**Lab Description:**

In README file

\*Not on Write up

**Main:**

#include "payroll.h"

#include "PayRollStack.h"

#include <iostream>

int main() {

PayRoll p1("Ryan", 25, 15); // Objects that will be used in testing

PayRoll p2("Skylar", 32, 50);

PayRoll p3("John", 10, 25);

PayRoll p4("Mary", 12, 25);

PayRoll p5("Marvin", 12, 10);

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

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

PayRollStack stack;

stack.push(p1);

stack.push(p2);

stack.push(p3);

stack.push(p4);

stack.push(p5);

stack.printStack();

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

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

std::cout << "Stack 1 \n";

stack.printStack();

PayRollStack stack2(stack);

std::cout << "Stack 2 \n";

stack2.printStack();

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

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

std::cout << "Stack 1 \n";

stack.printStack();

PayRollStack stack3;

stack3 = stack;

std::cout << "Stack 2 \n";

stack3.printStack();

std::cout << "Sample identical assignment: " << std::endl;

stack = stack;

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

std::cout << "Pop Function Test (returns a PayRoll object): \n"; // Test the pop function

std::cout << "Current stack: " << std::endl;

stack.printStack();

std::cout << "First pop(): \n";

stack.pop().printInfo();

std::cout << "Second pop(): \n";

stack.pop().printInfo();

std::cout << "Third pop(): \n";

stack.pop().printInfo();

std::cout << "Stack after pops: " << std::endl;

stack.printStack();

std::cout << "Popping rest of nodes to empty list: " << std::endl;

stack.pop();

stack.pop();

std::cout << "Empty stack: " << std::endl;

stack.printStack();

stack.pop();

std::cout << "Example empty stack pop: \n";

PayRollStack stack4;

stack4.pop();

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

std::cout << "Pop Function Test (a reference to a PayRoll object is passed): \n"; // Test other pop function

std::cout << "Repopulating previous stack... \n";

stack.push(p1);

stack.push(p2);

stack.push(p5);

stack.push(p3);

stack.push(p4);

std::cout << "Current stack: \n";

stack.printStack();

PayRoll pRef;

PayRoll pRef2;

stack.pop(pRef);

stack.pop(pRef2);

std::cout << "Info from popped nodes: " << std::endl;

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

pRef.printInfo();

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

pRef2.printInfo();

std::cout << "Stack after popped nodes: " << std::endl;

stack.printStack();

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

std::cout << "Size() Function Test: \n"; // Test the size function

std::cout << "First stack: " << std::endl;

stack.printStack();

std::cout << "Size of stack: " << stack.size() << "\n";

std::cout << "Second stack: " << std::endl;

PayRollStack stack5;

stack5.push(p4);

stack5.push(p2);

stack5.printStack();

std::cout << "Size of stack: " << stack5.size() << "\n";

std::cout << "Empty stack example: " << std::endl;

std::cout << "Size of stack: " << stack4.size() << "\n";

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

return 0;

}

**Sample Output:**

Testing for Lab-5

Push() Function Test:

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

Name: John

PayRate: 10

Hours: 25

Total Pay: 250

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

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

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

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

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

|Top|

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

Copy Constructor Test:

Stack 1

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

Name: John

PayRate: 10

Hours: 25

Total Pay: 250

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

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

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

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

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

|Top|

Stack 2

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

Name: John

PayRate: 10

Hours: 25

Total Pay: 250

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

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

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

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

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

|Top|

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

Overloaded Assignment Test:

Stack 1

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

Name: John

PayRate: 10

Hours: 25

Total Pay: 250

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

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

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

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

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

|Top|

Stack 2

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

Name: John

PayRate: 10

Hours: 25

Total Pay: 250

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

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

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

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

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

|Top|

Sample identical assignment:

Cannot assign, Objects are identical

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

Pop Function Test (returns a PayRoll object):

Current stack:

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

Name: John

PayRate: 10

Hours: 25

Total Pay: 250

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

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

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

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

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

|Top|

First pop():

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

Second pop():

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

Third pop():

Name: John

PayRate: 10

Hours: 25

Total Pay: 250

Stack after pops:

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

|Top|

Popping rest of nodes to empty list:

Empty stack:

|Bottom|

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

|Top|

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

Cannot pop, stack is empty

Example empty stack pop:

Cannot pop, stack is empty

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

Pop Function Test (a reference to a PayRoll object is passed):

Repopulating previous stack...

