Course: ENSF 337 – Fall 2020

Lab #: Lab 8

Instructor: M. Moussavi

Student Name: Quentin Jennings

Lab Section: B03

Submission Date: 2020-11-25

Exercise A OLList.cpp:

```cpp
// OLList.cpp
// ENSF 337 Fall 2020 Lab 8 Exercise A

#include <iostream>
#include <stdlib.h>
using namespace std;
#include "OLList.h"

OLList::OLList()
: headM(0)
{
}

OLList::OLList(const OLList& source)
{
    copy(source);
}

OLList& OLList::operator =(const OLList& rhs)
{
    if (this != &rhs) {
        destroy();
        copy(rhs);
    }
    return *this;
}

OLList::~OLList()
{
    destroy();
}

void OLList::print() const
{
    cout << '[';
    if (headM != 0) {
        cout << ' ' << headM->item;
        for (const Node *p = headM->next; p != 0; p = p->next)
            cout << ", " << p->item;
    }
    cout << " ]\n";
}

```

```
void OLList::insert(const ListItem& itemA)
{
    Node *new_node = new Node;
    new_node->item = itemA;

    if (headM == 0 || itemA <= headM->item ) {
        new_node->next = headM;
        headM = new_node;
        // point one
    }
    else {
        Node *before = headM;       // will point to node in front of new node
        Node *after = headM->next; // will be 0 or point to node after new node
        while(after != 0 && itemA > after->item) {
            before = after;
            after = after->next;
        }
        new_node->next = after;
        before->next = new_node;
        // point two
    }
}

void OLList::remove(const ListItem& itemA)
{
    // if list is empty, do nothing
    if (headM == 0 || itemA < headM->item)
        return;

    Node *doomed_node = 0;

    if (itemA == headM->item) {
        doomed_node = headM;
        headM = headM->next;
    }
    else {
        Node *before = headM;
        Node *maybe_doomed = headM->next;
        while(maybe_doomed != 0 && itemA > maybe_doomed->item)
        {
            before = maybe_doomed;
            maybe_doomed = maybe_doomed->next;
            //if statement is the only added part
            if(maybe_doomed->item == itemA)
            {
                before->next = maybe_doomed->next;
                free(maybe_doomed);
                break;
            }
        }
        // point three
    }
}
```

```
97
98  void OLList::destroy()
99  {
100     //added code
101     Node* tempNext;
102     while(headM != NULL)
103     {
104         tempNext = headM->next;
105         free(headM);
106         headM = tempNext;
107     }
108 }
109
110 void OLList::copy(const OLList& source)
111 {
112     //added code
113     Node* cBefore = new Node;
114     headM = cBefore;
115     cBefore->item = source.headM->item;
116     Node* sCurrent = source.headM->next;
117
118     while(sCurrent != NULL)
119     {
120         Node* newNode = new Node;
121         cBefore->next = newNode;
122         newNode->item = sCurrent->item;
123         newNode->next = NULL;
124         cBefore = newNode;
125         sCurrent = sCurrent->next;
126     }
127 }
```

Exercise A Output:

```
List just after creation. expected to be [ ]
[ ]
the_list after some insertions. Expected to be: [ 99, 110, 120, 220, 330, 440, 5
50 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
testing for copying lists ...
other_list as a copy of the_list: expected to be [ 99, 110, 120, 220, 330, 440,
550 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
third_list as a copy of the_list: expected to be: [ 99, 110, 120, 220, 330, 440,
 550 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
testing for removing and chaining assignment operator...
the_ist after some removals: expected to be: [ 99, 110, 120, 220, 440 ]
[ 99, 110, 120, 220, 440 ]
printing other_list one more time: expected to be: [ 99, 110, 120, 220, 330, 440
, 550 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
printing third_list one more time: expected to be: [ 99, 110, 120, 220, 330, 440
, 550 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
chaining assignment operator ...
the_list after chaining assignment operator: expected to be: [ 99, 110, 120, 220
, 440 ]
[ 99, 110, 120, 220, 440 ]
other_list after chaining: expected to be: [ 99, 110, 120, 220, 440 ]
[ 99, 110, 120, 220, 440 ]
third_list after chaining: expected to be: [ 99, 110, 120, 220, 440 ]
[ 99, 110, 120, 220, 440 ]
```

