ENSF 694 – Summer 2024
Lab 4
Department of Electrical & Computer Engineering
University of Calgary
Written by: M. Moussavi, PhD, PEng.

Objective:

The objectives of this lab assignment are to practice and understand:

1. The concept of data structure lookup tables.
2. Manipulation of data from text files in C++.
3. Usage of array of pointers and pointer-to-pointers.

Due Dates: Regarding upcoming midterm exam, the number of exercises in this lab is less than previous labs and its due date is extended to: **Friday July 26, before 11:59 PM**

Marking scheme:

- Exercise A 25 marks
- Exercise B 4 marks
- Exercise C 4 marks
- Exercise D 12 marks
-

Total: 45

Exercise A: Data Structure – LookupTable

Lookup Table is a data structure, which is generally an association of unique keys with some values. (Other common names for this type of data structures are Map and Dictionary). *Lookup Tables* are very useful abstract data types (ADT) that contain a collection of items called *keys*, which are typically numbers. Associated with each key is another item that will be called a *datum* in this exercise. (``Datum" is singular form of the plural noun "data".)

Typical operations for a *Lookup Table* include inserting a key with an associated datum, removing a key/datum pair by specifying a key, and searching for a pair by specifying a key. Lookup Tables can be implemented using different data structure such as arrays, vectors, or linked lists. In this exercise a linked list implementation, called `LookupTable` class is introduced. Class `LookupTable`, in addition to a node-pointer that usually points to the first node in the linked list has another node-pointer called `cursor` that is used for accesses to the individual key/datum pairs.

What to Do:
This exercise has three parts. Part one and two will be marked. Part three will not be marked and you don't have to submit anything. However, the third part is as important as other two parts and you are strongly recommended to complete part three, as well.

Part One (15 marks):
Create a directory called `Lab4_ExA_Part1` and download files, `lookupTable.h, lookupTable_tester-pat1.cpp`, and `data-part1.txt` into this directory.

In lookupTable.h there are three user-defined data types: `struct Pair`, `struct LT_Node`, and `class LookupTable`. Also please notice the usage of `typedef` for the convenience of changing the type of datum. In this part `Type` is an alias name for C++ library class `string` type.
You must also notice that I have created constructors for structs `Pair` and `LT_Node` (just for the convenience of creating objects of this structures).

Now, take a look at the file `lookupTable_tester_part1.cpp` , in this file there are few things to understand:

1. The C function `freopen`, allows to read the input from a text file instead of reading from keyboard.
2. The input will be read from file `data-part1.txt`, that must be in your working directory.

3. If all line in file `data-part1.txt` runs successfully the program should produce an output like this

```
Starting Test Run. Using input file.
Line 1 >> is comment
Line 2 >>  Passed
Line 3 >>  Passed
Line 4 >>  Passed
...
...
Line n >> Passed
```

And more test results.

After reading the downloaded files carefully, and understanding the structure of a program, your task is to create a file called `lookupTable.cpp`, and implement all the member functions that are declared in `lookupTable.h`.

Then you should compile, run, and test your program.

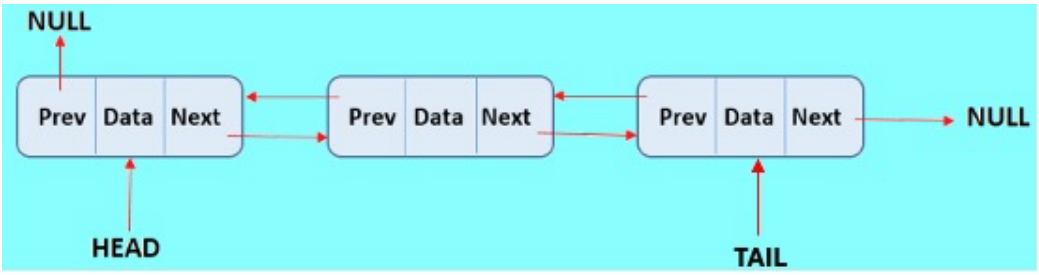
Part Two (10 marks):
For this part you need to take the following steps:

1. Create a new directory called `Lab4_ExA_PartII`.
2. Download file `Point.h`, and `data-part2.txt`, and `lookupTable_tester2.cpp`, into this directory
3. Copy your file `lookkupTable.h` and `lookupTable.cpp` from part one into this directory.
4. Open file `lookupTable.h` and change the line: `typedef string Type` to:

```
typedef Point Type;
```

5. Create a new file called `Point.cpp`, write the implementation of all the member functions declared in `Point.h` header file.
6. Now if you try to compile and run the program, you will face a few errors. Why? Because some statements don't fully comply with the requirements of `Point` objects. You need to make some changes to all files except the file `data-part2.txt` and `lookupTable_tester2.cpp`.
7. Once you made all the necessary changes and you could eliminate all compilation errors, test your program and make sure all required features work fine.

Part Three (not marked and you shouldn't submit anything for this part):
Create a new director called `Lab4_ExA_Part3` and copy all files from Part Two. In this part you are expected to convert your `LookupTable` class to **doubly linked** list lookup table class. A doubly linked list is a list that in each node there is pointer next, pointing to the next node and a pointer previous, pointing to the previous node. Also, in addition to the head, the linked list has a pointer called tail pointing to the last node. These additional features help some of the operations to become more efficient.
Here is the picture of a doubly linked list:



Exercise B: C++ File I/O (4 marks)

The objective of this exercise is to help you understand the basics of file I/O in C++

What to Do:

Download the file `lab4exe_B.cpp` from D2L. If you read this file carefully you will find out that this simple C++ program creates a binary file that contain several records, where each record is an object of a struct, type called `City`. The program contains several functions; the implementation of one of them called `print_from_binary` is missing. This function is supposed to read the content of the binary file created by the program and print the records in the following format:

```
Name: Calgary, x coordinate: 100, y coordinate: 50
```

Note: If your compiler is using C++17 instead of C++11, since lab exercises in this lab are developed based on C++11, the declaration of `const int size` in the program may give an ambiguity error. Therefore, you can switch to C++11, and get rid of this error message by compiling your code as follows:

```
g++ -std=c++11 -Wall …
```

What to Submit:

Submit the definition of your function `print_from_binary` and your program's output as part of your lab report in PDF format.

Exercise C: Working with Array of Pointers (4 marks)

What to Do:

Download the file `lab4exe_C.cpp` from D2L and read it carefully to understand what it does. Then:

1. Draw a memory diagram for point 1
2. Predict what is the program output at point one.
3. Compile and run the program to find out if your prediction is correct.
4. Read the function interface comment and the definition of function `insertion_sort`, which sorts an array of n integers. Its function prototype is as follows:

   ```
   void insertion_sort(int *int_array, int n);
   ```

5. Change the pre-processor directive `#if 0` to `#if 1`, and write the definition of the other overloaded definition of `insertion_sort` with the following prototype:

   ```
   void insertion_sort(const char** str_array, int n);
   ```

   The first argument of this function is a pointer that points to an array of `n` C-strings. The job of the function is to rearrange the pointers in `str_array` in a way that, lexicographically: `str_array[0]` points to the smallest string, `str_array[1]` points to the second smallest,`...`, `str_array[n-2]` points to second largest, and finally `str_array[n-1]` points to the largest string.

   The following code segment shows the concept of rearranging the pointers in an array of pointers, and referring to their target strings in a non-decreasing order:

   ```
   const char* str_array[3] = {"xyz", "klm" , "abc"}

   const char* tmp = str_array[0];
   ```

```
str_array[0] = str_array[2];
str_array[2] = tmp;

for(int j=0; j < 3; j++)
   cout << str[i] << endl;
```

And, here is the output of this code segment:

```
abc
klm
xyz
```

What to Submit:

Submit your AR diagram, the modified copy of the file `lab4exe_C.cpp`, and the program output, as part of your lab report in PDF format.

