

ENSF 607 – Software Design and Architecture I

Winter 2020

Lab Assignment #6: Introduction to C++

Due Dates	
Lab Assignment	Submit electronically on D2L **before 11:59 AM on Friday March 27th**

This is an individual assignment.

The objective of this lab is to gain experience with:

1. C++ Classes and Constructs


The following rules apply to this lab and all other lab assignments in future:

1. Before submitting your lab reports, take a moment to make sure that you are handing in all the material that is required. If you forget to hand something in, that is your fault; you can't use `I forgot' as an excuse to hand in parts of the assignment late.
2. **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.


Lab (25 marks)

Exercise 1: Tracing LinkedList Code (5 marks)

Download the files in Exercise 1 directory.

Points one has been marked in the definition of OLList::remove. Draw a diagram when program reached this point for the first time.

Leave static storage out of your diagrams.

What to Submit for Exercise 1: Submit your AR diagram on D2L.

Exercise 2: Coding the OLList Functions

Read This First - Part One

The `OLList::copy` function 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.

What To Do

In the files downloaded for exercise 1, 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.

Write a main program file to demonstrate successful copying of lists. Demonstrate the following cases:

- copying of an empty list;
- copying of a list with only one item;
- copying of a list with many items;

In each case write code to show that the copies are independent of each other--if `a` and `b` are `OLList` objects, and `a` is a copy of `b`, insertions to and removals from `a` after the copying is done should not affect `b`.

Read This First - Part Two

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 for-loop or a while-loop.



What To Do

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.

Read This First - Part Three

What To Do

The given `OLList::remove` function 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.

What to submit: `OLList.cpp`, your main program file, and your program's output that shows member function copy and destroy work.

Exercise 3: Analysis, Design, and Implementation of 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 revivers annual records, and allows some statistic 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 refresh your memory about a 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.h*:

```
#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;

Notice that you can similarly use other functions such as fail(), eof(), similar to what you learned about *cin*, in the previous labs.

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 your file is for example 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.

What to Do

H-E-F is a Hydropower Engineering Firm that works on hydropower plants. This company has asked you to 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)

1963      321.5
1961      322.7
```

Your program should be able to calculate the average and the median flow in the list, and also 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:

```
H-E-F Hydropower Studies – Winter 2019
Program: Flow Studies
Version: 1.0
Lab section: B??
Produced by: Your name

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

Then it should read its input from a text file (*flow.txt*), and create a linked list of the data (year, flow). **You are not allowed to use arrays to store these data**.

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 contain ListItems. This class can be similar to class OLList, in the previous exercise, but it has some other member functions and one additional data member called `cursorM`, with following specifications:
   a. `cursorM` is a `Node` pointer that is set to `NULL` (`cursorM` is `OFF`), when an empty list is initially created.
   b. If `FlowList` is not empty (i.e., `headM != NULL`), can be set to the first node in the list, by calling the member function `reset`.
   c. If `cursorM` is not `OFF` (i.e., `cursorM != NULL`), it can be move forward to the next node, by calling the member function `forward`, and sets `cursorM` to `NULL` if it is already pointing to the last node.
   d. `cursorM` can be accessed by calling the member function `cursor`.
   e. You should be able to check if cursor is ON (pointing to a node, and is not equal to NULL), by calling the member `isOn`.
   f. When a new node is inserted into the list, or an existing node is removed from the list, `cursorM` will be set to `NULL`.
   g. You should be able to retrieve an item to which `cursorM` is attached by calling the member function, `getItem()`.

Here is the partial definition of class FlowList:

```
class FlowList {
public:

 // CONSTRUCTOR, DESTRUCTOR, ETC. GOES HERE

 ...

 const ListItem& getItem() const;
 // REQUIRES: cursorM != NULL.
 // PROMISES: returns the item to which cursorM is attached to.

 void reset();
 // PROMISES: cursorM is equal to headM.

 bool isOn() const;
 // PROMISES: returns true if cursorM != NULL, otherwise
 //           returns false

 const Node* cursor()const;
 // PROMISES: returns cursorM.
```


```
void forward();
// PROMISES: if cursorM != NULL, moves cursorM to the next.

void insert(const ListItem& itemA);
// PROMISES: A node with a copy of itemA is added in a way
//           that preserves the non-decreasing order of flows
//           in nodes. Then, sets cursorM to NULL.

void remove(int target_year);
 // PROMISES: If a node has an item matching the
 // target_year, list is unchanged and gives a clear message
 // that the target_year (e.g. 2000) Not Found. Otherwise
 // exactly one node with its item.year == itemA.year is
 // removed. Then, sets cursorM to NULL.

 int count ()const;
 // PROMISES: returns the number of nodes in the list.

 private:
 // always points to the first node in the list.
 Node *headM;

 // Initially is set to NULL, but it may point to any node.
 Node *cursorM;

 void  copy(const FlowList& source);

 void destroy();
  };
```

Second Module:

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

- `main`
- `displayHeader` - that display the introduction screen
- `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.
- `menu` - that displays a menu and returns the user's choice (an integer 1 to 7).
- `display` - that displays years and flows, and shows the **average** and the **median** of the flows in the list (calling functions `averge`, and `median`).
- `addData` - that prompts the user to enter new data, inserts the data into the linked list, and updates the number of records.
- `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.
- `average` – that returns the flow average in the given list
- `median` – that returns the median flow.
- `saveData` - that opens the flow.txt file for writing and writes the contents of the linked list (annual flow records) into the file.
- `pressEnter` - that displays <<<Press Enter to Continue>>>, and waits for the user to press <Return> .



Given Data File:

The data file to test you program called, *flow.txt*, can be downloaded from D2L.

Samples run of the program:

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

```
HEF Hydropower Studies – Winter 2019
Program: Flow Studies
Version: 1.0
Lab section: Your lab section
Instructor : Your lab section's instructor name
Teaching Assistant(s): The name of your marking-TA(s)
Produced by: Your name(s)

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

When user presses <Return>, 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 data. **Then, closes the file**, and displays the following menu:

```
Please select on the following operations
   1. Display flow list, average and median
   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

based on the flow data:

```
Year       Flow (in billions of cubic meters)
2003         99.5
2002        100.0
1964        222.7
1963        321.5
…
The annual average of the flow is: ... millions cubic meter
The median flow is:... millions cubic meter.

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

When user presses <Return> the menu will be displayed again;

```
 Please select on the following operations
1. Display flow list, average and median
   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:

```
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 flows) 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: 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 you want to be removed:

```
Pleas 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>>>>
```

(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. It should look like following partial code/pseudo-code (some code are missing)

```
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;
  }

   // Creating a copy of FlowList x called copy1
   FlowList copy1;
   copy1 = x;
   // In the following section call addData to add
   // following records (year, flow) to copy1:
   // 2012  459.99
   // 2013  2000.34

   // Use the following code to create a copy of copy1 called copy2
   // FlowList copy2 = copy1;

   // removing three records from copy2
   copy2.remove(1922);
   copy2.remove(2003);
   copy2.remove(2001);

// Display the values in three list copy 2, copy1, and x
   cout << "\n values in copy2 are: ";
   display(copy2);

   cout << "\n values in copy1 are: ";
   display(copy1);

   cout << "\n values in x are: ";
   display(x);
  return 0;
}
```

Submit your program files, and the programs output that shows your program work.

How to submit: Include all your files for the lab section in one folder, zip your folder and upload it in D2L before the deadline.