**Lab Description:**

\*Lab Description in README file

\*\*Not on lab write up

**Main.cpp:**

#include "payroll.h" // PayRoll object

#include "MyStack.h" // MyStack list

#include <iostream> // cout, endl

int main() {

std::cout << "Testing for Lab-5\n";

std::cout << "Test with PayRoll Objects: " << std::endl; // PayRoll tests

MyStack<PayRoll> m;

PayRoll p1("Ryan", 25, 52);

PayRoll p2("Skylar", 30, 55);

PayRoll p3("Mary", 12, 30);

std::cout << "Push() Function Test: " << std::endl; // Test the push function

std::cout << "Empty list:\n";

m.printStack();

std::cout << "First push:\n";

m.push(p1);

m.printStack();

std::cout << "Second push:\n";

m.push(p2);

m.printStack();

std::cout << "Third push:\n";

m.push(p3);

m.printStack();

std::cout << "==================================" << std::endl;

std::cout << "Copy Constructor Test: " << std::endl; // Test the copy constructor

std::cout << "Previous List: " << std::endl;

m.printStack();

MyStack<PayRoll> s(m);

std::cout << "New Stack at Initialization:" << std::endl;

s.printStack();

std::cout << "==================================" << std::endl;

std::cout << "Overloaded Assignment Test: " << std::endl; // Test the overloaded assignment

std::cout << "Assign previous list to empty list: " << std::endl;

MyStack<PayRoll> z;

z = m;

z.printStack();

std::cout << "Assign a new list to a populated list: " << std::endl;

std::cout << "List being copied over: " << std::endl;

MyStack<PayRoll> j;

j.push(p3);

j.push(p1);

j.push(p2);

j.printStack();

std::cout << "Original list being assigned to: " << std::endl;

z.printStack();

std::cout << "Revised list after assigning: " << std::endl;

z = j;

z.printStack();

std::cout << "Self assignment test: " << std::endl;

j = j;

std::cout << "==================================" << std::endl;

std::cout << "Pop Function Test: \n"; // Test the pop function

std::cout << "List being popped off: " << std::endl;

z.printStack();

auto pop1 = z.pop();

std::cout << "First pop: \n" << pop1 << std::endl;

auto pop2 = z.pop();

std::cout << "Second pop: \n" << pop2 << std::endl;

auto pop3 = z.pop();

std::cout << "Third pop: \n" << pop3 << std::endl;

std::cout << "Empty list pop:" << std::endl;

z.pop();

std::cout << "==================================" << std::endl;

std::cout << "Peek Function Test: \n"; // Test the peek function

std::cout << "Empty list peek(): " << std::endl;

auto peek1 = z.peek();

std::cout << "List being peeked off of: " << std::endl;

j.printStack();

std::cout << "Peek off the top: " << std::endl;

auto peek2 = j.peek();

std::cout << "Peeked item: " << peek2 << std::endl;

std::cout << "==================================" << std::endl;

std::cout << "Test with ints: " << std::endl; // Int tests

MyStack<int> m1;

int p4 = 10;

int p5 = 20;

int p6 = 30;

std::cout << "Push() Function Test: " << std::endl; // Test the push function

std::cout << "Empty list:\n";

m1.printStack();

std::cout << "First push:\n";

m1.push(p4);

m1.printStack();

std::cout << "Second push:\n";

m1.push(p5);

m1.printStack();

std::cout << "Third push:\n";

m1.push(p6);

m1.printStack();

std::cout << "==================================" << std::endl;

std::cout << "Copy Constructor Test: " << std::endl; // Test the copy constructor

std::cout << "Previous List: " << std::endl;

m1.printStack();

MyStack<int> s1(m1);

std::cout << "New Stack at Initialization:" << std::endl;

s1.printStack();

std::cout << "==================================" << std::endl;

std::cout << "Overloaded Assignment Test: " << std::endl; // Test the overloaded assignment

std::cout << "Assign previous list to empty list: " << std::endl;

MyStack<int> z1;

z1 = m1;

z1.printStack();

std::cout << "Assign a new list to a populated list: " << std::endl;

std::cout << "List being copied over: " << std::endl;

