LAB ASSIGNMENT #8 REPORT

Course: ENSF 337 - Programming Fundamentals for Software and Computer

Lab #: 8

Instructor: Dr. Maan Khedr

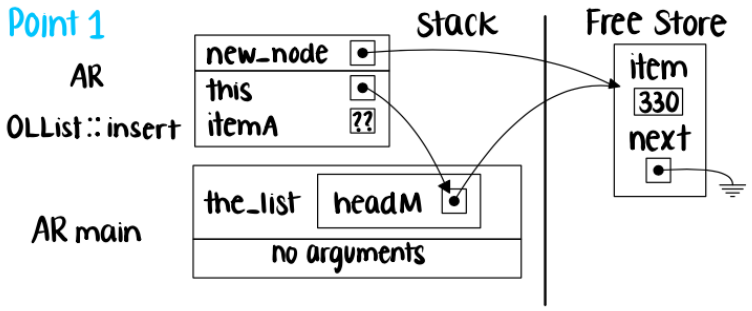
Student Name: Sadia Khandaker

Lab Section: B04

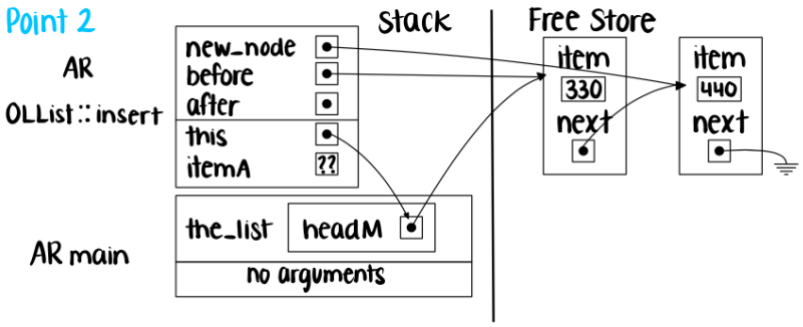
Submission Date: November 25, 2021

Exercise A: Tracing Linked List

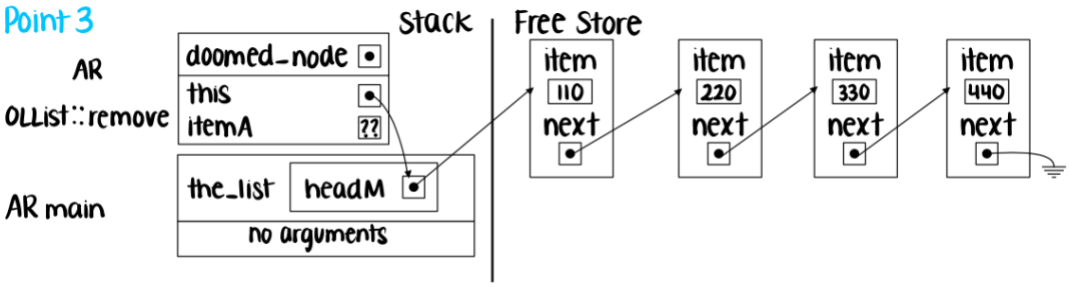
AR Diagram when program reaches Point 1 for the 1st time.



AR Diagram when program reaches Point 2 for the 1st time.



AR Diagram when program reaches Point 3 for the 1st time.



Exercise B: Tracing Linked List Code

Program that Performs a Variety of Operations on Linked Lists

```
/* File Name: OLList.cpp
 * Assignment: Lab 8 Exercise B
 * Lab Section: B04
 * Completed by: Sadia Khandaker
 * Submission Date: November 25, 2021
 */

#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 (headM == 0 || itemA < headM->item)
        return;

    Node *doomed_node = 0;

    if (itemA == headM->item) {
        doomed_node = headM;
        headM = headM->next;
        delete doomed_node;
    }
    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 (maybe_doomed != NULL) {
            before -> next = maybe_doomed -> next;
            delete maybe_doomed;
        }
    }
}

void OLList::destroy()
{
    Node* erase = headM;
    while(headM != NULL) {
        headM = headM ->next;
        delete headM;
        erase = headM;
    }
    headM = NULL;
}
```

```
void OLList::copy(const OLList& source)
{
    Node* current = source.headM;
    if (current != NULL)
    {
        Node *copy = new Node;
        copy->item = current->item;
        copy->next = NULL;
        headM = copy;
        current = source.headM->next;
        while (current != NULL)
        {
            Node *nn = new Node;
            nn->item = current->item;
            nn->next = NULL;
            copy->next = nn;
            copy = nn;
            current = current->next;
        }
    }
    else
    {
        headM = NULL;
    }
}
```

