**Lab Description:**

In README.txt file

\*Not on Write up

**Main:**

#include <iostream> // cout, endl

#include "payroll.h" // PayRoll class

#include "PayRollList.h" // PayRollList class

int main() {

PayRollList list;

std::cout << "Length Test cases: " << std::endl; // Length Test Cases

std::cout << "Current empty list length: " << std::endl;

std::cout << list.length() << std::endl;

list.insert("Ryan", 25, 10);

list.insert("John", 10, 15);

list.insert("Skylar", 12, 40);

list.printPayChecks();

std::cout << "Now current list length after 3 additions: " << std::endl;

std::cout << list.length() << std::endl;

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

std::cout << "Operator[] test cases: " << std::endl; // Operator[] Test Cases

std::cout << "Name of first item in list: " << std::endl;

std::cout << list[1]->getName() << std::endl;

std::cout << "Payrate of second item in list: " << std::endl;

std::cout << list[2]->getRate() << std::endl;

std::cout << "Hours of third item in list: " << std::endl;

std::cout << list[3]->getHours() << std::endl;

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

std::cout << "List remove() function test cases: " << std::endl; // remove() Test Cases

std::cout << "Removing the second position in the list..." << std::endl;

list.remove(2);

list.printPayChecks();

std::cout << "Removing the head of the list..." << std::endl;

list.remove(1);

list.printPayChecks();

std::cout << "Repopulating list real quick..." << std::endl;

list.insert("Margaret", 14, 23);

list.insert("Mary", 25, 58);

list.insert("Marvin", 42, 10);

list.printPayChecks();

std::cout << "Removing tail of list..." << std::endl;

list.remove(4);

list.printPayChecks();

std::cout << "Example of invalid index..." << std::endl;

list.remove(-5);

list.remove(6);

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

PayRoll p1("Beatrice", 45, 23); // Example objects created

PayRoll p2("Roberto", 12, 40);

std::cout << "Assign Function Test cases: " << std::endl; // Assign function Test Cases

std::cout << "Current size of list: " << list.length() << std::endl;

std::cout << "Example Invalid Input: " << std::endl;

list.assign(7, p1);

list.assign(0, p2);

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

list.printPayChecks();

std::cout << "Assigning Beatrice info to subscript 2: " << std::endl;

list.assign(2, p1);

std::cout << "Assigning Roberto info to tail of subscript: " << std::endl;

list.assign(3, p2);

std::cout << "Assigning Beatrice info to head: " << std::endl;

list.assign(1, p1);

list.printPayChecks();

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

std::cout << "Overloaded insert function using position Test cases: " << std::endl; // Overloaded Insert Test Cases

PayRollList list2;

std::cout << "Current New list size: " << list2.length() << std::endl;

std::cout << "Inserting Beatrice info into empty list at beginnning: " << std::endl;

list2.insert(1, p1);

list2.printPayChecks();

std::cout << "Inserting Roberto info at tail of list: " << std::endl;

std::cout << "Current list size before insert: " << list2.length() << std::endl;

list2.insert(2, p2);

list2.printPayChecks();

PayRoll p3("Jamison", 14, 25);

PayRoll p4("Skylar", 36, 45);

std::cout << "Inserting Jamison info at tail of list: " << std::endl;

std::cout << "Current list size before insert: " << list2.length() << std::endl;

list2.insert(3, p3);

list2.printPayChecks();

std::cout << "Inserting Skylar info at second position in list: " << std::endl;

std::cout << "Current list size before insert: " << list2.length() << std::endl;

list2.insert(2, p4);

list2.printPayChecks();

std::cout << "Inserting Skylar info at head in list: " << std::endl;

std::cout << "Current list size before insert: " << list2.length() << std::endl;

list2.insert(1, p4);

list2.printPayChecks();

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

std::cout << "Copy Constructor Test Case: " << std::endl; // Copy Constructor Test Cases

