(check==0) {
      //meta data column start being used to know the first index of the string rows
                    data2->colstart = tablecols;
                    //Setting check to 1 to prevent entering if statement
                    check=1;
                }
                strcpy(data2->columns[tablecols]->stringrows[tablerows], data-
                                >columns[i]->stringrows[j]);
            }
            //Incrementing tablecols
            tablecols++;
        }
        //Incrementing tablerows and resetting tablecols
        tablerows++;
        tablecols=0;
    }
    //Returning DataTable_t pointer
    return data2;
}
```

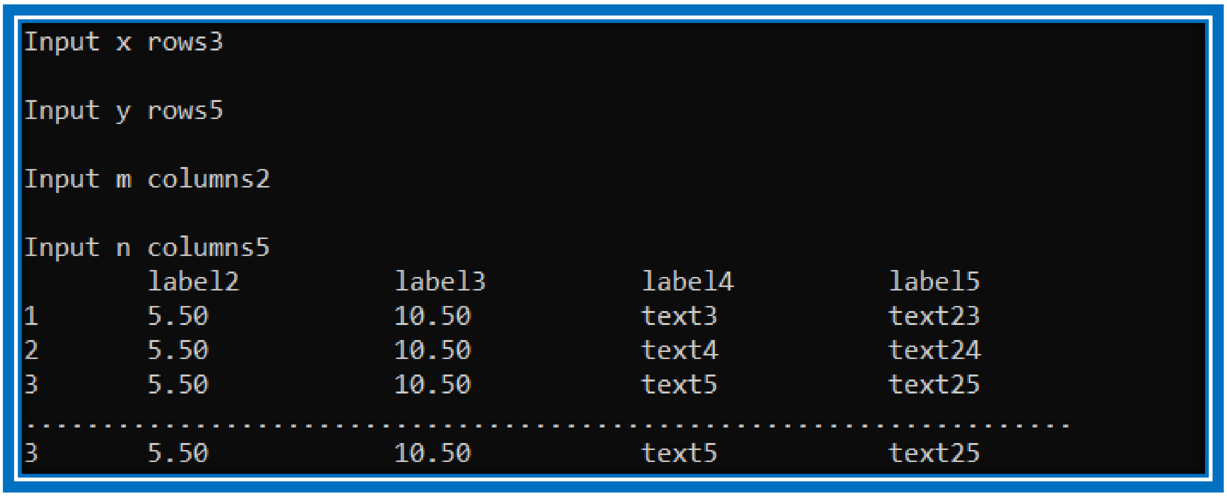
Testing User input (Similar validation for m & n):

Input	Type of data	Output
x=3,y=5	Valid data	Valid data
x=5,y=3	Invalid data X must be smaller than y	Invalid data X must be smaller than y
X=abc,y=-3	Invalid data Required to enter positive integers	Invalid data Required to enter positive integers
X=0,y=1	Invalid data Not in range	Invalid data Not in range



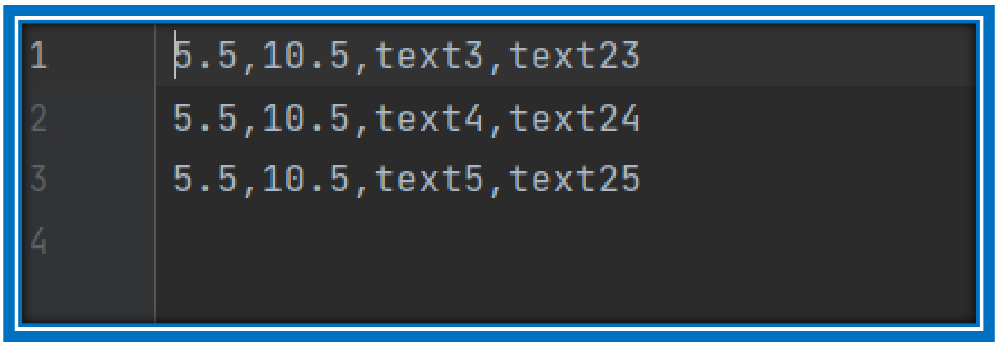
```
Input x rows5

Input y rows3

y must be greater than x
Input x rowsabc

Input y rows-3

Required to enter positive integers
Input x rows0

Input y rows1

x must be in range 1 - 998
Input x rows3

Input y rows5

Input m columns
```

Utilising the showDT method for testing purposes to show the data in the new DataTable_t structure:



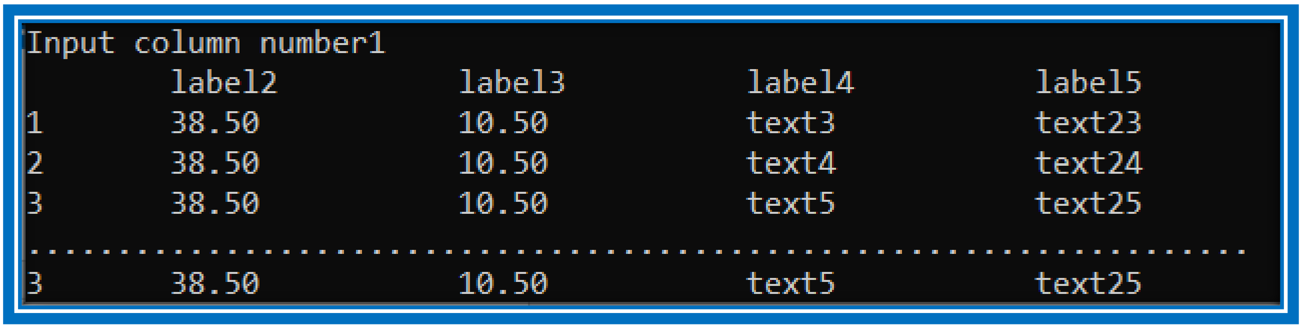
```
Input x rows3

Input y rows5

Input m columns2

Input n columns5
        label2          label3          label4          label5
1       5.50            10.50           text3           text23
2       5.50            10.50           text4           text24
3       5.50            10.50           text5           text25

.....................................................................
3       5.50            10.50           text5           text25
```

Utilising the exportDT method for testing purposes to show utilisation of the new DataTable_t structure:



```
5.5,10.5,text3,text23
5.5,10.5,text4,text24
5.5,10.5,text5,text25
```

Utilising the mutateDT and showDT methods for testing purposes and mutating column 1 to show utilisation of the new DataTable_t structure:



```
Input column number1
        label2          label3          label4          label5
1       38.50           10.50           text3           text23
2       38.50           10.50           text4           text24
3       38.50           10.50           text5           text25
....................................................................
3       38.50           10.50           text5           text25
```

2a)mutateDT()

Explanation

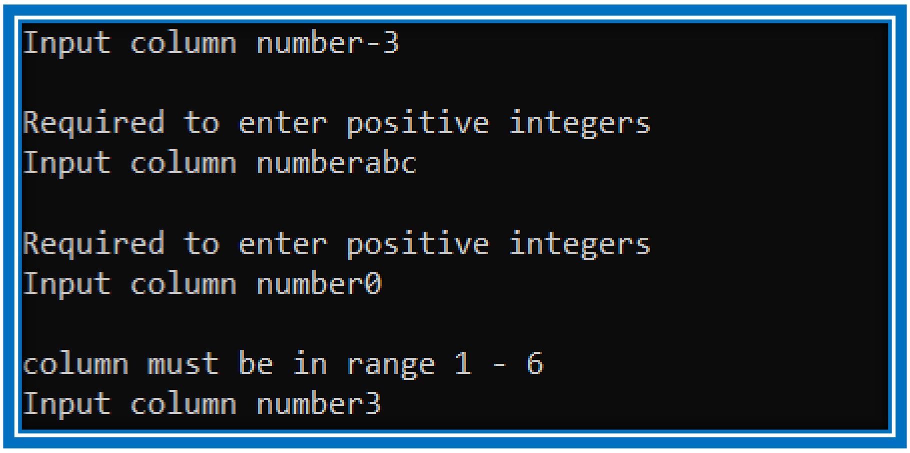
This method accepts a user-defined function and changes the respective column chosen by the user. If the column is a float, it will multiply all the rows in said column by 7. If the column is a string it will replace any character which is a 't' into an 'a'. Proper validation is ensured on the user input column, in case the user input; is not in the required range, is not of an integer data type.

The mutateDT method in functionClass2a.c class:

```
//Declaration of method mutateDT with DataTable_T pointer parameter
void mutateDT(DataTable_t *data,float (*floatfunction)(float num,int change),void
(*stringfunction)(char* string,char strchange,char strreplace)){
    //Declaration of variabls inptcol,notvalid
    int inptcol,notvalid;
    //Declaration of integer variable which will be used to multiply float rows by 7
    int change=7;
    //Declaring of temporary string str1
    char str1[SSIZE];
//Declaration of characters t and s which will be used to transpose the character t
into a in string columns
    char strchange='t',strreplace='a';
    //Displaying options
    printf("\nChoose a column if a string column is chosen 't' will be replaced by 'a'
           and \nif an integer column is chosen the column will be multiplied by 7\n");
    //User validation for input
    do {
        //User input String
        printf("\nInput column number");
        scanf("%s", str1);
        notvalid=1;
//Looping through all the character in the string and checking if they are digits, if
they are not notvalid is set to 0
        for(int i=0;i<strlen(str1);i++) {
            if(isdigit(str1[i])==0){
                notvalid=0;
            }
        }
        //If notvalid==0 then message is displayed
        if(notvalid==0){
            printf("\nRequired to enter positive integers");
        }
        //Else converting the strings into integers
        else {
            inptcol = atoi(str1);
            //Checking if inptol is in required range if not notvalid is set to 0
            if (inptcol <= 0|| inptcol > data->colend) {
                printf("\ncolumn must be in range %d - %d",1, data->colend);
                notvalid = 0;
            }
        }
        //Looping if notvalid==0
    }while(notvalid==0);
    //Decrementing both counts since in C index starts at 0
    inptcol--;
    //Looping through all the columns
    for(int i=0;i<data->nrows;i++) {
        //If column is string column else column is integer column
        if (inptcol >= 3) {
            //Calling user defined function
            stringfunction(data->columns[inptcol]->stringrows[i],strchange,strreplace);
        } else {//If column is float
            //Calling user defined function
            float temp=data->columns[inptcol]->floatrows[i];
             data->columns[inptcol]->floatrows[i]=floatfunction(temp,change);
        }
    }
}
```

Testing User input:

Input	Type of data	Output
Inptcol=3	Valid data	Valid data
Inptcol=-3,inptcol=abc	Invalid data Required to enter positive integers	Invalid data Required to enter positive integers
inptcol=0	Invalid data Not in range	Invalid data Not in range

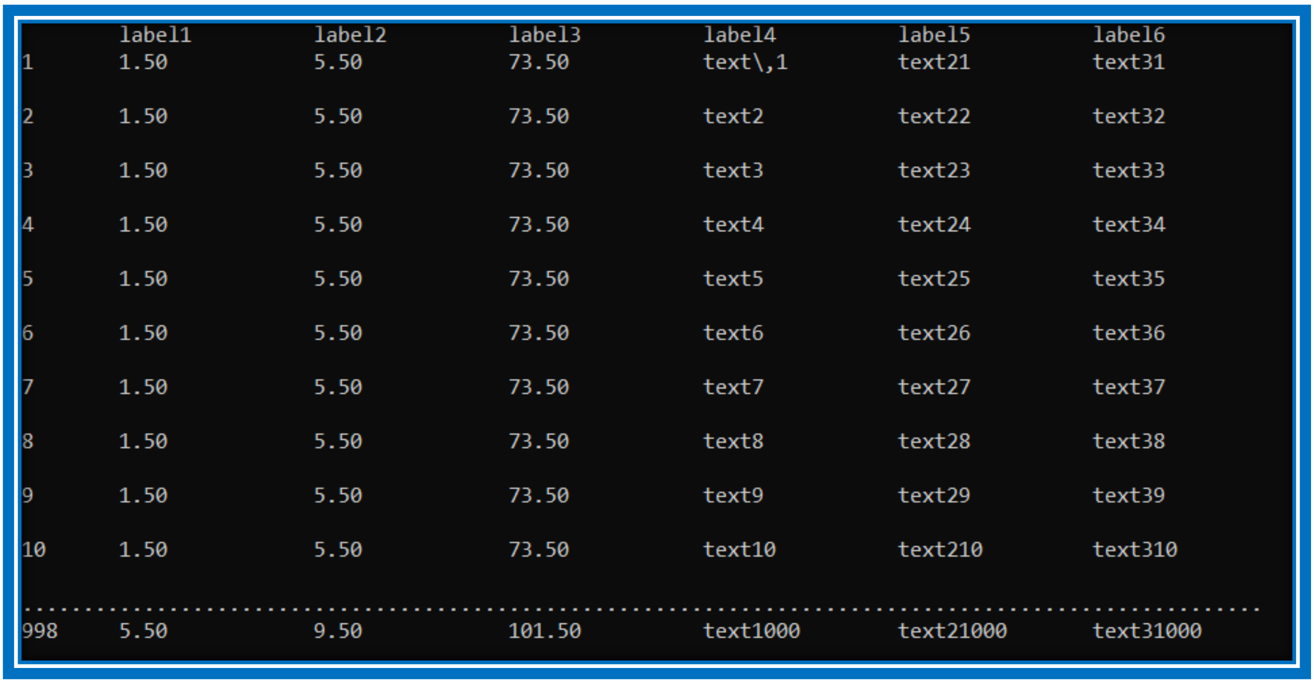


```
Input column number-3

Required to enter positive integers
Input column numberabc

Required to enter positive integers
Input column number0

column must be in range 1 - 6
Input column number3
```

The following is the user defined function to multiply the float cell with change(value of change is always 7):

```
//User defined functions:
float floatnumbers(float num, int change)
{
    return (float)change*num;
}
```

Utilising the showDT method for testing purposes to show the data change in the DataTable_t structure (performing mutation on column 3):

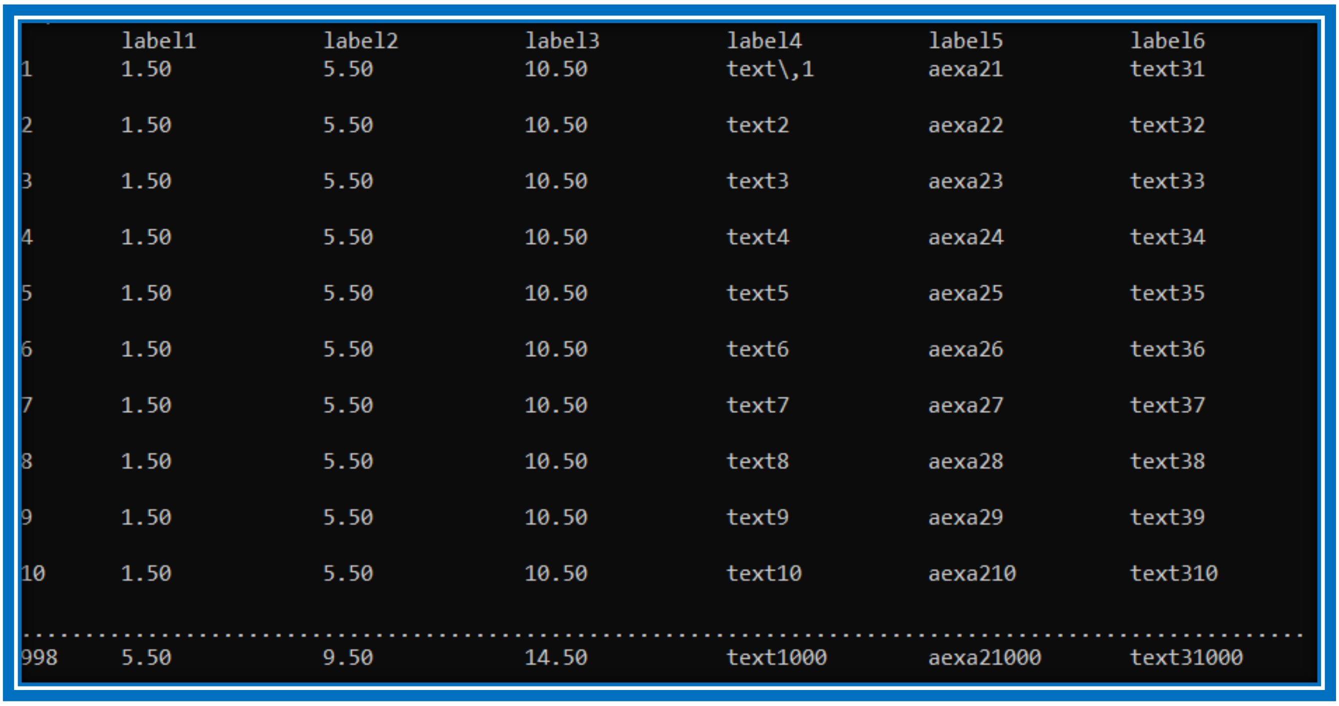


```
        label1          label2          label3          label4          label5          label6
1       1.50            5.50            73.50           text\,1         text21          text31

2       1.50            5.50            73.50           text2           text22          text32

3       1.50            5.50            73.50           text3           text23          text33

4       1.50            5.50            73.50           text4           text24          text34

5       1.50            5.50            73.50           text5           text25          text35

6       1.50            5.50            73.50           text6           text26          text36

7       1.50            5.50            73.50           text7           text27          text37

8       1.50            5.50            73.50           text8           text28          text38

9       1.50            5.50            73.50           text9           text29          text39

10      1.50            5.50            73.50           text10          text210         text310

......................................................................................................
998     5.50            9.50            101.50          text1000        text21000       text31000
```

The following is the user defined function to replace a ‘t’ with an ‘a’:

```
void stringinput(char* string,char strchange,char strreplace)
{
    int pos=0;//Position to hold position of string array
    //Checking if tmp has reached end of string
    while(string[pos]!='\0'){
        //If tmp[pos] is 't' then tmp[pos] is changed to 'a'
        if(string[pos]==strchange){
            string[pos]=strreplace;
        }
        //Incrementing position
        pos++;
    }
}
```

Utilising the showDT method for testing purposes to show the data change in the DataTable_t structure (performing mutation on column 5):



```
        label1         label2         label3         label4         label5         label6
1       1.50           5.50           10.50          text\,1        aexa21         text31

