**Course: Programming Fundamental – ENSF 337**

**Lab #: Lab 8**

**Instructor: M. Moussavi**

**Student Name: Michael Jeremy Olea**

**Lab Section: B01**

**Date submitted: Nov. 21, 2018**

**Exercise D;**

**//OLList.cpp**

#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;

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->item == itemA)

{

before->next = maybe\_doomed->next;

delete maybe\_doomed;

}

// point three

}

}

*void* OLList::destroy()

{

while(headM != nullptr)

{

Node\* prev\_node = headM;

headM = headM->next;

delete prev\_node;

}

headM = 0;

}

*void* OLList::copy(const OLList& source)

{

Node\* position = source.headM;

Node\* prev\_node = nullptr;

headM = nullptr;

while(position != nullptr)

{

Node\* new\_node = new Node;

new\_node->item = position->item;

new\_node->next = nullptr;

position = position->next;

if(headM == nullptr)

{

headM = new\_node;

prev\_node = new\_node;

}

else

{

prev\_node->next = new\_node;

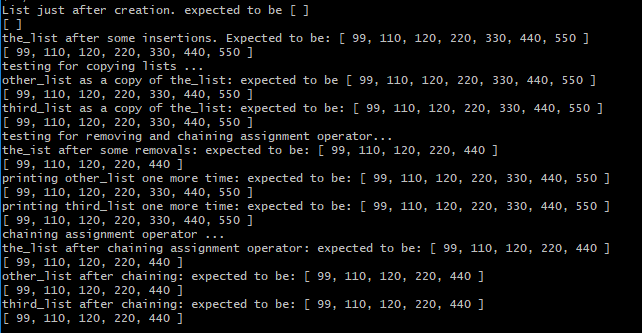
prev\_node = new\_node;

}

}

}

**Output: next page**



**Exercise E:**

**//List.h**

#ifndef list\_h

#define list\_h

*struct* ListItem {

*int* year;

*double* flow;

};

*struct* Node {

ListItem item;

Node \*next;

};

*class* FlowList {

public:

FlowList();

*void* add(*int* year, *double* flow);

*void* insert(const ListItem& itemA);

*void* print() const;

Node\* get\_headM() const;

*void* remove(*int* year);

private:

Node \*headM;

};

#endif

**//hydro.h**

#ifndef hydro\_h

#define hydro\_h

*void* displayHeader();

*int* readData(FlowList &list);

*int* menu();

*void* display(FlowList &list, *int* size);

*void* addData(FlowList &list, *int* &numRecords);

*void* removeData(FlowList &list);

*double* average(FlowList &list, *int* size);

*double* median(FlowList &list, *int* size);

*void* saveData(FlowList &list);

*void* pressEnter();

#endif

**//list.cpp**

#include <iostream>

#include <fstream>

#include <stdlib.h>

using *namespace* std;

#include "list.h"

#include "hydro.h"

FlowList::FlowList(): headM(0)

{

}

*void* FlowList::insert(const ListItem& itemA)

{

Node \*new\_node = new Node;

new\_node->item = itemA;

if (headM == 0 || itemA.flow <= headM->item.flow) {

new\_node->next = headM;

headM = new\_node;

}

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.flow > after->item.flow) {

before = after;

after = after->next;

}

new\_node->next = after;

before->next = new\_node;

}

}

*void* FlowList::print() const

{

cout << "Year \t Flow (in billions of cubic meters)\n";

if (headM != 0) {

cout << headM->item.year << " \t " << headM->item.flow;

for (const Node \*p = headM->next; p != 0; p = p->next)

cout << "\n" << p->item.year << " \t " << p->item.flow;

}

else

cout << "Linked-list is empty";

}

Node\* FlowList::get\_headM() const

{

return headM;

}

*void* FlowList::remove(const *int* year)

{

if(headM == 0)

return;

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

{

Node \*doomed = headM;

headM = headM->next;

delete doomed;

}

else {

Node \*before = headM;

