Homework #2

Shao Hua, Huang
DEE2505 - Data Structures

October 31, 2017

1 Objective

1. To understand how an implementation of an ADT is used by an application program.
2. To become familiar with how to implement Linked List.
3. To understand how to use list in STL library in c++.

2 Pseudo code

Declare variable order (CLOCKWISE or COUNTERCLOCKWISE).
Write a class Node whose members are id number and letters the person called out.
Following are Node class declaration and pseudo codes of the program.

```
enum {CLOCKWISE, COUNTERCLOCKWISE} order;
class Node {
  public:
    Node(int id, string letter);
    ~Node(){}
    int getid();
    string getletter();
    void addletter(char, string);
  private:
    int id;
    string letter;
}
```

Algorithm 2.1 main

```
1: fin.open(argv[1])
2: fout.open(argv[2])
3: list⟨Node⟩mylist
4: list⟨Node⟩result
5: list⟨Node⟩::iterator it
6: order ← CLOCKWISE
7: get input from fin, and push back id number to mylist until negative number received
8: use iterator to find which person to be the first one, assign to it
9: while get string str in fin do
10:     callTheWord(mylist, it, str)
11:     changeOrderIfNeed(str)
12:     oneOutFromGame(mylist, result, it)
13: end while
14: push the last node in mylist to result list
15: use iterator to print out id numbers and letters in result list
16: fin.close()
17: fout.close()
```

Algorithm 2.2 callTheWord(list⟨Node⟩&, list⟨Node⟩::iterator &, string)

```
1: for i from 0 to str.size() − 2 do
2:     if order = CLOCKWISE then
3:         add letter str[i] to current node's letter from back
4:     else if order = COUNTERCLOCKWISE then
5:         add letter str[i] to current node's letter from front
6:     end if
7:     putItToNextPlace(mylist, it)
8: end for
9: if order = CLOCKWISE then
10:     add letter str[str.size() − 1] to current node's letter from back
11: else if order = COUNTERCLOCKWISE then
12:     add letter str[str.size() − 1] to current node's letter from front
13: end if
```

Algorithm 2.3 changeOrderIfNeed(string)

```
1: if str[str.size() − 1] is not vowel then
2:     order ← COUNTERCLOCKWISE
3: else
4:     order ← CLOCKWISE
5: end if
```

Algorithm 2.4 oneOutFromGame(list⟨Node⟩&, list⟨Node⟩&, list⟨Node⟩::iterator &)

1: copy current node and push back to *result* list
2: pop current node by using erase function
3: $putItToNextPlace(mylist, it)$

Algorithm 2.5 putItToNextPlace(list⟨Node⟩&, list⟨Node⟩::iterator &)

1: **if** *order* = CLOCKWISE **then**
2: let the iterator points to the next node
3: **if** the node does not exist, *i.e.* *mylist.end*() **then**
4: $it \leftarrow mylist.begin()$
5: **end if**
6: **else if** *order* = COUNTERCLOCKWISE **then**
7: **if** the node is the first node, *i.e.* *mylist.begin*() **then**
8: $it \leftarrow mylist.end()$
9: **end if**
10: let the iterator points to the previous node
11: **end if**

3 Time complexity analysis

Let n be number of participants in this game.
Let c be the upper bound letters of each word.

3.1 main

Line 7: push all id numbers to the list $\rightarrow O(n)$
Line 8: find one number in the list $\rightarrow O(n)$
Line 9: get $n-1$ strings in while loop $\rightarrow O(n)$
Line 10: see 3.2 $\rightarrow O(c)$
Line 11: see 3.3 $\rightarrow O(1)$
Line 12: see 3.4 $\rightarrow O(1)$
Line 15: print result list $\rightarrow O(n)$
Other lines: $O(1)$

3.2 callTheWord

Line 1: enter for loop $str.size() - 1$ times $\rightarrow O(c)$
Line 7: see 3.5 $\rightarrow O(1)$
Line 9 to 13: push $str[str.size() - 1]$ to the list $\rightarrow O(1)$

3.3 changeOrderIfNeed

Line 1 to 5: change order if it needed $\rightarrow O(1)$

3.4 oneOutFromGame

Line 1 to 2: push current node to *result* and erase the same node in *mylist* $\rightarrow O(1)$
Line 3: see 3.5 $\rightarrow O(1)$

3.5 putItToNextPlace

Line 1 to 11: enter if to move iterator $\rightarrow O(1)$

3.6 Total time complexity

In main, we can see that
Line 7 to 8: $O(n)$
Line 9 to 12: $O(nc)$
Line 15: $O(n)$
Other line: $O(1)$
Therefore, the total time complexity will be $O(nc)$ where n is total participants and c is upper bound letters of each word.

4 Conclusion

In this homework, I learned how to implement linked list by using list in STL library.
List provided easy functions for me to use, and I also wrote a class called Node to store data.
While getting each string input, the program is also doing tasks to simulate the game.
It is not that difficult to implement the whole program, but it surely takes time and efforts.
Besides, I also tried to make my code more understandable by refactoring.
It improves my coding skill, and I am looking forward to the next homework.