2       1.50           5.50           10.50          text2          aexa22         text32

3       1.50           5.50           10.50          text3          aexa23         text33

4       1.50           5.50           10.50          text4          aexa24         text34

5       1.50           5.50           10.50          text5          aexa25         text35

6       1.50           5.50           10.50          text6          aexa26         text36

7       1.50           5.50           10.50          text7          aexa27         text37

8       1.50           5.50           10.50          text8          aexa28         text38

9       1.50           5.50           10.50          text9          aexa29         text39

10      1.50           5.50           10.50          text10         aexa210        text310

.................................................................................................
998     5.50           9.50           14.50          text1000       aexa21000      text31000
```

The functionClass2a.h included the declaration of the following functions to be used in the test driver:

```
//Declaring of methods:
//Declaration of method initDT with interger parameters choice
DataTable_t *initDT(int choice);
//Declaration of method deinitDT with DataTable_T pointer parameter
void deinitDT(DataTable_t *data);
//Declaration of method loadDT with DataTable_T pointer parameter
void loadDT(DataTable_t *data);
//Declaration of method exportDT with DataTable_T pointer parameter
void exportDT(DataTable_t *data);
//Declaration of method showDT with DataTable_T pointer parameter
void showDT(DataTable_t *data);
//Declaration of method projectDT with DataTable_T pointer parameter
DataTable_t *projectDT(DataTable_t *data);
//Declaration of method mutateDT with DataTable_T pointer and user defined functions as
parameters
void mutateDT(DataTable_t *data,float (*floatfunction)(float num,int change),void
(*stringfunction)(char* string,char strchange,char strreplace));

//User defined functions:
float floatnumbers(float num,int change);

void stringinput(char* string,char strchange,char strreplace);
```

The test driver class: main2a.c

```
//Runner file
//Header file used in the program:
#include "functionClass2a.h"

int main() {
//Calling the respective methods from the functionClass.c
// Calling initDT method with choice-0 , since creating a new DataTable from file
    DataTable_t *data=initDT(0);
    loadDT(data);
    showDT(data);
    exportDT(data);
    DataTable_t *data2=projectDT(data);
    showDT(data2);
    deinitDT(data2);
    mutateDT(data,floatnumbers,stringinput);
    showDT(data);
    deinitDT(data);

    return 0;
}
```

3. A DataTable library b)

The following functions were implemented in the class functionClass2b.c and utilised the following structs from the functionClass2b.h header file:

```
//Header file which allows main class to access methods of functionClass.c files

/*General types definition */
//Declaration of union location
typedef union{
    //Assigning a float value
    float floatrows;
    //Assigning an integer value
    int introws;
    //Assigning a pointer to a char array
    char *stringrows;
}location;

//Declaration of struct DataTable_t
typedef struct{
    //A pointer to an array containing pointers to union location to have both float, integer and character string values
    location **columns;
    //An integer to hold the number of populated rows
    int nrows;
    //A pointer pointing to strings (i.e. char*)
    char **labels;
    //An array containing the column types
    int *coltype;
    //An integer to hold the number of populated columns
    int ncols;
}DataTable_t;
```

The Class made use of the following libraries as can be seen below and the header class functionClass2b.h:

```
//functionClass file which includes methods to be used by runner file
//Libraries used in the program:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
//Header file used in the program
#include "functionClass2b.h"
```

For testing purposes, a Testing file containing 1003 rows and 7 columns was used for Task2b), the columns had different datatypes.

2b)initDT()

Explanation

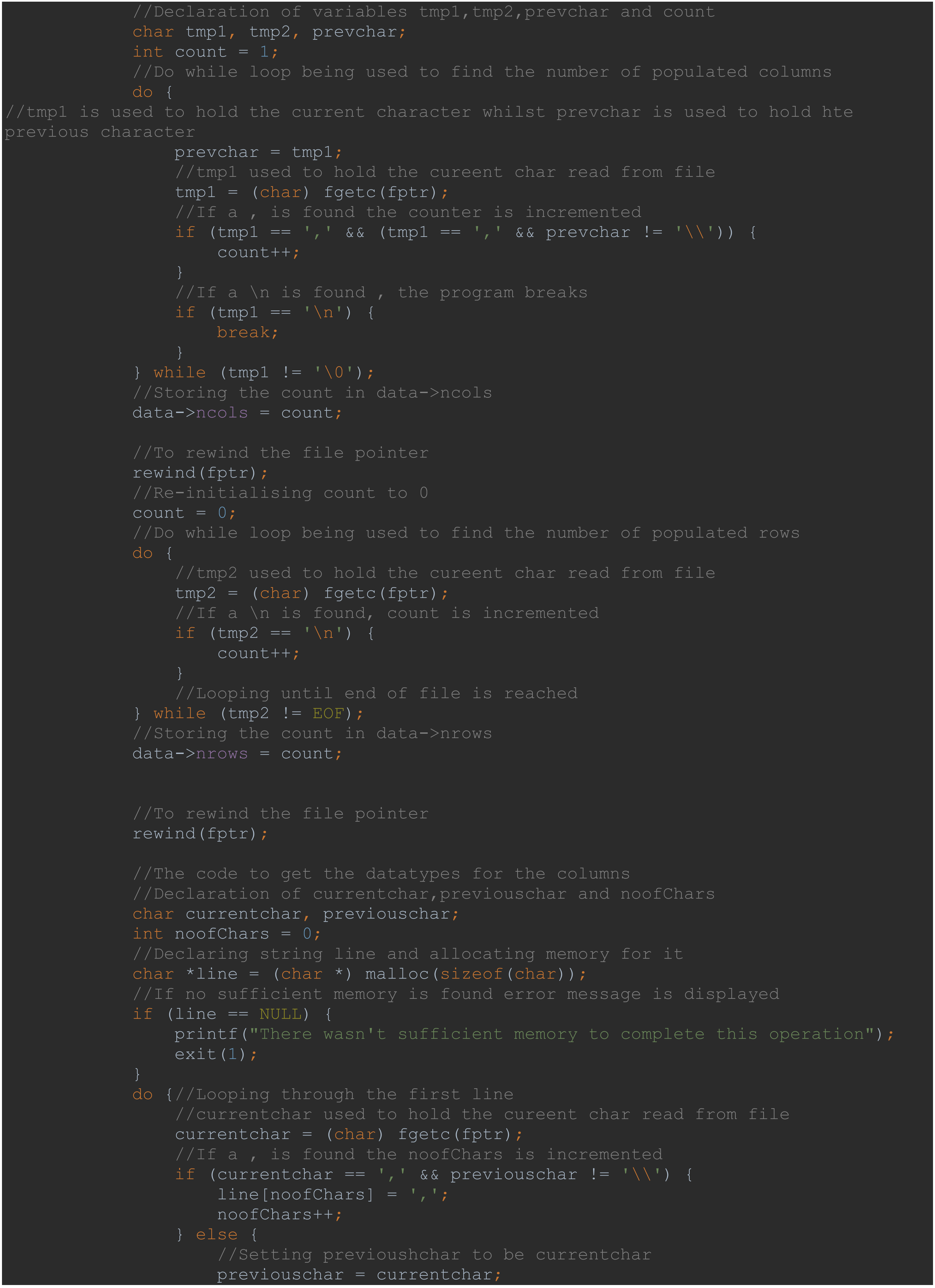
In this method a new DataTable_t structure is initialised, by allocating memory for said DataTable on the heap. The function allows 3 integer parameters to be passed, the first one being the choice, and the other two are the number of columns and number of rows respectively. If the parameter is 0, this implies an initial commitment of the DataTable_t, but if the parameter is 1 then a new DataTable_t is created upon specific specifications (i.e., specific rows and columns). When the initial commitment is chosen, the Testing2.csv file is opened in reading mode to calculate the number of rows, columns, and column data types in the DataTable_t structure.

- To get the number of columns needed for the DataTable_t structure; a do while loop is being utilised, when a ‘,’ is present in the file (implying a new column) the column counter is incremented. The method loops until ‘\n’ is reached.
- To get the number of rows needed for the DataTable_t structure; a do while loop is being utilised, when a ‘\n’ is present in the file (implying a new row) the row counter is incremented. The method loops until end of file is reached.
- To get the column data types needed for the DataTable_t structure; the first line of the csv file is stored in a string, this is achieved through a do while loop. In continuation a for loop is being used to loop through all the characters in said string, if a ‘.’ Is found the points counter is incremented, else if a digit is found the digits counter is incremented, else if a letter is found the chars counter is incremented, else the others counter is incremented. These 4 counters were utilised to find the relevant types of each column. If the column is float, the meta data array for column type is set to signal bit 1, if column is integer, the signal bit is set to 2 and if the column is string, signal bit is set to 4.

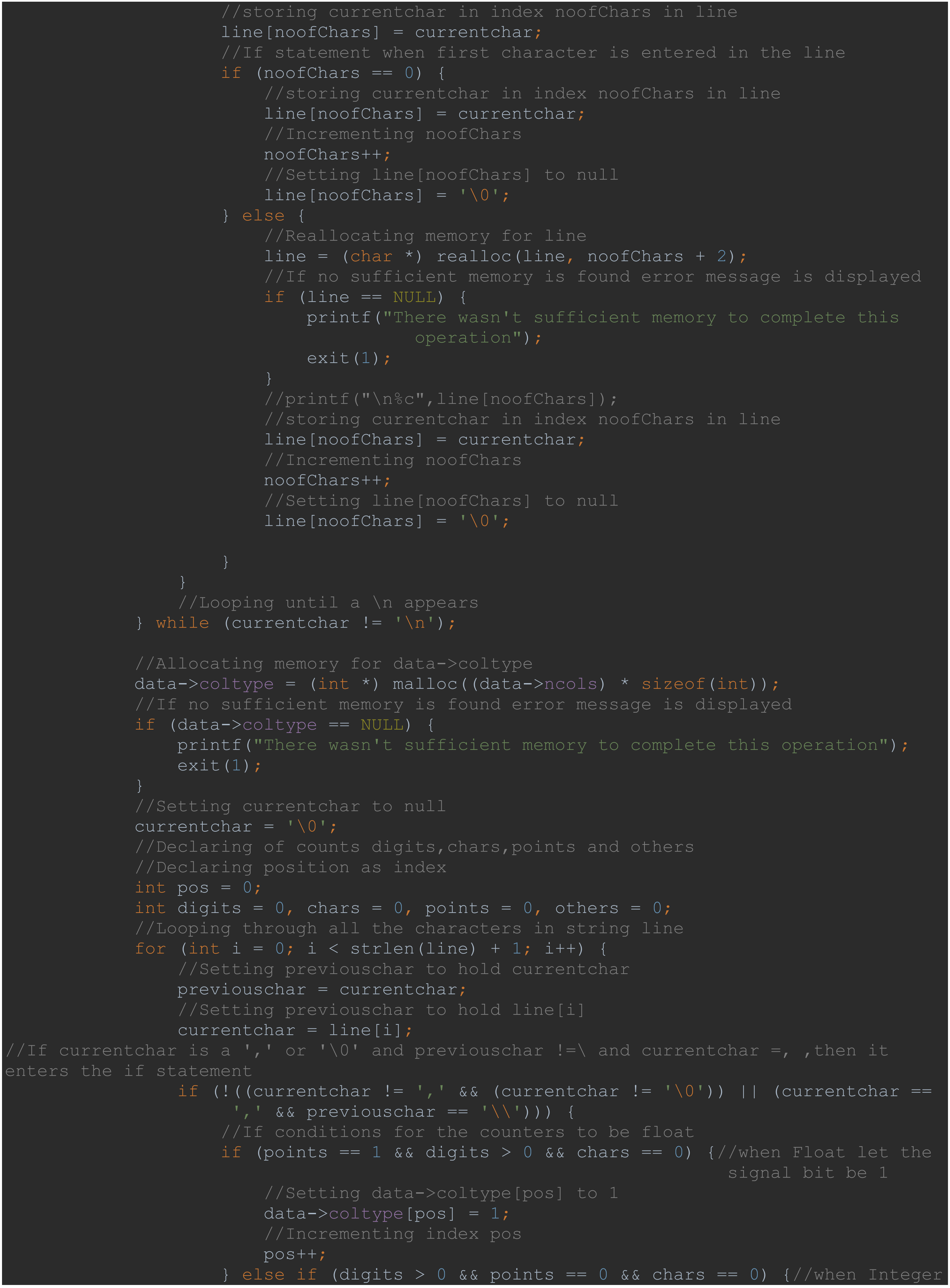
The user is also prompted to enter the column labels for each column, which is achieved through a dynamic string array, adding a character at a time to a temporary string and then copying contents of said string to the label array in DataTable_t structure.

The initDT method in functionClass2b.c class:

```
//Declaration of method initDT with interger parameters choice,noofcols and noofrows
DataTable_t *initDT(int choice , int noofcols,int noofrows) {
    //Declaration of DataTable_t pointer
    DataTable_t *data;

    //If choice is 0 , then the DataTable is initialised from a file
    if(choice==0) {
        //Creating size in heap for DataTable_t
        data = (DataTable_t *) malloc(sizeof(DataTable_t));
        //If no sufficient memory is found error message is displayed
        if (data == NULL) {
            printf("There wasn't sufficient memory to complete this operation");
            exit(1);
        }

        //Declaring file pointer and opening file in read mode
        FILE *fptr;
        if ((fptr = fopen("Testing2.csv", "r")) == NULL) {
            //If file is not found an Error message is presented
            printf("Error file not found");
        } else {
```

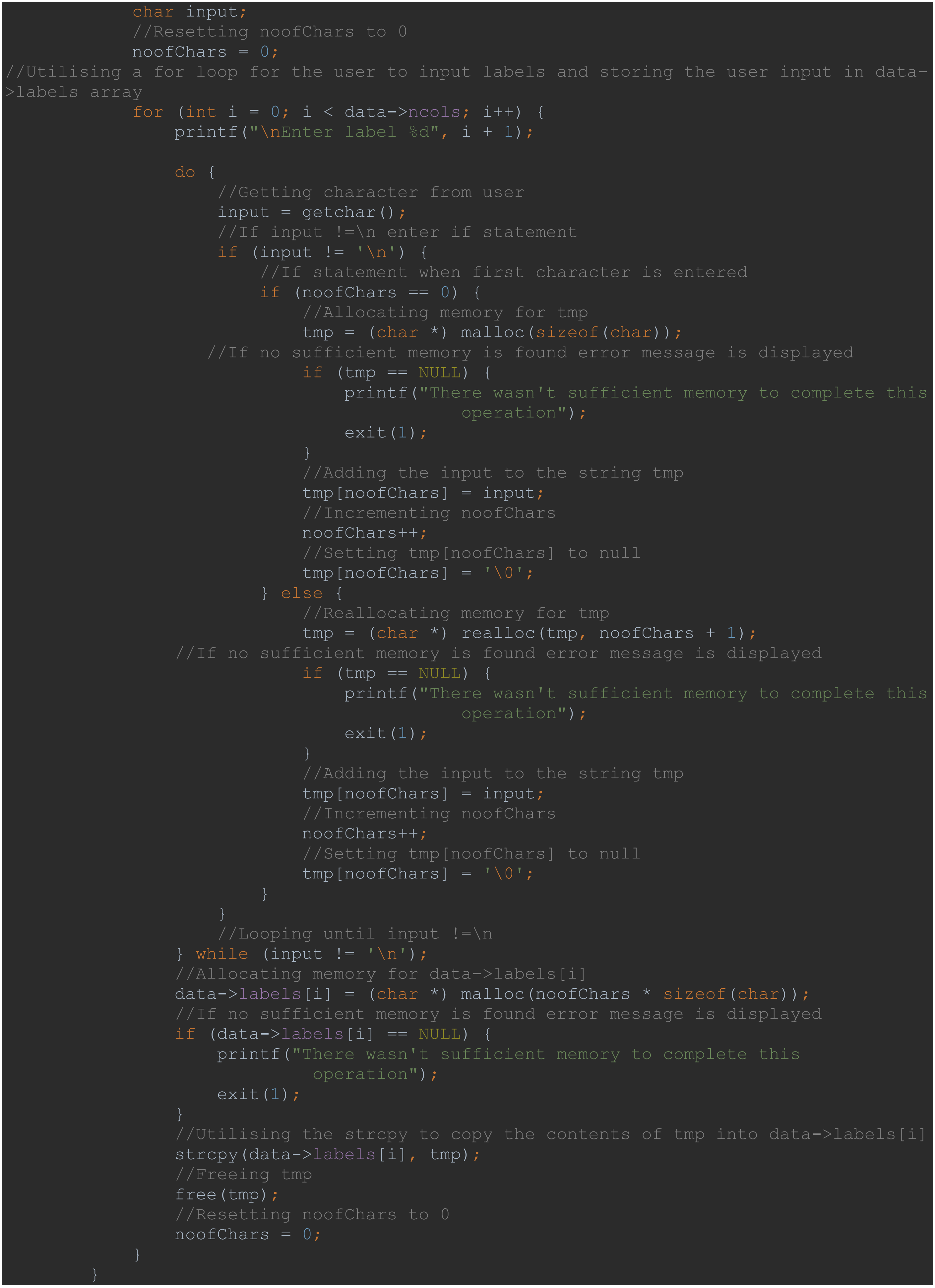


```
            //Declaration of variables tmp1,tmp2,prevchar and count
            char tmp1, tmp2, prevchar;
            int count = 1;
            //Do while loop being used to find the number of populated columns
            do {
//tmp1 is used to hold the current character whilst prevchar is used to hold hte
previous character
                prevchar = tmp1;
                //tmp1 used to hold the cureent char read from file
                tmp1 = (char) fgetc(fptr);
                //If a , is found the counter is incremented
                if (tmp1 == ',' && (tmp1 == ',' && prevchar != '\\')) {
                    count++;
                }
                //If a \n is found , the program breaks
                if (tmp1 == '\n') {
                    break;
                }
            } while (tmp1 != '\0');
            //Storing the count in data->ncols
            data->ncols = count;

            //To rewind the file pointer
            rewind(fptr);
            //Re-initialising count to 0
            count = 0;
            //Do while loop being used to find the number of populated rows
            do {
                //tmp2 used to hold the cureent char read from file
                tmp2 = (char) fgetc(fptr);
                //If a \n is found, count is incremented
                if (tmp2 == '\n') {
                    count++;
                }
                //Looping until end of file is reached
            } while (tmp2 != EOF);
            //Storing the count in data->nrows
            data->nrows = count;