Node \*maybe\_doomed = headM->next;

while(maybe\_doomed != 0 && maybe\_doomed->item.year != year) {

before = maybe\_doomed;

maybe\_doomed = maybe\_doomed->next;

}

if(maybe\_doomed->item.year == year)

{

before->next = maybe\_doomed->next;

delete maybe\_doomed;

}

}

return;

}

**//hydro.cpp**

#include <iostream>

#include <fstream>

using *namespace* std;

#define FILENAME "flow.txt"

#include "list.h"

#include "hydro.h"

#include <stdlib.h>

*int* main(*void*)

{

FlowList x;

*int* numRecords;

displayHeader();

numRecords = readData(x);

*int* quit = 0;

while(1)

{

switch(menu())

{

case 1:

display(x, numRecords);

pressEnter();

break;

case 2:

addData(x, numRecords);

pressEnter();

break;

case 3:

saveData(x);

pressEnter();

break;

case 4:

removeData(x);

pressEnter();

break;

case 5:

cout << "\nProgram terminated!\n\n";

quit = 1;

break;

default:

cout << "\nNot a valid input.\n";

}

if(quit == 1) break;

}

}

*void* displayHeader()

{

cout << "\nProgram: Flow Studies, Fall 2017";

cout << "\nVersion: 1.0";

cout << "\nLab Section: B01";

cout << "\nProduced by: Michael Jeremy Olea";

pressEnter();

}

*void* pressEnter()

{

cout << "\n<<<Press Enter to Continue>>>\n";

while(1)

{

if(cin.get() == '\n')

return;

}

}

*int* menu()

{

cout << "\nPlease select on the following operations\n";

cout << "\t1. Display flow list, average and median\n";

cout << "\t2. Add data.\n";

cout << "\t3. Save data into the file\n";

cout << "\t4. Remove data\n";

cout << "\t5. Quit\n";

cout << "Enter your choice (1, 2, 3, 4, or 5)\n";

*int* selected;

cin >> selected;

return selected;

}

*int* readData(FlowList &list)

{

*int* y;

*double* f;

*int* length = 0;

ifstream input;

input.open(FILENAME);

if(input.fail())

{

cout << "\nError: cannot open file 'flow.txt'\n";

exit(1);

}

while(!input.eof())

{

input >> y >> f;

length++;

ListItem \*data = new ListItem {y,f};

list.insert(\*data);

}

return length;

}

*void* display(FlowList &list, *int* size)

{

list.print();

cout << "\n\n";

cout << "the average is " << average(list, size) << " in billions of cubic meters\n";

cout << "the median is " << median(list, size) << " in billions of cubic meters";

}

*double* average(FlowList &list, *int* size)

{

Node\* select = list.get\_headM();

*double* sum = 0;

while(select)

{

sum += select->item.flow;

select = select->next;

}

return (sum/size);

}

*double* median(FlowList &list, *int* size)

{

Node\* med = list.get\_headM();

for(*int* i = 0; i < (size-1)/2; i++)

{

med = med->next;

}

return med->item.flow;

}

*void* addData(FlowList &list, *int* &numRecords)

{

Node \*check = list.get\_headM();

ListItem \*new\_item = new ListItem;

cout << "\nInput a year and a flow\n";

cin >> new\_item->year >> new\_item->flow;

if(cin.fail())

{

cout << "\nInvalid input\n";

return;

}

while(check)

{

if(check->item.year == new\_item->year)

{

cout << "\nError: Duplicate year\n";

return;

}

check = check->next;

}

numRecords++;

list.insert(\*new\_item);

cout << "\nData successfully added\n";

}

*void* saveData(FlowList &list)

{

ofstream output;

output.open(FILENAME);

if(output.fail())

{

cout << "\nError: cannot open file 'flow\_Output.txt'\n" << endl;

exit(1);

}

Node \*select = list.get\_headM();

while(select)

{

output << select->item.year << "\t" << select->item.flow;

output << "\n";

select = select->next;

}

output.close();

cout << "\nFile successfully saved\n";

