Fundação Getulio Vargas
Estruturas de Dados e Algoritmos
Prof. Jorge Poco

Homework 2
Due:: March 22, 2019

Homework 2

1 LinkedList Improvements

(1.0pt) Constructor: You must improve the constructor. In this version, we must be able to create a list using one of the following two options:

```
int main() {
    // Option 1: Variadic Templates
    LinkedList list1(1, 2, 10, 2, 3);
    list1.print();
    // Option 2: Initialization List
    LinkedList list2({1, 2, 10, 2, 3});
    list1.print();
}
```

For the first option you might use the *initializer_list* from STL and for the second option the *Variadic Templates* feature. An example for both cases can be found at `https://bit.ly/2EFy4fc`. Any of them is a valid answer but I recommend to try both options.

(0.5pt) Destructor: Your job here is to release the dynamic memory reserved by the new operator. In this method, you must call delete pNode for each node in the list. To check that your code is working print something in the Node destructor to check that all the nodes are destructed.

(1.0pt) Templates: Modify your class so it must support any data type. Remember we saw templates in classes. The following code should work if succeed this step.

```
int main() {
    // create a linked list for three data types
    LinkedList<int> ilist(1, 10, 2);
    ilist.print();
    // output: 1 10 2
    LinkedList<float> flist(1.2, 1.4, 100000);
    flist.print();
    // output: 1.2 1.4 100000
    LinkedList<std::string> slist("one", "two", "three");
    slist.print();
    // output: one two three
}
```

(1.5pt) Iterators: Here, you must implement all the necessary to make your LinkedList to work using iterators. It should be similar to the STL data structures in C++. The following code should work if you succeed.

```
int main() {
    LinkedList<int> ilist(1, 2, 10, 2, 3);
    LinkedList<int>::Iterator it;
    for (it=ilist.begin(); it!=ilist.end(); ++it) {
        std::cout << *it << " ";
    }
    cout << endl;
}
```

Just because this might be your first-time implementation and Iterator, you must include the following class inside your LinkedList class. Be careful, the code below is only a skeleton, you have to implement the methods.

```
class Iterator {
public:
    Node<T> *pNodo;
public:
    Iterator() { ... }
    Iterator(Node<T> *p) { ... }

    bool operator!=(Iterator it) { ... }
    Iterator operator++() { ... }
    T& operator*() { ... }
};
```

In addition, you have to implement the following methods in the LinkedList class.

```
    Iterator begin()  { ... }
    Iterator end() { ... }
```

(1.0pt) Exceptions: Our current implementation of the remove method does nothing if the element is not found in the list. However, the correct way to handle an exception is to throw an exception if something unusual happens. First, you must create an exception class (check this link to have an idea `https://bit.ly/2HXAwx3`). Then, modify your remove method to throws an exception if the element is not found. I will test using the following code:

```
int main() {
    LinkedList<int> ilist(1, 2, 10, 2, 3);
    // if we remove an item that doesn't exist it should throw an error
    ilist.remove(20);
    // output: libc++abi.dylib: terminating with uncaught
    //         exception of type NotFoundException: Element not found

}
```

Finally, you must use handle correctly the exception using try and catch structure.

```
int main() {
    // Correct way to handle exceptions
    try {
        ilist.remove(20);
    } catch (const NotFoundException& e) {
        cerr << e.wh\begin{center}
```

```
7
8           \end{center}
9           at();
10      }
11      // output: Element not found
12  }
```

2 Correctness of bubblesort

Bubblesort is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order.

Algorithm 1: BUBBLESORT(A)

```
1 for i = 1 to A.length − 1 do
2 |   for j = A.length downto i + 1 do
3 |   |   if A[j] < A[j − 1] then
4 |   |   |   exchange A[j] with A[j − 1]
5 |   |   end
6 |   end
7 end
```

A **(0.5pt)** Let A' denote the output of BUBBLESORT(A). To prove that BUBBLESORT is correct, we need to prove that it terminates and that

$$A'[1] \leq A'[2] \leq ... \leq A'[n] \tag{1}$$

where $n = A.length$. In order to show that BUBBLESORT actually sorts, what else do we need to prove?

The next two parts will prove inequality (1).

B **(1.0pt)** State precisely a loop invariant **for** the for loop in lines 2–6, and prove that this loop invariant holds. Your proof should use the structure of the loop invariant proof presented in this chapter.

C **(1.0pt)** Using the termination condition of the loop invariant proved in part (B), state a loop invariant for the for loop in lines 1–7 that will allow you to prove inequality (1). Your proof should use the structure of the loop invariant proof presented in this chapter.

D **(0.5pt)** What is the worst-case running time of BUBBLESORT? How does it compare to the running time of insertion sort?

3 Growth of Functions

A For each of the following pairs of functions, either $f(n)$ is in $O(g(n))$, $f(n)$ is in $\Omega(g(n))$, or $f(n) = \Theta(g(n))$. Determine which relationship is correct and briefly explain why.

- **(0.5pt)** $f(n) = \log n^2$; $g(n) = \log n + 5$
- **(0.5pt)** $f(n) = \log^2 n$; $g(n) = \log n$
- **(0.5pt)** $f(n) = n \log n + n$; $g(n) = \log n$
- **(0.5pt)** $f(n) = 2^n$; $g(n) = 10n^2$

B **(0.5pt)** Prove that $n^3 - 3n^2 - n + 1 = \Theta(n^3)$.

C **(0.5pt)** Prove that $n^2 = O(2^n)$.