# Computing Structures – Fall 2023
# Project 1
## Due: September 15th 2023 at 11:59 PM

## *Objective:*

This project requires you to create tabular data through C++. The tabular data will be heterogeneous and may contain headers for the columns.
In order to implement this in C++, you will be required to use some STL (Standard Template Library) classes, like vector, to store the data. The data will be stored using a user defined class. You will be provided with a boilerplate code to get started with this project.

## *Description:*

You will be required to read two files, the first one being a standard text file and the other being a Comma Separated Value (CSV) format file. The text file will have information about the data in the csv file, i.e. the number of columns in the csv file, the data type of each column, the number of rows in the csv file and whether the csv file has headers for the data or not. Thus, you can say that the text file stores metadata on the csv file.

The project will have two user defined classes: “aRow” and “Table”. The class “aRow” will store information for a single row. This information will be stored as a vector of class **string**. The class “Table” will hold information about all the rows in a file. It will have a vector of our user defined class “aRow”. In addition to this, the class will also store information about the headers in a csv file, as well as the data type of each column.

## *Input explanation:*

**Following is a sample csv file (addresses.csv):**

```
"Name",     "Sex", "Age", "Height (in)", "Weight (lbs)"
"Alex",      "M",   41,      74,      170
"Bert",      "M",   42,        ,      166
"Carl",      "M",   32,      70,      155
"Dave",       "M",   39,        ,      167
"Elly",      "F",   30,      66,
```

This csv file has 5 records (rows), 5 fields (columns) and headers. Not all fields in all rows will be populated, as you can see in the example above. You need to account for that when reading data from the file. One way to handle missing data is to use **nullptr**.

The corresponding text input file (input1.txt) will be:

```
1 biostats.csv
5
String
Char
Int
Int
Int
5
```

The **first line** is interpreted as follows:
The csv file has a header, followed by the csv file name (biostats.csv). If the line was instead:
0 addresses.csv
The interpretation would be: The csv file does not have a header, and the name of the file is addresses.csv.

The **second line** is interpreted as the number of columns in the csv file.

The **next five** lines represent the data types of the columns.

The **last line** represents the number of records in the csv file.

Even though the columns have different data types, they will all be stored as strings.

You will read the text input file through redirected input and then proceed to reading the csv file through file I/O, and storing the data in the user defined class objects mentioned.

## ***Class definition:***

The user defined classes will have the following layout:

```
class aRow{
public:
   vector<string> myRow; //for storing the row data

   //methods
   aRow (); // default constructor
   display (); //display method
   string getColValue (int col); //get myRow[col]
   ~aRow();//destructor
   //other methods if you need them
};
```

```
class Table {
public:
   vector<string> colTypes; //storing column data types.
   vector<string> colNames; //storing column headers, if applicable.
   vector<aRow> myTable; // storing the rows with vector of user defined class.

   //methods
   Table (); //constructor
   ~Table(); // destructor
   Table* selectColumns (int* colNums); //returns a pointer to a Table object which has data
                                   // for the array of column numbers in the argument.
   Table* subsetTable (int bRow, int eRow, int bCol, int eCol); // returns a pointer to a Table
object with
                                                  // beginning and end row and column numbers.
   string columnAverage (int colNum); // returns the average of values in the column colNum.
   string columnMax (int colNum); // returns the max value of the column colNum.
   string columnMin (int colNum); // returns the min value of the column colNum.
   int missingValues (int colNum); // returns the number of missing values in the column
colNum.
   displayTableSummary(); //Displays number of rows, number of columns, total number of
missing values,
                         //table Headers (1, 2, 3, ... in case there are no headers), followed by the tabular
data.
   display (); //display the table with column names
};
```

You will have to implement the constructor, destructor as well as the functions provided in the class definitions.

The **stringstream** class can be used to tokenize the row of data into individual fields, using the comma ‘,’ delimiter. The statement,
#include <sstream>
includes the class in the program.

The **vector** is a very handy class in the STL for storing one-dimensional data of a given type. It has also been included in the boilerplate code.

## ***Output:***

The output is defined by the main function in the boilerplate code. A sample output file will be provided shortly.

```
int main () {
    Table* aTable;

    int headerOrNot;
    string csvFileName;

    ifstream csvFile (csvFileName);
    int numColumns;
    int numRows;

    cin << headerOrNot << csvFileName;
    cin << numColumns;

    //Create a Table instance
    aTable = new Table ();

    // next read the data types and store this *aTable object - Write code

    cin << numRows;

    // open and read the csv files that has numRows and numCols and store this in aTable - Write
code

    cout << (*aTable).displayTableSummary();

    int cols = [0,1,4,7];
    Table* bTable = (*aTable).selectColumns (cols);
    (*bTable).displayTableSummary();

    Table* cTable = (*bTable).subsetTable (0, 99, 2, 3);
    (*cTable).displayTableSummary();
    (*cTable).display();

    cout << (*aTable).columnAverage (3);
    cout << (*bTable).columnMax (2);
    cout << (*cTable).columnMin (1);
    cout  << (*aTable).missingValues (4);
}
```

## ***Submission:***

Submission will be through *GradeScope*. Your program will be autograded. You may submit how many ever times to check if your program passes the test cases provided. One test case will be released at the beginning and later other test cases will be released while grading. For the autograder to work, the program you upload must be named as ***project1.cpp***.

Your final submission must contain your source file named project1.cpp. You access GradeScope using the tab on the left in our course page on canvas.

## ***Constraints:***

1. None of the projects is a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.
2. Please review academic integrity policies in the modules.

## ***Programming advice:***

1. Before working on data manipulation, please make sure that you are reading the data correctly. Use cout statements to print out the values you are storing. Data properly read and stored is more than half the job done in my opinion.
2. Try to create the program in modules. Be consistent in you programming effort.
3. If your program works for reading data from the input files provided, try creating your own files and see if your program reads them properly too.

## ***How to ask for help:***

1. Please attend the help sessions. You can even stay there while you work.
2. Please always have a lookout for announcements on canvas. This class has video lectures that were recorded in the past, so the current information on the course will be mainly through announcements.
3. Email the TA with your precise questions. Please attach a snapshot of the error and your source file to the email if the question is regarding error(s) in compilation or runtime.