**Project Description:**

\*Not on project write up

\*\*In README file

**Sample Output:**

**1.**

Int Queue Tests:

Queue Array enqueue Function Tests:

First Queue:

Queue is Empty

First Queue after 4 enqueues:

Front:

5

8

25

7

Back:

First Queue, auto resize then 3 more enqueues:

Front:

5

8

25

7

40

2

4

Back:

First Queue, second auto resize then 2 more enqueues:

Front:

5

8

25

7

40

2

4

16

1000

4500

Back:

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

Queue Array dequeue Function Tests:

Current Queue being worked with:

Front:

10

25

0

Back:

First dequeue: 10

Second dequeue: 25

Third dequeue: 0

List now:

Queue is Empty

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

Queue Array size Function Tests:

Current List:

Queue is Empty

Size of queue: 0

Current List:

Front:

5

8

25

7

40

2

4

16

1000

4500

Back:

Size of queue: 10

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

Queue Array isEmpty Function Tests:

Current List:

Queue is Empty

Queue is empty!!!

Current List:

Front:

5

8

25

7

40

2

4

16

1000

4500

Back:

Queue is not empty!!!

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

Queue Array printQueue Function Tests:

Print Queue function demonstrated through all tests

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

Queue Array overloaded assignment Tests:

First List:

Front:

5

8

25

7

40

2

4

16

1000

4500

Back:

Second List:

Queue is Empty

First copied to Second:

Front:

5

8

25

7

40

2

4

16

1000

4500

Back:

Third List:

Front:

41

88

92

Back:

Third copied to First:

Front:

41

88

92

Back:

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

Queue Array copy constructor Tests:

Current List:

Front:

41

88

92

Back:

New list Initialized:

Front:

41

88

92

Back:

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

Double Queue Tests:

Queue Array enqueue Function Tests:

First Queue:

Queue is Empty

First Queue after 4 enqueues:

Front:

5.2

8.4

25.621

7

Back:

First Queue, auto resize then 3 more enqueues:

Front:

5.2

8.4

25.621

7

40.15

2.2

4.441

Back:

First Queue, second auto resize then 2 more enqueues:

Front:

5.2

8.4

25.621

7

40.15

2.2

4.441

16.5

1000.5

4500.62

Back:

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

Queue Array dequeue Function Tests:

Current Queue being worked with:

Front:

10.14

25.18

0

Back:

First dequeue: 10.14

Second dequeue: 25.18

Third dequeue: 0

List now:

Queue is Empty

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

Queue Array size Function Tests:

Current List:

Queue is Empty

Size of queue: 0

Current List:

Front:

5.2

8.4

25.621

7

40.15

2.2

4.441

16.5

1000.5

4500.62

Back:

Size of queue: 10

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

Queue Array isEmpty Function Tests:

Current List:

Queue is Empty

Queue is empty!!!

Current List:

Front:

5.2

8.4

25.621

7

40.15

2.2

4.441

16.5

1000.5

4500.62

Back:

Queue is not empty!!!

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

Queue Array printQueue Function Tests:

Print Queue function demonstrated through all tests

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

Queue Array overloaded assignment Tests:

First List:

Front:

5.2

8.4

25.621

7

40.15

2.2

4.441

16.5

1000.5

4500.62

Back:

Second List:

Queue is Empty

First copied to Second:

Front:

5.2

8.4

25.621

7

40.15

2.2

4.441

16.5

1000.5

4500.62

Back:

Third List:

Front:

41.5

88.991

92.78

Back:

Third copied to First:

Front:

41.5

88.991

92.78

Back:

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

Queue Array copy constructor Tests:

Current List:

Front:

41.5

88.991

92.78

Back:

New list Initialized:

Front:

41.5

88.991

92.78

Back:

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

PayRoll Queue Tests:

Queue Array enqueue Function Tests:

First Queue:

Queue is Empty

First Queue after 4 enqueues:

Front:

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Back:

First Queue, auto resize then 3 more enqueues:

Front:

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Back:

First Queue, second auto resize then 2 more enqueues:

Front:

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Back:

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

Queue Array dequeue Function Tests:

Current Queue being worked with:

Front:

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Back:

First dequeue: Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Second dequeue: Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Third dequeue: Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

List now:

Queue is Empty

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

Queue Array size Function Tests:

Current List:

Queue is Empty

Size of queue: 0

Current List:

Front:

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Back:

Size of queue: 10

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

Queue Array isEmpty Function Tests:

Current List:

Queue is Empty

Queue is empty!!!

Current List:

Front:

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Back:

Queue is not empty!!!

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

Queue Array printQueue Function Tests:

Print Queue function demonstrated through all tests

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

Queue Array overloaded assignment Tests:

First List:

Front:

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Back:

Second List:

Queue is Empty

First copied to Second:

Front:

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Back:

Third List:

Front:

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Back:

Third copied to First:

Front:

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Back:

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

Queue Array copy constructor Tests:

Current List:

Front:

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Back:

New list Initialized:

Front:

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Back:

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

Does myList contain 100?

100 not in the list

3

2

6

1

4

5

++++++++++++

Name: Eve

PayRate: 30

Hours: 35

Total Pay: 1050

Name: Alice

PayRate: 10

Hours: 35

Total Pay: 350

Name: Bob

PayRate: 20

Hours: 35

Total Pay: 700

Name: Charlie

PayRate: 25

Hours: 35

Total Pay: 875

Name: Diana

PayRate: 10

Hours: 35

Total Pay: 350

++++++++++++

++++++++++++

+++++++ Queue Lad +++++++++

1523

2

++++++++++++

Name: Bob

PayRate: 20

Hours: 35

Total Pay: 700

Name: Alice

PayRate: 10

Hours: 35

Total Pay: 350

Name: Charlie

PayRate: 25

Hours: 35

Total Pay: 875

Name: Diana

PayRate: 10

Hours: 35

Total Pay: 350

Name: Eva

PayRate: 30

Hours: 35

Total Pay: 1050

+++++++ popped lad +++++++++

Name: Bob

PayRate: 20

Hours: 35

Total Pay: 700

++++++++++++

Name: Alice

PayRate: 10

Hours: 35

Total Pay: 350

Name: Charlie

PayRate: 25

Hours: 35

Total Pay: 875

Name: Diana

PayRate: 10

Hours: 35

Total Pay: 350

Name: Eva

PayRate: 30

Hours: 35

Total Pay: 1050

**2.**

Int Stack Tests:

Stack Array push Function Tests:

First Stack:

Stack is Empty

First Stack after 4 pushes:

Top:

7

25

8

5

Bottom:

First Stack, auto resize then 3 more pushes:

Top:

4

2

40

7

25

8

5

Bottom:

First Stack, second auto resize then 2 more push:

Top:

4500

1000

16

4

2

40

7

25

8

5

Bottom:

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

Stack Array pop Function Tests:

Current Stack being worked with:

Top:

0

25

10

Bottom:

First pop: 0

Second pop: 25

Third pop: 10

List now:

Stack is Empty

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

Stack Array size Function Tests:

Current List:

Stack is Empty

Size of stack: 0

Current List:

Top:

4500

1000

16

4

2

40

7

25

8

5

Bottom:

Size of stack: 10

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

Stack Array isEmpty Function Tests:

Current List:

Stack is Empty

Stack is empty!