            //To rewind the file pointer
            rewind(fptr);

            //The code to get the datatypes for the columns
            //Declaration of currentchar,previouschar and noofChars
            char currentchar, previouschar;
            int noofChars = 0;
            //Declaring string line and allocating memory for it
            char *line = (char *) malloc(sizeof(char));
            //If no sufficient memory is found error message is displayed
            if (line == NULL) {
                printf("There wasn't sufficient memory to complete this operation");
                exit(1);
            }
            do {//Looping through the first line
                //currentchar used to hold the cureent char read from file
                currentchar = (char) fgetc(fptr);
                //If a , is found the noofChars is incremented
                if (currentchar == ',' && previouschar != '\\') {
                    line[noofChars] = ',';
                    noofChars++;
                } else {
                    //Setting previoushchar to be currentchar
                    previouschar = currentchar;
```



```
                    //storing currentchar in index noofChars in line
                    line[noofChars] = currentchar;
                    //If statement when first character is entered in the line
                    if (noofChars == 0) {
                        //storing currentchar in index noofChars in line
                        line[noofChars] = currentchar;
                        //Incrementing noofChars
                        noofChars++;
                        //Setting line[noofChars] to null
                        line[noofChars] = '\0';
                    } else {
                        //Reallocating memory for line
                        line = (char *) realloc(line, noofChars + 2);
                        //If no sufficient memory is found error message is displayed
                        if (line == NULL) {
                            printf("There wasn't sufficient memory to complete this
                                      operation");
                            exit(1);
                        }
                        //printf("\n%c",line[noofChars]);
                        //storing currentchar in index noofChars in line
                        line[noofChars] = currentchar;
                        //Incrementing noofChars
                        noofChars++;
                        //Setting line[noofChars] to null
                        line[noofChars] = '\0';

                    }
                }
                //Looping until a \n appears
            } while (currentchar != '\n');

            //Allocating memory for data->coltype
            data->coltype = (int *) malloc((data->ncols) * sizeof(int));
            //If no sufficient memory is found error message is displayed
            if (data->coltype == NULL) {
                printf("There wasn't sufficient memory to complete this operation");
                exit(1);
            }
            //Setting currentchar to null
            currentchar = '\0';
            //Declaring of counts digits,chars,points and others
            //Declaring position as index
            int pos = 0;
            int digits = 0, chars = 0, points = 0, others = 0;
            //Looping through all the characters in string line
            for (int i = 0; i < strlen(line) + 1; i++) {
                //Setting previouschar to hold currentchar
                previouschar = currentchar;
                //Setting previouschar to hold line[i]
                currentchar = line[i];
//If currentchar is a ',' or '\0' and previouschar !=\ and currentchar =, ,then it
enters the if statement
                if (!((currentchar != ',' && (currentchar != '\0')) || (currentchar ==
                     ',' && previouschar == '\\'))) {
                    //If conditions for the counters to be float
                    if (points == 1 && digits > 0 && chars == 0) {//when Float let the
                                                                   signal bit be 1
                        //Setting data->coltype[pos] to 1
                        data->coltype[pos] = 1;
                        //Incrementing index pos
                        pos++;
                    } else if (digits > 0 && points == 0 && chars == 0) {//when Integer
```

```
               let the signal bit be 2
            // Setting data->coltype[pos] to 2
            data->coltype[pos] = 2;
            //Incrementing index pos
            pos++;
        } else {//when String let the signal bit be 4
            //Setting data->coltype[pos] to 4
            data->coltype[pos] = 4;
            //Incrementing index pos
            pos++;
        }
        //Resetting all counters to 0
        digits = 0, chars = 0, points = 0, others = 0;
    } else {
        //If currentchar is a letter , chars is incremented
        if (isalpha(currentchar) != 0) {
            chars++;
            //If currentchar is a digit , digits is incremented
        } else if (isdigit(currentchar) != 0) {
            digits++;
            //If currentchar is a ., points is incremented
        } else if (currentchar == '.') {
            points++;
            //Else others is incremented
        } else {
            others++;
        }
    }
}
//Freeing the line
free(line);
//Closing the file
fclose(fptr);

//Allocating size for all column rows
data->columns = (location **) malloc((data->ncols) * sizeof(location
                                                 *));//number of rows
//If no sufficient memory is found error message is displayed
if (data->columns == NULL) {
    printf("There wasn't sufficient memory to complete this operation");
    exit(1);
}
//Allocating size for all column columns
for (int i = 0; i < data->ncols; i++) {
    //Allocating more memory than required to be safe
    data->columns[i] = (location *) malloc((2*data->nrows+1) *
                                  sizeof(location));//number of columns
    //If no sufficient memory is found error message is displayed
    if (data->columns[i]== NULL) {
        printf("There wasn't sufficient memory to complete this
                        operation");
        exit(1);
    }
}

//Allocating size for column titles:
data->labels = (char **) malloc((data->ncols) * sizeof(char *));
//If no sufficient memory is found error message is displayed
if (data->labels == NULL) {
    printf("There wasn't sufficient memory to complete this operation");
    exit(1);
}
//Declaring string tmp and char input variabls
char *tmp;
```



```
            char input;
            //Resetting noofChars to 0
            noofChars = 0;
//Utilising a for loop for the user to input labels and storing the user input in data-
>labels array
            for (int i = 0; i < data->ncols; i++) {
                printf("\nEnter label %d", i + 1);

                do {
                    //Getting character from user
                    input = getchar();
                    //If input !=\n enter if statement
                    if (input != '\n') {
                        //If statement when first character is entered
                        if (noofChars == 0) {
                            //Allocating memory for tmp
                            tmp = (char *) malloc(sizeof(char));
                   //If no sufficient memory is found error message is displayed
                            if (tmp == NULL) {
                                printf("There wasn't sufficient memory to complete this
                                           operation");
                                exit(1);
                            }
                            //Adding the input to the string tmp
                            tmp[noofChars] = input;
                            //Incrementing noofChars
                            noofChars++;
                            //Setting tmp[noofChars] to null
                            tmp[noofChars] = '\0';
                        } else {
                            //Reallocating memory for tmp
                            tmp = (char *) realloc(tmp, noofChars + 1);
                //If no sufficient memory is found error message is displayed
                            if (tmp == NULL) {
                                printf("There wasn't sufficient memory to complete this
                                           operation");
                                exit(1);
                            }
                            //Adding the input to the string tmp
                            tmp[noofChars] = input;
                            //Incrementing noofChars
                            noofChars++;
                            //Setting tmp[noofChars] to null
                            tmp[noofChars] = '\0';
                        }
                    }
                    //Looping until input !=\n
                } while (input != '\n');
                //Allocating memory for data->labels[i]
                data->labels[i] = (char *) malloc(noofChars * sizeof(char));
                //If no sufficient memory is found error message is displayed
                if (data->labels[i] == NULL) {
                    printf("There wasn't sufficient memory to complete this
                             operation");
                    exit(1);
                }
                //Utilising the strcpy to copy the contents of tmp into data->labels[i]
                strcpy(data->labels[i], tmp);
                //Freeing tmp
                free(tmp);
                //Resetting noofChars to 0
                noofChars = 0;
            }
        }
```

```
    }
//If choice is 1 , then the DataTable is initialised from an already existing Datatable
to have specific rows and columns
    else if(choice==1){
        //Creating size in heap for DataTable_t
        data=(DataTable_t*)malloc(sizeof(DataTable_t));
        //If no sufficient memory is found error message is displayed
        if (data == NULL) {
            printf("There wasn't sufficient memory to complete this operation");
            exit(1);
        }
//Assigning the passed parameters to number of columns and number of rows of the
Datatable
        data->ncols=noofcols;
        data->nrows=noofrows;
        //Allocating memory for data->labels
        //Allocating more memory than required to be safe
        data->labels= (char **) malloc((2*data->ncols+1) * sizeof(char *));
        //If no sufficient memory is found error message is displayed
        if (data->labels == NULL) {
            printf("There wasn't sufficient memory to complete this operation");
            exit(1);
        }
        //Allocating memory for data->coltype
        //Allocating more memory than required to be safe
        data->coltype = (int *) malloc((2*data->ncols+1) * sizeof(int));
        //If no sufficient memory is found error message is displayed
        if (data->coltype == NULL) {
            printf("There wasn't sufficient memory to complete this operation");
            exit(1);
        }

        //Allocating size for all column rows
        //Allocating more memory than required to be safe
        data->columns = (location **) malloc((2*data->ncols+1) * sizeof(location
                                  *));//number of rows
        //If no sufficient memory is found error message is displayed
        if (data->columns == NULL) {
            printf("There wasn't sufficient memory to complete this operation");
            exit(1);
        }
        //Allocating size for all column columns
        for (int i = 0; i < data->ncols; i++) {
            //Allocating more memory than required to be safe
            data->columns[i] = (location *) malloc((2*data->nrows+1) *
                         sizeof(location));//number of columns
            //If no sufficient memory is found error message is displayed
            if (data->columns[i] == NULL) {
                printf("There wasn't sufficient memory to complete this operation");
                exit(1);
            }
        }

    }
    //Returning DataTable_t pointer
    return data;
}
```

2b)deinitDT()

Explanation

In this method all sources pertaining to the DataTable_t structure are relinquished through the free function.

The deinitDT method in functionClass2b.c class:

```
//Declaration of method deinitDT with DataTable_T pointer parameter
void deinitDT(DataTable_t *data){

    //Looping through all the columns
    for( int i=0;i<data->ncols;i++){
        //Freeing columns[i]and labels[i] from memory
        free(data->columns[i]);
        free(data->labels[i]);
    }
    //Freeing coltype,labels and columns from memory
    free(data->coltype);
    free(data->labels);
    free(data->columns);
    //Freeing the DataTable_t from memory
    free(data);

}
}
```

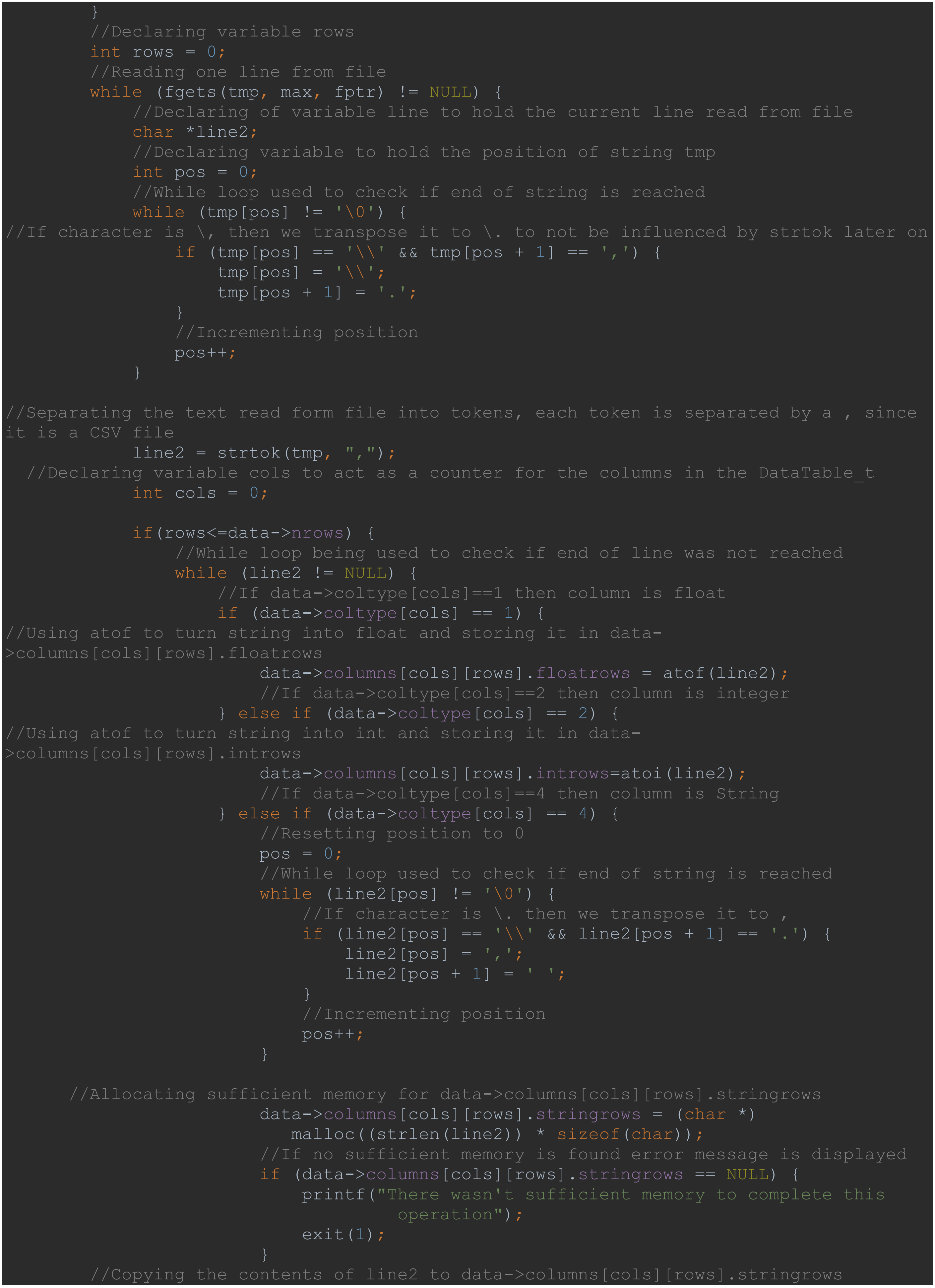
2b)loadDT()

Explanation

In this method contents of a csv file are loaded into a newly initialised DataTable_t structure. This was executed by first opening the file in read mode (If file is not found, error message is displayed) and reading a line at a time from the file and placing it in a string called line. The string was then parsed to check whether a \, occurs, if so, it was replaced with a \., to not be recognised as a column separator. Afterwards, the string line was split into tokens, each token being a cell in the DataTable. Based on the respective data types, the cells were added to the DataTable_t. In case the cell is string \. Is parsed into a , followed by a space. Following this, the csv file was closed by utilising fclose() and the meta data for the number of populated rows was set to the counter rows. Due to the columns and rows having dynamic features, the csv file can have more than 6 columns and more than 1000 rows, as well as have different data types.

The loadDT method in functionClass2b.c class:

```
//Declaration of method loadDT with DataTable_T pointer parameter
void loadDT(DataTable_t *data){
    //Declaration of file pointer fptr
    FILE *fptr;
    //Opening the file in reading mode
    if((fptr=fopen("Testing2.csv","r"))==NULL)
    {
        //If file is not found an Error message is presented
        printf("Error file not found");
    }
    else {
        //To find the maximum number of characters in a line
        //Declaring variables linecount , max and filechar
        int linecount=0,max=0;
        char filechar;
        do{//Looping through the whole file
            //Getting a character from the file and storing it in filechar
            filechar=fgetc(fptr);
            //If filechar =='\n' or EOF, enter if statement
            if(filechar=='\n'||filechar==EOF){
                //Finding the max number of characters in a line
                if(max<linecount){
                    max=linecount;
                }
                //Resetting linecount
                linecount=0;
            }
                //Incrementing linecount
            else{
                linecount++;
            }
            //Looping until end of file is reached
        }while(filechar!=EOF);

        //Rewinding file pointer
        rewind(fptr);
//Declaring of variable tmp and allocating memory for it to act as a temporary string
        char *tmp= (char *) malloc((max+1)*sizeof(char));
        //If no sufficient memory is found error message is displayed
        if (tmp == NULL) {
            printf("There wasn't sufficient memory to complete this operation");
            exit(1);
```

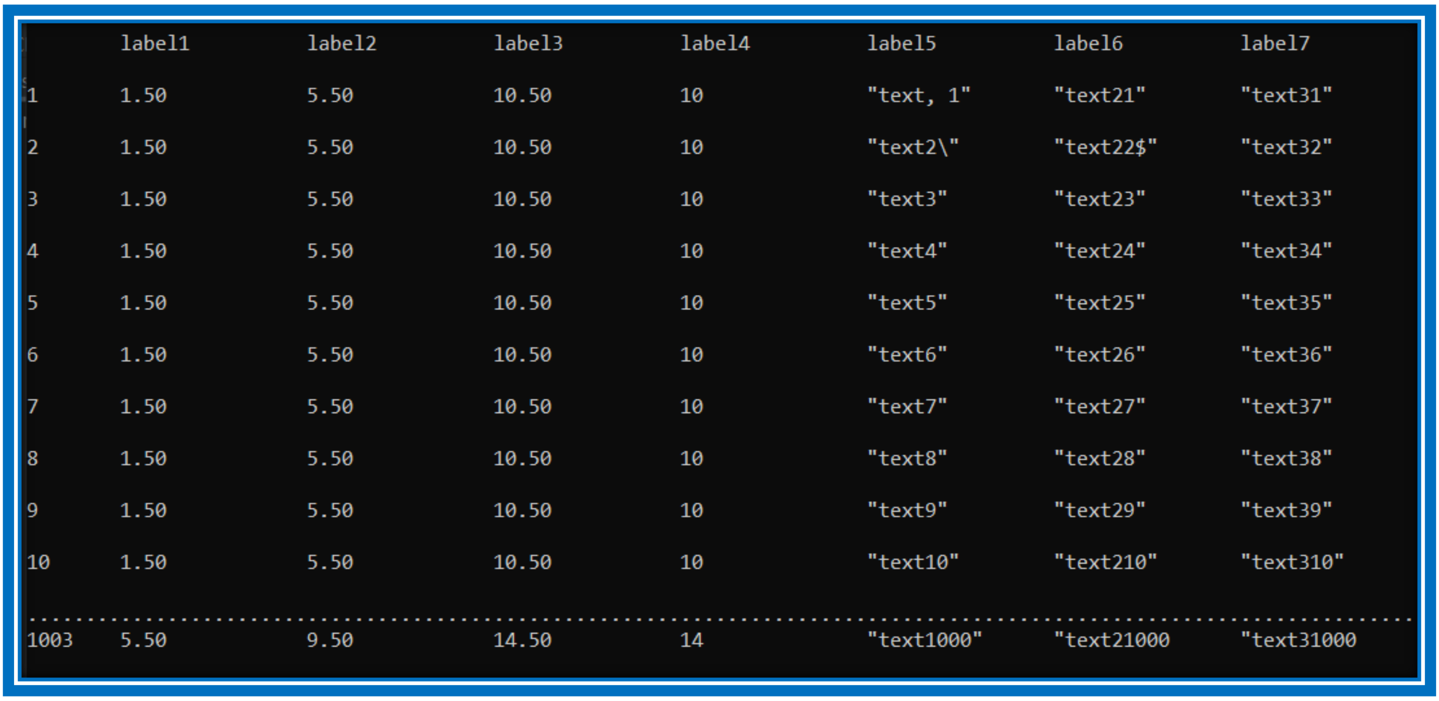


