**Department of Electrical and Computer Engineering**

# ENSF 337: Programming Fundamentals
# Lab-9 — Week of November 22, 2021

*Written by: M. Moussavi, PhD, P.Eng*

## Note: This is an individual lab. You CANNOT work with a partner.

## Introduction:

Some of the materials in this lab include:

- Another exercise on linked list
- A simple exercise to read binary data from a binary file
- An exercise on using the C++ library classes: vector and string
- Application of pointer-to-pointers (Material on this topic will be covered on Monday Nov 25th.
    - Using an array of pointers to access the data in another array in a specific order.
    - Using a pointer-to-pointer to build a matrix of double numbers, dynamically.

## Due Dates: Thursday December 3rd, before 11 AM.

## Marking scheme:

Total mark for the exercises in this lab is: 23 **marks**

- Exercise A: Not marked
- Exercise B: 4 marks
- Exercise C: 4 marks
- Exercise D: 4 marks
- Exercise E: 11 marks

## Exercise A: A linked list with nodes of class type objects

### Introduction:

This exercise introduces a C++ linked list class that uses a class type `Node` instead of struct type. Which means, node objects must be created by calling its constructor, and its data member can be accessed only by calling its member function (getters and setters). In addition to a `Node` pointer, class `SimpleList` uses another data member that keeps track of the number of nodes inserted into the list. This additional piece of information allows us to implement operations such as removing a node from i$^{th}$ position in the list or inserting a node into the i$^{th}$ position. For the purpose this exercise, if an object of `SimpleList` has `n` nodes, we consider the first node is located at the position 0 and the last node at the position `n-1` (similar to the concept of arrays).

### A Brief Note on C++ `nullptr`

Since C++11 there has been a replacement for `NULL` (defined constant for zero), whenever its supposed to be used for a pointer type. This new keyword is: `nullptr`. This change is due to an ambiguity that can happen when an integer zero is used as an integer constant and as a value for a pointer. Because C++ is a language that supports function overloading, the issue of ambiguity can arise in situations as follows. Lets assume we have two functions overloaded as follows:

```
void foo (char*);
void foo(int);
```

The call: `foo(NULL);` generates an ambiguity error since this call can be resolved to both functions. Moreover, since C++ forbids implicit conversion from `void*` to other pointer types, this new keyword, `nullptr`, serves as a distinguished null pointer constant.
Please note in some systems you may need to use the following command to switch to C++11, compiler to recognize the `nullptr` keywork:

```
g++ -std=c++11 -Wall …
```

**What To Do:**

Download the files `SimpleList.cpp`, `SimpleList.h`, `node.cpp`, `node.h`, and `lab9_exA.cpp` from D2L. Read these files carefully, then draw AR diagrams for points in member functions **`set_next`** (located in node.cpp), **`push_front`**, both **`at`** functions, and **`copy`** (five points in total). You should draw diagrams at these points, when the program reaches there for the **first time**.

**Notes:**

- Points in this exercise are not numbered and you need to trace the program very carefully, line by line, to reach each point without missing any of them.
- In your diagrams you should indicate the actual order-number of the point that is called (Point 1, 2, 3, and so on).

**What to Submit:**

Nothing to submit in this exercise

## Exercise B: C++ File I/O

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

**What to Do:**

Download the files `lab9ExB.cpp` from D2L. If you read this file carefully you will realize 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 in the binary file created by the program and print the records in the following format:

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

**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: Using C++ library classes, `vector` and `string`

The objective of this exercise is to gain some experience in understanding the C++ library classes, `vector`, and `string`.

**What to Do:**

Download the files `lab9ExC.cpp` from D2L. In this file there is a declaration of `vector <string>`. If you compile and run this program it creates the following output:

```
ABCD
EFGH
IJKL
MNOP
QRST
```

Lets visualize this output as a matrix of letters (5 rows and 4 columns):

| | | | |
|---|---|---|---|
| A | B | C | D |
| E | F | G | H |
| I | J | K | L |
| M | N | O | P |
| Q | R | S | T |

Your job is to complete the definition of the function called `transpose` that creates a new object of `vector<string>` where its strings are the transpose of the original vector:

| | | | | |
|---|---|---|---|---|
| A | E | I | M | Q |
| B | F | J | N | R |
| C | G | K | O | S |
| D | H | L | P | T |

To test your program you can change the values of the constants `ROWS` and `COLS`, in the `main` function to make sure your function works with other sizes of the `String_Vector`.

**What to Submit:**

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

## Exercise D: Working with Array of Pointers

**What to Do:**

Download the file `lab9ExD.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 `lab9ExD.cpp`, and the program output, as part of your lab report in PDF format.

## Exercise E: Pointer-to-Pointers and Command-line Arguments

**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`:

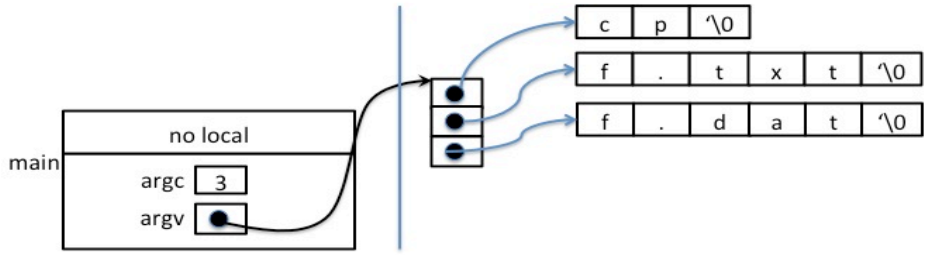
```
cp f.txt f.dat
```

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 `lab9ExE.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`.

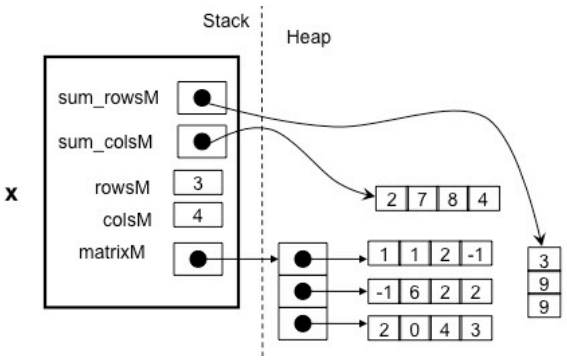
   The class `Matrix` also contains the following private data members:
   `rowsM`: which holds the number of rows in a matrix object
   `colsM`: which holds the number of columns in a matrix object
   `sum_rowsM`: is a pointer to double that is supposed to point to an array which is used for storing the 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 lab9ExE.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.