ENSF 337: Programming Fundamentals
Lab 8 Assignments
Fall 2022

Department of Electrical & Computer Engineering
University of Calgary
M. Moussavi
This lab contains material written by S. Norman, and M. Moussavi for an earlier version of this course

Important Notes:

- This is an **individual lab,** and you are not allowed to work with a partner.
- In exercise C in addition to your regular lab report in PDF format you should create a zip file that contains all your actual source code (.cpp and .h files).

Objective:

Understanding the details operations on linked list, and designing and writing a complete program that uses Linked List.

Marking scheme:

The total mark for the exercises in this lab is: **46 marks**
Exercise A - 12 marks
Exercise B – 12 marks
Exercise C – 22 marks

Due Date: Friday Nov 25, before 9:30 AM.

Exercise A: Tracing linked list code

What to Do

Download the files `OOList.cpp, OLList.h` and `lab8ExA.cpp` from D2L. Read the files carefully. Points one and two have been marked in the definition of `OLList::insert` and point three in function `OLList::remove`. Draw memory diagram when program reached these points **for the first time**.

Submit your diagrams. Please provide a digital copy of your diagram using a scanner and post it as part of your lab report on the D2L. Also make sure the scanned copy of your diagram is clear and readable.

Exercise B:

Part I:

Download files `OLList.cpp, OLList.h ,` if you haven't done yet. In the file `OLList.cpp,` you will notice there is a member function called `OLList::copy`, which is a helper for both the copy constructor and the assignment operator. Its job is to generate a new chain of nodes with items identical to the items in an existing `OLList` object. In this part you should rewrite the definition of `OLList::copy` so that it does what it is supposed to do. Do not call `insert` from your `copy` function; instead allocate nodes using `new`, and manipulate pointer variables to create links. There are two reasons to avoid `insert`:

1. Use of `insert` in this situation is inefficient--each insertion causes an unnecessary list traversal.
2. You need practice manipulating pointers in linked lists.

Now download file `lab8ExB.cpp` from D2L. This file contains a main function to be used for testing your exercise A. Change the first preprocess directive from `#if 0` to `#if 1`. Then compile and run the program to test if your `OLList::copy` function works.

Part Two:

The given `OLList::remove` function also doesn't do the right job to remove some nodes (only removes the nodes located at the beginning of the list properly). There is a missing code segment that you should add.

Now, in the file l`ab8ExB.cpp` change the second preprocess directive from `#if 0` to `#if 1`. Then compile and run the program to test if your `OLList::remove` function works.

Part Three:

The `OLList::destroy` function is a helper for both the copy constructor and the assignment operator. Its job is to remove the entire set of nodes in a liked list. In other words, it should delete the entire nodes in an existing linked list, one by one, using a loop.
The given `OLList::destroy` function doesn't do the right job to remove dynamically allocated nodes of a linked list. In fact the given code for the `OLList::destroy` function causes a memory leak. Your job is to rewrite the definition of `OLList::destroy` so that it does what it is supposed to do.

What to submit: `OLList.cpp`, and the program output as part of your lab report.

Exercise C: A Complete Program - Designing a Linked List Application

In this exercise you are going to analyze, design, and develop a program that reads river flow data stored in a text file, and creates a linked list that represent a database for the rivers annual records, and allows some statistical analysis.

Read this First

Basically, the techniques for file input and output in C++ are virtually identical to the standard input /output when reading from keyboard and writing to the screen. The purpose of this section is to briefly explain the file I/O in C++ (reading from and writing into text files). This section only reviews the basics of file I/O that you need to complete this exercise.

To open a text file for input (reading data), you need to take the following steps:

First, you must include a header called *fstream*:

```
#include <fstream>
using namespace std;
```

Second, you have to create an object of a class called *`ifstream`*. Here is an example of creating an `ifstream` object:

```
ifstream  inObj;
```

Third, to read data from a file called *mydata.txt*, you need to open the file. To open a file located in the current working directory, you write:

```
inObj.open ("mydata.txt");
```

You can put a complete file path between double quotation marks.

Fourth step is to use *inObj* exactly like *cin* to read data from the text file *mydata.txt*. For example, to read two integer number and store them in integer variables a, and b, you can write:

```
inObj >> a >> b;
```

You can also use functions such as `inObj.eof()` to check for end of file. This function returns true if process of reading encounters an end of file.

You need to take similar steps to open a text file for output (writing data). First, you must have included the header file, fstream.h (the same file that is also required for input). Then, you need to create an object of a class called *ofstream*, and to open the output file. See the following example:

```
ofstream    outObj;
outObj.open("myoutput.txt);
```

Now, you can use *outObj*, similar to *cout*, to write any data into *myoutput.txt*. For example, you can write the values of two integers *a* and *b* into *myoutput.txt*.

```
outObj   << setw (10) << a << setw (15) << b << endl;
```

Although all opened files will be automatically closed, when the C++ programs terminate, it is always a good practice to close them manually, whenever you don't need them anymore:

```
inObj.close();
outObj.close();
```