```
        }
        //Declaring variable rows
        int rows = 0;
        //Reading one line from file
        while (fgets(tmp, max, fptr) != NULL) {
            //Declaring of variable line to hold the current line read from file
            char *line2;
            //Declaring variable to hold the position of string tmp
            int pos = 0;
            //While loop used to check if end of string is reached
            while (tmp[pos] != '\0') {
//If character is \, then we transpose it to \. to not be influenced by strtok later on
                if (tmp[pos] == '\\' && tmp[pos + 1] == ',') {
                    tmp[pos] = '\\';
                    tmp[pos + 1] = '.';
                }
                //Incrementing position
                pos++;
            }

//Separating the text read form file into tokens, each token is separated by a , since
it is a CSV file
            line2 = strtok(tmp, ",");
  //Declaring variable cols to act as a counter for the columns in the DataTable_t
            int cols = 0;

            if(rows<=data->nrows) {
                //While loop being used to check if end of line was not reached
                while (line2 != NULL) {
                    //If data->coltype[cols]==1 then column is float
                    if (data->coltype[cols] == 1) {
//Using atof to turn string into float and storing it in data-
>columns[cols][rows].floatrows
                        data->columns[cols][rows].floatrows = atof(line2);
                        //If data->coltype[cols]==2 then column is integer
                    } else if (data->coltype[cols] == 2) {
//Using atof to turn string into int and storing it in data-
>columns[cols][rows].introws
                        data->columns[cols][rows].introws=atoi(line2);
                        //If data->coltype[cols]==4 then column is String
                    } else if (data->coltype[cols] == 4) {
                        //Resetting position to 0
                        pos = 0;
                        //While loop used to check if end of string is reached
                        while (line2[pos] != '\0') {
                            //If character is \. then we transpose it to ,
                            if (line2[pos] == '\\' && line2[pos + 1] == '.') {
                                line2[pos] = ',';
                                line2[pos + 1] = ' ';
                            }
                            //Incrementing position
                            pos++;
                        }

      //Allocating sufficient memory for data->columns[cols][rows].stringrows
                        data->columns[cols][rows].stringrows = (char *)
                           malloc((strlen(line2)) * sizeof(char));
                        //If no sufficient memory is found error message is displayed
                        if (data->columns[cols][rows].stringrows == NULL) {
                            printf("There wasn't sufficient memory to complete this
                                     operation");
                            exit(1);
                        }
        //Copying the contents of line2 to data->columns[cols][rows].stringrows
```

```
                        strcpy(data->columns[cols][rows].stringrows, line2);

                    }



                    //Incrementing cols
                    cols++;
                    //Updating the location of the next token
                    line2 = strtok(NULL, ",");
                }
                //Incrementing rows and resetting cols
                rows++;
                cols = 0;
            }
        }
        //Freeing tmp from memory
        free(tmp);
        //Closing the file
        fclose(fptr);

    }
}
```

Testing

Utilising the showDT method for testing purposes to show the data loaded from Testing2.csv into the DataTable_t:



```
        label1          label2          label3          label4          label5          label6          label7

1       1.50            5.50            10.50           10              "text, 1"       "text21"        "text31"

2       1.50            5.50            10.50           10              "text2\"        "text22$"       "text32"

3       1.50            5.50            10.50           10              "text3"         "text23"        "text33"

4       1.50            5.50            10.50           10              "text4"         "text24"        "text34"

5       1.50            5.50            10.50           10              "text5"         "text25"        "text35"

6       1.50            5.50            10.50           10              "text6"         "text26"        "text36"

7       1.50            5.50            10.50           10              "text7"         "text27"        "text37"

8       1.50            5.50            10.50           10              "text8"         "text28"        "text38"

9       1.50            5.50            10.50           10              "text9"         "text29"        "text39"

10      1.50            5.50            10.50           10              "text10"        "text210"       "text310"

.........................................................................................................................
1003    5.50            9.50            14.50           14              "text1000"      "text21000      "text31000
```

2b)exportDT()

Explanation

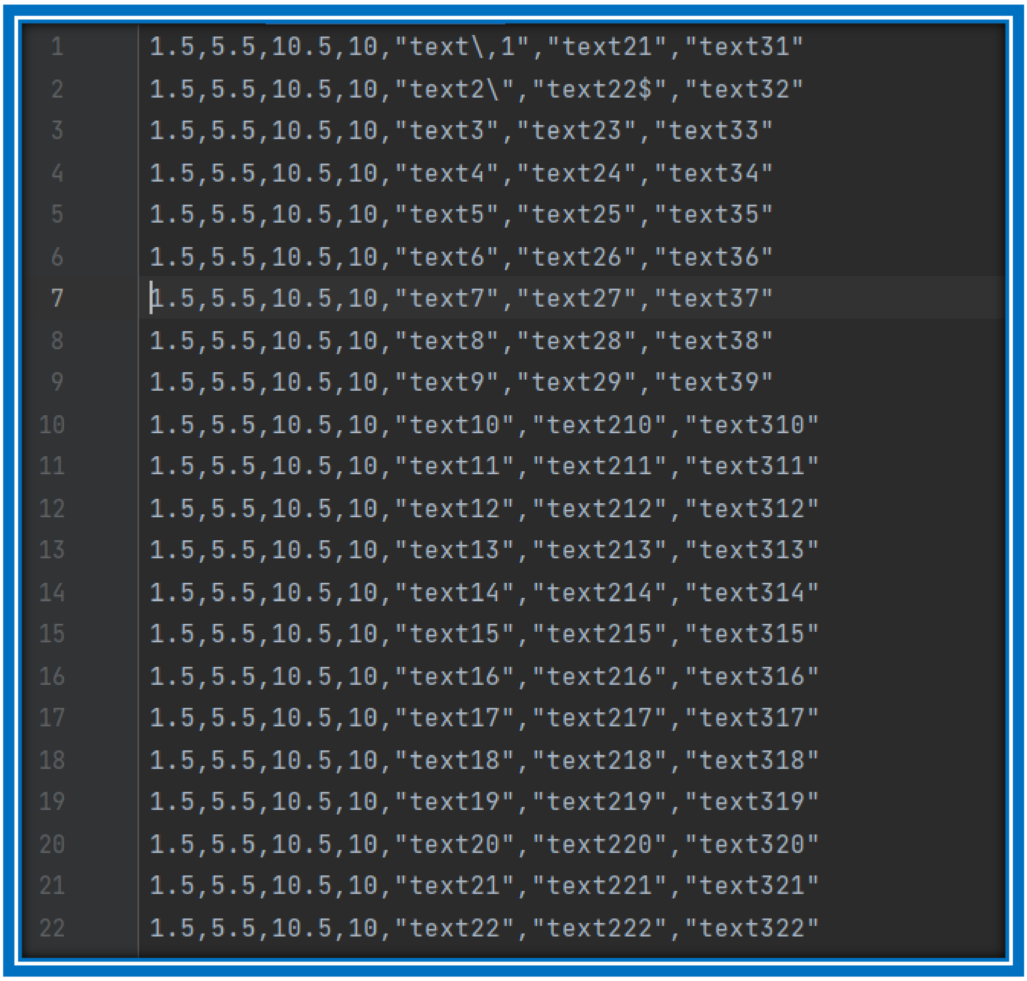
In this method contents of DataTable_t structure are exported into a csv file. This was executed by first opening the OutputFile.csv in writing mode. The method also required opening the Testing2.csv file in reading mode (if file is not found, error message is displayed) to find the maximum number of characters present in a row, to allocate sufficient memory to be able to write to the OutputFile. The method loops through all the cells in the DataTable_t structure and appends the cells to a string, separating each cell with a ‘,’. After the method loops through a whole row, it resorts to print the string onto the OutputFile.csv. If the last character of the string is not a \n, the method will print a \n to the file to signify end of row. The string cells which had a ’ ,’ followed by a space were parsed to store a \, in the csv file. The files were closed by the fclose() function.

The exportDT method in functionClass2b.c class:

```
//Declaration of method exportDT with DataTable_T pointer parameter
void exportDT(DataTable_t *data){
    //Declaration of file pointer fptr
    FILE *fptr;
    //Opening the file in writing mode
    fptr=fopen("OutputFile.csv","w");
    //Opening the file in reading mode
    FILE *fptr2;
    if((fptr2=fopen("Testing2.csv","r"))==NULL)
    {
        //If file is not found an Error message is presented
        printf("Error file not found to complete this action");
    }
    else {
        //To find the maximum number of characters in a line
        //Declaring variables linecount , max and filechar
        int linecount = 0, max = 0;
        char filechar;
        do {//Looping through the whole file
            //Getting a character from the file and storing it in filechar
            filechar = fgetc(fptr2);
            //If filechar =='\n' or EOF, enter if statement
            if (filechar == '\n' || filechar == EOF) {
                //Finding the max number of characters in a line
                if (max < linecount) {
                    max = linecount;
                }
                //Resetting linecount
                linecount = 0;
            } else {
                //Incrementing linecount
                linecount++;
            }
            //Looping until end of file is reached
        } while (filechar != EOF);


        //Looping through all the populated rows
        for (int i = 0; i < data->nrows; i++) {
  //Allocating sufficient memory for temp and resetting all values to '\0' via memset
            //Allocating more memory than required to be safe
            char *temp= (char *) malloc((max+10)*sizeof(char));
            //If no sufficient memory is found error message is displayed
            if (temp == NULL) {
                printf("There wasn't sufficient memory to complete this operation");
                exit(1);
            }
            memset(temp, '\0', (max+10)*sizeof(char));
            //Looping through the float columns
            for (int j = 0; j <data->ncols ; j++) {
                //If data->coltype[cols]==1 then column is float
                if(data->coltype[j]==1) {
                    //Declaring variable max digits
                    int maxdigits=0;
                    //Storing data->columns[j][i].floatrows in temp2
                    float temp2=data->columns[j][i].floatrows;
                    //Finding the number of digits in temp2
                    while(temp2>0.001){
                        temp2 = temp2 / 10;
                        maxdigits++;
                    }
                    //Allocation of sufficient memory for text
                    //Allocating more memory than required to be safe
                    char *text= (char *) malloc((maxdigits+10)*sizeof(char));
                    //If no sufficient memory is found error message is displayed
                    if (text == NULL) {
                        printf("There wasn't sufficient memory to complete this
                                     operation");
                        exit(1);
                    }
                    //Transforming the float into string using gcvt
                    gcvt( data->columns[j][i].floatrows, maxdigits, text);
                    //Appending the float to the string text
                    strcat(temp, text);
//If statement being used to append a , to the string temp to signify end of entity,
besides when final column is reached
                    if (j != data->ncols - 1) {
                        strcat(temp, ",");
                    }
                    //Freeing text
                    free(text);
                    text=NULL;
                }
                    //If data->coltype[cols]==2 then column is integer
                else if(data->coltype[j]==2){
                    //Declaring variable max digits
                    int maxdigits=0;
                    //Storing data->columns[j][i].introws in temp2
                    int temp2=data->columns[j][i].introws;
                    //Finding the number of digits in temp2
                    while(temp2>0){
                        temp2 = temp2 / 10;
                        maxdigits++;
                    }
                    //Allocation of sufficient memory for text
                    //Allocating more memory than required to be safe
                    char *text= (char *) malloc((maxdigits+10)*sizeof(char));
                    //If no sufficient memory is found error message is displayed
                    if (text == NULL) {
                        printf("There wasn't sufficient memory to complete this
                                    operation");
                        exit(1);
                    }
                    //Transforming the float into string using itoa
                    itoa( data->columns[j][i].introws,  text,10);//10 be the base
                                                                 decimal
                    //Appending the float to the string text
                    strcat(temp, text);
//If statement being used to append a , to the string temp to signify end of entity,
besides when final column is reached
                    if (j != data->ncols - 1) {
                        strcat(temp, ",");
                    }
                    //Freeing text
                    free(text);
                    text=NULL;
                }
                    //If data->coltype[cols]==4 then column is string
                else if(data->coltype[j]==4){//String
//Allocating sufficient memory for words and setting it to hold data-
>columns[j][i].stringrows
                    //Allocating more memory than required to be safe
                    char *words= (char *) malloc(sizeof(data-
                             >columns[j][i].stringrows)+10);
                    //If no sufficient memory is found error message is displayed
                    if (words == NULL) {
                        printf("There wasn't sufficient memory to complete this
                               operation");
                        exit(1);
                    }
                    strcpy(words,data->columns[j][i].stringrows);
                    //Declaring variable to hold the position of string tmp
                    int pos = 0;
                    //While loop used to check if end of string is reached
                    while (words[pos] != '\0') {
                        //If character is , then we transpose it to \,
                        if (words[pos] == ',' && words[pos +1] == ' ') {
                            words[pos] = '\\';
                            words[pos + 1] = ',';
                            pos++;
                        }
                        //Incrementing position
                        pos++;
                    }
                    //Appending the words to the string temp
                    strcat(temp, words);
//If statement being used to append a , to the string temp to signify end of entity,
besides when final column is reached
                    if (j != data->ncols - 1) {
                        strcat(temp, ",");
                    }
                    //Freeing words
                    free(words);
                    words=NULL;

