University of Calgary
Department of Electrical and Computer Engineering
Advanced Systems Analysis and Software Design (ENSF 614)
Lab 1

M. Moussavi, PhD, PEng

Important Notes:

- This is a group assignment, and you can work with a partner (groups of three or more **are NOT allowed**). Working with a partner usually should give you the opportunity to discuss some of details of the topics and learn from each other. Also, it will give you the opportunity to practice one of the popular methods of program development called, **pair-programming.** This method which is normally associated with "Agile Software Development" technique, two programmers work together normally on the same workstation (you may consider a zoom session for this purpose). While one partner, the driver, writes the code, the other partner, acts as observer, looks over his/her shoulder making sure the syntax and solution logic is correct. Partners should switch roles frequently in a way that both of them have equivalent opportunity to practice both roles.

- When submitting your source code (.cpp, or .h), make sure the following information appears at the top of your file:
    a. File Name
    b. Assignment and exercise number
    c. Lab section
    d. Your name
    e. Submission Date:

  Here is an example:

```
/*
 *  File Name: lab1exe_F.cpp
 *  Assignment: Lab 1 Exercise F
 *  Completed by: Your Name (or both team members name for group exercises)
 *  Submission Date: Sept 18, 2025
 */
```

Start Date: Wed. Sept 10, 2025

Due Dates: Wed Sept 17, before 3:00 PM

Objectives:

As mentioned during the lectures, the focus of the first part of the course is code-level design concepts. These topics includes concepts such as overloading operators, friend concept, inheritance, multiple inheritance, realization, aggregation, and delegation in C++. However, the purpose of this lab is:

1. First to review basic C++ concepts.
2. Understand and apply the basic low-level object-oriented class design and programming concept such as "Abstraction", "Encapsulation", "Modularity", and "Hierarchy".

Note:
Your lab reports must have a cover page with the following information. This is just for helping your TAs to easily record your marks on the D2L system.

Name (s): If working with a partner both students' name must be written
Course Name: Principles of Software Design
Course Code: ENSF 614
Assignment Number: For example, Lab-1
Submission Date and Time: DD/MM/YYYY

Marking scheme:

Please notice that only some of the lab exercises in this lab will be marked. Since each TA should mark several lab assignments and for each lab on average, we will have 3 or 4 exercises, it is almost impossible for them to mark all the lab exercises. Therefore, in this lab we only mark exercise B and D. But it doesn't mean that remaining exercises are unimportant or less important. All exercises are equally important, and you are strongly urged to complete all them, as similar questions will likely appear in upcoming midterm exams.

Exercise B: 12 marks
Exercise D: 20 marks

Total: 32 marks.

Exercise A – Review of C++ Fundamentals

The purpose of this exercise is to refresh your memory about some of the basic constructs of C++.

What to do:

Step1. Download files `mystring.h`, `mystring.cpp` and `exAmain.cpp` from D2L.

Step 2. If you are using Cygwin, or Mac Terminal, compile the files `mystring.cpp` and `exAmain.cpp`, using the following command:

```
g++ -Wall  -o myprog  mystring.cpp exAmain.cpp
```

Note: `myprog` is the program's executable file name.

Step 3. Run the program from your working directory (exA), using the following command:

```
./myprog
```

Step 4. Use your text editor to open files: `mystring.h`, `mystring.cpp`, `exAmain.cpp`, and try to understand how class `Mystring` and its functions work.

Step 5. Now to better understand this class, draw AR diagrams for points `one,` and `two`, when the program reaches these points for the first time. If you need to review and refresh your C++ programming background, please study the slides and documents posted on the D2L

There is a copy of the following table in MS Word format available on the D2L. You should download the given file (called `AnswerSheet.doc`) and type your answers into the given table.

The answer for the first output is given, as an example for the expected details of your answer.

	Program output and its order	**Your explanation (why and where this output was created)**
1	**constructor with int argument is called.**	it is called at line 12 in exAmain. The statement, Mystring c = 3 is interpreted by the compiler as a call to the constructor `Mystring::Mystring(int n).`
2	**default constructor is called.** **default constructor is called.**	
3	**constructor with char* argument is called.**	
4	**copy constructor is called.**	

	copy constructor is called.	
5	**destructor is called.** **destructor is called.**	
6	**copy constructor is called.**	
7	**assignment operator called.**	
8	**constructor with char* argument is called.** **constructor with char* argument is called.**	
	destructor is called. **destructor is called.** **destructor is called.** **destructor is called.** **destructor is called.**	
7	**constructor with char* argument is called.**	
8	**Program terminated successfully.**	
9	**destructor is called.** **destructor is called**	

What to Submit:

This exercise will not be marked, and you don't need to submit anything.