Here is a complete example of a C++ program that opens a file for writing, writes two integers into that file and then closes the file.

```
#include <iostream.h>
#include <fstream.h>

int main() {
   char* infile = " /usr/mydirectory/myoutput.txt";
   int a = 2543,  b = 465;

   // Create an ofstream object named outObj and open the target file
   ofstream outObj ;
   outObj.open(infile)

   // Make sure if file was opened successfully
   if (! outObj)  {
        cout << "Error: cannot open the file << filename << endl;
        exit(1);
   }

   // store values of two integers,  a, and b, in the file
   outObj <<setw(15) << a << setw(15) << b << endl;

   outObj.close();
   return 0;
}
```

Note: If you want to run this program under the Microsoft Windows environment, and for example your file is located in the directory: c:\mydirectory, you have to assign the file path to infile, as follows:

```
char* infile = "c:\\mydirectory\\myoutput.txt"
```

Any backslash must be preceded with another backslash.

When you open a file for writing, there are two common file open modes, "truncate" and "append". By default, if no mode is specified and the file exists, it will be truncated (its data will be lost).

To append data (add the data to the end of file), you can open the file in append mode:

```
OutObj.open("/usr/mydirectory/myoutput.txt", ios::append);
```

Further details about different file open modes, and file types such as binary files are deferred to future lectures or lab instructions.

Read This Second – What is `switch` Statment

Another option to implement a multi-way selection structure in C and C++ is a `switch` statement. The general syntax for a switch statement:

```
switch(expression)
{
   case conatant-1:
   statement(s);
   break;  // optional
   case constant-2:
   statement(s);
   break;  // optional
   case constant-3:
   statement(s);
   break;  // optional
   default:
   statement(s);
}
```

More details about `switch`:

- **Expression** must be either a variable or expression that results in an integral type (or enumerated type that we have not discussed it in this course, so far). For example. if variable n is an integer any of the following expression are valid: `n, n*2, n +2, …`
- Each case must be followed by numeric or character constant such as: `1, 2, 3, 'A', 'B', 'C'`
- Each code block within a case statement can optionally have a break statement. The break statement terminates the witch and program's flow of control jumps to the line after the switch statement. Without a break statement the program's flow of control goes from current case to the next case.
- The **`default`** case works like an else statement withing if … else if … else statement. Means if none of the cases happen the default case will execute its statements, then switch terminates.

Here is a flowchart that shows how switch statement works:

```
int main () {
   int n = 2;

   switch(n) {
      case 1 :
         cout << "n is 1\n";
         break;
```

```
      case 2 :
         cout << "n is 2\n";
      case 3 :
         cout << "n is 3\n";
         break;
      case 4 :
         cout << "n is 4\n" );
         break;
      default :
         cout << "n is a number other than: 1, 2, 3, 4\n";
   }

   return 0;
}
```

Since the value of n is 2, the flow of control enters casse 2. However, since it doesn't have a break statement, it continues to the case 3 and prints:

```
n is 2
n is 3
```

What to Do

Assume an engineering firm that works on hydropower plants has asked you to design and write a program in C++ to help them to study the variation of annual water flow in a river of their interest. There should be only one value of flow for each year (no duplications). Here is an example of the format of the records to be used:

Year	Flow (in billions of cubic meters)
1961	322.7
1963	321.5

Your program should be able to calculate the average flow in the list, and also to be able to add new data to the list, or remove data from the list. An example of the sequence of operations that your program must perform is explained in the rest of this section:

The program starts with displaying a title and brief information about the program. For example:

```
Program: Flow Studies – Fall 2020
Version: 1.0
Lab section: B??
Produced by: Your name(s)

<<< Press Enter to Continue>>>>
```

Then it should read its input from a text file called *flow.txt* (posted on the D2L), and create a linked list of the data (year, flow). **You are not allowed to use arrays to store these data**. The program should be written in several modules (a collection of `.cpp` and `.h` file, is called a module module)

First Module:

The program should have a module that contains a header file called `list.h` and `list.cpp`. This module will contain two structures and one class as follows:

1. Structure called ListItem that maintains an annual flow record (year and flow):

```
struct  ListItem {
```

```
        int year;
        double flow;
};
```

2. Structure called Node that contains the data (an instance of ListItem) and a pointer to Node:

```
struct Node {
        ListItem item;
        Node *next;
};
```

3. A class called `FlowList` that holds and manages a set of nodes that contains `ListItems`. This class can be similar to class `OLList`, in the previous exercise, that keeps the data in each node in order of years. You may need to add more member functions as needed. For this purpose you can also add another data member of type `Node*`, to be able to set it to different nodes when traversing through the list and have access to the item in the node (this is just a suggestion and you can come up with different solutions as you like).