Current List:

Top:

4500

1000

16

4

2

40

7

25

8

5

Bottom:

Stack is not empty!

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

Stack Array printStack Function Tests:

Print Stack function demonstrated through all tests

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

Stack Array overloaded assignment Tests:

First List:

Top:

4500

1000

16

4

2

40

7

25

8

5

Bottom:

Second List:

Stack is Empty

First copied to Second:

Top:

4500

1000

16

4

2

40

7

25

8

5

Bottom:

Third List:

Top:

92

88

41

Bottom:

Third copied to First:

Top:

92

88

41

Bottom:

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

Stack Array copy constructor Tests:

Current List:

Top:

92

88

41

Bottom:

New list Initialized:

Top:

92

88

41

Bottom:

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

Double Stack Tests:

Stack Array push Function Tests:

First Stack:

Stack is Empty

First Stack after 4 pushes:

Top:

7.898

25.2

8.6

5.25

Bottom:

First Stack, auto resize then 3 more pushes:

Top:

4.5

2.2

40.32

7.898

25.2

8.6

5.25

Bottom:

First Stack, second auto resize then 2 more push:

Top:

4500.7

1000

16.56

4.5

2.2

40.32

7.898

25.2

8.6

5.25

Bottom:

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

Stack Array pop Function Tests:

Current Stack being worked with:

Top:

0.7

25.65

10.2

Bottom:

First pop: 0.7

Second pop: 25.65

Third pop: 10.2

List now:

Stack is Empty

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

Stack Array size Function Tests:

Current List:

Stack is Empty

Size of stack: 0

Current List:

Top:

4500.7

1000

16.56

4.5

2.2

40.32

7.898

25.2

8.6

5.25

Bottom:

Size of stack: 10

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

Stack Array isEmpty Function Tests:

Current List:

Stack is Empty

Stack is empty!

Current List:

Top:

4500.7

1000

16.56

4.5

2.2

40.32

7.898

25.2

8.6

5.25

Bottom:

Stack is not empty!

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

Stack Array printStack Function Tests:

Print Stack function demonstrated through all tests

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

Stack Array overloaded assignment Tests:

First List:

Top:

4500.7

1000

16.56

4.5

2.2

40.32

7.898

25.2

8.6

5.25

Bottom:

Second List:

Stack is Empty

First copied to Second:

Top:

4500.7

1000

16.56

4.5

2.2

40.32

7.898

25.2

8.6

5.25

Bottom:

Third List:

Top:

92.43

88.01

41.4

Bottom:

Third copied to First:

Top:

92.43

88.01

41.4

Bottom:

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

Stack Array copy constructor Tests:

Current List:

Top:

92.43

88.01

41.4

Bottom:

New list Initialized:

Top:

92.43

88.01

41.4

Bottom:

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

PayRoll Stack Tests:

Stack Array push Function Tests:

First Stack:

Stack is Empty

First Stack after 4 pushes:

Top:

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Bottom:

First Stack, auto resize then 3 more pushes:

Top:

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Bottom:

First Stack, second auto resize then 2 more push:

Top:

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Bottom:

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

Stack Array pop Function Tests:

Current Stack being worked with:

Top:

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Bottom:

First pop: Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Second pop: Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Third pop: Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

List now:

Stack is Empty

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

Stack Array size Function Tests:

Current List:

Stack is Empty

Size of stack: 0

Current List:

Top:

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Bottom:

Size of stack: 10

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

Stack Array isEmpty Function Tests:

Current List:

Stack is Empty

Stack is empty!

Current List:

Top:

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Bottom:

Stack is not empty!

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

Stack Array printStack Function Tests:

Print Stack function demonstrated through all tests

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

Stack Array overloaded assignment Tests:

First List:

Top:

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Bottom:

Second List:

Stack is Empty

First copied to Second:

Top:

Name: Andrea

PayRate: 15

Hours: 55

Total Pay: 825

Name: Elena

PayRate: 30

Hours: 55

Total Pay: 1650

Name: Cricket

PayRate: 45

Hours: 50

Total Pay: 2250

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Name: Leonardo

PayRate: 25

Hours: 28

Total Pay: 700

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Margaret

PayRate: 1

Hours: 23

Total Pay: 23

Name: Skylar

PayRate: 55

Hours: 50

Total Pay: 2750

Name: James

PayRate: 21

Hours: 23

Total Pay: 483

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Bottom:

Third List:

Top:

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Bottom:

Third copied to First:

Top:

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Bottom:

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

Stack Array copy constructor Tests:

Current List:

Top:

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Bottom:

New list Initialized:

Top:

Name: Hemp

PayRate: 20

Hours: 22

Total Pay: 440

Name: Ryan

PayRate: 25

Hours: 32.4

Total Pay: 810

Name: Jamison

PayRate: 22

Hours: 59

Total Pay: 1298

Bottom:

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

Does myList contain 100?

100 not in the list

3

2

6

1

4

5

++++++++++++

Name: Eve

PayRate: 30

Hours: 35

Total Pay: 1050

Name: Alice

PayRate: 10

Hours: 35

Total Pay: 350

Name: Bob

PayRate: 20

Hours: 35

Total Pay: 700

Name: Diana

PayRate: 25

Hours: 35

Total Pay: 875

Name: Diana

PayRate: 10

Hours: 35

Total Pay: 350

++++++++++++

-----=========-------

Name: Eve

PayRate: 30

Hours: 35

Total Pay: 1050

Name: Bob

PayRate: 20

Hours: 35

Total Pay: 700

Name: Alice

PayRate: 10

Hours: 35

Total Pay: 350

Name: Charlie

PayRate: 25

Hours: 35

Total Pay: 875

Name: Diana

PayRate: 10

Hours: 35

Total Pay: 350

+++++++ popped lad +++++++++

Name: Eve

PayRate: 30

Hours: 35

Total Pay: 1050

++++++++++++

Name: Bob

PayRate: 20

Hours: 35

Total Pay: 700

Name: Alice

PayRate: 10

Hours: 35

Total Pay: 350

Name: Charlie

PayRate: 25

Hours: 35

Total Pay: 875

Name: Diana

PayRate: 10

Hours: 35

Total Pay: 350

++++++++++++

10

102

1

50

+++++++ =operator +++++++++

Name: Bob

PayRate: 20

Hours: 35

Total Pay: 700

Name: Alice

PayRate: 10

Hours: 35

Total Pay: 350

Name: Charlie

PayRate: 25

Hours: 35

Total Pay: 875

Name: Diana

PayRate: 10

Hours: 35

Total Pay: 350

**SUStackDriver.cpp:**

#include "SUStack.cpp"

#include "payroll.h"

#include "SUList.h"

#include <iostream>

#include <stdio.h>

int main() {

SUStackArr<int> intStack;

std::cout << "Int Stack Tests: " << std::endl;

std::cout << "Stack Array push Function Tests: " << std::endl;

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

intStack.printStack();

intStack.push(5);

intStack.push(8);

intStack.push(25);

intStack.push(7);

std::cout << "First Stack after 4 pushes: " << std::endl;

intStack.printStack();

intStack.push(40);

intStack.push(2);

intStack.push(4);

std::cout << "First Stack, auto resize then 3 more pushes: " << std::endl;

intStack.printStack();

intStack.push(16);

intStack.push(1000);

intStack.push(4500);

std::cout << "First Stack, second auto resize then 2 more push: " << std::endl;

intStack.printStack();