MyStack<int> j1;

j1.push(p6);

j1.push(p4);

j1.push(p5);

j1.printStack();

std::cout << "Original list being assigned to: " << std::endl;

z1.printStack();

std::cout << "Revised list after assigning: " << std::endl;

z1 = j1;

z1.printStack();

std::cout << "Self assignment test: " << std::endl;

j1 = j1;

std::cout << "==================================" << std::endl;

std::cout << "Pop Function Test: \n"; // Test the pop function

std::cout << "List being popped off: " << std::endl;

z1.printStack();

auto pop4 = z1.pop();

std::cout << "First pop: \n" << pop4 << std::endl;

auto pop5 = z1.pop();

std::cout << "Second pop: \n" << pop5 << std::endl;

auto pop6 = z1.pop();

std::cout << "Third pop: \n" << pop6 << std::endl;

std::cout << "Empty list pop:" << std::endl;

z1.pop();

std::cout << "==================================" << std::endl;

std::cout << "Peek Function Test: \n"; // Test the peek function

std::cout << "Empty list peek(): " << std::endl;

auto peek3 = z1.peek();

std::cout << "List being peeked off of: " << std::endl;

j1.printStack();

std::cout << "Peek off the top: " << std::endl;

auto peek4 = j1.peek();

std::cout << "Peeked item: " << peek4 << std::endl;

std::cout << "==================================" << std::endl;

std::cout << "Test with Strings: " << std::endl; // String tests

MyStack<std::string> m2;

std::string p7 = "Hello there";

std::string p8 = "Hi how are you";

std::string p9 = "I am great how are you";

std::cout << "Push() Function Test: " << std::endl; // Test the push function

std::cout << "Empty list:\n";

m2.printStack();

std::cout << "First push:\n";

m2.push(p7);

m2.printStack();

std::cout << "Second push:\n";

m2.push(p8);

m2.printStack();

std::cout << "Third push:\n";

m2.push(p9);

m2.printStack();

std::cout << "==================================" << std::endl;

std::cout << "Copy Constructor Test: " << std::endl; // Test the copy constructor

std::cout << "Previous List: " << std::endl;

m2.printStack();

MyStack<std::string> s2(m2);

std::cout << "New Stack at Initialization:" << std::endl;

s2.printStack();

std::cout << "==================================" << std::endl;

std::cout << "Overloaded Assignment Test: " << std::endl; // Test the overloaded assignment

std::cout << "Assign previous list to empty list: " << std::endl;

MyStack<std::string> z2;

z2 = m2;

z2.printStack();

std::cout << "Assign a new list to a populated list: " << std::endl;

std::cout << "List being copied over: " << std::endl;

MyStack<std::string> j2;

j2.push(p9);

j2.push(p7);

j2.push(p8);

j2.printStack();

std::cout << "Original list being assigned to: " << std::endl;

z2.printStack();

std::cout << "Revised list after assigning: " << std::endl;

z2 = j2;

z2.printStack();

std::cout << "Self assignment test: " << std::endl;

j2 = j2;

std::cout << "==================================" << std::endl;

std::cout << "Pop Function Test: \n"; // Test the pop function

std::cout << "List being popped off: " << std::endl;

z2.printStack();

auto pop7 = z2.pop();

std::cout << "First pop: \n" << pop7 << std::endl;

auto pop8 = z2.pop();

std::cout << "Second pop: \n" << pop8 << std::endl;

auto pop9 = z2.pop();

std::cout << "Third pop: \n" << pop9 << std::endl;

std::cout << "Empty list pop:" << std::endl;

z2.pop();

std::cout << "==================================" << std::endl;

std::cout << "Peek Function Test: \n"; // Test the peek function

std::cout << "Empty list peek(): " << std::endl;

auto peek5 = z2.peek();

std::cout << "List being peeked off of: " << std::endl;

j2.printStack();

std::cout << "Peek off the top: " << std::endl;

auto peek6 = j2.peek();

std::cout << "Peeked item: " << peek6 << std::endl;

std::cout << "==================================" << std::endl;

return 0;

}

**Sample Output:**

Testing for Lab-5

Test with PayRoll Objects:

Push() Function Test:

Empty list:

The list is empty

First push:

Bottom

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Top

Second push:

Bottom

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Top

Third push:

Bottom

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Top

==================================

Copy Constructor Test:

Previous List:

Bottom

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Top

New Stack at Initialization:

Bottom

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Top

==================================

Overloaded Assignment Test:

Assign previous list to empty list:

Bottom

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Top

Assign a new list to a populated list:

List being copied over:

Bottom

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Top

Original list being assigned to:

Bottom

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Top

Revised list after assigning:

Bottom

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Top

Self assignment test:

Cannot assign, Objects are identical

==================================

Pop Function Test:

List being popped off:

Bottom

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Top

First pop:

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Second pop:

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Third pop:

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Empty list pop:

Cannot pop, stack is empty

==================================

Peek Function Test:

Empty list peek():

The stack is empty

List being peeked off of:

Bottom

Name: Mary

PayRate: 12

Hours: 30

Total Pay: 360

Name: Ryan

PayRate: 25

Hours: 52

Total Pay: 1300

Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

Top

Peek off the top:

Peeked item: Name: Skylar

PayRate: 30

Hours: 55

Total Pay: 1650

==================================

Test with ints:

Push() Function Test:

Empty list:

The list is empty

First push:

Bottom

10

Top

Second push:

Bottom

10

20

Top

Third push:

Bottom

10

20

30

Top

==================================

Copy Constructor Test:

Previous List:

Bottom

10

20

30

Top

New Stack at Initialization:

Bottom

10

20

30

Top

==================================

Overloaded Assignment Test:

Assign previous list to empty list:

Bottom

10

20

30

Top

Assign a new list to a populated list:

List being copied over:

Bottom

30

10

20

Top

Original list being assigned to:

Bottom

10

20

30

Top

Revised list after assigning:

Bottom

30

10

20

Top

Self assignment test:

Cannot assign, Objects are identical

==================================

Pop Function Test:

List being popped off:

Bottom

30

10

20

Top

First pop:

20

Second pop:

10

Third pop:

30

Empty list pop:

Cannot pop, stack is empty

==================================

Peek Function Test:

Empty list peek():

The stack is empty

List being peeked off of:

Bottom

30

10

20

Top

Peek off the top:

Peeked item: 20

==================================

Test with Strings:

Push() Function Test:

Empty list:

The list is empty

First push:

Bottom

Hello there

Top

Second push:

Bottom

Hello there

Hi how are you

Top

Third push:

Bottom

Hello there

Hi how are you

I am great how are you

Top

==================================

Copy Constructor Test:

Previous List:

Bottom

Hello there

Hi how are you

I am great how are you

Top

New Stack at Initialization:

Bottom

Hello there

Hi how are you

I am great how are you

Top

==================================

Overloaded Assignment Test:

Assign previous list to empty list:

Bottom

Hello there

Hi how are you

I am great how are you

Top

Assign a new list to a populated list:

List being copied over:

Bottom

I am great how are you

Hello there

Hi how are you

Top

Original list being assigned to:

Bottom

Hello there

Hi how are you

I am great how are you

Top

Revised list after assigning:

Bottom

I am great how are you

Hello there

Hi how are you

Top

Self assignment test:

Cannot assign, Objects are identical

==================================

Pop Function Test:

List being popped off:

Bottom

I am great how are you

Hello there

Hi how are you

Top

First pop:

Hi how are you

Second pop:

Hello there

Third pop:

I am great how are you

Empty list pop:

Cannot pop, stack is empty

==================================

Peek Function Test:

Empty list peek():

The stack is empty

List being peeked off of:

Bottom

I am great how are you

Hello there

Hi how are you

Top

Peek off the top:

Peeked item: Hi how are you

==================================

**MyStack.h:**

#ifndef \_PAYROLLSTACK\_H\_

#define \_PAYROLLSTACK\_H\_

#include "payroll.h"

#include <iostream>

template<class T>

class MyStack {

private:

struct StackNode { // Node

T data;

StackNode\* next;

};

StackNode\* head; // Bottom of the stack or head of linked list

public:

MyStack(); // Default ctor

~MyStack(); // Destructor

MyStack(const MyStack&); // Copy ctor

MyStack operator=(const MyStack&); // Oeverloaded assignment

void printStack(); // Prints the data members of the stack

void push(T obj); // Adds object to stack

T pop(); // Same but uses a reference to the pop function

T peek(); // Shows the item on the top of the stack

};

/\* Default Ctor:

\* Initializes all values to nullptr

\*/

template<class T>

MyStack<T>::MyStack() {

head = nullptr;

}

/\* Destructor:

\* Deletes dynamically allocated data

\*/

template<class T>

MyStack<T>::~MyStack() {

if (head == nullptr) {

return;

}

StackNode\* cursor = head;

while (cursor) {

cursor = cursor->next;

delete head;

head = cursor;

}

}

/\* Copy Ctor:

\* Copies all elements of linked list stack over to another

\*/

template<class T>

MyStack<T>::MyStack(const MyStack& rhs) {

head = nullptr;

StackNode\* rhsCursor = rhs.head;

while (rhsCursor) {

push(rhsCursor->data);

rhsCursor = rhsCursor->next;

}

}

/\* Overloaded Assignment:

\* Allows the user to assign one stack to another

\*/

template<class T>

MyStack<T> MyStack<T>::operator=(const MyStack& rhs) {

if (this == &rhs) {

std::cout << "Cannot assign, Objects are identical" << std::endl;

return \*this;

}

if (head != nullptr) {

StackNode\* cursor = head;

while (cursor) {

cursor = cursor->next;

delete head;

head = cursor;

}

head = nullptr;

}

StackNode\* rhsCursor = rhs.head;

while (rhsCursor) {

push(rhsCursor->data);

rhsCursor = rhsCursor->next;

}

return \*this;

}

/\* Push Function:

\* Adds an object to the stack

\*/

template<class T>

void MyStack<T>::push(T obj) {

StackNode\* newNode = new StackNode;

newNode->data = obj;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

return;

}

StackNode\* cursor = head;

while (cursor->next) {

cursor = cursor->next;

}

cursor->next = newNode;

}

/\* Pop Function:

\* Returns the number of elements on the stack

\*/

template<class T>

T MyStack<T>::pop() {

if (head == nullptr) {

std::cout << "Cannot pop, stack is empty" << std::endl;

T obj;

return obj;

}

if (head->next == nullptr) {

StackNode\* temp = head;

head = nullptr;

T obj = temp->data;

delete temp;

return obj;

}

StackNode\* cursor = head;

StackNode\* previous = head;

while (cursor->next) {

previous = cursor;

cursor = cursor->next;

}

previous->next = nullptr;

T rtnObj = cursor->data;

delete cursor;

return rtnObj;

}

/\* PrintStack Function:

\* Prints the stack to the screen for testing and visual output

\*/

template<class T>

void MyStack<T>::printStack() {

if (head == nullptr) {

std::cout << "The list is empty" << std::endl;

return;

}

StackNode\* cursor = head;

std::cout << "Bottom\n" << std::endl;

std::cout << cursor->data << std::endl;

while (cursor->next) {

cursor = cursor->next;

std::cout << cursor->data << std::endl;

}

std::cout << "Top\n" << std::endl;

}

/\* Peek Function:

\* Shows the top item on the stack

\*/

template<class T>

T MyStack<T>::peek() {

if (head == nullptr) {

std::cout << "The stack is empty" << std::endl;

T ded;

return ded;

}

StackNode\* cursor = head;

while (cursor->next) {

cursor = cursor->next;

}

auto rtnMe = cursor->data;

return rtnMe;

}

#endif

**payroll.h:**

#ifndef PAYROLL\_H\_

#define PAYROLL\_H\_

#include <string>

class PayRoll {

friend std::ostream& operator<<(std::ostream&, PayRoll&);

private:

std::string name;

double payrate;

double hours;

public:

PayRoll(); // default ctor

PayRoll(std::string, double, double); // non-default ctor

double getRate(); // returns payrate

double getHours(); // returns hours

void setRate(double); // assigns payrate

void setName(std::string); // assigns name

void setHours(double); // assigns hours

double calculatePay(); // "getTotal()" returns the pay

void printInfo(); // prints info of all data members plus total pay

std::string getName(); // returns name

};

#endif

**payroll.cpp:**

#include "payroll.h" // Header file

#include <iostream> // cout, endl

#include <string>

/\* Ryan Rosiak

\*

\* Implementaion file for payroll.h:

\*

\*/

/\*

\* Default Constructor:

\* Initializes all members to default values.

\*/

PayRoll::PayRoll() {

name = "";

payrate = 0.0;

hours = 0.0;

}

/\*

\* Non-Default Constructor:

\* Takes a string, double, and another double as arguments and assigns them to

\* name, payrate, and hours.

\*/

PayRoll::PayRoll(std::string n, double pr, double hr) {

name = n;

payrate = pr;

hours = hr;

}

/\*

\* Payrate Setter:

\* Takes a double as an argument and assigns it to payrate

\*/

void PayRoll::setRate(double pr) {

payrate = pr;

}

/\*

\* Name Setter:

\* Takes a string as an argument and assigns it to name

\*/

void PayRoll::setName(std::string n) {

name = n;

}

/\*

\* Hours Setter:

\* Takes an double as an arugment and assigns it to hours

\*/

void PayRoll::setHours(double h) {

hours = h;

}

/\*

\* Calculate Function:

\* Calculates the total pay by returning the value of hours

\* multiplied by pay

\*/

double PayRoll::calculatePay() {

return hours \* payrate;

}

/\*

\* PrintInfo Function:

\* Displays all possible info in given class

\*/

void PayRoll::printInfo() {

std::cout << "Name: " << name << std::endl;

std::cout << "PayRate: " << payrate << std::endl;

std::cout << "Hours: " << hours << std::endl;

std::cout << "Total Pay: " << calculatePay() << std::endl;

}

/\*

\* Name Getter:

\* Returns a string that is the name of said object

\*/

std::string PayRoll::getName() {

return name;

}

/\*

\* PayRate Getter:

\* Returns the rate of the current object

\*/

double PayRoll::getRate() {

return payrate;

}

/\*

\* Hours Getter:

\* Returns the hours of the current object

\*/

double PayRoll::getHours() {

return hours;

}

/\*

\* Overloaded operator<< Function:

\* Returns a reference to the ouput stream and prints out payroll members

\*/

std::ostream& operator<<(std::ostream& os, PayRoll& p) {

p.printInfo();

return os;

}