Second Module:

Your program should have a second module that contains two files (`hydro.cpp, and hydro.h`). Some of the required functions to be defined in this module include:

- `main` that creates and uses the objects of `FlowList`.
- `displayHeader` - that display the introduction screen:

```
Program: Flow Studies – Fall 2020
Version: 1.0
Lab section: B??
Produced by: Your name
<<< Press Enter to Continue>>>>
```

- Function `readData` - that reads records years and flows from input file (`flow.txt`), inserts them into the list, and returns the number of records in the file.
- Function `menu` - that displays a menu and returns the user's choice (an integer 1 to 5).
- Function `display` - that displays years and flows, and shows the **average** of the flows in the list (calling function `averge`).
- Function `addData` - that prompts the user to enter new data, inserts the data into the linked list, and updates the number of records.
- Function `removeData` – that prompts the user to indicate what year to be removed, removes a single record from the list, and updates the number of records.
- Function `average` – that returns the flow average in the given list
- Function `saveData` - that opens the flow.txt file for writing and writes the contents of the linked list (annual flow records) into the file.
- Function `pressEnter` - that displays `<<<Press Enter to Continue>>>`, and waits for the user to press the <Return Key> . You can implement this function by printing the message followed by a call to function cin.get() that works similar to functions fgetc() or getc()in C. Here is the statements needed:

```
cout << "\n<<< Press Enter to Continue>>>>\n";
cin.get();
```

Note: These functions are all global functions like main (), not a class member function of a class.

Samples run of the program:

To give you a better idea that how the program is supposed to work, here are some screenshots of the sample run of the program. The program starts with displaying a title and brief information about the program. For example:

```
Program: Flow Studies, Fall 2020
Version: 1.0
Lab section: Your lab section
Produced by: Your name

<<< Press Enter to Continue>>>>
```

When user presses <Enter>, the program opens an input file called *flow.txt*. The program should give an error message and terminate, if for some reason it cannot open the file.
If the file exits it reads the data (years and flows) and insert the data into the linked list in ascending order, based on the flow years. **Then, closes the file**, and displays the following menu:

```
Please select on the following operations
   1. Display flow list, and the average.
   2. Add data.
   3. Save data into the file
   4. Remove data
   5. Quit
   Enter your choice (1, 2, 3, 4, of 5):
```

If user enters 1: the program displays the content of the list on the screen in the ascending order of years. For example:

```
Year       Flow (in billions of cubic meters)
1963        321.5
1964        222.7
2002        100.0
2003         99.5
…

The annual average of the flow is: ... billions of cubic meter

<<< Press Enter to Continue>>>>
```

When user presses <Enter> the menu will be displayed again:

```
Please select on the following operations
   1. Display flow list, and the average
   2. Add data.
   3. Save data into the file
   4. Remove data
   5. Quit
   Enter your choice (1, 2, 3, 4, of 5):
```

When user selects 2, the program prompts the user to enter a year and then the flow. For example user can enter the following values (in bold):

```
Please enter a year: 1995
Please enter the flow: 102.99
```

If data for the same year doesn't exit, the program should insert the record in proper order (ascending order, based on year) in the list, and display the following message:

```
New record inserted successfully.
<<< Press Enter to Continue>>>>
```

If the year already exists in the list, the program displays the following message:

```
Error: duplicate data.
<<< Press Enter to Continue>>>>
```

If user selects 3 from the menu options: the program opens `flow.txt` again, but this time for writing into the file, by creating an *ofstream* object. Then, it writes the data (years and flow) into the file, closes the file, and displays the following messages on the screen.

```
Data are saved into the file.
<<< Press Enter to Continue>>>>
```

When user selects 4, the program prompts the user to enter the year that should be removed. Here is an example:

```
Please enter the year that you want to remove: 1995
```

If data exist, the program should remove the record from the list, then displays the following message:

```
Record was successfully removed.

<<< Press Enter to Continue>>>>
```

(Note: The program at this point doesn't rewrite the records into the data file)
If the requested year doesn't exist in the list, the program displays the following message:

```
Error: No such a data.
<<< Press Enter to Continue>>>>
```

If user selects 5: the program terminates, showing the following message:

```
Program terminated successfully.
```

Your main function should test and shows all the functionalities of your program. As illustrated in the following example a good option would be to use a C++ `switch` statement.

```
int main(void) {
  FlowList x;
  int numRecords;
  displayHeader();
  numRecords = readData(x);
  int quit =0;

  while(1)
  {
    switch(menu()){
                case 1:
                        // call display function;
                         // call pressEnter;
                        break;
                case 2:
                        // call addData function
                        // call pressEnter;
                        break;
                case 3:
                        // call saveData function;
                         // call pressEnter;
                        break;
                case 4:
                        // call removeData
                        // call presenter;
                        break;
                case 5:
                        cout << "\nProgram terminated!\n\n";
                        quit = 1;
                        break;
                default:
                        cout <<  "\nNot a valid input.\n";
                        // pressEnter();
    }
    if(quit == 1) break;
```

```
}
```

Normally you are expected that you have learned about `switch` statement in previous courses. However if you didn't, please study this topic in one of your C or C++ textbooks.

What to Submit:

1. *Your regular lab report in PDF that contains the copy of your source code (.cpp and .h file), plus a copy of your sample run of the program that shows your program works. Be sure to show all of the possible actions from the menu, including attempts at duplicate year entries and removing a non-existent year.*

2. *Provide a zip file called `lab8_EXC.zip`, that contains your actual source files: `list.h, list.cpp, hydro.h, hydro.cpp`, then submit your zip file on the D2L Dropbox.*