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

std::cout << "Stack Array pop Function Tests: " << std::endl;

SUStackArr<int> intStack2;

intStack2.push(10);

intStack2.push(25);

intStack2.push(0);

std::cout << "Current Stack being worked with: " << std::endl;

intStack2.printStack();

int pop1, pop2, pop3;

intStack2.pop(pop1);

intStack2.pop(pop2);

intStack2.pop(pop3);

std::cout << "First pop: " << pop1 << "\nSecond pop: " << pop2 << "\nThird pop: " << pop3 << std::endl;

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

intStack2.printStack();

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

std::cout << "Stack Array size Function Tests: " << std::endl;

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

intStack2.printStack();

std::cout << "Size of stack: " << intStack2.size() << std::endl;

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

intStack.printStack();

std::cout << "Size of stack: " << intStack.size() << std::endl;

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

std::cout << "Stack Array isEmpty Function Tests: " << std::endl;

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

intStack2.printStack();

if (intStack2.isEmpty()) {

std::cout << "Stack is empty!" << std::endl;

}

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

intStack.printStack();

if (!intStack.isEmpty()) {

std::cout << "Stack is not empty!" << std::endl;

}

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

std::cout << "Stack Array printStack Function Tests: " << std::endl;

std::cout << "Print Stack function demonstrated through all tests" << std::endl;

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

std::cout << "Stack Array overloaded assignment Tests: " << std::endl;

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

intStack.printStack();

SUStackArr<int> intStack3;

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

intStack3.printStack();

intStack3 = intStack;

std::cout << "First copied to Second: " << std::endl;

intStack3.printStack();

SUStackArr<int> intStack4;

intStack4.push(41);

intStack4.push(88);

intStack4.push(92);

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

intStack4.printStack();

std::cout<< "Third copied to First: " << std::endl;

intStack = intStack4;

intStack.printStack();

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

std::cout << "Stack Array copy constructor Tests: " << std::endl;

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

intStack.printStack();

std::cout << "New list Initialized: " << std::endl;

SUStackArr<int> intStack5(intStack);

intStack5.printStack();

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

SUStackArr<double> doubleStack;

std::cout << "Double Stack Tests: " << std::endl;

std::cout << "Stack Array push Function Tests: " << std::endl;

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

doubleStack.printStack();

doubleStack.push(5.25);

doubleStack.push(8.6);

doubleStack.push(25.2);

doubleStack.push(7.898);

std::cout << "First Stack after 4 pushes: " << std::endl;

doubleStack.printStack();

doubleStack.push(40.32);

doubleStack.push(2.2);

doubleStack.push(4.5);

std::cout << "First Stack, auto resize then 3 more pushes: " << std::endl;

doubleStack.printStack();

doubleStack.push(16.56);

doubleStack.push(1000.0);

doubleStack.push(4500.7);

std::cout << "First Stack, second auto resize then 2 more push: " << std::endl;

doubleStack.printStack();

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

std::cout << "Stack Array pop Function Tests: " << std::endl;

SUStackArr<double> doubleStack2;

doubleStack2.push(10.2);

doubleStack2.push(25.65);

doubleStack2.push(0.7);

std::cout << "Current Stack being worked with: " << std::endl;

doubleStack2.printStack();

double pop4, pop5, pop6;

doubleStack2.pop(pop4);

doubleStack2.pop(pop5);

doubleStack2.pop(pop6);

std::cout << "First pop: " << pop4 << "\nSecond pop: " << pop5 << "\nThird pop: " << pop6 << std::endl;

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

doubleStack2.printStack();

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

std::cout << "Stack Array size Function Tests: " << std::endl;

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

doubleStack2.printStack();

std::cout << "Size of stack: " << doubleStack2.size() << std::endl;

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

doubleStack.printStack();

std::cout << "Size of stack: " << doubleStack.size() << std::endl;

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

std::cout << "Stack Array isEmpty Function Tests: " << std::endl;

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

doubleStack2.printStack();

if (doubleStack2.isEmpty()) {

std::cout << "Stack is empty!" << std::endl;

}

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

doubleStack.printStack();

if (!doubleStack.isEmpty()) {

std::cout << "Stack is not empty!" << std::endl;

}

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

std::cout << "Stack Array printStack Function Tests: " << std::endl;

std::cout << "Print Stack function demonstrated through all tests" << std::endl;

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

std::cout << "Stack Array overloaded assignment Tests: " << std::endl;

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

doubleStack.printStack();

SUStackArr<double> doubleStack3;

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

doubleStack3.printStack();

doubleStack3 = doubleStack;

std::cout << "First copied to Second: " << std::endl;

doubleStack3.printStack();

SUStackArr<double> doubleStack4;

doubleStack4.push(41.4);

doubleStack4.push(88.01);

doubleStack4.push(92.43);

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

doubleStack4.printStack();

std::cout<< "Third copied to First: " << std::endl;

doubleStack = doubleStack4;

doubleStack.printStack();

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

std::cout << "Stack Array copy constructor Tests: " << std::endl;

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

doubleStack.printStack();

std::cout << "New list Initialized: " << std::endl;

SUStackArr<double> doubleStack5(doubleStack);

doubleStack5.printStack();

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

SUStackArr<PayRoll> payrollStack;

PayRoll p1("Ryan", 25, 32.4);

PayRoll p2("James", 21, 23);

PayRoll p3("Skylar", 55, 50);

PayRoll p4("Margaret", 1, 23);

PayRoll p5("Hemp", 20, 22);

PayRoll p6("Leonardo", 25, 28);

PayRoll p7("Jamison", 22, 59);

PayRoll p8("Cricket", 45, 50);

PayRoll p9("Elena", 30, 55);

PayRoll p10("Andrea", 15, 55);

std::cout << "PayRoll Stack Tests: " << std::endl;

std::cout << "Stack Array push Function Tests: " << std::endl;

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

payrollStack.printStack();

payrollStack.push(p1);

payrollStack.push(p2);

payrollStack.push(p3);

payrollStack.push(p4);

std::cout << "First Stack after 4 pushes: " << std::endl;

payrollStack.printStack();

payrollStack.push(p5);

payrollStack.push(p6);

payrollStack.push(p7);

std::cout << "First Stack, auto resize then 3 more pushes: " << std::endl;

payrollStack.printStack();

payrollStack.push(p8);

payrollStack.push(p9);

payrollStack.push(p10);

std::cout << "First Stack, second auto resize then 2 more push: " << std::endl;

payrollStack.printStack();

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

std::cout << "Stack Array pop Function Tests: " << std::endl;

SUStackArr<PayRoll> payrollStack2;

payrollStack2.push(p10);

payrollStack2.push(p2);

payrollStack2.push(p3);

std::cout << "Current Stack being worked with: " << std::endl;

payrollStack2.printStack();

PayRoll p11, p12, p13;

payrollStack2.pop(p11);

payrollStack2.pop(p12);

payrollStack2.pop(p13);

std::cout << "First pop: " << p11 << "\nSecond pop: " << p12 << "\nThird pop: " << p13 << std::endl;

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

payrollStack2.printStack();

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

std::cout << "Stack Array size Function Tests: " << std::endl;

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

payrollStack2.printStack();

std::cout << "Size of stack: " << payrollStack2.size() << std::endl;

