University of Calgary
Department of Electrical and Computer Engineering
Principles of Software Design (ENSF 480)
Lab 1– September 14, 2018

Instructor: M. Moussavi

Due Dates: Friday Sept 21, at 2:00 PM.

Objectives:

The purpose of this lab is:

1. A quick review of basic C++ concepts, learned in the previous course, ENCM 339. If you need to refresh your memory about those C++ topics, please study the set of slides on the D2L called: "`00-Quick Review of Basics Learned in ENCM339`".
2. Understand the concepts such as `friend` and overloading operators in C++.

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:
Course Name: Principles of Software Design
Course Code: ENSF 480
Assignment Number: For example, Lab-1
Submission Date and Time: DD/MM/YYYY

Marking scheme:

The total mark for the exercises in this lab is: **42 marks**

- Exercise A (12 marks)
- Exercise B (14 marks)
- Exercise C (16 marks)

Note: 20% marks will be deducted from the assignments handed in up to 24 hours after each due date. It means if your mark is X out of Y, you will only gain 0.8 times X. There will be no credit for assignments turned in later than 24 hours after the due dates; they will be returned unmarked.

Exercise A (12 marks) – Review of C++ Fundamentals Learned in ENCM 339

The purpose of this exercise is to refresh your memory about some of the basic constructs of C++ classes that you have learned in ENCM 339.

What to do:

Create a directory called lab8, and under this directory create another directory called `exA`. Then, follow the steps below:
Step1. Copy files `mystring.h`, `mystring.cpp` and `exAmain.cpp` from D2L in your directory, exA.

Step 2. 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. Using your favorite text editor 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 have never taken the previous course, ENCM 339, or if you have taken this course long time ago and do not remember about details of AR notations for C/C++ please ask for help immediately.

Step 6. If you compile and run this program you will see several lines of output as shown in the left side of the following table. On the right side of this table, explain why and at which point in the file `exAmain.cpp`, these outputs will be produced. Your explanation must be clear and backed with some reasoning.

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)**
constructor with int argument is called.	it is called at line 12 in exAmain when: Mystring c = 3. Constructor `Mystring::Mystring(int n)` is called
default constructor is called. **default constructor is called.**	
constructor with char* argument is called.	
copy constructor is called. **copy constructor is called.**	
destructor is called. **destructor is called.**	
copy constructor is called.	
assignment operator called.	
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.**	
constructor with char* argument is called.	
Program terminated successfully.	
destructor is called. **destructor is called**	

What to Submit:

Submit you AR diagrams, and the printout of the above table.

Exercise B (14 marks):

Part I – Drawing an AR Diagram for a Dictionary Data Structure (5 marks):

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".)

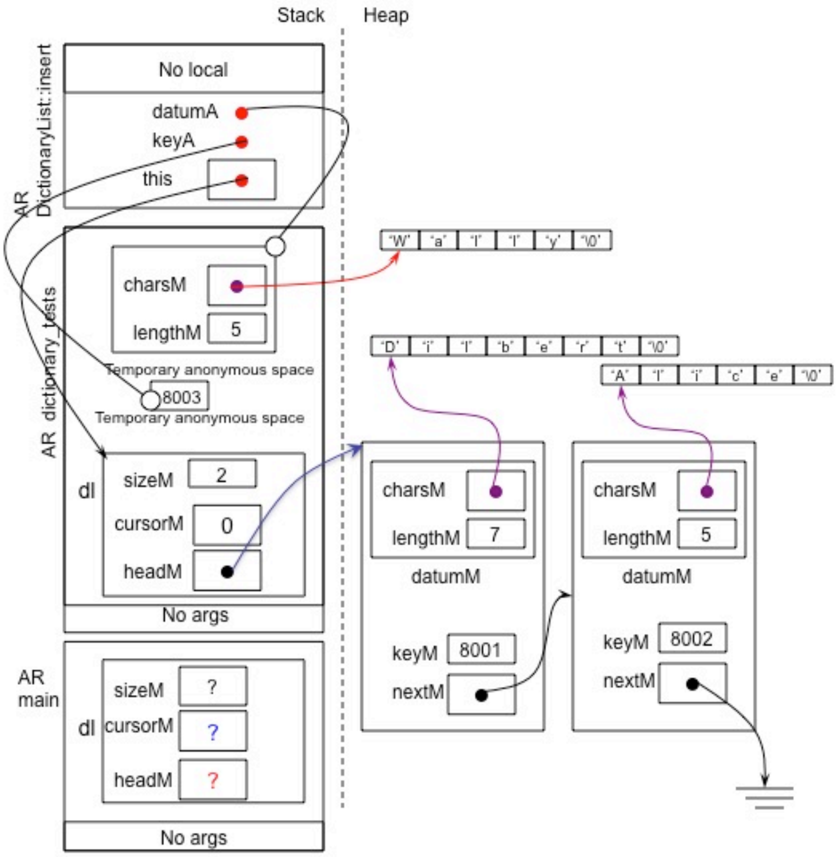
Typical operations for a Dictionary include: inserting a key with an associated datum, removing a key/datum pair by specifying a key, and searching for a pair by specifying a key. Dictionaries can be implemented using different data structure such as arrays, vectors, or linked lists. In this exercise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