Program Output:

```
List just after creation. expected to be [ ]
[ ]
the_list after some insertions. Expected to be: [ 99, 110, 120, 220, 330, 440, 550 ]
[ 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 C: A Complete Program - Designing a Linked List Application

Program Development for Flow Rate Analysis

list.h

```
/* File Name: list.h
 * Assignment: Lab 8 Exercise C
 * Lab Section: B04
 * Completed By: Sadia Khandaker
 * Submission Date: November 25, 2021
 */
#ifndef LIST_H
#define LIST_H

struct ListItem {
    int year;
    double flow;
};
struct Node {
    ListItem item;
    Node *next;
};
class FlowList {
public:
    FlowList();
    ~FlowList();
    int insert(const ListItem& itemA);
    int remove(const ListItem& itemA);
    Node* headM;

private:
    void destroy();
};
#endif
```

list.cpp

```
/* File Name: list.cpp
 * Assignment: Lab 8 Exercise C
 * Lab Section: B04
 * Completed By: Sadia Khandaker
 * Submission Date: November 25, 2021
 */


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

FlowList::FlowList()
        : headM(0)
{
```

```
}
FlowList::~FlowList() {
    destroy();
}

int FlowList::insert(const ListItem& itemA) {
    Node* crsr = headM;
    while (crsr != 0) {
        if(crsr -> item.year == itemA.year) {
            return 0;
        }
        crsr = crsr -> next;
    }
    Node *aNode = new Node;
    aNode -> item.year = itemA.year;
    aNode -> item.flow = itemA.flow;
    if (headM == 0 || itemA.flow <= headM->item.flow) {
        aNode->next = headM;
        headM = aNode;
    }
    else {
        Node *before = headM;
        Node *after = headM->next;
        while( after != 0 && itemA.flow > after -> item.flow) {
            before = after;
            after = after -> next;
        }
            aNode -> next = after;
            before->next = aNode;
        }
        return 1;
    }


int FlowList::remove(const ListItem& itemA)
{
    if (headM == 0)
        return 0;

    Node *doomed_node = 0;

    if (itemA.year == headM->item.year) {

        doomed_node = headM;
        headM = headM->next;
    }
    else {
        Node *before = headM;
        Node *maybe_doomed = headM->next;

        while(maybe_doomed != nullptr && itemA.year != maybe_doomed-
>item.year) {
            before = maybe_doomed;
            maybe_doomed = maybe_doomed->next;
        }
        if (maybe_doomed != nullptr) {
            before -> next = maybe_doomed -> next;
```

```
            delete maybe_doomed;
            return 1;
        }
    }
    return 0;
}

void FlowList::destroy() {
    Node *before = headM;
    while (before != nullptr) {
        Node *temp = before->next;
        delete before;
        before = temp;
    }
    headM = nullptr;
}
```

hydro.h

```
/* File Name: hydro.h
 * Assignment: Lab 8 Exercise C
 * Lab Section: B04
 * Completed By: Sadia Khandaker
 * Submission Date: November 25, 2021
 */


#ifndef HYDRO_H
#define HYDRO_H
#include "list.h"

int main();
void displayHeader();
int readData(FlowList &x);
int menu();
void display(int n, FlowList &x);
int addData(FlowList &x, int &n);
int removeData(FlowList &x, int & n);
double average(int n, FlowList &x);
void saveData(FlowList &x, int n);
void pressEnter();

#endif
```

hydro.cpp

```
/* File Name: hydro.cpp
 * Assignment: Lab 8 Exercise C
 * Lab Section: B04
 * Completed By: Sadia Khandaker
 * Submission Date: November 25, 2021
 */
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "hydro.h"
using namespace std;
int main() {
```

```
    FlowList x;
    int numRecords
    int quit =0;
    displayHeader();
    numRecords = readData(x);
    while(true) {
       switch(menu()) {
           case 1:
               display(numRecords, x);
               pressEnter();
               break;
           case 2:
               addData(x, numRecords);
               pressEnter();
               break;
           case 3:
               saveData(x, numRecords);
               pressEnter();
               break;
           case 4:
               removeData(x, numRecords);
               pressEnter();
               break;
           case 5:
               cout << "\nProgram terminated!\n\n";
               quit = 1;
               break;
           default:
               cout << "\nNot a valid input.\n";
               pressEnter();
       }
       if(quit == 1) break;
    }
}

void displayHeader()
{
    cout<<"Program: Flow Studies - Fall 2021\n";
    cout<<"Version: 1.0\n";
    cout<<"Lab section: B04\n";
    cout<<"Produced by: Sadia Khandaker\n";
}

int readData(FlowList &x) {
    ifstream inObj;
    inObj.open("flow.txt");
    int year,i = 0;
    double flow;
    if(!inObj) {
        cout<<"Error: cannot open the file: flow.txt.";
        exit(1);
    }
    while(!inObj.eof()) {
        inObj >> year >> flow;
        ListItem temp = {year, flow};
        x.insert(temp);
        i++;
```