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

payrollStack.printStack();

std::cout << "Size of stack: " << payrollStack.size() << std::endl;

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

std::cout << "Stack Array isEmpty Function Tests: " << std::endl;

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

payrollStack2.printStack();

if (payrollStack2.isEmpty()) {

std::cout << "Stack is empty!" << std::endl;

}

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

payrollStack.printStack();

if (!payrollStack.isEmpty()) {

std::cout << "Stack is not empty!" << std::endl;

}

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

std::cout << "Stack Array printStack Function Tests: " << std::endl;

std::cout << "Print Stack function demonstrated through all tests" << std::endl;

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

std::cout << "Stack Array overloaded assignment Tests: " << std::endl;

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

payrollStack.printStack();

SUStackArr<PayRoll> payrollStack3;

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

payrollStack3.printStack();

payrollStack3 = payrollStack;

std::cout << "First copied to Second: " << std::endl;

payrollStack3.printStack();

SUStackArr<PayRoll> payrollStack4;

payrollStack4.push(p7);

payrollStack4.push(p1);

payrollStack4.push(p5);

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

payrollStack4.printStack();

std::cout<< "Third copied to First: " << std::endl;

payrollStack = payrollStack4;

payrollStack.printStack();

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

std::cout << "Stack Array copy constructor Tests: " << std::endl;

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

payrollStack.printStack();

std::cout << "New list Initialized: " << std::endl;

SUStackArr<PayRoll> payrollStack5(payrollStack);

payrollStack5.printStack();

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

// SUStack tests, SUList in SUQueueDriver file

/\*

\* @Author: Grant Dawson

\*/

PayRoll x;

int y;

SUList<int> myList;

SUList<PayRoll> pList;

SUStackList<int> iStackList;

SUStackList<PayRoll> pStackList;

SUStackList<PayRoll> pStackList2;

/\*\*

\* Test the SUList using ints

\*/

myList.putFront(1);

myList.putFront(6);

myList.putFront(2);

myList.putBack(4);

myList.putBack(5);

myList.putFront(3);

std::cout<<"Does myList contain 100?"<<std::endl;

if(myList.contains(100)){

std::cout<<"100 is in the list"<<std::endl;

}else{

std::cout<<"100 not in the list"<<std::endl;

}

myList.display();

std::cout << "++++++++++++\n";

/\*

\* Test the SUList using PayRoll

\*/

pList.putFront(PayRoll("Bob", 20, 35));

pList.putFront(PayRoll("Alice", 10, 35));

pList.putBack(PayRoll("Diana", 25, 35));

pList.putBack(PayRoll("Diana", 10, 35));

pList.putFront(PayRoll("Eve", 30, 35));

pList.display();

std::cout << "++++++++++++\n";

/\*\*

\* Test the SUStack using PayRoll

\*/

pStackList.push(PayRoll("Bob", 20, 35));

pStackList.push(PayRoll("Alice", 10, 35));

pStackList.push(PayRoll("Charlie", 25, 35));

pStackList.push(PayRoll("Diana", 10, 35));

pStackList.push(PayRoll("Eve", 30, 35));

pStackList.pop(x);

pStackList.printStack();

std::cout<<"+++++++ popped lad +++++++++"<<std::endl;

x.printInfo();

std::cout << "++++++++++++\n";

pStackList.printStack();

std::cout << "++++++++++++\n";

iStackList.push(10);

iStackList.push(102);

iStackList.push(1);

iStackList.push(50);

iStackList.printStack();

std::cout << "+++++++ =operator +++++++++\n";

pStackList2 = pStackList;

pStackList2.printStack();

return 0;

}

**SUQueueDriver.cpp:**

#include "SUQueue.cpp"

#include "payroll.h"

#include "SUList.h"

#include <iostream>

#include <stdio.h>

int main() {

/\*

\* @Author: Ryan Rosiak

\*/

SUQueueArr<int> intQueue;

std::cout << "Int Queue Tests: " << std::endl;

std::cout << "Queue Array enqueue Function Tests: " << std::endl;

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

intQueue.printQueue();

intQueue.enqueue(5);

intQueue.enqueue(8);

intQueue.enqueue(25);

intQueue.enqueue(7);

std::cout << "First Queue after 4 enqueues: " << std::endl;

intQueue.printQueue();

intQueue.enqueue(40);

intQueue.enqueue(2);

intQueue.enqueue(4);

std::cout << "First Queue, auto resize then 3 more enqueues: " << std::endl;

intQueue.printQueue();

intQueue.enqueue(16);

intQueue.enqueue(1000);

intQueue.enqueue(4500);

std::cout << "First Queue, second auto resize then 2 more enqueues: " << std::endl;

intQueue.printQueue();

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

std::cout << "Queue Array dequeue Function Tests: " << std::endl;

SUQueueArr<int> intQueue2;

intQueue2.enqueue(10);

intQueue2.enqueue(25);

intQueue2.enqueue(0);

std::cout << "Current Queue being worked with: " << std::endl;

intQueue2.printQueue();

int deq1, deq2, deq3;

intQueue2.dequeue(deq1);

intQueue2.dequeue(deq2);

intQueue2.dequeue(deq3);

std::cout << "First dequeue: " << deq1 << "\nSecond dequeue: " << deq2 << "\nThird dequeue: " << deq3 << std::endl;

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

intQueue2.printQueue();

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

std::cout << "Queue Array size Function Tests: " << std::endl;

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

intQueue2.printQueue();

std::cout << "Size of queue: " << intQueue2.size() << std::endl;

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

intQueue.printQueue();

std::cout << "Size of queue: " << intQueue.size() << std::endl;

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

std::cout << "Queue Array isEmpty Function Tests: " << std::endl;

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

intQueue2.printQueue();

if (intQueue2.isEmpty()) {

std::cout << "Queue is empty!!!" << std::endl;

}

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

intQueue.printQueue();

if (!intQueue.isEmpty()) {

std::cout << "Queue is not empty!!!" << std::endl;

}

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

std::cout << "Queue Array printQueue Function Tests: " << std::endl;

std::cout << "Print Queue function demonstrated through all tests" << std::endl;

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

std::cout << "Queue Array overloaded assignment Tests: " << std::endl;

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

intQueue.printQueue();

SUQueueArr<int> intQueue3;

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

intQueue3.printQueue();

intQueue3 = intQueue;

std::cout << "First copied to Second: " << std::endl;

intQueue3.printQueue();

SUQueueArr<int> intQueue4;

intQueue4.enqueue(41);

intQueue4.enqueue(88);

intQueue4.enqueue(92);

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

intQueue4.printQueue();

std::cout << "Third copied to First: " << std::endl;

intQueue = intQueue4;

intQueue.printQueue();

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

std::cout << "Queue Array copy constructor Tests: " << std::endl;

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

intQueue.printQueue();

std::cout << "New list Initialized: " << std::endl;

SUQueueArr<int> intQueue5(intQueue);

intQueue5.printQueue();

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

SUQueueArr<double> doubleQueue;

std::cout << "Double Queue Tests: " << std::endl;

std::cout << "Queue Array enqueue Function Tests: " << std::endl;