Exercise D: Pointer-to-Pointers and Command-line Arguments (12 marks)

Read this First

Up to this point in our C or C++ programs we have always used the main function without any argument(s). However, C and C++ support a means to pass some information to a program via command-line arguments. A set of good examples of the programs that use this feature of C/C++ is Linux and Unix commands such as: `cp`, `mv`, `g++`. For example, the following `cp` command receives the name of two file, and makes `f.dat` as a copy of `f.txt`:

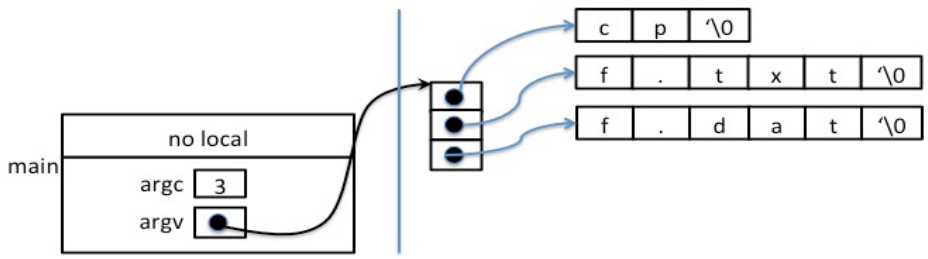
```
cp f.txt f.dat
```

The first token in the above `cp-command` is the program's executable filename followed by two other pieces of information that can be used by the program. How does it work?

To access the command-line arguments, a C/C++ main function can have argument as follow:

```
int main(int argc, char **argv)
{
   // MORE CODE
   return 0;
}
```

Where, `argc` is an integer that holds the number of tokens on the command-line. As an example, in the above-mentioned `cp-command` the value of `argc` is 3. The delimiter to count for the number of tokens on the command- line is one or more spaces. The second argument is a pointer-to-pointer, which points to an array of pointers. Each pointer in this array points to one of the string tokens on the command line. The following figure show how `argv[0]`, `argv[1]`, and `argv[2]` point to the tokens on the command line:



The exact location of the memory allocated for command-line arguments depends on the underlying OS and the compiler, but for most of the C/C++ systems it is a special area on the stack that is not used for the activation records.

What to Do:

1. Download files `matrix.h, matrix.cpp,` and `lab4exe_D.cpp`, from D2L.
2. Read these files carefully to understand how the class `Matrix` dynamically allocates memory for an array-of-pointers called `matrixM`. Each element of this array is supposed to point to a dynamically allocated array of `doubles`.

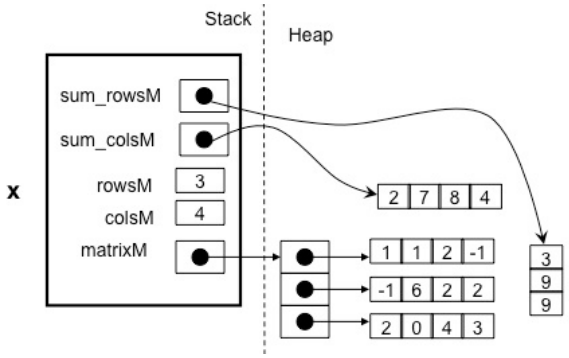
   The class `Matrix` also contains the following private data members:
   `rowsM:` which holds the number of rows in a matrix object
   `colsM:` which holds the number of columns in a matrix object
   `sum_rowsM:` is a pointer to double that is supposed to point to an array which is used for storing the sum of the values in each row of the matrix.
   `sum_colsM:` is a pointer to double that is supposed to point to an array which is used for storing the sum of the values in each column of the matrix.
3. Compile the program using the following command:

```
g++ matrix.cpp lab4exe_D.cpp -o matrix
```

4. Then run the program using the following command:

```
./matrix.exe  3  4
```

5. The given program will use the command-line argument to create an object of class `Matrix` with 3 rows and 4 columns. Here is picture of such an object, called x:



6. Check the program output and review the given codes again to understand how the constructor and other given member functions of class `Matrix` work. The above picture is an example of the product that will be generated by the constructor.
7. Now in the main function change the preprocessor directive #if 0 to if 1.
8. Compile the program and run it again with the command-line arguments for the number of rows and columns and check the output again. Now you will see some messages that indicates four member functions of class Matrix are incomplete/defective (`sum_of_rows, sum_of_cols, copy`, and `destroy`). Your job in this exercise is to complete all those incomplete functions and get rid of those messages.

What to Submit:

Submit your modified version of the file `matrix.cpp`, and your program output as part of your lab report in PDF format.