Exercise B:

The purpose of this exercise is to refresh your memory about some of the important and basic details of designing a C++ class. Particularly concepts such: class constructor, destructor, copy constructor, and assignment operator.

Dictionary 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 Lookup Table). *Dictionaries* are very useful abstract data types (ADT) that contain a collection of items called *keys*, which are typically strings or 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".)

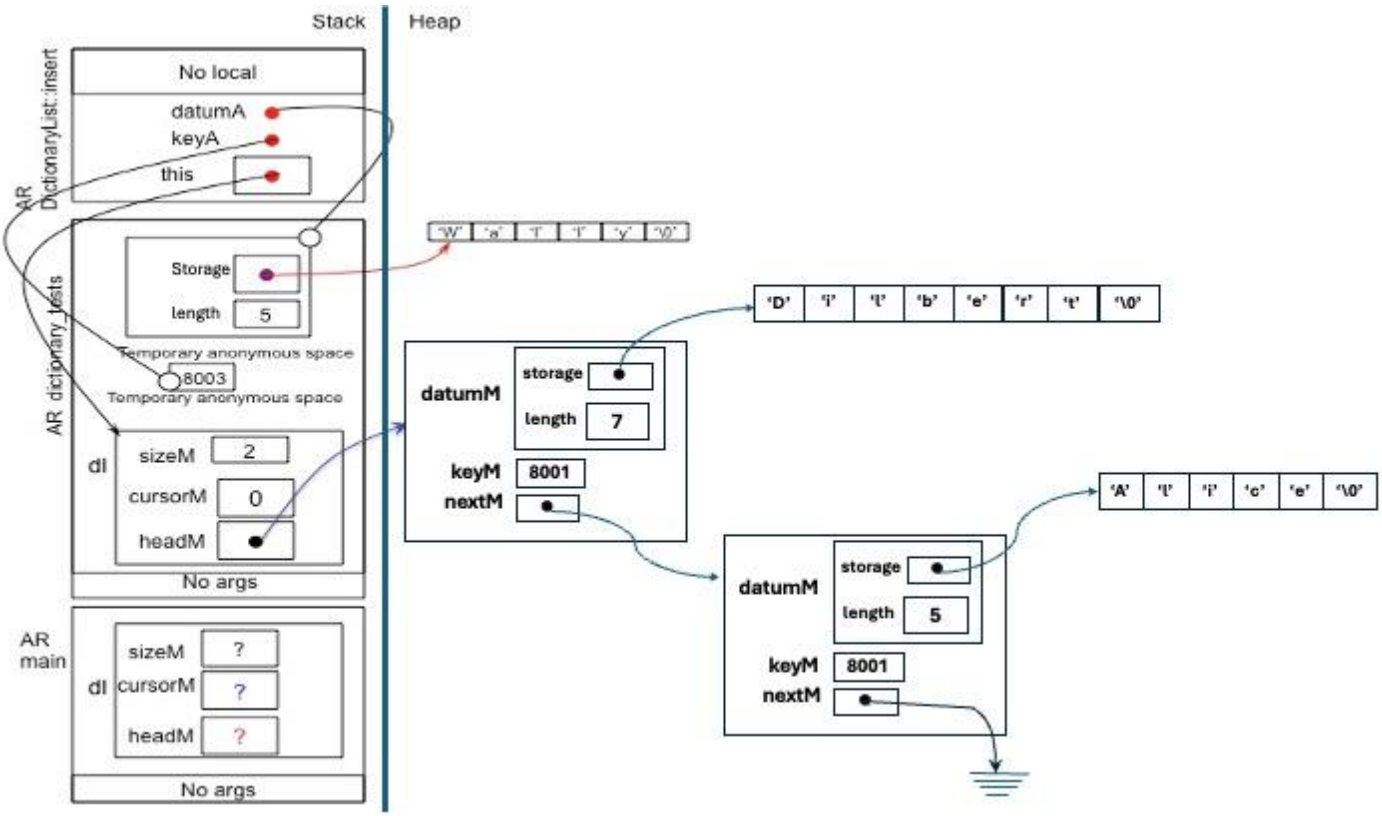
Typical operations for a Dictionary include: inserting a key with an associated datum, removing a key/datum pair by specifying a key, and searching for a pair by specifying a key. Dictionaries can be implemented using different data structure such as arrays, vectors, or linked lists. In this exercise a linked list implementation, called `DictionaryList` class is introduced. Class `DictionaryList`, 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:

Download files `dictionaryList.h`, `dictionaryList.cpp`, `my,` and `exBmain` from D2L. Read them carefully and try to understand what the functionalities of these classes are doing. Also notice that the definition of class `Node` contains three private data members: `keyM`, `datumM`, and `nextM`, a pointer of type `Node`. Additionally, class `Node` declares class `DictionaryList` as a `friend`, which means class `Node` gives the privilege of access to its data members to the class `DictionaryList`.