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

doubleQueue.printQueue();

doubleQueue.enqueue(5.2);

doubleQueue.enqueue(8.4);

doubleQueue.enqueue(25.621);

doubleQueue.enqueue(7.0);

std::cout << "First Queue after 4 enqueues: " << std::endl;

doubleQueue.printQueue();

doubleQueue.enqueue(40.15);

doubleQueue.enqueue(2.2);

doubleQueue.enqueue(4.441);

std::cout << "First Queue, auto resize then 3 more enqueues: " << std::endl;

doubleQueue.printQueue();

doubleQueue.enqueue(16.5);

doubleQueue.enqueue(1000.5);

doubleQueue.enqueue(4500.616);

std::cout << "First Queue, second auto resize then 2 more enqueues: " << std::endl;

doubleQueue.printQueue();

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

std::cout << "Queue Array dequeue Function Tests: " << std::endl;

SUQueueArr<double> doubleQueue2;

doubleQueue2.enqueue(10.14);

doubleQueue2.enqueue(25.18);

doubleQueue2.enqueue(0.0);

std::cout << "Current Queue being worked with: " << std::endl;

doubleQueue2.printQueue();

double deq4, deq5, deq6;

doubleQueue2.dequeue(deq4);

doubleQueue2.dequeue(deq5);

doubleQueue2.dequeue(deq6);

std::cout << "First dequeue: " << deq4 << "\nSecond dequeue: " << deq5 << "\nThird dequeue: " << deq6 << std::endl;

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

doubleQueue2.printQueue();

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

std::cout << "Queue Array size Function Tests: " << std::endl;

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

doubleQueue2.printQueue();

std::cout << "Size of queue: " << doubleQueue2.size() << std::endl;

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

doubleQueue.printQueue();

std::cout << "Size of queue: " << doubleQueue.size() << std::endl;

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

std::cout << "Queue Array isEmpty Function Tests: " << std::endl;

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

doubleQueue2.printQueue();

if (doubleQueue2.isEmpty()) {

std::cout << "Queue is empty!!!" << std::endl;

}

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

doubleQueue.printQueue();

if (!doubleQueue.isEmpty()) {

std::cout << "Queue is not empty!!!" << std::endl;

}

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

std::cout << "Queue Array printQueue Function Tests: " << std::endl;

std::cout << "Print Queue function demonstrated through all tests" << std::endl;

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

std::cout << "Queue Array overloaded assignment Tests: " << std::endl;

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

doubleQueue.printQueue();

SUQueueArr<double> doubleQueue3;

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

doubleQueue3.printQueue();

doubleQueue3 = doubleQueue;

std::cout << "First copied to Second: " << std::endl;

doubleQueue3.printQueue();

SUQueueArr<double> doubleQueue4;

doubleQueue4.enqueue(41.5);

doubleQueue4.enqueue(88.991);

doubleQueue4.enqueue(92.78);

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

doubleQueue4.printQueue();

std::cout << "Third copied to First: " << std::endl;

doubleQueue = doubleQueue4;

doubleQueue.printQueue();

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

std::cout << "Queue Array copy constructor Tests: " << std::endl;

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

doubleQueue.printQueue();

std::cout << "New list Initialized: " << std::endl;

SUQueueArr<double> doubleQueue5(doubleQueue);

doubleQueue5.printQueue();

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

SUQueueArr<PayRoll> payrollQueue;

PayRoll p1("Ryan", 25, 32.4);

PayRoll p2("James", 21, 23);

PayRoll p3("Skylar", 55, 50);

PayRoll p4("Margaret", 1, 23);

PayRoll p5("Hemp", 20, 22);

PayRoll p6("Leonardo", 25, 28);

PayRoll p7("Jamison", 22, 59);

PayRoll p8("Cricket", 45, 50);

PayRoll p9("Elena", 30, 55);

PayRoll p10("Andrea", 15, 55);

std::cout << "PayRoll Queue Tests: " << std::endl;

std::cout << "Queue Array enqueue Function Tests: " << std::endl;

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

payrollQueue.printQueue();

payrollQueue.enqueue(p1);

payrollQueue.enqueue(p2);

payrollQueue.enqueue(p3);

payrollQueue.enqueue(p4);

std::cout << "First Queue after 4 enqueues: " << std::endl;

payrollQueue.printQueue();

payrollQueue.enqueue(p5);

payrollQueue.enqueue(p6);

payrollQueue.enqueue(p7);

std::cout << "First Queue, auto resize then 3 more enqueues: " << std::endl;

payrollQueue.printQueue();

payrollQueue.enqueue(p8);

payrollQueue.enqueue(p9);

payrollQueue.enqueue(p10);

std::cout << "First Queue, second auto resize then 2 more enqueues: " << std::endl;

payrollQueue.printQueue();

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

std::cout << "Queue Array dequeue Function Tests: " << std::endl;

SUQueueArr<PayRoll> payrollQueue2;

payrollQueue2.enqueue(p10);

payrollQueue2.enqueue(p2);

payrollQueue2.enqueue(p3);

std::cout << "Current Queue being worked with: " << std::endl;

payrollQueue2.printQueue();

PayRoll p11, p12, p13;

payrollQueue2.dequeue(p11);

payrollQueue2.dequeue(p12);

payrollQueue2.dequeue(p13);

std::cout << "First dequeue: " << p11 << "\nSecond dequeue: " << p12 << "\nThird dequeue: " << p13 << std::endl;

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

payrollQueue2.printQueue();

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

std::cout << "Queue Array size Function Tests: " << std::endl;

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

payrollQueue2.printQueue();

std::cout << "Size of queue: " << payrollQueue2.size() << std::endl;

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

payrollQueue.printQueue();

std::cout << "Size of queue: " << payrollQueue.size() << std::endl;

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

std::cout << "Queue Array isEmpty Function Tests: " << std::endl;

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

payrollQueue2.printQueue();

if (payrollQueue2.isEmpty()) {

std::cout << "Queue is empty!!!" << std::endl;

}

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

payrollQueue.printQueue();

if (!payrollQueue.isEmpty()) {

std::cout << "Queue is not empty!!!" << std::endl;

}

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

std::cout << "Queue Array printQueue Function Tests: " << std::endl;

std::cout << "Print Queue function demonstrated through all tests" << std::endl;

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

std::cout << "Queue Array overloaded assignment Tests: " << std::endl;

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

payrollQueue.printQueue();

SUQueueArr<PayRoll> payrollQueue3;

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

payrollQueue3.printQueue();

payrollQueue3 = payrollQueue;

std::cout << "First copied to Second: " << std::endl;

payrollQueue3.printQueue();

SUQueueArr<PayRoll> payrollQueue4;

payrollQueue4.enqueue(p7);

payrollQueue4.enqueue(p1);

payrollQueue4.enqueue(p5);

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

payrollQueue4.printQueue();

std::cout << "Third copied to First: " << std::endl;

payrollQueue = payrollQueue4;

payrollQueue.printQueue();

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

std::cout << "Queue Array copy constructor Tests: " << std::endl;

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

payrollQueue.printQueue();

std::cout << "New list Initialized: " << std::endl;

SUQueueArr<PayRoll> payrollQueue5(payrollQueue);

payrollQueue5.printQueue();

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

// SUQueue tests and SUList tests

/\*

\* @Author: Grant Dawson

\*/

PayRoll x;

int y;

SUList<int> myList;

SUList<PayRoll> pList;