                }

            }

            //Printing to file the string temp
            fprintf(fptr, "%s", temp);
            //Printing \n to file to indicate a new row in case there wasn't
            if(temp[strlen(temp)-1]!='\n'){
                //Printing to file the string temp
                fprintf(fptr, "%s", "\n");
            }
            //Freeing temp
            free(temp);
            temp=NULL;
;
        }
    }
    //Closing the files
    fclose(fptr);
    fclose(fptr2);
}
```

Testing

Output in OutputFile.csv:



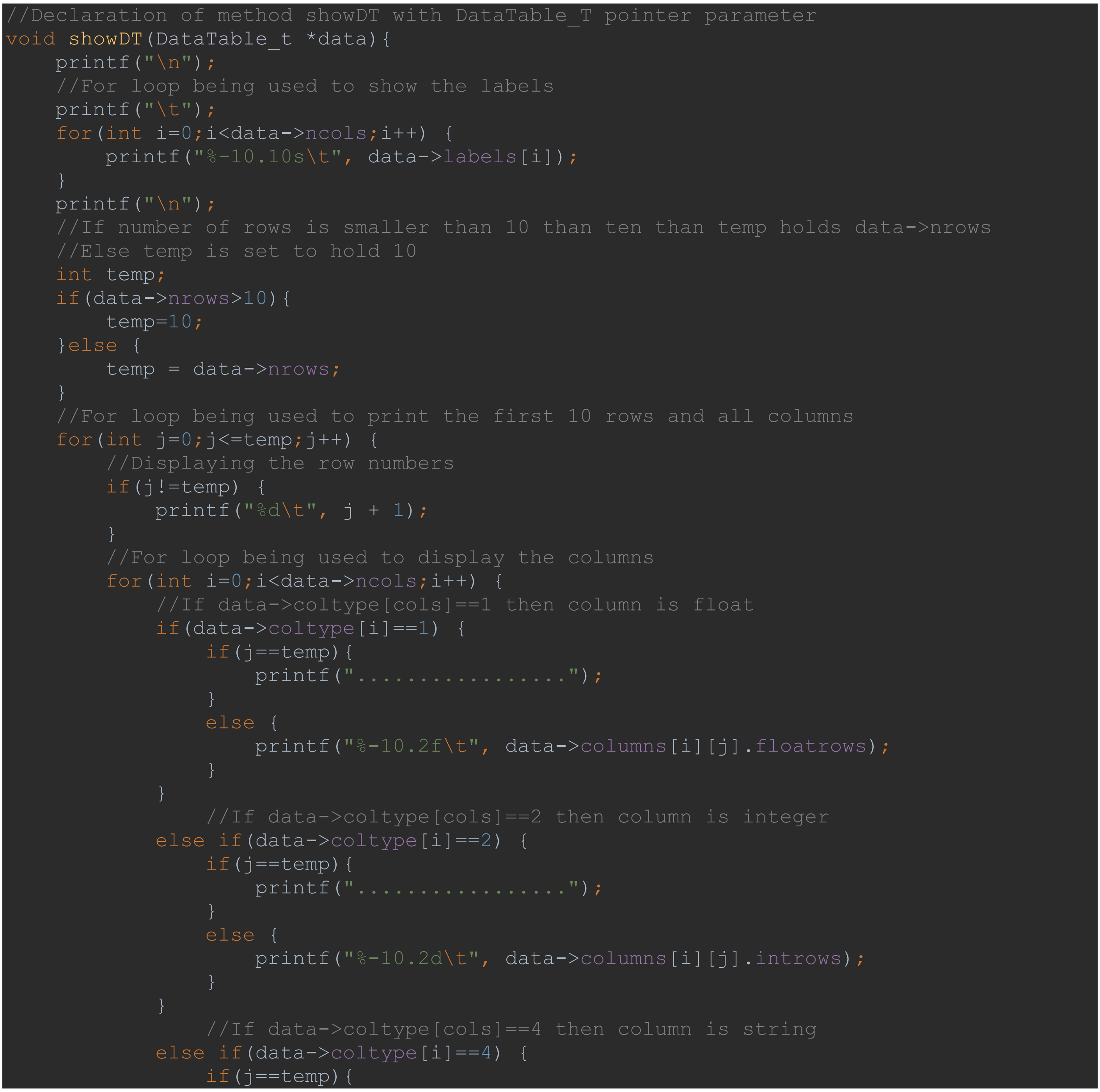
```
1.5,5.5,10.5,10,"text\,1","text21","text31"
1.5,5.5,10.5,10,"text2\","text22$","text32"
1.5,5.5,10.5,10,"text3","text23","text33"
1.5,5.5,10.5,10,"text4","text24","text34"
1.5,5.5,10.5,10,"text5","text25","text35"
1.5,5.5,10.5,10,"text6","text26","text36"
1.5,5.5,10.5,10,"text7","text27","text37"
1.5,5.5,10.5,10,"text8","text28","text38"
1.5,5.5,10.5,10,"text9","text29","text39"
1.5,5.5,10.5,10,"text10","text210","text310"
1.5,5.5,10.5,10,"text11","text211","text311"
1.5,5.5,10.5,10,"text12","text212","text312"
1.5,5.5,10.5,10,"text13","text213","text313"
1.5,5.5,10.5,10,"text14","text214","text314"
1.5,5.5,10.5,10,"text15","text215","text315"
1.5,5.5,10.5,10,"text16","text216","text316"
1.5,5.5,10.5,10,"text17","text217","text317"
1.5,5.5,10.5,10,"text18","text218","text318"
1.5,5.5,10.5,10,"text19","text219","text319"
1.5,5.5,10.5,10,"text20","text220","text320"
1.5,5.5,10.5,10,"text21","text221","text321"
1.5,5.5,10.5,10,"text22","text222","text322"
```

2b)showDT()

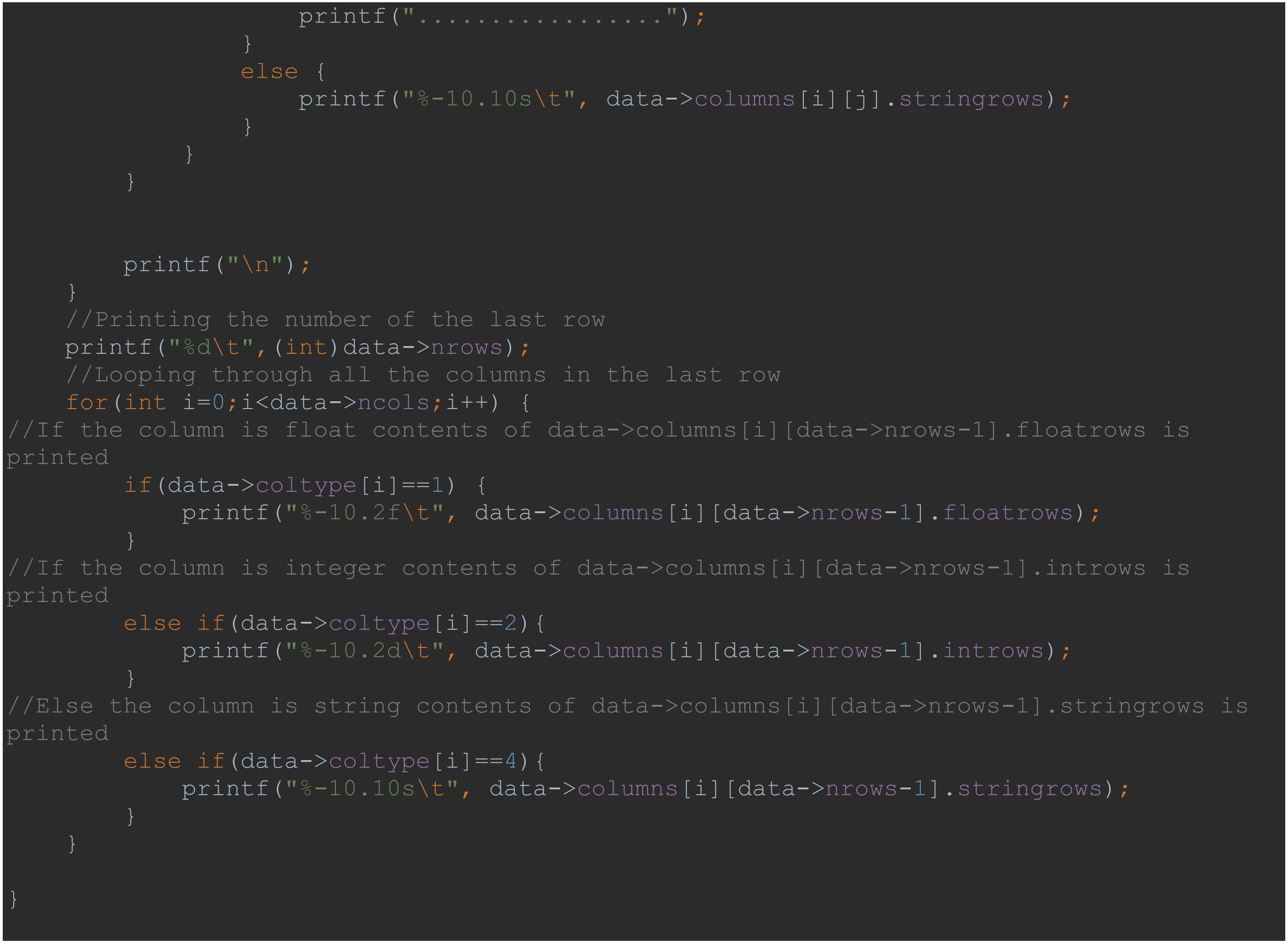
Explanation

In this method contents of DataTable_t structure are displayed, showing the labels followed by the first 10 rows, and the last row. This was achieved by utilising a nested for loop to loop through all the columns and the first 10 rows, printing the relevant cells respectively. If there are less than 10 rows, all the rows are printed. After the first 10 rows are printed a line of ' …' is printed which is followed by printing all the columns in the last row. Row numbers are also clearly displayed at the beginning of each row. Depending on the respective column type, the relevant datatype is printed. If each cell has more than 10 characters, only the first 10 characters are displayed.

The showDT method in functionClass2b.c class:



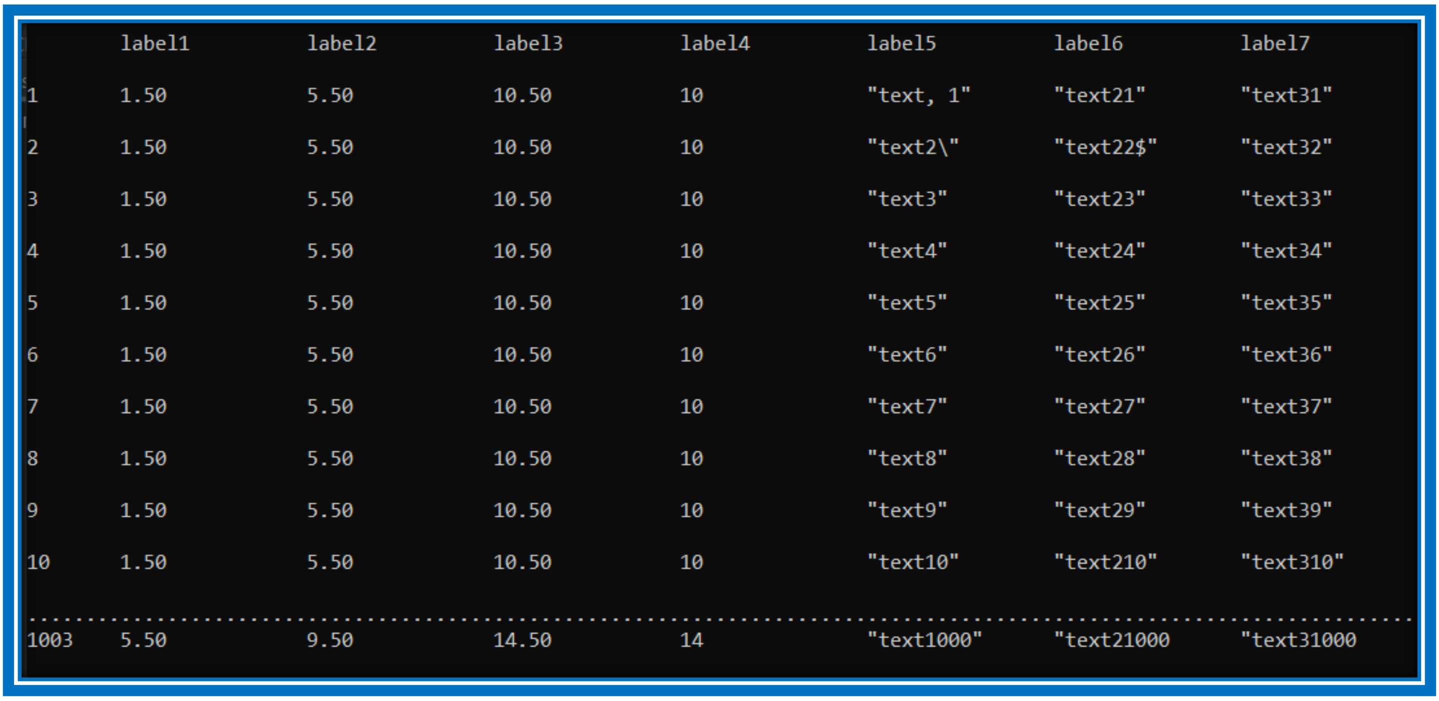
```c
//Declaration of method showDT with DataTable_T pointer parameter
void showDT(DataTable_t *data){
    printf("\n");
    //For loop being used to show the labels
    printf("\t");
    for(int i=0;i<data->ncols;i++) {
        printf("%-10.10s\t", data->labels[i]);
    }
    printf("\n");
    //If number of rows is smaller than 10 than ten than temp holds data->nrows
    //Else temp is set to hold 10
    int temp;
    if(data->nrows>10){
        temp=10;
    }else {
        temp = data->nrows;
    }
    //For loop being used to print the first 10 rows and all columns
    for(int j=0;j<=temp;j++) {
        //Displaying the row numbers
        if(j!=temp) {
            printf("%d\t", j + 1);
        }
        //For loop being used to display the columns
        for(int i=0;i<data->ncols;i++) {
            //If data->coltype[cols]==1 then column is float
            if(data->coltype[i]==1) {
                if(j==temp){
                    printf(".................");
                }
                else {
                    printf("%-10.2f\t", data->columns[i][j].floatrows);
                }
            }
                //If data->coltype[cols]==2 then column is integer
            else if(data->coltype[i]==2) {
                if(j==temp){
                    printf(".................");
                }
                else {
                    printf("%-10.2d\t", data->columns[i][j].introws);
                }
            }
                //If data->coltype[cols]==4 then column is string
            else if(data->coltype[i]==4) {
                if(j==temp){
```



```
                    printf("................");
                }
                else {
                    printf("%-10.10s\t", data->columns[i][j].stringrows);
                }
            }
        }


        printf("\n");
    }
    //Printing the number of the last row
    printf("%d\t",(int)data->nrows);
    //Looping through all the columns in the last row
    for(int i=0;i<data->ncols;i++) {
//If the column is float contents of data->columns[i][data->nrows-1].floatrows is
printed
        if(data->coltype[i]==1) {
            printf("%-10.2f\t", data->columns[i][data->nrows-1].floatrows);
        }
//If the column is integer contents of data->columns[i][data->nrows-1].introws is
printed
        else if(data->coltype[i]==2){
            printf("%-10.2d\t", data->columns[i][data->nrows-1].introws);
        }
//Else the column is string contents of data->columns[i][data->nrows-1].stringrows is
printed
        else if(data->coltype[i]==4){
            printf("%-10.10s\t", data->columns[i][data->nrows-1].stringrows);
        }
    }

}
```

Testing

Utilising the showDT method to print the DataTable_t structure:



```
        label1      label2      label3      label4      label5      label6      label7

1       1.50        5.50        10.50       10          "text, 1"   "text21"    "text31"

2       1.50        5.50        10.50       10          "text2\"    "text22$"   "text32"

3       1.50        5.50        10.50       10          "text3"     "text23"    "text33"

4       1.50        5.50        10.50       10          "text4"     "text24"    "text34"

5       1.50        5.50        10.50       10          "text5"     "text25"    "text35"

6       1.50        5.50        10.50       10          "text6"     "text26"    "text36"

7       1.50        5.50        10.50       10          "text7"     "text27"    "text37"

8       1.50        5.50        10.50       10          "text8"     "text28"    "text38"

9       1.50        5.50        10.50       10          "text9"     "text29"    "text39"

10      1.50        5.50        10.50       10          "text10"    "text210"   "text310"

..........................................................................................
1003    5.50        9.50        14.50       14          "text1000"  "text21000  "text31000
```

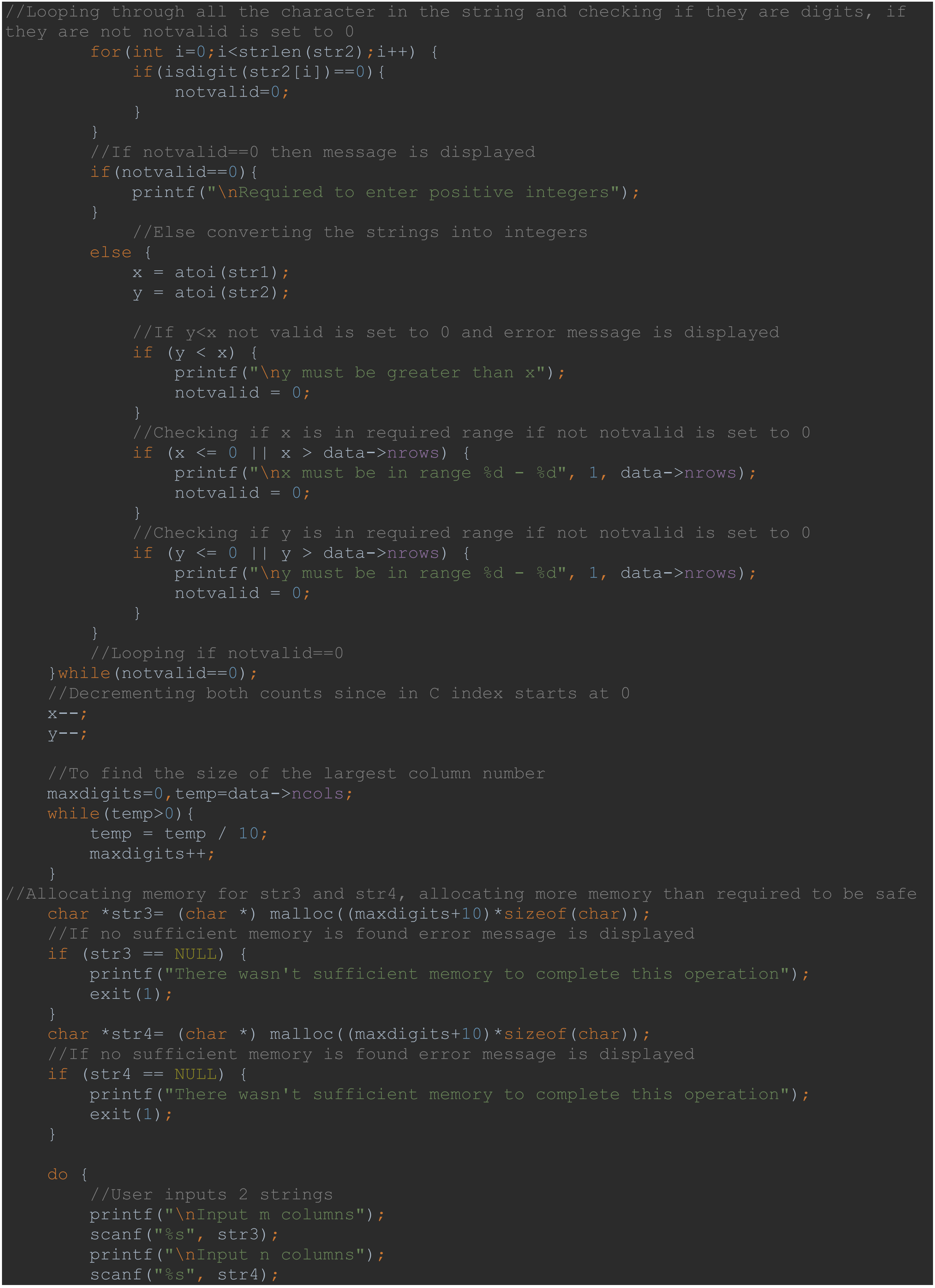
2b)projectDT()

Explanation

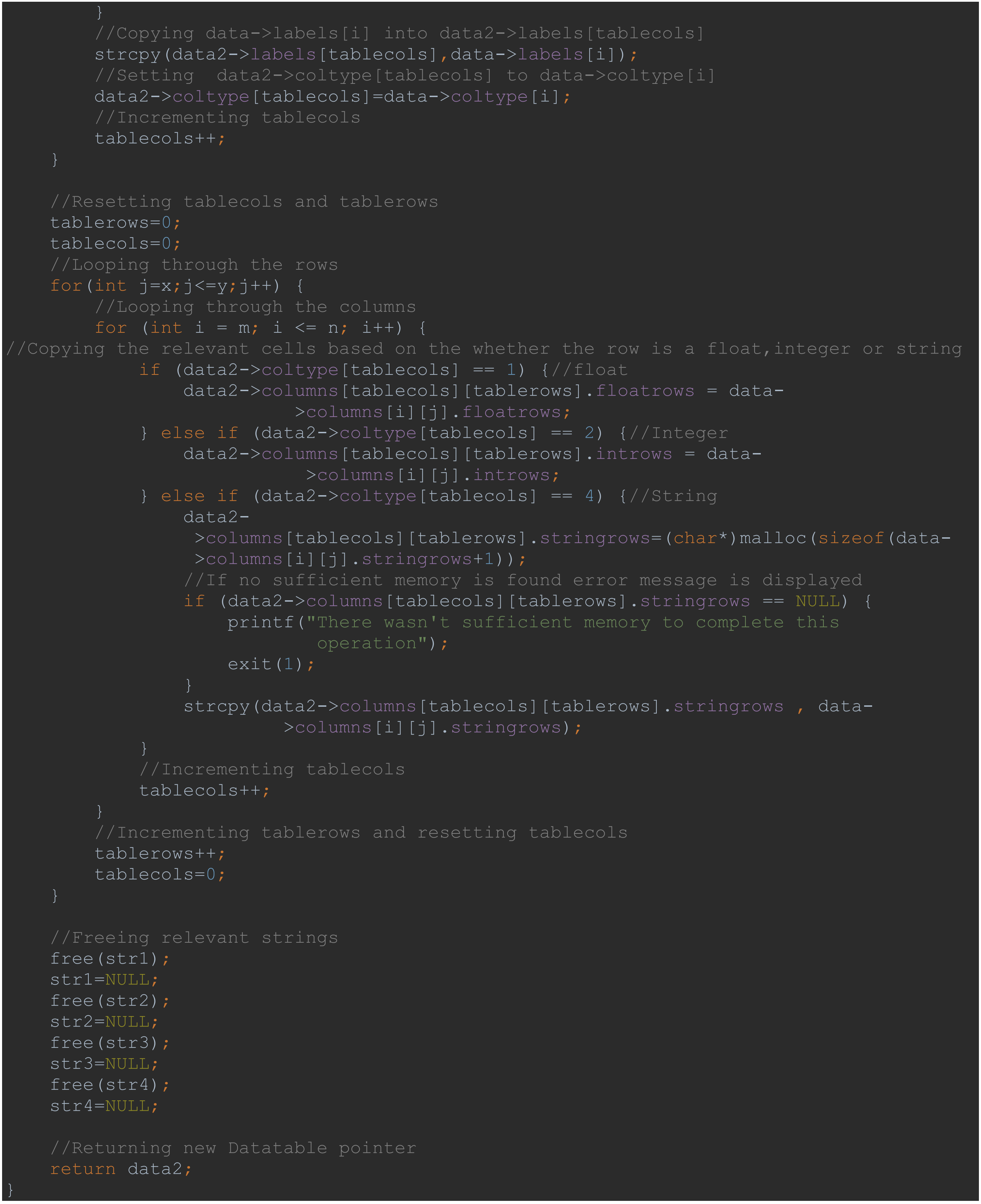
This method returns a pointer to a new DataTable_t structure but containing the rows from x to y and columns from m to n. This method requires the user to input x, y, m and n and proper validation is ensured for all inputs, in case the user input; is not in the required range, is not of an integer data type or if the second input is smaller than the first one. The method then declares and initialises a new DataTable_t, reusing the initDT function but passing 1 as a parameter as well as number of columns and rows, as a new DataTable is created with specific row and column specifications. The relevant cells were copied from the initial DataTable to the new one.

The projectDT method in functionClass2b.c class:

```
//Declaration of method projectDT with DataTable_T pointer parameter
DataTable_t *projectDT(DataTable_t *data){

    //To find the size of the largest row number
    int maxdigits=0,temp=data->nrows;
    while(temp>0){
        temp = temp / 10;
        maxdigits++;
    }
//Allocating sufficient memory for str1 and str2, allocating more memory than required
to be safe
    char *str1= (char *) malloc((maxdigits+7)* sizeof(char));
    //If no sufficient memory is found error message is displayed
    if (str1 == NULL) {
        printf("There wasn't sufficient memory to complete this operation");
        exit(1);
    }
    char *str2= (char *) malloc((maxdigits+7)* sizeof(char));
    //If no sufficient memory is found error message is displayed
    if (str2 == NULL) {
        printf("There wasn't sufficient memory to complete this operation");
        exit(1);
    }

    //Declaring integer parameters
    int x,y,m,n;
    //Declaring variables notvalid to be used in loop
    int notvalid;
    printf("\n");

    //Performing validation on inputs
    do {
        //User inputs 2 strings
        printf("\nInput x rows");
        scanf("%s", str1);
        printf("\nInput y rows");
        scanf("%s", str2);
        notvalid = 1;//Setting notvalid to 1
//Looping through all the character in the string and checking if they are digits, if
they are not notvalid is set to 0
        for(int i=0;i<strlen(str1);i++) {
            if(isdigit(str1[i])==0){
                notvalid=0;
            }
        }
```



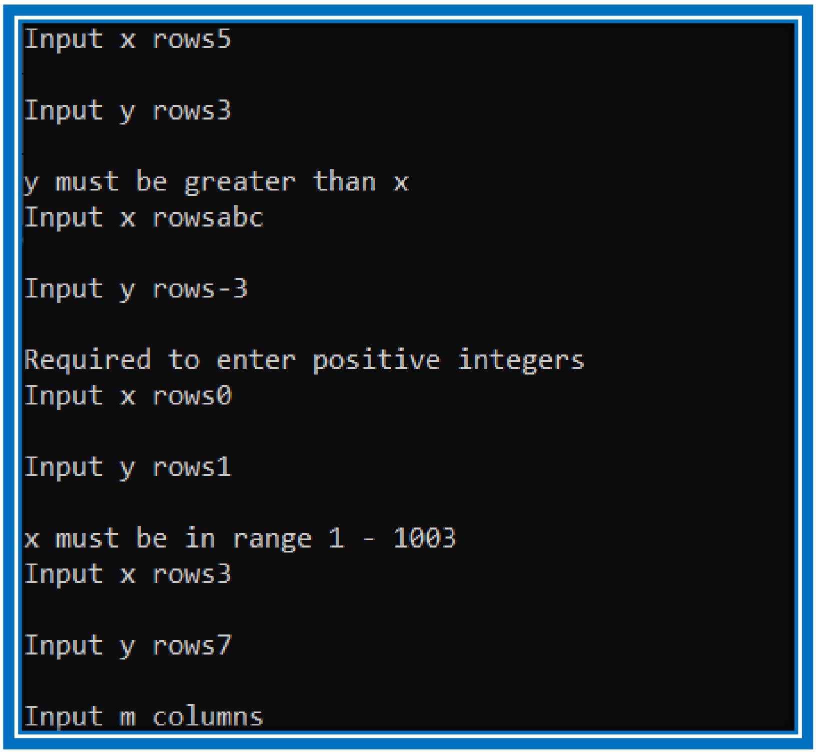
```
//Looping through all the character in the string and checking if they are digits, if
they are not notvalid is set to 0
        for(int i=0;i<strlen(str2);i++) {
            if(isdigit(str2[i])==0){
                notvalid=0;
            }
        }
        //If notvalid==0 then message is displayed
        if(notvalid==0){
            printf("\nRequired to enter positive integers");
        }
            //Else converting the strings into integers
        else {
            x = atoi(str1);
            y = atoi(str2);

            //If y<x not valid is set to 0 and error message is displayed
            if (y < x) {
                printf("\ny must be greater than x");
                notvalid = 0;
            }
            //Checking if x is in required range if not notvalid is set to 0
            if (x <= 0 || x > data->nrows) {
                printf("\nx must be in range %d - %d", 1, data->nrows);
                notvalid = 0;
            }
            //Checking if y is in required range if not notvalid is set to 0
            if (y <= 0 || y > data->nrows) {
                printf("\ny must be in range %d - %d", 1, data->nrows);
                notvalid = 0;
            }
        }
        //Looping if notvalid==0
    }while(notvalid==0);
    //Decrementing both counts since in C index starts at 0
    x--;
    y--;

    //To find the size of the largest column number
    maxdigits=0,temp=data->ncols;
    while(temp>0){
        temp = temp / 10;
        maxdigits++;
    }
//Allocating memory for str3 and str4, allocating more memory than required to be safe
    char *str3= (char *) malloc((maxdigits+10)*sizeof(char));
    //If no sufficient memory is found error message is displayed
    if (str3 == NULL) {
        printf("There wasn't sufficient memory to complete this operation");
        exit(1);
    }
    char *str4= (char *) malloc((maxdigits+10)*sizeof(char));
    //If no sufficient memory is found error message is displayed
    if (str4 == NULL) {
        printf("There wasn't sufficient memory to complete this operation");
        exit(1);
    }

    do {
        //User inputs 2 strings
        printf("\nInput m columns");
        scanf("%s", str3);
        printf("\nInput n columns");
        scanf("%s", str4);
        notvalid=1;//Setting notvalid to 1
//Looping through all the character in the string and checking if they are digits, if 
they are not notvalid is set to 0
        for(int i=0;i<strlen(str3);i++) {
            if(isdigit(str3[i])==0){
                notvalid=0;
            }
        }
//Looping through all the character in the string and checking if they are digits, if 
they are not notvalid is set to 0
        for(int i=0;i<strlen(str4);i++) {
            if(isdigit(str4[i])==0){
                notvalid=0;
            }
        }
        //If notvalid==0 then message is displayed
        if(notvalid==0){
            printf("\nRequired to enter positive integers");
        }
            //Else converting the strings into integers
        else {
            m = atoi(str3);
            n = atoi(str4);
            //If n<m not valid is set to 0 and error message is displayed
            if (n < m) {
                printf("\nn must be greater than m");
                notvalid=0;
            }
            //Checking if n is in required range if not notvalid is set to 0
            if (n <= 0 || n > data->ncols) {
                printf("\nn must be in range %d - %d", 1, data->ncols);
                notvalid = 0;
            }
            //Checking if m is in required range if not notvalid is set to 0
            if (m <= 0 || m > data->ncols) {
                printf("\nm must be in range %d - %d", 1, data->ncols);
                notvalid = 0;
            }
        }
        //Looping if notvalid==0
    }while(notvalid==0);
    //Decrementing both counts since in C index starts at 0
    m--;
    n--;


    //Calculating the amount of rows and columns the new Datatable will have:
    int tablerows=(y-x)+1;
    int tablecols=(n-m)+1;
//Calling initDT method and passing choice as 1 since a new DataTable is created based 
on DataTable data
    //and respective tablecols and tablerows as parameters
    DataTable_t *data2= initDT(1,tablecols,tablerows);
    //Resetting tablecols to 0
    tablecols=0;

    //Looping through respective columns in initial DataTable
    for(int i=m;i<=n;i++) {
        //Allocating memory to label[i]
        data2->labels[tablecols] = (char *) malloc(sizeof(data->labels[i])+1);
        //If no sufficient memory is found error message is displayed
        if (data->labels[tablecols] == NULL) {
            printf("There wasn't sufficient memory to complete this operation");
            exit(1);
```



```
        }
        //Copying data->labels[i] into data2->labels[tablecols]
        strcpy(data2->labels[tablecols],data->labels[i]);
        //Setting  data2->coltype[tablecols] to data->coltype[i]
        data2->coltype[tablecols]=data->coltype[i];
        //Incrementing tablecols
        tablecols++;
    }

    //Resetting tablecols and tablerows
    tablerows=0;
    tablecols=0;
    //Looping through the rows
    for(int j=x;j<=y;j++) {
        //Looping through the columns
        for (int i = m; i <= n; i++) {
//Copying the relevant cells based on the whether the row is a float,integer or string
            if (data2->coltype[tablecols] == 1) {//float
                data2->columns[tablecols][tablerows].floatrows = data-
                          >columns[i][j].floatrows;
            } else if (data2->coltype[tablecols] == 2) {//Integer
                data2->columns[tablecols][tablerows].introws = data-
                           >columns[i][j].introws;
            } else if (data2->coltype[tablecols] == 4) {//String
                data2-
                 >columns[tablecols][tablerows].stringrows=(char*)malloc(sizeof(data-
                 >columns[i][j].stringrows+1));
                //If no sufficient memory is found error message is displayed
                if (data2->columns[tablecols][tablerows].stringrows == NULL) {
                    printf("There wasn't sufficient memory to complete this
                            operation");
                    exit(1);
                }
                strcpy(data2->columns[tablecols][tablerows].stringrows , data-
                         >columns[i][j].stringrows);
            }
            //Incrementing tablecols
            tablecols++;
        }
        //Incrementing tablerows and resetting tablecols
        tablerows++;
        tablecols=0;
    }

    //Freeing relevant strings
    free(str1);
    str1=NULL;
    free(str2);
    str2=NULL;
    free(str3);
    str3=NULL;
    free(str4);
    str4=NULL;

    //Returning new Datatable pointer
    return data2;
}
```

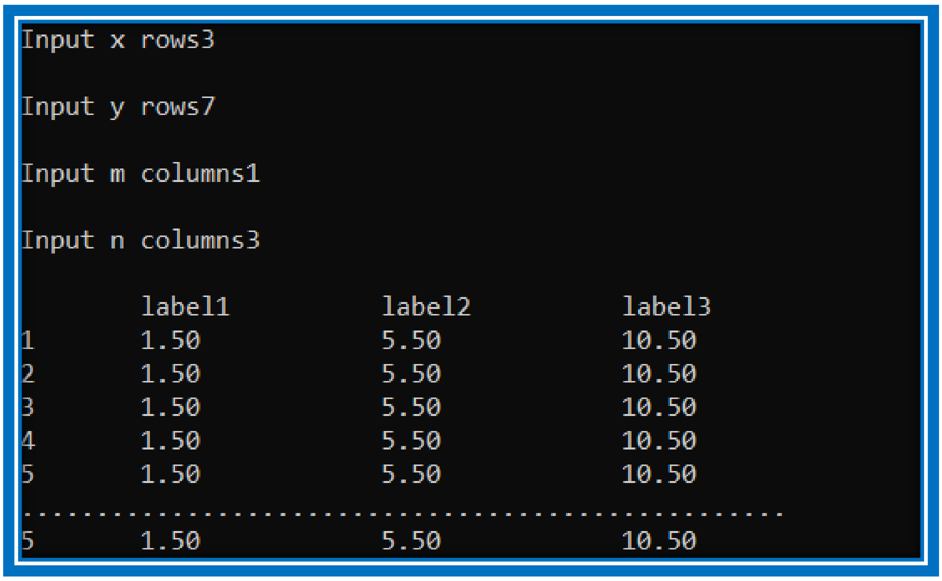
Testing User input (Similar validation for m & n):

Input	Type of data	Output
x=3,y=7	Valid data	Valid data
x=5,y=3	Invalid data X must be smaller than y	Invalid data X must be smaller than y
X=abc,y=-3	Invalid data Required to enter positive integers	Invalid data Required to enter positive integers
X=0,y=1	Invalid data Not in range	Invalid data Not in range



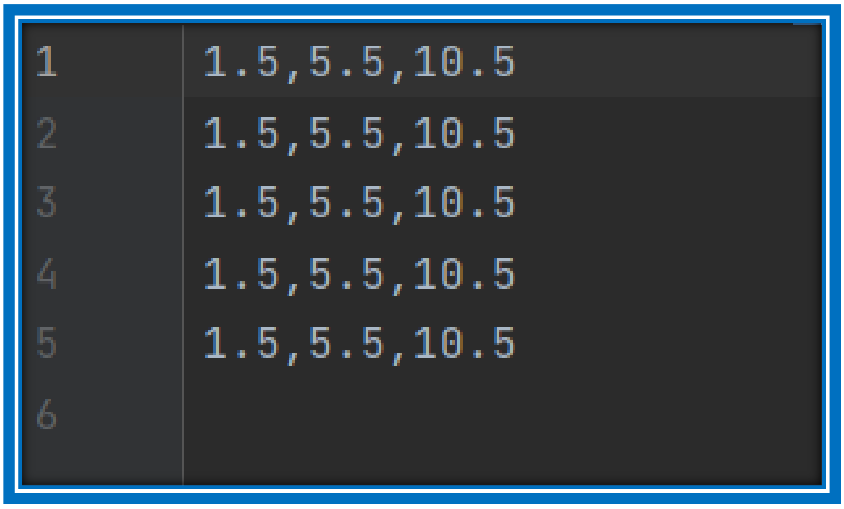
```
Input x rows5

Input y rows3

y must be greater than x
Input x rowsabc

Input y rows-3

Required to enter positive integers
Input x rows0

Input y rows1

x must be in range 1 - 1003
Input x rows3

Input y rows7

Input m columns
```

Utilising the showDT method for testing purposes to show the data in the new DataTable_t structure:



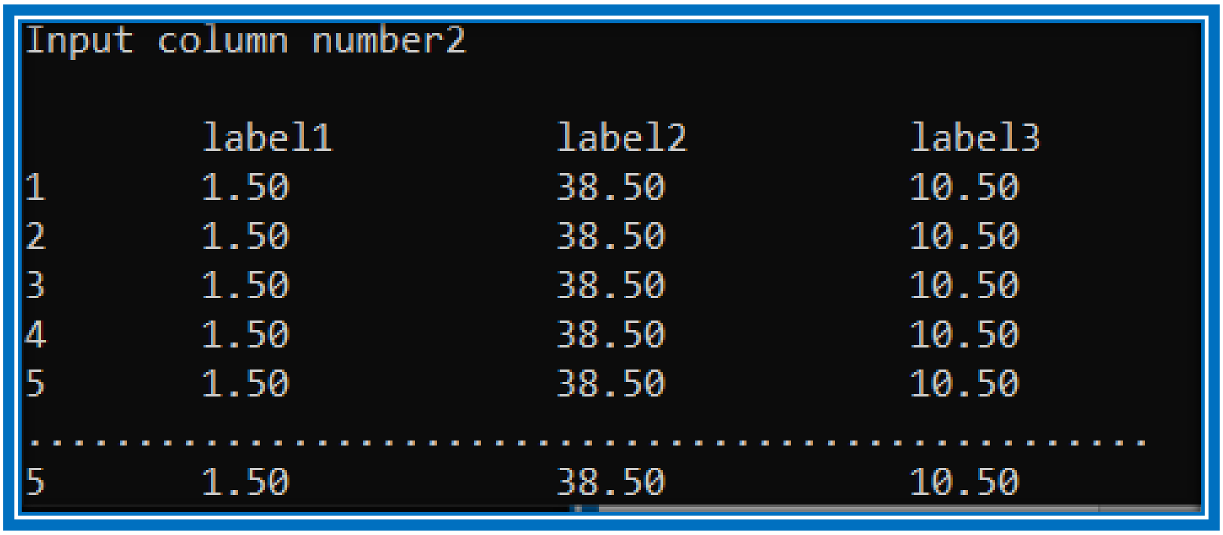
```
Input x rows3

Input y rows7

Input m columns1

Input n columns3

        label1          label2          label3
1       1.50            5.50            10.50
2       1.50            5.50            10.50
3       1.50            5.50            10.50
4       1.50            5.50            10.50
5       1.50            5.50            10.50
...................................................
5       1.50            5.50            10.50
```

Utilising the exportDT method for testing purposes to show utilisation of the new DataTable_t structure:



```
1.5,5.5,10.5
1.5,5.5,10.5
1.5,5.5,10.5
1.5,5.5,10.5
1.5,5.5,10.5

```

Utilising the mutateDT method for testing purposes and mutating column 2 to show utilisation of the new DataTable_t structure:



```
Input column number2

        label1          label2          label3
1       1.50            38.50           10.50
2       1.50            38.50           10.50
3       1.50            38.50           10.50
4       1.50            38.50           10.50
5       1.50            38.50           10.50
..................................................
5       1.50            38.50           10.50
```

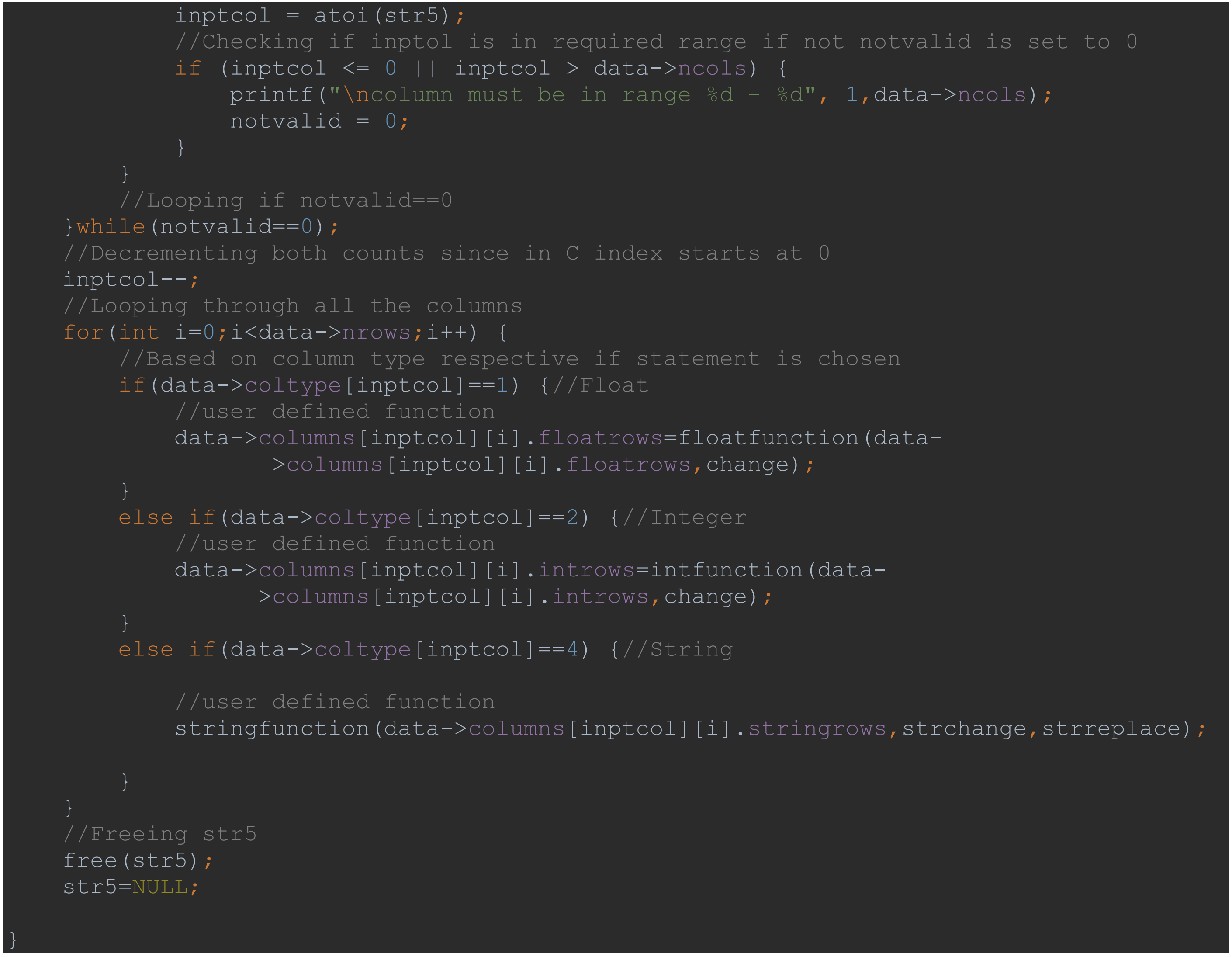
2b)mutateDT()

Explanation

This method accepts a user-defined function and changes the respective column chosen by the user. If the column is a float or integer, it will multiply all the rows in said column by 7. If the column is a string it will replace any character which is a ‘t’ into an ‘a’. Proper validation is ensured on the user input column, in case the user input; is not in the required range, is not of an integer data type.

The mutateDT method in functionClass2b.c class:

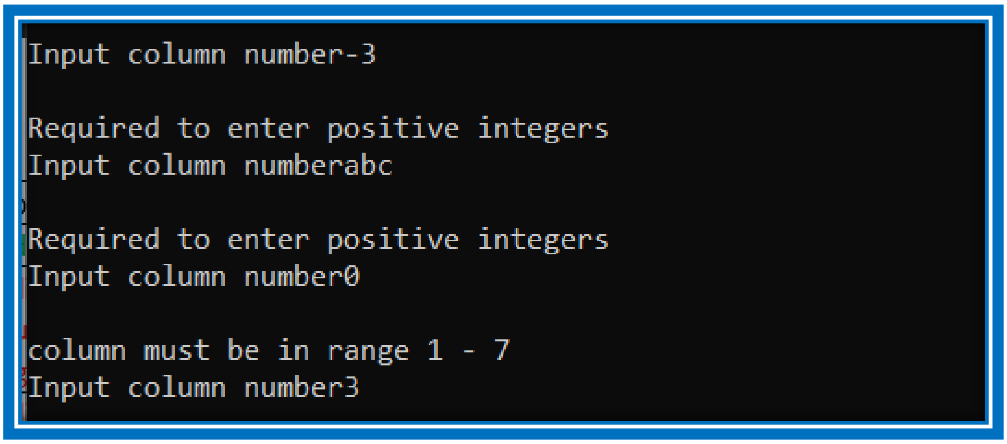
```
//Declaration of method mutateDT with DataTable_T pointer and user defined functions as
parameters
void mutateDT(DataTable_t *data,float (*floatfunction)(float num,int change),int
(*intfunction)(int num,int change),void (*stringfunction)(char* string,char
strchange,char strreplace)){
    //Declaration of variables inptcol,notvalid
    int inptcol,notvalid;
//Declaration of integer variable which will be used to multiply float rows and int
rows by 7
    int change=7;
    //To find the size of the largest row number
    int maxdigits=0,temp=data->ncols;
    while(temp>0){
        temp = temp / 10;
        maxdigits++;
    }
//Declaring and allocating memory for str5, allocating more memory than required to be
safe
    char *str5= (char *) malloc((maxdigits+10)*sizeof(char));
    //If no sufficient memory is found error message is displayed
    if (str5 == NULL) {
        printf("There wasn't sufficient memory to complete this operation");
        exit(1);
    }
//Declaration of character variables which will be used to transpose the character t
into a in string columns
    char strchange='t',strreplace='a';

    //Displaying options
    printf("\nChoose a column if a string column is chosen 't' will be replaced by 'a'
and \nif an integer or float column is chosen the column will be multiplied by 7\n");
    //User validation for input
    do {
        //User input String
        printf("\nInput column number");
        scanf("%s", str5);
        notvalid=1;
//Looping through all the character in the string and checking if they are digits, if
they are not notvalid is set to 0
        for(int i=0;i<strlen(str5);i++) {
            if(isdigit(str5[i])==0){
                notvalid=0;
            }
        }
        //If notvalid==0 then message is displayed
        if(notvalid==0){
            printf("\nRequired to enter positive integers");
        }
            //Else converting the strings into integers
        else {
```



```
            inptcol = atoi(str5);
            //Checking if inptol is in required range if not notvalid is set to 0
            if (inptcol <= 0 || inptcol > data->ncols) {
                printf("\ncolumn must be in range %d - %d", 1,data->ncols);
                notvalid = 0;
            }
        }
        //Looping if notvalid==0
    }while(notvalid==0);
    //Decrementing both counts since in C index starts at 0
    inptcol--;
    //Looping through all the columns
    for(int i=0;i<data->nrows;i++) {
        //Based on column type respective if statement is chosen
        if(data->coltype[inptcol]==1) {//Float
            //user defined function
            data->columns[inptcol][i].floatrows=floatfunction(data-
                   >columns[inptcol][i].floatrows,change);
        }
        else if(data->coltype[inptcol]==2) {//Integer
            //user defined function
            data->columns[inptcol][i].introws=intfunction(data-
                  >columns[inptcol][i].introws,change);
        }
        else if(data->coltype[inptcol]==4) {//String

            //user defined function
            stringfunction(data->columns[inptcol][i].stringrows,strchange,strreplace);

        }
    }
    //Freeing str5
    free(str5);
    str5=NULL;

}
```

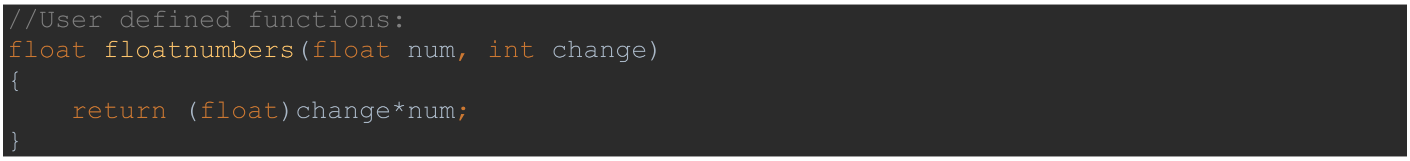
Testing User input:

Input	Type of data	Output
Inptcol=3	Valid data	Valid data
Inptcol=-3,inptcol=abc	Invalid data Required to enter positive integers	Invalid data Required to enter positive integers
inptcol=0	Invalid data Not in range	Invalid data Not in range



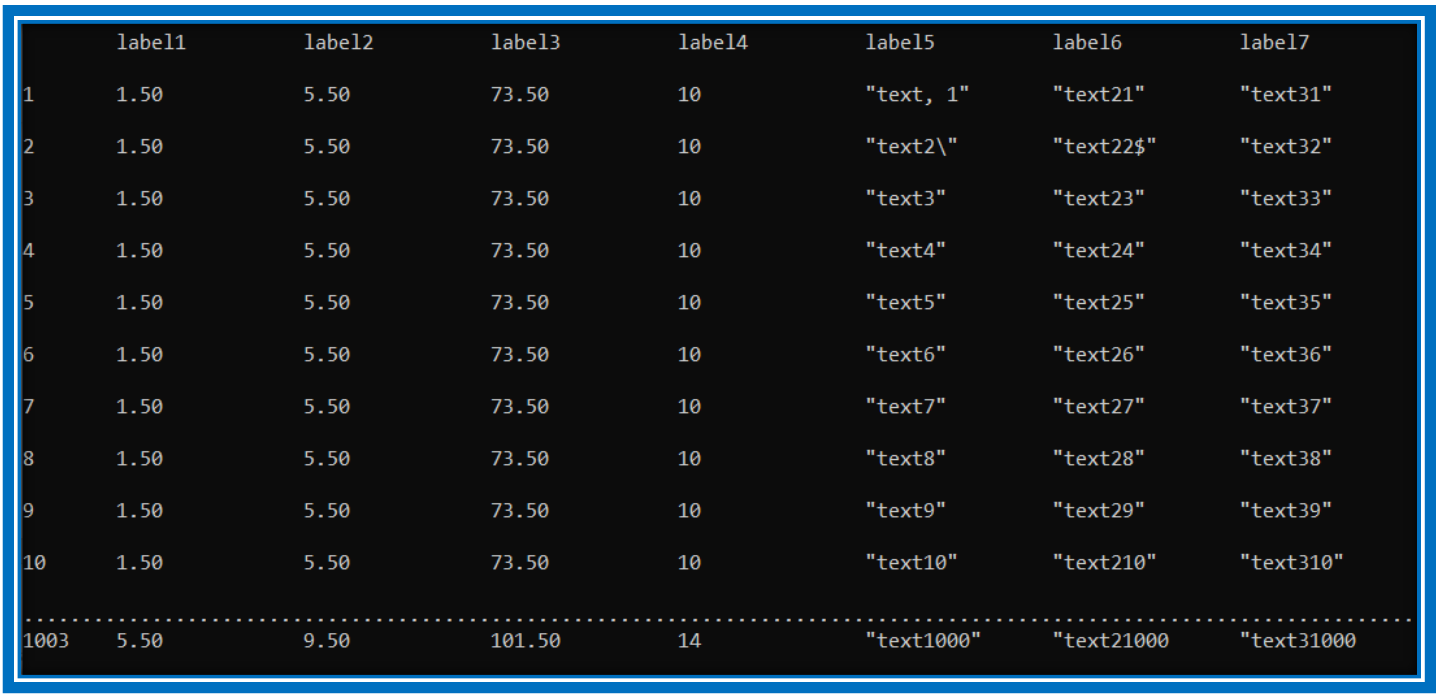
```
Input column number-3

Required to enter positive integers
Input column numberabc

Required to enter positive integers
Input column number0

column must be in range 1 - 7
Input column number3
```

The following is the user defined function to multiply the float cell with change(value of change is always 7):



```
//User defined functions:
float floatnumbers(float num, int change)
{
    return (float)change*num;
}
```

Utilising the showDT method for testing purposes to show the data change in the DataTable_t structure (performing mutation on column 3):

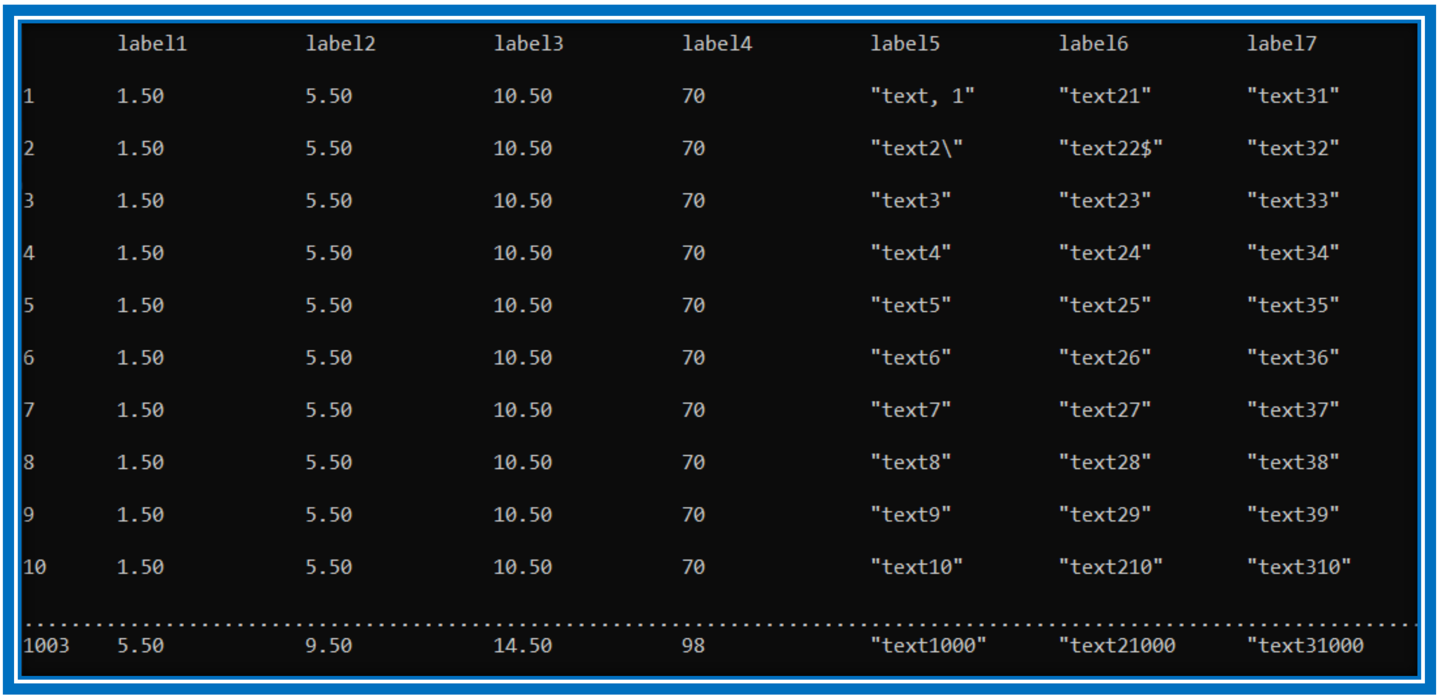


```
        label1          label2          label3          label4          label5          label6          label7

1       1.50            5.50            73.50           10              "text, 1"       "text21"        "text31"

2       1.50            5.50            73.50           10              "text2\"        "text22$"       "text32"

3       1.50            5.50            73.50           10              "text3"         "text23"        "text33"

4       1.50            5.50            73.50           10              "text4"         "text24"        "text34"

5       1.50            5.50            73.50           10              "text5"         "text25"        "text35"

6       1.50            5.50            73.50           10              "text6"         "text26"        "text36"

7       1.50            5.50            73.50           10              "text7"         "text27"        "text37"

8       1.50            5.50            73.50           10              "text8"         "text28"        "text38"

9       1.50            5.50            73.50           10              "text9"         "text29"        "text39"

10      1.50            5.50            73.50           10              "text10"        "text210"       "text310"