Current stack:

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

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

Name: John

PayRate: 10

Hours: 25

Total Pay: 250

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

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

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

|Top|

Info from popped nodes:

First pop:

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

Second pop:

Name: John

PayRate: 10

Hours: 25

Total Pay: 250

Stack after popped nodes:

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

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

|Top|

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

Size() Function Test:

First stack:

|Bottom|

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

Name: Ryan

PayRate: 25

Hours: 15

Total Pay: 375

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

Name: Marvin

PayRate: 12

Hours: 10

Total Pay: 120

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

|Top|

Size of stack: 3

Second stack:

|Bottom|

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

Name: Mary

PayRate: 12

Hours: 25

Total Pay: 300

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

Name: Skylar

PayRate: 32

Hours: 50

Total Pay: 1600

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

|Top|

Size of stack: 2

Empty stack example:

Size of stack: 0

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

**PayRoll.h:**

#ifndef PAYROLL\_H\_

#define PAYROLL\_H\_

#include <string>

class 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;

}

**PayRollStack.h:**

#ifndef \_PAYROLLSTACK\_H\_

#define \_PAYROLLSTACK\_H\_

#include "payroll.h"

#include <iostream>

class PayRollStack {

private:

struct StackNode { // Node

PayRoll p;

StackNode\* next;

};

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

public:

PayRollStack(); // Default ctor

~PayRollStack(); // Destructor

PayRollStack(const PayRollStack&); // Copy ctor

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

void push(PayRoll p); // Adds object to stack

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

void pop(PayRoll&); // Returns a number of elements on the stack

int size(); // Returns the number of elements on the stack

void printStack(); // Prints the stack for testing purposes

};

#endif

**PayRollStack.cpp:**

#include "PayRollStack.h"

/\* Default Ctor:

\* Initializes all values to nullptr

\*/

PayRollStack::PayRollStack() {

head = nullptr;

}

/\* Destructor:

\* Deletes dynamically allocated data

\*/

PayRollStack::~PayRollStack() {

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

\*/

PayRollStack::PayRollStack(const PayRollStack& rhs) {

head = nullptr;

StackNode\* rhsCursor = rhs.head;

while (rhsCursor) {

push(rhsCursor->p);

rhsCursor = rhsCursor->next;

}

}

/\* Overloaded Assignment:

\* Allows the user to assign one stack to another

\*/

PayRollStack PayRollStack::operator=(const PayRollStack& rhs) {

if (this == &rhs) {

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

return \*this;

}

StackNode\* rhsCursor = rhs.head;

while (rhsCursor) {

push(rhsCursor->p);

rhsCursor = rhsCursor->next;

}

return \*this;

}

/\* Push Function:

\* Adds an object to the stack

\*/

void PayRollStack::push(PayRoll obj) {

StackNode\* newNode = new StackNode;

newNode->p = 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

\*/

PayRoll PayRollStack::pop() {

if (head == nullptr) {

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

PayRoll obj;

return obj;

}

if (head->next == nullptr) {

StackNode\* temp = head;

delete temp;

head = nullptr;

PayRoll obj;

return obj;

}

StackNode\* cursor = head;

StackNode\* previous = head;

while (cursor->next) {

previous = cursor;

cursor = cursor->next;

}

previous->next = nullptr;

PayRoll rtnObj = cursor->p;

delete cursor;

return rtnObj;

}

/\* Overloaded Pop Function:

\* Same as previous pop function, only implements using a reference to

\* the object that is popped off the stack

\*/

void PayRollStack::pop(PayRoll& obj) {

obj = pop();

}

/\* Size Function:

\* Returns the number of elements on the stack

\*/

int PayRollStack::size() {

if (head == nullptr) {

return 0;

}

StackNode\* cursor = head;

int counter = 0;

while (cursor) {

counter++;

cursor = cursor->next;

}

return counter;

}

/\* PrintStack Function:

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

\*/

void PayRollStack::printStack() {

std::cout << "|Bottom|" << std::endl;

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

if (head != nullptr) {

StackNode\* cursor = head;

while (cursor) {

cursor->p.printInfo();

cursor = cursor->next;

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

}

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

} else {

std::cout << "|Top|" << std::endl;

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

}

}