SUQueueList<int> iQueueList; // TODO

SUQueueList<PayRoll> pQueueList; // TODO

/\*\*

\* Test the SUList using ints

\*/

myList.putFront(1);

myList.putFront(6);

myList.putFront(2);

myList.putBack(4);

myList.putBack(5);

myList.putFront(3);

std::cout<<"Does myList contain 100?"<<std::endl;

if(myList.contains(100)){

std::cout<<"100 is in the list"<<std::endl;

}else{

std::cout<<"100 not in the list"<<std::endl;

}

myList.display();

std::cout << "++++++++++++\n";

/\*

\* Test the SUList using PayRoll

\*/

pList.putFront(PayRoll("Bob", 20, 35));

pList.putFront(PayRoll("Alice", 10, 35));

pList.putBack(PayRoll("Charlie", 25, 35));

pList.putBack(PayRoll("Diana", 10, 35));

pList.putFront(PayRoll("Eve", 30, 35));

pList.display();

std::cout << "++++++++++++\n";

std::cout << "++++++++++++\n";

std::cout << "+++++++ Queue Lad +++++++++\n";

iQueueList.enqueue(1000);

iQueueList.enqueue(1523);

iQueueList.enqueue(2);

iQueueList.dequeue(y);

iQueueList.printQueue();

std::cout << "++++++++++++\n";

pQueueList.enqueue(PayRoll("Bob", 20, 35));

pQueueList.enqueue(PayRoll("Alice", 10, 35));

pQueueList.enqueue(PayRoll("Charlie",25, 35));

pQueueList.enqueue(PayRoll("Diana", 10, 35));

pQueueList.enqueue(PayRoll("Eva", 30, 35));

pQueueList.printQueue();

pQueueList.dequeue(x);

std::cout<<"+++++++ popped lad +++++++++"<<std::endl;

x.printInfo();

std::cout << "++++++++++++\n";

pQueueList.printQueue();

return 0;

}

**SUList.h:**

#ifndef SULIST\_H

#define SULIST\_H

#include <iostream>

#include "PayRoll.h"

template <class DataType>

class SUList{

private:

struct ListNode{ // The nodes of the list

ListNode\* previous;//TODO make it double linked

DataType data; // The data stored in the node

ListNode\* next; // The next node in the list

};

ListNode\* head; // The front of the list

ListNode\* tail; // The last node of the list

public:

SUList(); // Default ctor

SUList(const SUList&); // Copy ctor

~SUList(); // Destructor

DataType getFront(); // Remove & return the front

DataType getBack(); // Remove & return the back

void putFront(const DataType&);// Add to the front

void putBack(const DataType&); // Add to the back

int size() const; // Returns the number of elements

bool contains(const DataType&);// Tests for membership

SUList<DataType>& operator=(const SUList<DataType>&); // Overloaded assignment

void display() const;

};

#include "SUList.cpp"

#endif

**SUList.cpp:**

//defualt constructor

template<class DataType>

SUList<DataType>::SUList(){

head=nullptr;

tail=nullptr;

}

//Copy constructor

template<class DataType>

SUList<DataType>::SUList(const SUList& old){

if(old.head==nullptr){//checks to see if there is a old head

head=nullptr;

tail=nullptr;

}else{//if there is then

ListNode\* oldCursor=old.head;

while(oldCursor){//pushes everything over

putBack(oldCursor->data);

oldCursor=oldCursor->next;

}

}

};

//equals overloaded operator

template<class DataType>

SUList<DataType>& SUList<DataType>::operator=(const SUList<DataType>& old){

if(this==&old){

return \*(this);

}

ListNode\* oldCursor=old.head;

while(oldCursor){

putBack(oldCursor->data);

oldCursor=oldCursor->next;

}

return \*(this);

}

//Deconstructor

template<class DataType>

SUList<DataType>::~SUList(){

ListNode\* cursor=head;

while(head){

cursor=head;

head=head->next;

delete cursor;

}

}

//Pops the front/head

// template<class DataType>

// DataType SUList<DataType>::getFront(){

// ListNode\* cursor=head;

// DataType rtn;

// //TODO Need to fix if there is no head

// if(!head){

// throw "Queue is empty!";

// }

// if(!head->next){

// rtn=head->data;

// delete head;

// head=nullptr;

// tail=nullptr;

// return rtn;

// }

// rtn=head->data;

// head=head->next;

// delete cursor;

// return rtn;

// }

template<class DataType>

DataType SUList<DataType>::getFront(){

ListNode\* cursor=head;

DataType rtn;

//TODO Need to fix if there is no head

if(!head){

throw "Queue is empty!";

}

if(!head->next){

rtn=head->data;

delete head;

head=nullptr;

tail=nullptr;

return rtn;

}

rtn=head->data;

head=head->next;

delete cursor;

return rtn;

}

//pops the end of the list

// template<class DataType>

// DataType SUList<DataType>::getBack(){

// ListNode\* cursor=head;

// ListNode\* prev=head;

// DataType temp;

// if(!head){//if the stack is empty when it tries to pop

// throw "Queue is empty!";

// }

// if(!head->next){//if there is only one item left than pop it

// temp = head->data;

// delete head;

// head=nullptr;

// return temp;

// }

// while(cursor->next){//finds end with a prev

// prev=cursor;

// cursor=cursor->next;

// }

// prev->next=nullptr;

// temp = cursor->data;

// delete cursor;//detatches and deltes end but before it sends it

// return temp;

// }

template<class DataType>

DataType SUList<DataType>::getBack(){

ListNode\* cursor=head;

ListNode\* prev=head;

DataType temp;

if(!head){//if the stack is empty when it tries to pop

throw "Queue is empty!";

}

if(head->next==tail){

cursor=tail;

temp=tail->data;

head->next=nullptr;

delete cursor;

return temp;

}

if(head==tail){

temp=head->data;

delete head;

return temp;

}

temp=tail->data;

prev=tail;

cursor=tail->previous;

cursor->next=nullptr;

delete prev;

return temp;

}

//This will push a new dataType onto the front of the list

// template<class DataType>

// void SUList<DataType>::putFront(const DataType&x){

// ListNode\* newNode = new ListNode;

// newNode->data=x;

// newNode->next=head;

// newNode->previous=nullptr;

// //head->prev

// head=newNode;

// }

template<class DataType>

void SUList<DataType>::putFront(const DataType&x){

ListNode\* newNode = new ListNode;

newNode->data=x;

if(!head){

head=newNode;

tail=newNode;

newNode->next=nullptr;

newNode->previous=nullptr;

}

newNode->next=head;

newNode->previous=nullptr;

head->previous=newNode;

head=newNode;

}

//This will push a new Datatype into the bakc of the list

// template<class DataType>

// void SUList<DataType>::putBack(const DataType&x){

// ListNode\* newNode = new ListNode;

// newNode->data=x;

// newNode->next=nullptr;

// tail=newNode;

// if(!head){

// head=newNode;

// return;

// }

// ListNode\* cursor=head;

// while(cursor->next){

// cursor=cursor->next;

// }

// cursor->next=newNode;

// }

template<class DataType>

void SUList<DataType>::putBack(const DataType&x){

ListNode\* newNode = new ListNode;

newNode->data=x;

newNode->next=nullptr;

if(!head){

head=newNode;

tail=newNode;

newNode->next=nullptr;

newNode->previous=nullptr;