Exercise B list.h:

```
//list.h
//ENSF 337 - Lab 8 Exercise B
//Quentin Jennings
#ifndef list_h
#define list_h

struct ListItem { //structure for the item in a node
   int year;
   double flow;
};

struct Node { //structure for a node in a linked list
   ListItem item;
   Node* next;
};

class FlowList { //class for the linked list
public:
   FlowList();
   void insert(const int srcYear, const double srcFlow);
   // A node will be created with a ListItem comprised of the input year and flow.
   // The new node will be inserted as to keep the order of the list by year.
   bool remove(const int srcYear);
   // The list will be traversed until a node with the year is found.
   // If said node exists, it is deleted and the before pointer is adjusted.
   // Returns true if node is removed, returns false otherwise.
   bool exists(const int srcYear)const;
   // Returns true if input number srcYear exists within the list.
   Node* getHead()const{return head;}
   // Getter function for head pointer
private:
   Node* head;
   // Pointer to the head of linked list
};
#endif
```

Exercise B list.cpp:

```
//list.cpp
//ENSF 337 - Lab 8 Exercise B
//Quentin Jennings
#include <iostream>
using namespace std;

#include <stdlib.h>
#include "list.h"

FlowList::FlowList(): head(0) {}

void FlowList::insert(const int srcYear, const double srcFlow)
{
    //creates the new node to be inserted
    Node* newNode = new Node;
    newNode->item.year = srcYear;
    newNode->item.flow = srcFlow;

    //special case needed for if the node is first - the node becomes the new head and the head
comes after it
    if(head == 0 || srcYear <= head->item.year)
    {
        newNode->next = head;
        head = newNode;
    }
    //otherwise finds where it needs to be inserted and adjusts next pointers
    else
    {
        Node *before = head; //the node before the new node
        Node *after = head->next; //the node after the new node
        while(after != 0 && srcYear > after->item.year) //traverses to needed location
        {
            before = after;
            after = after->next;
        }
        newNode->next = after;
        before->next = newNode;
    }
}
```

```
bool FlowList::remove(const int srcYear)
{
    //if the list is empty, do nothing
    if(head == 0 || srcYear < head->item.year)
    {
        return false;
    }

    Node *delete_this = 0;
    //if the head needs to be deleted
    if(srcYear == head->item.year)
    {
        delete_this = head;
        head = head->next;
    }
    //otherwise, checks the rest of the list and marks the node to be deleted if found
    else
    {
        Node *before = head;
        Node *current = head->next;
        while(current != 0 && srcYear > current->item.year)
        {
            before = current;
            current = current->next;
        }
        if(current != 0 && current->item.year == srcYear)
        {
            before->next = current->next;
            delete_this = current;
        }
    }

    if(delete_this==0)
        return false;
    else
    {
        free(delete_this);
        return true;
    }
}
```

```
bool FlowList::exists(const int srcYear)const
{
    //checks head first
    if(srcYear == head->item.year)
        return true;
    //then checks rest of list, similar to remove function
    else
    {
        Node* current = head->next;
        while(current != 0 && srcYear > current->item.year)
            current = current->next;
        if(current != 0 && current->item.year == srcYear)
            return true;
    }
    return false;
}
```