```
        }
    inObj.close();
    return i;

}

int menu(){
    cout<<"Please select on the following operations.\n";
    cout<<"1. Display flow list and the average\n";
    cout<<"2. Add data\n";
    cout<<"3. Save data into the file\n";
    cout<<"4. Remove data\n";
    cout<<"5. Quit.\n";
    cout<<"Enter your choice (1, 2, 3, 4, or 5):";
    int choice; cin>>choice;
    return choice;
}
void display(int n, FlowList &x) {
    Node* crsr = x.headM;
    cout << "Year    Flow(in billions of cubic meters)\n" ;
    for (int i = 0; i < n; i++) {
        cout << crsr -> item.year <<"             " << crsr ->
item.flow << endl;
        crsr = crsr ->next;
        }
    cout << "The annual average of the flow is: " << average(n,x) <<  "
billion cubic meters \n" ;
}

int addData(FlowList &x, int &n) {
    ListItem temp{};
    cout << "Please enter a year: "; cin >> temp.year;
    cout << "Please enter the flow: "; cin >> temp.flow;
    if ( x.insert(temp))
    {
        cout << "New record inserted successfully.";
        n++;
    }
    else
    {
        cout << "Error: duplicate data." ;
    }
    return 0;
}

int removeData(FlowList &x, int & n){
    int year;
    cout << "Please enter the year that you want to remove: " ; cin >>
year;
    ListItem temp = {year};
    if (x.remove(temp)) {
        cout << "Record was successfully removed.";
        return n--;
    }
    else {
        cout << "Error: No such a data.";
        return n;
```

```
    }
}

double average(int n, FlowList &x) {
    Node* head = x.headM;
    double sum = 0;
    int i = 0;
    while(head!=nullptr){
        sum += head->item.flow;
        i++;
        head = head->next;
    }
    return sum/n;
}

void saveData(FlowList &x, int n) {
    ofstream outObj;
    outObj.open("flow.txt");
    Node* head = x.headM;
    if(! outObj) {
        cout<<"\nError: cannot open the file."<<"flow.txt"<<endl;
        exit(1);
    }
    for(int i = 0; i < n; i++) {
        string text =  to_string(head->item.year) + " " +
to_string(head->item.flow);
        outObj<<text<<"\n";
        head = head -> next;
    }

    outObj.close();
    cout << "Data are saved into the file.";
    }

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

Program Output:

User Interface

```
Program: Flow Studies - Fall 2021
Version: 1.0
Lab section: B04
Produced by: Sadia Khandaker
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, or 5):
```

Menu 1

```
Enter your choice (1, 2, 3, 4, or 5):1
Year    Flow(in billions of cubic meters)
1970              100.34
2000              110.22
1999              110.99
1945              145.66
1922              192.99
1971              209.99
1901              210.11
2002              211.44
1990              214.98
1972              219.99
1900              220.11
2001              231.44
1989              234.98
1946              300.99
1947              310.99
The annual average of the flow is: 201.681 billion cubic meters
```

Menu 2

Outcome 1: Duplicate Input Value

```
Enter your choice (1, 2, 3, 4, of 5):2
Please enter a year: 2001
Please enter the flow: 423
Error: duplicate data.
<<< Press Enter to Continue >>>
```

Outcome 2: Unique Input Value

```
Enter your choice (1, 2, 3, 4, or 5):2
Please enter a year: 2016
Please enter the flow: 112.45
New record inserted successfully.
<<< Press Enter to Continue >>>
```

Menu 3

```
Enter your choice (1, 2, 3, 4, or 5):3
Data are saved into the file.
<<< Press Enter to Continue >>>
```

Menu 4

Outcome 1: Removing a value that's not on flow.txt

```
Enter your choice (1, 2, 3, 4, or 5):4
Please enter the year that you want to remove: 2035
Error: No such a data.
```

Outcome 2: Removing a value that's on flow.txt

```
Enter your choice (1, 2, 3, 4, or 5):4
Please enter the year that you want to remove: 2001
Record was successfully removed.
<<< Press Enter to Continue >>>
```

Menu 5

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

Program terminated!



...Program finished with exit code 0
Press ENTER to exit console.
```