........................................................................................................................
1003    5.50            9.50            101.50          14              "text1000"      "text21000      "text31000
```

The following is the user defined function to multiply the integer cell with change(value of change is always 7):

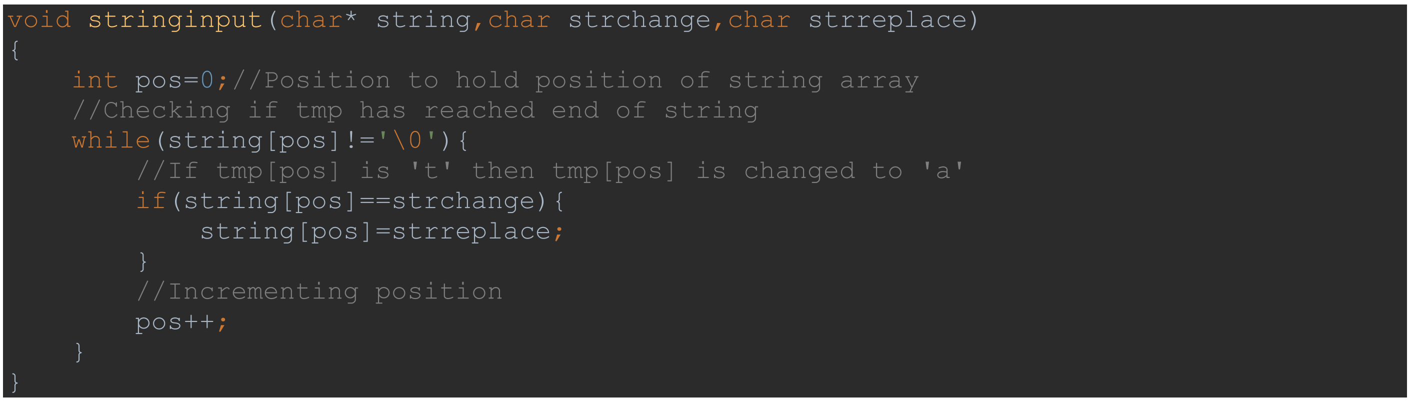
```
int intnumbers(int num, int change)
{
    return change*num;
}
```

Utilising the showDT method for testing purposes to show the data change in the DataTable_t structure (performing mutation on column 4):



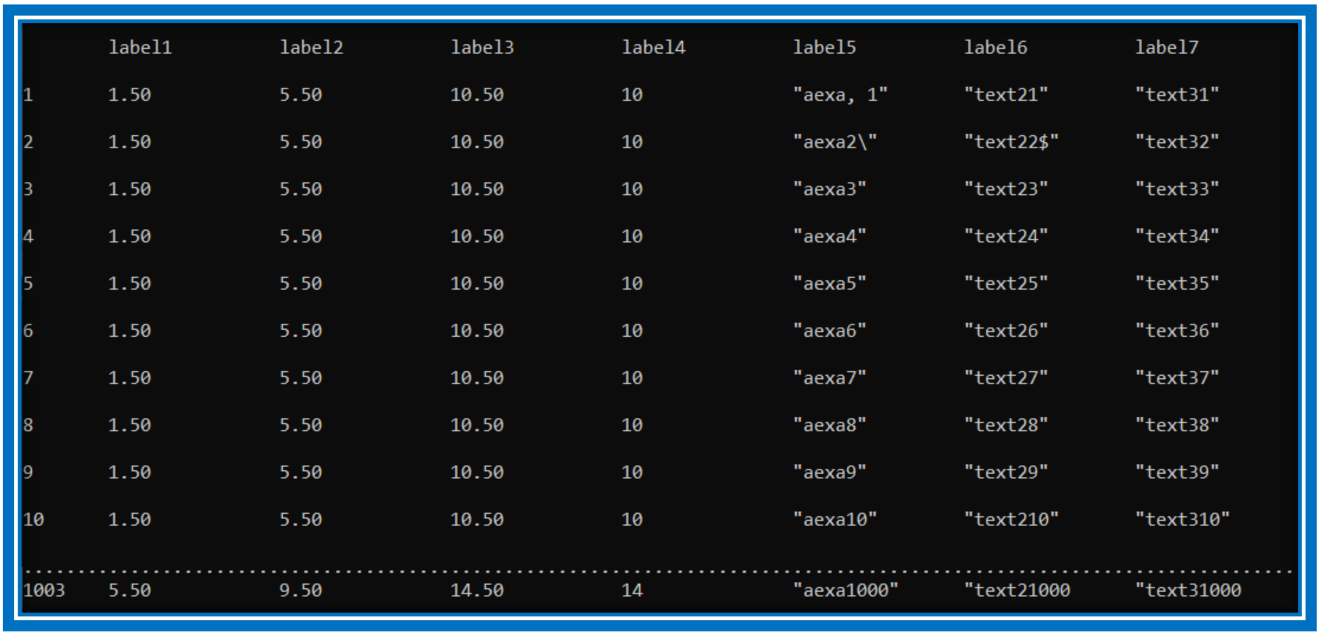
```
        label1         label2         label3         label4         label5         label6         label7

1       1.50           5.50           10.50          70             "text, 1"      "text21"       "text31"

2       1.50           5.50           10.50          70             "text2\"       "text22$"      "text32"

3       1.50           5.50           10.50          70             "text3"        "text23"       "text33"

4       1.50           5.50           10.50          70             "text4"        "text24"       "text34"

5       1.50           5.50           10.50          70             "text5"        "text25"       "text35"

6       1.50           5.50           10.50          70             "text6"        "text26"       "text36"

7       1.50           5.50           10.50          70             "text7"        "text27"       "text37"

8       1.50           5.50           10.50          70             "text8"        "text28"       "text38"

9       1.50           5.50           10.50          70             "text9"        "text29"       "text39"

10      1.50           5.50           10.50          70             "text10"       "text210"      "text310"

...........................................................................................................
1003    5.50           9.50           14.50          98             "text1000"     "text21000     "text31000
```

The following is the user defined function to replace a ‘t’ with an ‘a’:



```
void stringinput(char* string,char strchange,char strreplace)
{
    int pos=0;//Position to hold position of string array
    //Checking if tmp has reached end of string
    while(string[pos]!='\0'){
        //If tmp[pos] is 't' then tmp[pos] is changed to 'a'
        if(string[pos]==strchange){
            string[pos]=strreplace;
        }
        //Incrementing position
        pos++;
    }
}
```

Utilising the showDT method for testing purposes to show the data change in the DataTable_t structure (performing mutation on column 5):



```
        label1     label2     label3     label4     label5       label6       label7
1       1.50       5.50       10.50      10         "aexa, 1"    "text21"     "text31"
2       1.50       5.50       10.50      10         "aexa2\"     "text22$"    "text32"
3       1.50       5.50       10.50      10         "aexa3"      "text23"     "text33"
4       1.50       5.50       10.50      10         "aexa4"      "text24"     "text34"
5       1.50       5.50       10.50      10         "aexa5"      "text25"     "text35"
6       1.50       5.50       10.50      10         "aexa6"      "text26"     "text36"
7       1.50       5.50       10.50      10         "aexa7"      "text27"     "text37"
8       1.50       5.50       10.50      10         "aexa8"      "text28"     "text38"
9       1.50       5.50       10.50      10         "aexa9"      "text29"     "text39"
10      1.50       5.50       10.50      10         "aexa10"     "text210"    "text310"

.........................................................................................
1003    5.50       9.50       14.50      14         "aexa1000"   "text21000   "text31000
```

The functionClass2b.h included the declaration of the following functions to be used in the test driver:

```
/*operation: initialises DataTable_t struct*/
/*preconditions: integer parameters choice,noofcols and noofrows */
/*                choice can be either 0 or 1 and noofcols and noofrows refer*/
/* to number of columns and rows in DataTabel_t struct respectively */
/*postconditions: The DataTable_t is initialised to empty */
DataTable_t *initDT(int choice , int noofcols,int noofrows);

/*operation: Frees the DataTable_t from memory*/
/*preconditions: DataTable_t pointer */
/*postconditions: Relinquishes all sources of DataTable_t*/
void deinitDT(DataTable_t *data);

/*operation: loads a csv file into the DataTable_t struct*/
/*preconditions: DataTable_t pointer */
/*postconditions: DataTable_t struct initialised with data from csv file */
void loadDT(DataTable_t *data);

/*operation: exports the contents of the DataTable_t struct into a csv file*/
/*preconditions: DataTable_t pointer */
/*postconditions: csv file contains data from DataTable_t struct*/
void exportDT(DataTable_t *data);

/*operation: prints DataTable_t struct contents*/
/*preconditions: DataTable_t pointer */
/*postconditions: DataTable_t struct is displayed */
void showDT(DataTable_t *data);

/*operation: Returns a copy of the DataTable_t with specific rows and columns*/
/*preconditions: DataTable_t pointer */
/*postconditions: pointer to new DataTable_t struct */
DataTable_t *projectDT(DataTable_t *data);

/*operation: Accepts a user defined function to transform columns*/
/*preconditions: DataTable_t pointer and 3 user defined function with types float,int
and char* respectively */
/*postconditions: DataTable_t struct including transformed column */
void mutateDT(DataTable_t *data,float (*floatfunction)(float num,int change),int
(*intfunction)(int num,int change),void (*stringfunction)(char* string,char
strchange,char strreplace));

/*User defined function used for testing:*/
float floatnumbers(float num,int change);

/*User defined function used for testing:*/
int intnumbers(int num,int change);

/*User defined function used for testing:*/
void stringinput(char* string,char strchange,char strreplace);
```

The test driver class: main2b.c

```c
//Runner file
//Header file used in the program:
#include "functionClass2b.h"

int main() {
//Calling the respective methods from the functionClass.c
//Calling initDT method with choice-0 , since creating a new DataTable from file ,
//and setting noofcols and rows to 0 since they are not needed
    DataTable_t *data=initDT(0,0,0);
    loadDT(data);
    showDT(data);
    exportDT(data);
    DataTable_t *data2= projectDT(data);
    showDT(data2);
    deinitDT(data2);
    mutateDT(data,floatnumbers,intnumbers,stringinput);
    showDT(data);
    deinitDT(data);
    return 0;
}
```

4. A DataTable library c)

To complete the full implementation as a shared library, the contents of functionClass2b.h was copied to functionClass2c.h, functionClass2b.c was copied to functionClass2c.h and main2b.c copied to main2c.c respectively. The CMakeLists.txt was altered as can be seen below to account for a shared library interface.

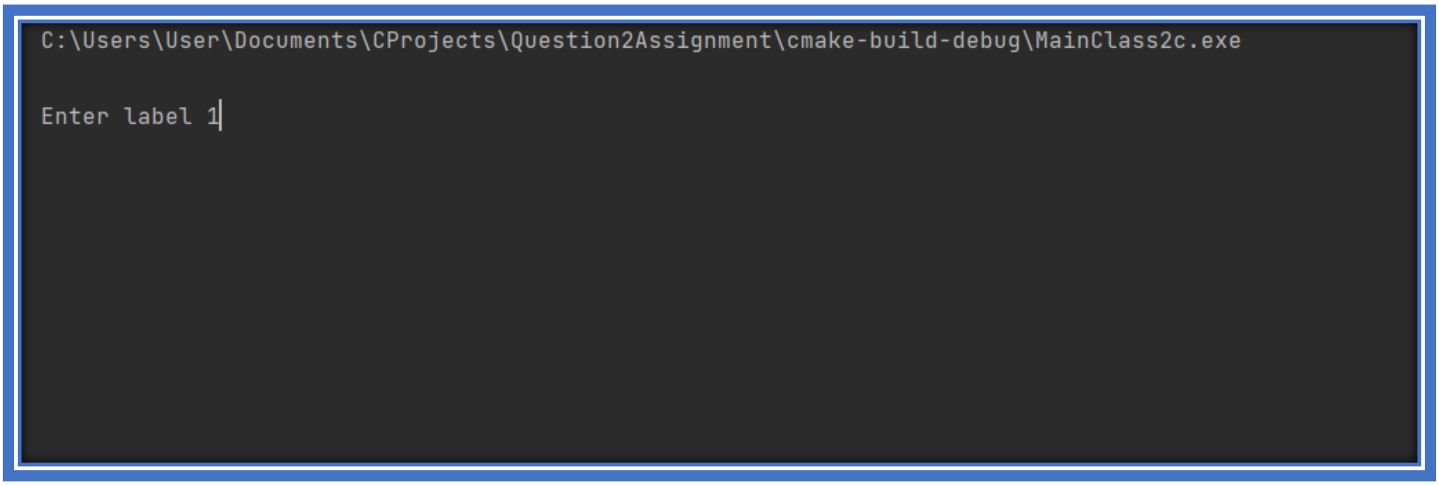
```
set(SOURCE_FILES1 functionClass2c.c)
add_library(MainClass SHARED ${SOURCE_FILES1})

set(SOURCE_FILES2  main2c.c)
add_executable(MainClass2c ${SOURCE_FILES2})
target_link_libraries(MainClass2c MainClass)
```

ADT interface documentation located in the functionClass2c.h header files, which outline the operation, preconditions and post conditions required:

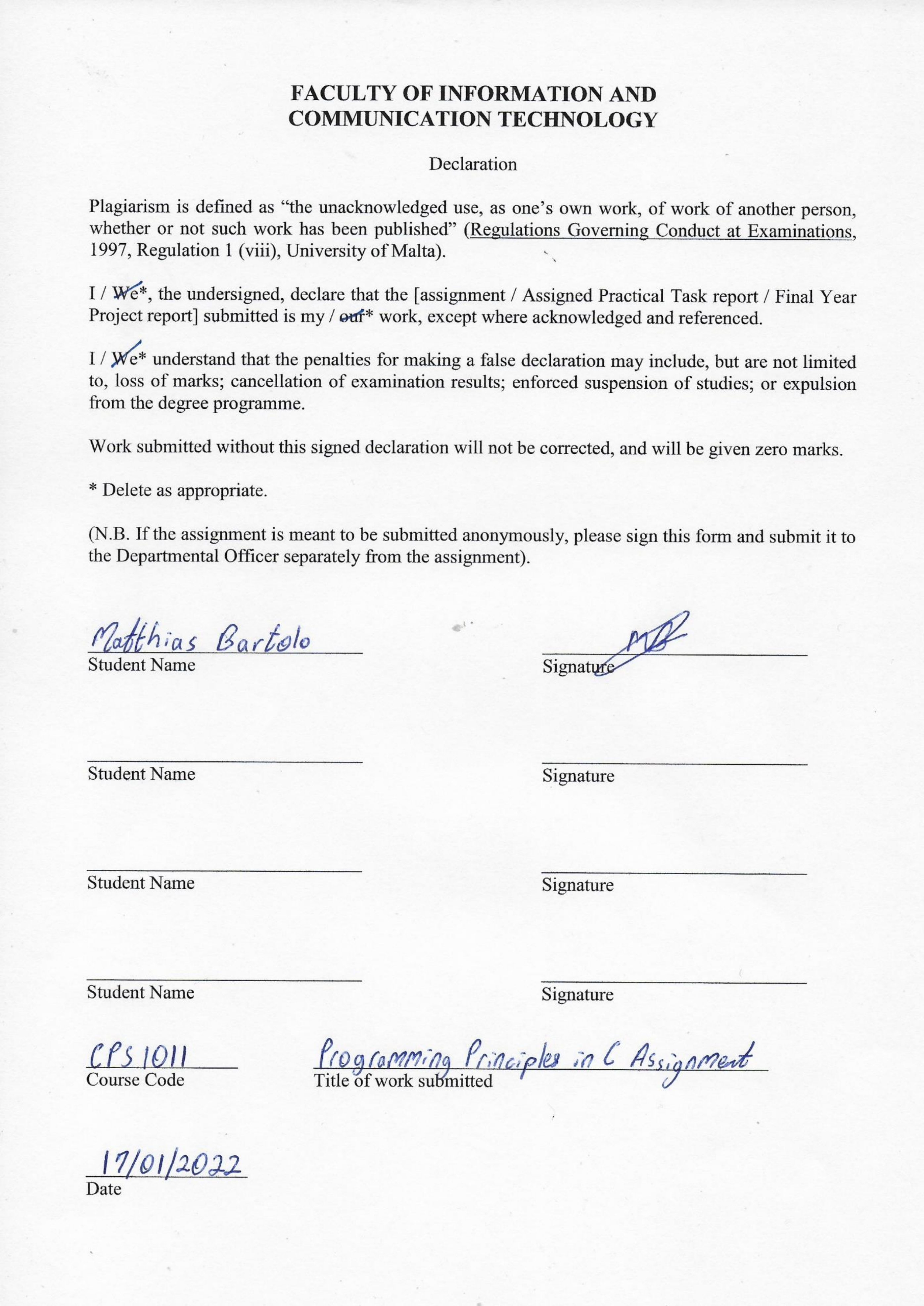
```
/*operation: initialises DataTable_t struct*/
/*preconditions: integer parameters choice,noofcols and noofrows */
/*               choice can be either 0 or 1 and noofcols and noofrows refer*/
/* to number of columns and rows in DataTabel_t struct respectively */
/*postconditions: The DataTable_t is initialised to empty */
DataTable_t *initDT(int choice , int noofcols,int noofrows);

/*operation: Frees the DataTable_t from memory*/
/*preconditions: DataTable_t pointer */
/*postconditions: Relinquishes all sources of DataTable_t*/
void deinitDT(DataTable_t *data);

/*operation: loads a csv file into the DataTable_t struct*/
/*preconditions: DataTable_t pointer */
/*postconditions: DataTable_t struct initialised with data from csv file */
void loadDT(DataTable_t *data);

/*operation: exports the contents of the DataTable_t struct into a csv file*/
/*preconditions: DataTable_t pointer */
/*postconditions: csv file contains data from DataTable_t struct*/
void exportDT(DataTable_t *data);

/*operation: prints DataTable_t struct contents*/
/*preconditions: DataTable_t pointer */
/*postconditions: DataTable_t struct is displayed */
void showDT(DataTable_t *data);

/*operation: Returns a copy of the DataTable_t with specific rows and columns*/
/*preconditions: DataTable_t pointer */
/*postconditions: pointer to new DataTable_t struct */
DataTable_t *projectDT(DataTable_t *data);

/*operation: Accepts a user defined function to transform columns*/
/*preconditions: DataTable_t pointer and 3 user defined function with types float,int
and char* respectively */
/*postconditions: DataTable_t struct including transformed column */
void mutateDT(DataTable_t *data,float (*floatfunction)(float num,int change),int
(*intfunction)(int num,int change),void (*stringfunction)(char* string,char
strchange,char strreplace));

/*User defined function used for testing:*/
```

```
float floatnumbers(float num,int change);

/*User defined function used for testing:*/
int intnumbers(int num,int change);

/*User defined function used for testing:*/
void stringinput(char* string,char strchange,char strreplace);
```

Showing successful library linking in MainClass2c, through initDT method functioning properly:



```
C:\Users\User\Documents\CProjects\Question2Assignment\cmake-build-debug\MainClass2c.exe

Enter label 1
```



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as “the unacknowledged use, as one’s own work, of work of another person, whether or not such work has been published” (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I / ~~We~~*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / ~~our~~* work, except where acknowledged and referenced.

I / ~~We~~* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Matthias Bartolo Student Name	Signature
Student Name	Signature
Student Name	Signature
Student Name	Signature

CPS1011
Course Code

Programming Principles in C Assignment
Title of work submitted

17/01/2022
Date