PayRollList originalList;

originalList.insert("Jerry", 10, 10);

originalList.insert("Magnolia", 12, 15);

originalList.insert("Veridian", 12, 16);

originalList.insert("Virgo", 2, 14);

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

originalList.printPayChecks();

PayRollList copiedList(originalList);

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

copiedList.printPayChecks();

return 0;

}

**Sample Output:**

Length Test cases:

Current empty list length:

0

Name: Ryan

PayRate: 25

Hours: 10

Total Pay: 250

Name: Skylar

PayRate: 12

Hours: 40

Total Pay: 480

Name: John

PayRate: 10

Hours: 15

Total Pay: 150

Now current list length after 3 additions:

3

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

Operator[] test cases:

Name of first item in list:

Ryan

Payrate of second item in list:

12

Hours of third item in list:

15

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

List remove() function test cases:

Removing the second position in the list...

Name: Ryan

PayRate: 25

Hours: 10

Total Pay: 250

Name: John

PayRate: 10

Hours: 15

Total Pay: 150

Removing the head of the list...

Name: John

PayRate: 10

Hours: 15

Total Pay: 150

Repopulating list real quick...

Name: Marvin

PayRate: 42

Hours: 10

Total Pay: 420

Name: Mary

PayRate: 25

Hours: 58

Total Pay: 1450

Name: Margaret

PayRate: 14

Hours: 23

Total Pay: 322

Name: John

PayRate: 10

Hours: 15

Total Pay: 150

Removing tail of list...

Name: Marvin

PayRate: 42

Hours: 10

Total Pay: 420

Name: Mary

PayRate: 25

Hours: 58

Total Pay: 1450

Name: Margaret

PayRate: 14

Hours: 23

Total Pay: 322

Example of invalid index...

Invalid! Index starts at 1.

Index larger than size of current list

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

Assign Function Test cases:

Current size of list: 3

Example Invalid Input:

Index larger than size of current list

Invalid! Index starts at 1.

Current List:

Name: Marvin

PayRate: 42

Hours: 10

Total Pay: 420

Name: Mary

PayRate: 25

Hours: 58

Total Pay: 1450

Name: Margaret

PayRate: 14

Hours: 23

Total Pay: 322

Assigning Beatrice info to subscript 2:

Assigning Roberto info to tail of subscript:

Assigning Beatrice info to head:

Name: Beatrice

PayRate: 45

Hours: 23

Total Pay: 1035

Name: Beatrice

PayRate: 45

Hours: 23

Total Pay: 1035

Name: Roberto

PayRate: 12

Hours: 40

Total Pay: 480

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

Overloaded insert function using position Test cases:

Current New list size: 0

Inserting Beatrice info into empty list at beginnning:

Name: Beatrice

PayRate: 45

Hours: 23

Total Pay: 1035

Inserting Roberto info at tail of list:

Current list size before insert: 1

Name: Beatrice

PayRate: 45

Hours: 23

Total Pay: 1035

Name: Roberto

PayRate: 12

Hours: 40

Total Pay: 480

Inserting Jamison info at tail of list:

Current list size before insert: 2

Name: Beatrice

PayRate: 45

Hours: 23

Total Pay: 1035

Name: Roberto

PayRate: 12

Hours: 40

Total Pay: 480

Name: Jamison

PayRate: 14

Hours: 25

Total Pay: 350

Inserting Skylar info at second position in list:

Current list size before insert: 3

Name: Beatrice

PayRate: 45

Hours: 23

Total Pay: 1035

Name: Skylar

PayRate: 36

Hours: 45

Total Pay: 1620

Name: Roberto

PayRate: 12

Hours: 40

Total Pay: 480

Name: Jamison

PayRate: 14

Hours: 25

Total Pay: 350

Inserting Skylar info at head in list:

Current list size before insert: 4

Name: Skylar

PayRate: 36

Hours: 45

Total Pay: 1620

Name: Beatrice

PayRate: 45

Hours: 23

Total Pay: 1035

Name: Skylar

PayRate: 36

Hours: 45

Total Pay: 1620

Name: Roberto

PayRate: 12

Hours: 40

Total Pay: 480

Name: Jamison

PayRate: 14

Hours: 25

Total Pay: 350

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

Copy Constructor Test Case:

Original List:

Name: Magnolia

PayRate: 12

Hours: 15

Total Pay: 180

Name: Jerry

PayRate: 10

Hours: 10

Total Pay: 100

Name: Veridian

PayRate: 12

Hours: 16

Total Pay: 192

Name: Virgo

PayRate: 2

Hours: 14

Total Pay: 28

Copied List:

Name: Magnolia

PayRate: 12

Hours: 15

Total Pay: 180

Name: Jerry

PayRate: 10

Hours: 10

Total Pay: 100

Name: Veridian

PayRate: 12

Hours: 16

Total Pay: 192

Name: Virgo

PayRate: 2

Hours: 14

Total Pay: 28

**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;

}

**PayRollList.h:**

#ifndef \_PAYROLLLIST\_H

#define \_PAYROLLLIST\_H

#include "payroll.h"

#include <string>

#include <iostream>

class PayRollList {

private:

struct ListNode { // nodes for linked list

PayRoll p;

ListNode\* next;

};

ListNode\* head; // head of linked list

public:

PayRollList(); // Default ctor

~PayRollList(); // Destructor

PayRollList(PayRollList&); // Copy Constructor

int length(); // Returns length of list

PayRoll\* operator[](int); // Overloaded index operator

void remove(int); // Removes the list item at the specified position

void assign(int, PayRoll); // Copys a PayRoll into a specific position

void insert(int, PayRoll); // inserts a PayRoll into the list at the specified position

void insert(std::string, double, double); // insert into linked list using 3 parameters

void insert(PayRoll); // insert into linked list using PayRoll object

void printPayChecks(); // Print function for all items in linked list

};

#endif

**PayRollList.cpp:**

#include "PayRollList.h"

/\*

\* Default Constructor

\* Takes no parameters and initializes head pointer to null

\*/

PayRollList::PayRollList() {

head = nullptr;

}

/\*

\* Insert Function:

\* Takes name, rate, and hours worked as parameters for a new ListNode.

\* Then calls overidden insert function to insert in order of payrate.

\*/

void PayRollList::insert(std::string n, double pr, double h) {

PayRoll input(n, pr, h);

insert(input);

}

/\*

\* Overidden insert:

\* Takes a PayRoll object directly.

\* Then, inserts the object into the linked list by order of payrate.

\*/

void PayRollList::insert(PayRoll newP) {

ListNode\* newNode = new ListNode;

newNode->p.setName(newP.getName());

newNode->p.setRate(newP.getRate());

newNode->p.setHours(newP.getHours());

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

return;

}

if (head->p.getRate() < newNode->p.getRate()) {

ListNode\* temp = head;

head = newNode;

newNode = temp;

head->next = newNode;

return;

}

ListNode\* cursor = head;

while (cursor->next != nullptr) {

ListNode\* forward = cursor->next;

if ((cursor->p.getRate() > newNode->p.getRate()) && (forward->p.getRate() < newNode->p.getRate())) {

cursor->next = newNode;

newNode->next = forward;

return;

}

cursor = cursor->next;

}

cursor->next = newNode;

}

/\*

\* PrintPayChecks Function:

\* Prints each employee name and total pay out.