Exercise B hydro.h:

```
//hydro.h
//ENSF 337 - Lab 8 Exercise B
//Quentin Jennings
#include "list.h"
#ifndef hydro_h
#define hydro_h
void displayHeader();
//displays intro screen

int readData(FlowList& source);
//reads and records years/flows into the flow list, returns # of records

int menu();
//displays the menu, returns user's choice (1-5)

void display(const FlowList* source, int num);
//displays the years, flows, and average of the flows (calls average)

void addData(FlowList* source, int &numRecords);
//prompts the user to enter new data, inserts data into linked list, updates # of records

void removeData(FlowList* source, int &numRecords);
//prompts user to indicate a year to be removed, removes it from the list, updates # of records

double average(const FlowList* source, int num);
//returns average flow of given list

void saveData(const FlowList* source);
//opens flow.txt file for writing and writes contents of linked list into the file in same format

void pressEnter();
//prompts user to press enter to continue, uses cin.get() to stall

void clearCin();
//clears the cin stream

#endif
```

Exercise B hydro.cpp:

```
//hydro.cpp
//ENSF 337 - Lab 8 Exercise B
//Quentin Jennings

#include <fstream>
#include <stdlib.h>
#include <iomanip>
#include <iostream>
using namespace std;
#include "hydro.h"

#define VERSION "1.0"
#define LABSECTION "03"
#define PROGRAMNAME "Flow Studies - Fall 2020"
#define FILENAME "flow.txt"

int main(void)
{
    FlowList List;
    int numRecords;

    displayHeader();
    numRecords = readData(List);

    while(1)
    {
        switch(menu())
        {
            case 1: //displays flow list and avg
                display(&List, numRecords);
                break;
            case 2: //adds a data node to list
                addData(&List, numRecords);
                break;
            case 3: //saves data into file
                saveData(&List);
                break;
            case 4: //removes a data node from list
                removeData(&List, numRecords);
```

```
                break;
            case 5: //terminates program
                cout << "\nProgram terminated.\n";
                return 0;
            default: //invalid input
                cout << "\nError: Invalid input. (should be 1, 2, 3, 4, or 5)\n";
                break;
        }
        //clears the stream and waits for a response after each selection
        clearCin();
        pressEnter();
    }
}

void displayHeader()
{
    cout << "Program: " << PROGRAMNAME << endl;
    cout << "Version: " << VERSION << endl;
    cout << "Lab Section: B" << LABSECTION << endl;
    cout << "Produced By: Quentin Jennings" << endl;
    pressEnter();
}

void pressEnter()
{
    cout << "\n<<< Press Enter to Continue >>>\n";
    cin.get();
}

int readData(FlowList& source)
{
    ifstream inStream (FILENAME);
    if(inStream.fail())
    {
        cout << "Error: File " << FILENAME << " not found, could not read data. Closing
program.\n";
        exit(-1);
    }

    int yr;
    double fl;
```

```
    int num = 0;

    while(!inStream.eof())
    {
        inStream >> yr;
        inStream >> fl;
        source.insert(yr, fl);
        num++;
    }
    inStream.close();
    return num;
}

int menu()
{
    cout << "Please select one of the following operations:\n";
    cout << "1. Display flow list and the average flow.\n";
    cout << "2. Add data to the flow list.\n";
    cout << "3. Save data into file.\n";
    cout << "4. Remove data from the flow list.\n";
    cout << "5. Quit Program.\n";
    cout << "\nEnter your choice (1, 2, 3, 4, or 5):\n";

    int n = 0;
    cin >> n;
    return n;

}

void display(const FlowList* source, int num)
{
    cout << "Year:     Flow: (billions of cubic meters)\n";

    Node* current = source->getHead();
    while(current != 0)
    {
        cout << setw(10) << left << current->item.year << setiosflags(ios::fixed) <<
setprecision(2) << current->item.flow << endl;
        current = current->next;
    }
```

```
    cout << "\nThe annual average of the flow is: " << setiosflags(ios::fixed) << setprecision(2)
<< average(source, num) << endl;
}

double average(const FlowList* source, int num)
{
    double sum = 0;
    Node* current = source->getHead();

    while(current != 0)
    {
        sum += current->item.flow;
        current = current->next;
    }

    return sum / num;
}

void addData(FlowList* source, int &numRecords)
{
    int yr;
    double fl;

    cout << "\nPlease enter a year: ";
    cin >> yr;

    if(source->exists(yr))
    {
        cout  << "\nError: data already exists for the year " << yr << ".\n";
    }
    else
    {
        clearCin();
        cout << "Please enter the flow: ";
        cin >> fl;
        source->insert(yr, fl);
        numRecords++;
        cout << "\nNew data record inserted successfully.\n";
    }
}
```

```
void removeData(FlowList* source, int &numRecords)
{
    int yr;
    cout << "\nPlease enter a year: ";
    cin >> yr;

    if(source->remove(yr))
    {
        numRecords--;
        cout << "\nData node successfully removed.\n";
    }
    else
        cout << "\nData node not removed as no node was found for the year " << yr;



}

void saveData(const FlowList* source)
{
    ofstream outStream("flow.txt");
    if(outStream.fail())
    {
        cout << "Error: File output stream unsuccessful. Closing program.\n";
        exit(-1);
    }
    Node* current = source->getHead();
    while(current != 0)
    {
        outStream << setiosflags(ios::fixed) << setprecision(2) << current->item.year << "      "
<< current->item.flow << endl;
        current = current->next;
    }
    outStream.close();
    cout << "Data successfully saved into " << FILENAME << ".\n";
}

void clearCin()
{
    cin.clear();
    while((getchar()) != '\n');
}
```