}

*void* removeData(FlowList &list)

{

Node \*check = list.get\_headM();

*int* removeYear;

cout << "\nEnter year you want to remove\n";

cin >> removeYear;

if(cin.fail())

{

cout << "Invalid input";

return;

}

while(check)

{

if(check->item.year == removeYear)

{

list.remove(removeYear);

cout << "Year successfully removed";

return;

}

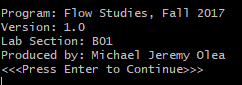
check = check->next;

}

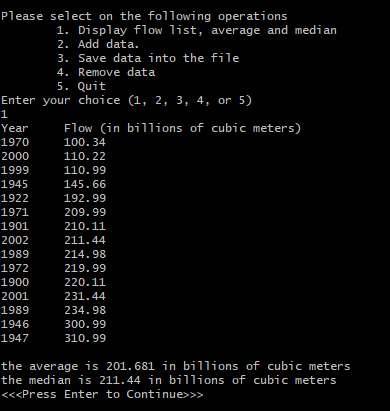
cout << "\nYear does not exist\n";

}

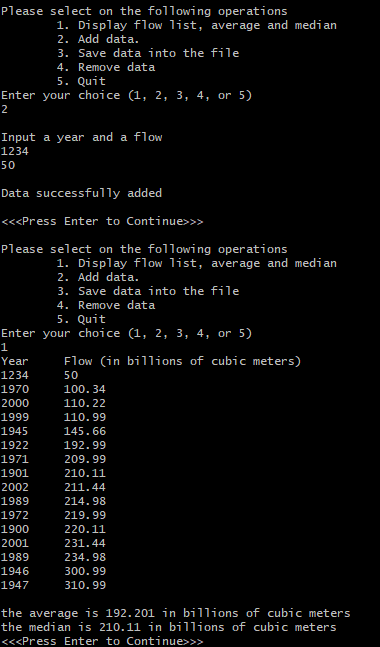
Output:



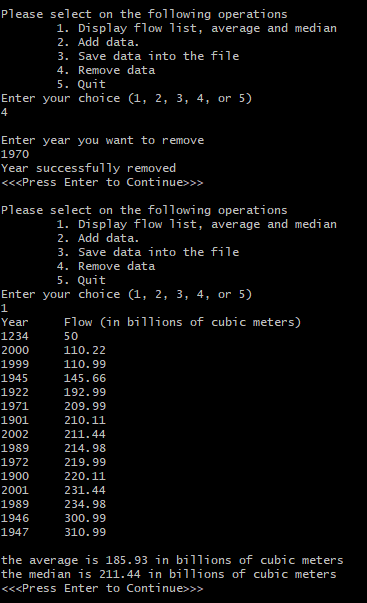
Enter 1 to display data



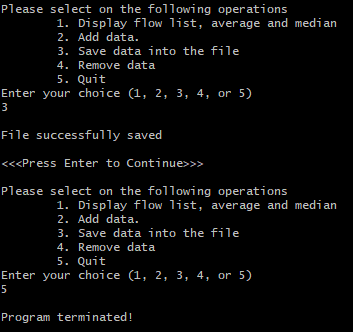
Enter 2 to add data, input a year and flow and display it to prove it works



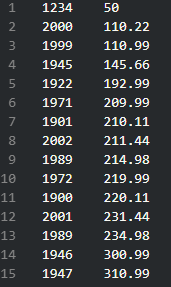
Enter 4 to remove data, remove year “1970” then display to prove it works



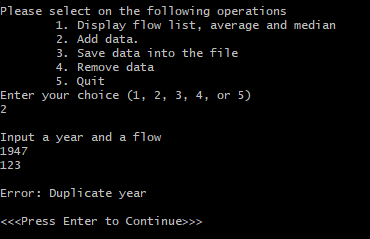
Enter 3 to save data, then press 5 to terminate



New “flow.txt” file



Enter 2 to add data and enter duplicate year to get error message



Enter 3 to remove data and enter non-existent year