Copy files `dictionaryList.h`, `dictionaryList.cpp`, `mystring_B.h`, `mystring_B.cpp`, and `exBmain` from D2L. Read them carefully and try to understand what the functionalities of these classes are doing. Note also the definition of class `Node` that contains three private data members: `keyM`, `datumM`, and `nextM`, a pointer of type `Node`. Also, class `Node` declares class `DictionaryList` as a `friend`. For details about `friend` keyword in C++, please refer to your lecture notes.

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` works. Further details about the operation of inserting a node into the list, can be learned from following AR diagram, which represents **point one** in this function, when reaches this point for the second time:



Now that you know how class `DictionaryList` and its member function insert work, draw an AR diagram for **point two** in function remove, when the program reaches this point for the first time.

What to Submit:

*Submit you AR diagram by providing a neat **pdf** document and submit it electronically on the D2L. Poor and unclear images of the diagrams are not acceptable and they may be returned unmarked.*

Part II – Defining Missing Functions (9 marks)

If you compile and run the files that you downloaded from D2L in part one of this exercise, you will notice that the program will not do very much, because the definitions of three member functions are not properly implemented. Here is all you will see as a screen output of the list just after its creation:

```
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---------------------------***

DictionaryList::copy is not implemented properly, so the program is calling exit.
```

The function definitions that are not properly implemented are: `find`, `copy`, and `destroy`. Your job in this part of the exercise B is to properly implement them.

Test your function definitions and note that you should develop incrementally--implement and test `find` before you start working on `copy`, and finish `copy` before starting `destroy`. To test your function `find`, you have to first uncomment the call to the function `test_finding` in `exBmain.cpp`.

What to Submit:

Please see the instructions at the end of this document.

Exercise C - Overloading Operators in C++ (16 marks)

In this exercise you are going to use the same files that you have used in exercise B, and you will overload several operators, for classes Mystring, and DictionaryList.

What to Do:

Open the file `exBmain.cpp` and uncomment the line that calls function `test_overloading`, and its prototype at the top of the file. Then, change the conditional compilation right above the implementation of `test_overloading` from `#if 0` to `#if 1` and try to compile or run the program. Now you will see a few errors. These errors are due to the fact that this function is trying to make some operations on Mystring or DictionaryList objects using common C++ relational operators such as, >, <, <=, !=, or other type of operators such as << or [] that are not by default defined by C++ compiler for objects of Mystring or DictionaryList. Your job in this assignment is to find out which operator is required to be overloaded and write the necessary code in `mystring_B.h, mystring_B.cpp, dictionalyList.h` and `dictionaryList.cpp`. For example, one of the lines in `exBmain.cpp` is:

```
if(dl.cursor_datum() >= (dl2.cursor_datum()))
    cout << endl << dl.cursor_datum() << " is greater than or equal " <<
dl2.cursor_datum();
```

In the above if-clause, the binary operator >= is used to compare two `Mystring` objects and find out which one is greater than the other one (lexicographically). Then it tries to print the value of string on the screen using C++ insertion operator <<.

Without overloading the operators >= , and << for class Mystring, this line will show errors such as: `Invalid operands to binary expression Mystring`

Since there are several operators to be overloaded, again the best strategy is to work incrementally. Means first comment out most of the lines in function terst_overloading, except the two lines that uses the operator >= and << . Then write the necessary code in both mystring.h and mystring.cpp files. If they work fine, then uncomment the next few lines to implement and test the next operator -- until all required operators are properly defined and tested.

Instructions to Submit Exercises B and C:

For exercises B and C submit the following files electronically on the D2L:

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