Exercise B Program Outputs:

```
Program: Flow Studies - Fall 2020
Version: 1.0
Lab Section: B03
Produced By: Quentin Jennings

<<< Press Enter to Continue >>>

Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
1
Year:     Flow: (billions of cubic meters)
1900      220.11
1901      210.11
1922      192.99
1945      145.66
1946      300.99
1947      310.99
1970      100.34
1971      209.99
1972      219.99
1989      234.98
1990      214.98
1999      110.99
2000      110.22
2001      231.44
2002      211.44

The annual average of the flow is: 201.68

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

```
Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
2

Please enter a year: 1900

Error: data already exists for the year 1900.

<<< Press Enter to Continue >>>

Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
2

Please enter a year: 1971

Error: data already exists for the year 1971.

<<< Press Enter to Continue >>>

Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
2

Please enter a year: 2002

Error: data already exists for the year 2002.

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

```
Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
2

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

New data record inserted successfully.

<<< Press Enter to Continue >>>

Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
2

Please enter a year: 2020
Please enter the flow: 120.12

New data record inserted successfully.

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

```
Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
4

Please enter a year: 1800

Data node not removed as no node was found for the year 1800
<<< Press Enter to Continue >>>

Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
4

Please enter a year: 1950

Data node not removed as no node was found for the year 1950
<<< Press Enter to Continue >>>

Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
4

Please enter a year: 2100

Data node not removed as no node was found for the year 2100
<<< Press Enter to Continue >>>
```

```
Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
4

Please enter a year: 2002

Data node successfully removed.

<<< Press Enter to Continue >>>

Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
4

Please enter a year: 1970

Data node successfully removed.

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

```
Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
1
Year:     Flow: (billions of cubic meters)
1900      220.11
1901      210.11
1922      192.99
1945      145.66
1946      300.99
1947      310.99
1971      209.99
1972      219.99
1989      234.98
1990      214.98
1995      102.99
1999      110.99
2000      110.22
2001      231.44
2020      120.12

The annual average of the flow is: 195.77

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

```
Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
3
Data successfully saved into flow.txt.

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

```
Please select one of the following operations:
1. Display flow list and the average flow.
2. Add data to the flow list.
3. Save data into file.
4. Remove data from the flow list.
5. Quit Program.

Enter your choice (1, 2, 3, 4, or 5):
5

Program terminated.
```

flow.txt - Notepad

File Edit Format View Help

```
1900      220.11
1901      210.11
1922      192.99
1945      145.66
1946      300.99
1947      310.99
1971      209.99
1972      219.99
1989      234.98
1990      214.98
1995      102.99
1999      110.99
2000      110.22
2001      231.44
2020      120.12
```

100% Unix (LF) UTF-8