}

newNode->previous=tail;

tail->next=newNode;

tail=newNode;

}

//checks to see if there is a desired DataType in the list

//NOTE: This function will require that the datta type has the == operator

template<class DataType>

bool SUList<DataType>::contains(const DataType&x){

ListNode\* cursor=head;

while(cursor){

if(cursor->data==x){

return true;

}

cursor=cursor->next;

}

return false;

}

//Display requires the overloaded << or you make your own overloaded tamplate

template<class DataType>

void SUList<DataType>::display() const{

ListNode\* cursor=head;

if(!head){

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

return;

}

while(cursor){

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

cursor=cursor->next;

}

}

//Display overloaded for Payroll

template<>

void SUList<PayRoll>::display() const{

ListNode\* cursor=head;

if(!head){

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

return;

}

while(cursor){

cursor->data.printInfo();

cursor=cursor->next;

}

}

**SUQueue.h:**

#ifndef \_SUQUEUE\_H\_

#define \_SUQUEUE\_H\_

#include <iostream>

#include "SUList.h"

/\*

\* @Author: Ryan Rosiak

\*/

template<class T>

class SUQueueArr {

private:

T\* arr;

int capacity;

int front;

int rear;

void enlarge();

void moveUp();

public:

SUQueueArr(); // Constructor

SUQueueArr(const SUQueueArr&); // Copy Constructor

~SUQueueArr(); // Destructor

int size() const; // Gets the number of elements in the queue

bool isEmpty() const; // Checks if the queue is empty

void enqueue(const T&); // Enqueues some data to the back of the queue

void dequeue(T&); // Gets the front element and stores it into the passed parameter

void printQueue() const; // Prints the queue from the front to the rear

SUQueueArr<T>& operator=(const SUQueueArr<T>&); // Overloaded Assignment

};

/\*

\* @Author: Grant Dawson

\*/

template <class DataType>

class SUQueueList{

private:

SUList<DataType> list;

public:

SUQueueList(); // Constructor

SUQueueList(const SUQueueList &); // Copy Constructor

~SUQueueList(); // Destructor

int size() const; // get the number of elements in the queue

bool isEmpty() const; // Check if the queue is empty

void enqueue(const DataType&); // Enqueues some data

void dequeue(DataType&); // Get the front element and store it

void printQueue() const; // Prints the queue from the front to the rear

SUQueueList<DataType>& operator=(const SUQueueList<DataType>&); // Assignment operator

};

#endif

**SUQueue.cpp:**

#include "SUQueue.h"

/\*

\* @Author: Ryan Rosiak

\* Default Constructor

\* No inputs or outputs

\* Initializes all values to default in queue

\*/

template<class T>

SUQueueArr<T>::SUQueueArr() {

arr = new T[4];

capacity = 4;

front = -1;

rear = -1;

}

/\*

\* @Author: Ryan Rosiak

\* enlarge Function

\* No inputs or outputs

\* Dynamically allocates more space for the queue

\*/

template<class T>

void SUQueueArr<T>::enlarge() {

capacity \*= 2;

T\* buff = new T[capacity];

for (int i = 0; i <= rear; i++) {

buff[i] = arr[i];

}

delete [] arr;

arr = buff;

}

/\*

\* @Author: Ryan Rosiak

\* moveUp Function

\* No inputs or ouputs

\* Shifts all of the elements in the queue array to the front of the allocated array space

\*/

template<class T>

void SUQueueArr<T>::moveUp() {

T\* buff = new T[capacity];

for (int i = 0; i <= rear; i++) {

buff[i] = arr[i + 1];

}

delete [] arr;

arr = buff;

}

/\*

\* @Author: Ryan Rosiak

\* Copy Constructor

\* No inputs or outputs

\* Copies the initialized queue into another queue

\*/

template<class T>

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

arr = new T[rhs.capacity];

capacity = rhs.capacity;

front = -1;

rear = -1;

for (int i = 0; i <= rhs.rear; i++) {

enqueue(rhs.arr[i]);

}

}

/\*

\* @Author: Ryan Rosiak

\* Destructor

\* No inputs or outputs

\* Deallocates dynamic memory

\*/

template<class T>

SUQueueArr<T>::~SUQueueArr() {

delete [] arr;

}

/\*

\* @Author: Ryan Rosiak

\* size Function

\* No inputs, returns an int

\* Gets the number of elements int the queue and returns an int

\*/

template<class T>

int SUQueueArr<T>::size() const {

return rear + 1;

}

/\*

\* @Author: Ryan Rosiak

\* isEmpty Function

\* No inputs, returns a boolean

\* Checks to see if the queue is empty or not

\*/

template<class T>

bool SUQueueArr<T>::isEmpty() const {

if (front == -1) {

return true;

}

return false;

}

/\*

\* @Author: Ryan Rosiak

\* enqueue Function

\* No outputs, input a class T data member to be put in the back of the queue

\* Places an item to the back of the queue

\*/

template<class T>

void SUQueueArr<T>::enqueue(const T& obj) {

if (rear == (capacity - 1)) {

enlarge();

}

if (front == -1) {

front++;

}

arr[++rear] = obj;

}

/\*

\* @Author: Ryan Rosiak

\* dequeue Function

\* No outputs, takes a reference to a class T data member and assigns the first value in the

\* queue to it

\* Dequeue's the front value from the queue

\*/

template<class T>

void SUQueueArr<T>::dequeue(T& assign) {

if (front == -1) {

std::cout << "Queue is empty" << std::endl;

return;

}

assign = arr[front];

if (rear >= 1) {

moveUp();

}

rear--;

if (rear == -1) {

front = -1;

}

}

/\*

\* @Author: Ryan Rosiak

\* printQueue Function

\* No inputs, outputs the queue to the screen

\* Prints the contents of the queue to the screen

\*/

template<class T>

void SUQueueArr<T>::printQueue() const {

if (front == -1) {

std::cout << "Queue is Empty" << std::endl;

return;

}

std::cout << "Front:" << std::endl;

for (int i = front; i <= rear; i++) {

std::cout << arr[i] << std::endl;

}

std::cout << "Back:" << std::endl;

}

/\*

\* @Author: Ryan Rosiak

\* Overloaded Assigment

\* No outputs, takes a reference to a queue array of the same type

\* Deallocates memory for queue being assigned to and then input paramter is copied over

\*/

template<class T>

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

if (this == &rhs) {

return \*this;

}

delete [] arr;

arr = new T[rhs.capacity];

capacity = rhs.capacity;

front = -1;

rear = -1;

for (int i = 0; i <= rhs.rear; i++) {

enqueue(rhs.arr[i]);

}

return \*this;

}

/\*

\* @Author: Grant Dawson

\*/

//defualt constructor

template<class DataType>

SUQueueList<DataType>::SUQueueList(){}

//Deconstructor

template<class DataType>

SUQueueList<DataType>::~SUQueueList(){

//delete list;

}

//Copy constructor

template<class DataType>

SUQueueList<DataType>::SUQueueList(const SUQueueList &old){

list=old.list;

}

//returns th current size of the queue at the time of call

template<class DataType>

int SUQueueList<DataType>::size() const{

return list.size();

}

//This will return if the queue is empty or not

template<class DataType>

bool SUQueueList<DataType>::isEmpty() const{

if(list.size()==0)

return true;

return false;