The idea of implementing a dictionary as a linked list is simple: each node contains a key, a datum, a pointer to the next node, and another node pointer called cursor that can either point to any node or can be set to NULL. For better understanding of the details of `DictionaryList`, try to understand how the function `insert`, and `remove` works. Further details about the operation of inserting a node into the list, can be learned from following activation record (AR) diagram, which represents a memory diagram, when CPU control reaches **point one** in this function, **for the second time**.

Please notice that in the following diagram we have assumed that a string object (string class from C++ library), contains a pointer called **storage** that points to an array of characters, representing a C-string, and another integer data member called **length**, that stores the length of the storage.



Once you understood how function insert works, compile the files that you downloaded from D2L and try to run the program. You will notice that the program will not work properly, as some functions are missing. Here is the program output that appears on the screen:

```
Printing table just after its creation ...
  Table is EMPTY.

Printing table after inserting 3 new keys ...
  8001  Dilbert
  8002  Alice
  8003  Wally

Printing table after removing two keys and inserting PointyHair ...
  8003  Wally
  8004  PointyHair

Printing table after changing data for one of the keys ...
  8003  Sam
  8004  PointyHair

Printing table after inserting 2 more keys ...
  8001  Allen
  8002  Peter
  8003  Sam
  8004  PointyHair
***----Finished dictionary tests----------------------------***
```

The function definitions that are not properly implemented are: `operator =`, `copy constructor`, `destructor`, `find`, `make_empty`. Your job in this exercise is to properly implement them. For this purpose, take the following steps:

1. In the main function change the `#if 0` to `#if 1` allowing function `test_copying` to be compiled and included in the program.
2. Complete the implementations of `destructor, copy constructor` and `assignment operator`, and `make_empty`, for class `DictionaryList`.
3. If step 2 is successfully completed and the above-mentioned functions are all fully functional, now change the `#if 0` to `#if 1` allowing function `test_finding` to be compiled and included in the program.
4. Complete the definition of function find.

What to Submit:

- As part of your lab report (PDF file), submit the copy of your source code, and the output of your program.
- Submit a zipped file that contains your actual source code (all .cpp and .h files)

Exercise C:

The purpose of this exercise is to help you understanding and practicing the application of four pillars of object-oriented programming, which includes:

- Abstraction
- Encapsulation and Information hiding
- Modularity
- Hierarchy

What to Do:

Assume the following definition of class, `Company`, belongs to only a portion of software application. Now, as a group of software developers, you have decided to refactor this code to make better and to comply with four principle and pillar of a good OOP.

Here is the existing code:

```
// company.cpp
#include <string>
#include <vector>
using namespace std;

struct Company{
    string companyName;
    string companyAdderss;

    vector<string> employees;       // vector of information about employees' information
                                    // (name, address, date of birth)
    string dateEstablished;         // the date that company was established

    vector <string> employeeState;  // information about employees' current states
                                    //(active, suspended, retired, fired)
    vector <string> customers;      // vector of information about customers
                                    // (name, address, phone)
};
```

This C++ program compiles with no syntax error, but even in the first glance, you can find several design issues that needs to be fixed. Although you don't need to compile and run this program, but if you want to do so, you need to write your own main function and then test it.

Notes:

- In your refactored code you should not use any C++ library classes other than `vector` and `string`.
- For simplicity purposes in this exercise, you don't need to worry about member functions and their implementation, at all. **Put your focus only on the data members of the class**.

What to Submit:
This exercise will not be marked, and you don't need to submit anything.

Exercise D:

In this exercise you should download the file `human_program.cpp` from D2L. In this file there is a program with two classes: class `Point,` and class `Human`. The program compiles and runs with no compilation or runtime error. However:

1. There are serious design flaws in the definition of class `Human`.
2. There are several bad style C++ programming issues in this code that doesn't protect the data members in classes `Point,` and `Human`.
3. Another issue with this program is that the definition of the classes is kept in single .cpp file. In general, in a properly designed program we should keep the outside view of the program (class definitions) separate from its implementation. Please make sure this concept is properly considered.
4. There might be serious logical issue with this code such as: bad memory management.

Your task in this exercise is to refactor the entire definition of the classes `Point,` and `Human`. Refactoring means you need to rewrite the program and apply every possible modification to make this program better in terms of **design, style, and documentation**.

Note:
You are NOT allowed to make any changes to the data member of any of the given classes or their types. Please only focus on the above-mentioned design issues or flaw.

What to Submit:
Submit the copy of your refactored .cpp and .h files, as part of your lab report in PDF format.