Note to grader: I have uploaded the example program I used for the output screenshots to the GitHub repository if you want to test it. I also have it on repl.it: https://replit.com/@Vulpolox/Assignment2

For context, the program from which the following screenshots were obtained is only meant to show that each required function is behaving as required by the assignment directions and to show that my Employee classes are working properly (it just outputs a bunch of stuff to the console without user input)

Linked List

Initialize/Declare

```
LinkedList<std::string> myList;
bool tempBool;
int tempInt;

myList.push_back("middle");
std::cout << "starting list: ";
myList.printList();
std::cout << "----------" << std::endl << std::endl;
```

```
> sh -c make -s
> ./main
starting list: middle
----------
```

push_back()

```
//push_back()
  std::cout << "calling myList.push_back(\"back\")" << std::endl;
  myList.push_back("back");
  std::cout << "current list: ";
  myList.printList();
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.push_back("back")
current list: middle back
----------
```

push_front()

```
//push_front()
  std::cout << "calling myList.push_front(\"front\")" << std::endl;
  myList.push_front("front");
  std::cout << "current list: ";
  myList.printList();
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.push_front("front")
current list: front middle back
----------
```

front()

```
//front()
  std::cout << "myList.front() = " << myList.front() << std::endl;
  std::cout << "----------" << std::endl << std::endl;
```

```
myList.front() = front
----------
```

back()

```
//back()
  std::cout << "myList.back() = " << myList.back() << std::endl;
  std::cout << "----------" << std::endl << std::endl;
```

```
myList.back() = back
----------
```

insert()

```
//insert()
  std::cout << "calling myList.insert(1, \"insert\")" << std::endl;
  myList.insert(1, "insert");
  std::cout << "current list: ";
  myList.printList();
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.insert(1, "insert")
current list: front insert middle back
----------
```

insert() [beyond end of list]

if the passed index is greater than the max index of the list
(in this case max index is 3 ; 100 is passed),
the function call is treated as push_back() as per
the assignment directions

```
//insert() [beyond end of list]
  std::cout << "calling myList.insert(100, \"insertBeyondEnd\")" << std::endl;
  myList.insert(100, "insertBeyondEnd");
  std::cout << "current list: ";
  myList.printList();
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.insert(100, "insertBeyondEnd")
current list: front insert middle back insertBeyondEnd
----------
```

remove()

```
//remove()
  std::cout << "calling myList.remove(4)" << std::endl;
  tempBool = myList.remove(4);
  std::cout << "current list: ";
  myList.printList();
  std::cout << "Boolean return value (1 if successful 0 if fail) = " << tempBool << std::endl;
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.remove(4)
current list: front insert middle back
Boolean return value (1 if successful 0 if fail) = 1
----------
```

remove() [beyond end of list]

if the passed index is greater than the max index of the list (current max index is 3 ; passed value is 100), function will not remove any element from the list and will return false

```
//remove() [beyond end of list]
  std::cout << "calling myList.remove(100)" << std::endl;
  tempBool = myList.remove(100);
  std::cout << "current list: ";
  myList.printList();
  std::cout << "Boolean return value (1 if successful 0 if fail) = " << tempBool << std::endl;
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.remove(100)
current list: front insert middle back
Boolean return value (1 if successful 0 if fail) = 0
----------
```

find()

```
//find()
  std::cout << "calling myList.find(\"insert\")" << std::endl;
  tempInt = myList.find("insert");
  std::cout << "current list: ";
  myList.printList();
  std::cout << "return value = " << tempInt << std::endl;
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.find("insert")
current list: front insert middle back
return value = 1
----------
```

find() [nonexistent value]

if a value that doesn't exist in the list is

passed to the function, it will return the size of the list

```
//find() [nonexistent value]
  std::cout << "calling myList.find(\"not here\")" << std::endl;
  tempInt = myList.find("not here");
  std::cout << "current list: ";
  myList.printList();
  std::cout << "return value = " << tempInt << std::endl;
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.find("not here")
current list: front insert middle back
return value = 4
----------
```

pop_back()

```
//pop_back()
  std::cout << "calling myList.pop_back()" << std::endl;
  myList.pop_back();
  std::cout << "current list: ";
  myList.printList();
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.pop_back()
current list: front insert middle
----------
```

pop_front()

```
//pop_front()
  std::cout << "calling myList.pop_front()" << std::endl;
  myList.pop_front();
  std::cout << "current list: ";
  myList.printList();
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.pop_front()
current list: insert middle
```

empty()

```
//empty()
  std::cout << "calling myList.empty()" << std::endl;
  tempBool = myList.empty();
  std::cout << "current list: ";
  myList.printList();
  std::cout << "Boolean return value = " << tempBool << std::endl;
  std::cout << "----------" << std::endl << std::endl;
```

```
calling myList.empty()
current list: insert middle
Boolean return value = 0
----------
```

Employee

Initialize/Declare

```
//initialize objects and store in LinkedList
  Employee* e1 = new Nonprofessional("Bob", 25.0, 36);
  Employee* e2 = new Professional("Mike", 2500.0, 3);
  LinkedList<Employee*> empList;
  empList.push_back(e1);
  empList.push_back(e2);
```

Runtime Polymorphism and Explaining Metrics

```
//runtime polymorphism / displaying employee's metrics / using my iterator class for ranged based for loop
  for (auto employee : empList)
    {
      employee->printEmployee();
      std::cout << std::endl;
    }
```

```
Nonprofessional Employee
------------------------
  Name: Bob
  Pay Rate: $25.00 / hour
  Hours Worked: 36.0
  Vacation Days Earned: 0.36
  Weelky Pay Earned: $900.00
  Health Care Contribution: $90.00

Professional Employee
------------------------
  Name: Mike
  Pay Rate: $2500.00 / month
  Months Worked: 3.0
  Vacation Days Earned: 9.00
  Weelky Pay Earned: $625.00
  Health Care Contribution: $125.00
```