\*/

void PayRollList::printPayChecks() {

if (head == nullptr) {

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

return;

}

ListNode\* cursor = head;

while (cursor) {

cursor->p.printInfo();

std::cout << std::endl;

cursor = cursor->next;

}

}

/\*

\* PayRollList Destructor:

\* Destroys dynamically allocated data

\*/

PayRollList::~PayRollList() {

if (head == nullptr) {

return;

}

ListNode\* cursor = head;

while (cursor) {

cursor = cursor->next;

delete head;

head = cursor;

}

}

/\*

\* Copy Constructor:

\* Copys all items from a list into a another list

\*/

PayRollList::PayRollList(PayRollList &rhs) {

head = nullptr;

ListNode\* rhsCursor = rhs.head;

while (rhsCursor) {

insert(rhsCursor->p);

rhsCursor = rhsCursor->next;

}

}

/\*

\* Length Function:

\* Returns the length of the current list

\*/

int PayRollList::length() {

int total = 0;

if (head == nullptr) {

return total;

}

ListNode\* cursor = head;

while (cursor) {

total++;

cursor = cursor->next;

}

return total;

}

/\*

\* Overloaded operator[]:

\* Takes an integer index and (if it exists) returns a reference to the PayRoll

\* inside it

\*/

PayRoll\* PayRollList::operator[](int input) {

if (input > length()) {

return nullptr;

}

if (head == nullptr) {

return nullptr;

}

int tracker = 0;

ListNode\* cursor = head;

while (cursor) {

if (tracker == (input - 1)) {

PayRoll\* ref = &cursor->p;

return ref;

}

cursor = cursor->next;

tracker++;

}

}

/\* Remove Function:

\* Remove the list item at the specified position

\*/

void PayRollList::remove(int input) {

if (input < 1) {

std::cout << "Invalid! Index starts at 1." << std::endl;

return;

}

if (input > length()) {

std::cout << "Index larger than size of current list" << std::endl;

return;

}

if (head == nullptr) {

std::cout << "List is empty already" << std::endl;

return;

}

if (head->next == nullptr) {

ListNode\* temp = head;

delete temp;

head = nullptr;

return;

}

if (input == 1) {

ListNode\* temp = head;

head = head->next;

delete temp;

return;

}

int tracker = 1;

ListNode\* cursor = head->next;

ListNode\* previous = head;

while (cursor) {

if (tracker == length()) {

previous->next = nullptr;

delete cursor;

return;

}

if (tracker == (input - 1)) {

previous->next = cursor->next;

delete cursor;

return;

}

tracker++;

previous = cursor;

cursor = cursor->next;

}

}

/\* Assign Function:

\* Copy a PayRoll into a specific position

\*/

void PayRollList::assign(int inputPos, PayRoll inputOBJ) {

if (inputPos < 1) {

std::cout << "Invalid! Index starts at 1." << std::endl;

return;

}

if (inputPos > length()) {

std::cout << "Index larger than size of current list" << std::endl;

return;

}

int tracker = 0;

ListNode\* cursor = head;

while (cursor) {

if (tracker == (inputPos - 1)) {

cursor->p = inputOBJ;

return;

}

tracker++;

cursor = cursor->next;

}

}

/\* Overloaded Insert:

\* Inserts the PayRoll into the list so that it is now at the specified position

\*/

void PayRollList::insert(int inputPos, PayRoll inputOBJ) {

if (inputPos < 1) {

std::cout << "Invalid! Index starts at 1." << std::endl;

return;

}

if (inputPos > (length() + 1)) {

std::cout << "Index larger than size of current list" << std::endl;

return;

}

ListNode\* newNode = new ListNode;

newNode->p.setName(inputOBJ.getName());

newNode->p.setRate(inputOBJ.getRate());

newNode->p.setHours(inputOBJ.getHours());

newNode->next = nullptr;

if (head == nullptr && inputPos == 1) {

head = newNode;

return;

}

if (inputPos == 1) {

ListNode\* temp = head;

head = newNode;

newNode->next = temp;

return;

}

if (head->next == nullptr && inputPos == 2) {

head->next = newNode;

return;

}

int tracker = 1;

ListNode\* cursor = head->next;

ListNode\* previous = head;

while(cursor) {

if (tracker == (inputPos - 1)) {

previous->next = newNode;

newNode->next = cursor;

return;

}

if (cursor->next == nullptr) {

break;

}

tracker++;

previous = cursor;

cursor = cursor->next;

}

// If made it to here than we want to add to tail

cursor->next = newNode;

}