}

//This will add a new Node to the front of the queue

template<class DataType>

void SUQueueList<DataType>::enqueue(const DataType& x){

list.putBack(x);

}

//This will pop the front of the queue off and return it

template<class DataType>

void SUQueueList<DataType>::dequeue(DataType& x){

try{

x=list.getFront();

}catch(std::string n){

throw "Queue is empty";

}catch(...){

throw "An unknown error has occured...";

}

}

//This will print the qholw queue

template<class DataType>

void SUQueueList<DataType>::printQueue() const{

list.display();

}

//

template<class DataType>

SUQueueList<DataType>& SUQueueList<DataType>::operator=(const SUQueueList<DataType> &old){

if(this==old){

return \*this;

}

list=old.list;

return \*this;

}

**SUStack.h:**

#ifndef \_SUSTACK\_H\_

#define \_SUSTACK\_H\_

#include <iostream>

#include "SUList.h"

/\*

\* @Author: Ryan Rosiak

\* Stack Array Class Definition

\* Implements a stack algorithm on a dynamic array

\*/

template<class T>

class SUStackArr {

private:

T\* arr;

int capacity;

int top;

void enlarge(); // Resizes the array stack and makes it bigger for more data

public:

SUStackArr(); // Constructor

SUStackArr(const SUStackArr&); // Copy Constructor

~SUStackArr(); // Destructor

int size() const; // Get the number of elements in the stack

bool isEmpty() const; // Check if the stack is empty

void push(const T&); // Pushes an object onto the stack

void pop(T&); // Pop an object off the stack and store it

void printStack() const; // Prints the stack from the top, down

SUStackArr<T>& operator=(const SUStackArr<T>&); // Overloaded Assignment

};

/\*

\* @Author: Grant Dawson

\*/

template<class DataType>

class SUStackList{

private:

SUList<DataType> list;

public:

SUStackList(); // Constructor

SUStackList(const SUStackList &); // Copy Constructor

~SUStackList(); // Destructor

int size() const; // get the number of elements in the stack

bool isEmpty() const; // Check if the stack is empty

void push(const DataType&); // Pushes an object onto the stack

void pop(DataType&); // Pop an object off the stack and store it

void printStack() const; // Prints the stack from the top, down

SUStackList<DataType>& operator=(const SUStackList<DataType>&); // Assignment operator

};

#endif

**SUStack.cpp:**

#include "SUStack.h"

/\*

\* @Author: Ryan Rosiak

\* Default Constructor

\* No inputs or outputs

\* Allocates memory for array of type T

\*/

template<class T>

SUStackArr<T>::SUStackArr() {

arr = new T[4];

capacity = 4;

top = -1;

}

/\*

\* @Author: Ryan Rosiak

\* enlarge Function

\* No inputs or outputs

\* Resizes the array stack and makes it bigger so more data can be stored

\*/

template<class T>

void SUStackArr<T>::enlarge() {

capacity \*= 2;

T\* buff = new T[capacity];

for (int i = 0; i <= top; i++) {

buff[i] = arr[i];

}

delete [] arr;

arr = buff;

}

/\*

\* @Author: Ryan Rosiak

\* Copy Constructor

\* No inputs or outputs

\* Allocates memory for array of type T assigned from another array of type T

\*/

template<class T>

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

arr = new T[rhs.capacity];

capacity = rhs.capacity;

top = -1;

for (int i = 0; i <= rhs.top; i++) {

push(rhs.arr[i]);

}

}

/\*

\* @Author: Ryan Rosiak

\* Destructor

\* No inputs or outputs

\* Deallocates memory of current stack array

\*/

template<class T>

SUStackArr<T>::~SUStackArr() {

delete [] arr;

}

/\*

\* @Author: Ryan Rosiak

\* size Function

\* No inputs, returns an int

\* Gets the number of elements in the stack and returns the counter

\*/

template<class T>

int SUStackArr<T>::size() const {

return top + 1;

}

/\*

\* @Author: Ryan Rosiak

\* isEmpty Function

\* No inputs, returns a boolean

\* Checks if the stack is empty

\*/

template<class T>

bool SUStackArr<T>::isEmpty() const {

if (top == -1) {

return true;

}

return false;

}

/\*

\* @Author: Ryan Rosiak

\* push Function

\* No outputs, takes a type T argument/object by reference that is pushed onto the stack array

\* Pushes an object of type T onto the stack in the array

\*/

template<class T>

void SUStackArr<T>::push(const T& obj) {

if (top == (capacity - 1)) {

enlarge();

}

arr[++top] = obj;

}

/\*

\* @Author: Ryan Rosiak

\* pop Function

\* No outputs, takes a type T argument reference and assigns the popped value to it

\* Assigns the popped value off the stack to the argument passed through

\*/

template<class T>

void SUStackArr<T>::pop(T& assign) {

if (top == -1) {

std::cout << "Stack is empty" << std::endl;

return;

}

assign = arr[top];

top--;

}

/\*

\* @Author: Ryan Rosiak

\* printStack Function

\* No inputs, outputs the array stack contents from top to bottom

\* Outputs the array contents from top to bottom

\*/

template<class T>

void SUStackArr<T>::printStack() const {

if (top == -1) {

std::cout << "Stack is Empty" << std::endl;

return;

}

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

for (int i = top; i >= 0; i--) {

std::cout << arr[i] << std::endl;

}

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

}

/\*

\* @Author: Ryan Rosiak

\* Overloaded Assignment

\* No inputs or outputs

\* Assigns the right hand side array stack to the left hand side array stack

\*/

template<class T>

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

if (this == &rhs) {

return \*this;

}

delete [] arr;

arr = new T[rhs.capacity];

capacity = rhs.capacity;

top = -1;

for (int i = 0; i <= rhs.top; i++) {

push(rhs.arr[i]);

}

return \*this;

}

/\*

\* @Author: Grant Dawson

\*/

//defualt constructor

template<class DataType>

SUStackList<DataType>::SUStackList(){

}

//copy constructor

template<class DataType>

SUStackList<DataType>::SUStackList(const SUStackList &old){

list=old.list;

}

//Deconstructor

template<class DataType>

SUStackList<DataType>::~SUStackList(){

//delete list;

}

//returns the size of the Stack at that moment

template<class DataType>

int SUStackList<DataType>::size() const{

return list.size();

}

//checks to see if the stack is empty or not

template<class DataType>

bool SUStackList<DataType>::isEmpty() const{

if(list.size()==0)

return true;

return false;

}

//pushs a newNode to the end of the list

template<class DataType>

void SUStackList<DataType>::push(const DataType& x){

list.putBack(x);

}

//pops the top/back of the stack off and returns the data what was in it

template<class DataType>

void SUStackList<DataType>::pop(DataType& x){

try{

x=list.getBack();

std::cout<<"-----=========-------\n";

x.printInfo();

}catch(std::string n){

throw n;

}catch(...){

throw "An unknown error has occured...";

}

}

//This will print the stack

template<class DataType>

void SUStackList<DataType>::printStack() const{

list.display();

}

//= operator will set the data in the right into the left

template<class DataType>

SUStackList<DataType>& SUStackList<DataType>::operator=(const SUStackList<DataType>& old){

if(this==&old){

return \*this;

}

list=old.list;

return \*this;

}

**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;

}