Exercises 4

张逸松

September 22, 2019

4.4

```
virtual bool interChange() {
    int curPos = currPos();
    E& temp = getValue();
    remove();
    next();
    if (curPos == currPos()) return false;
    prev();
    insert(temp);
}
```

4.6

```
void reverse() {
    if (head == tail) return;
    head = head->next; append(0);
    moveToStart();
    for (Link<E>* temp = curr->next; curr != NULL; curr = temp, temp =
   temp->next)
        curr->next = curr;
    Link<E> temp = head;
    head = tail;
    tail = temp;
}
```

4.13

```
template <typename E> class TwoStack: public Stack<E> {
private:
    int maxSize;
    int top1, top2;
    E* listArray;
public:
    TwoStack(int size = defaultSize) {
        maxSize = size; top1 = 0; top2 = size - 1; listArray = new E[
   size];
    }
    ~TwoStack() { delete [] listArray; }
    void clear(){ top1 = 0; top2 = maxSize - 1; }
    void push1(const E& it){
        Assert(top1 != top2, "Stack is full");
        listArray[top1++] = it;
    }
    void push2(const E& it){
        Assert(top1 != top2, "Stack is full");
        listArray[top2--] = it;
    }
    E pop1() {
        Assert(top1 != 0, "Stack1 is empty");
        return listArray[--top1];
    }
    E pop2() {
        Assert(top2 != maxSize - 1, "Stack2 is empty");
        return listArray[++top2];
    }
    const E& top1Value() {
        Assert(top1 != 0, "Stack1 is empty");
        return listArray[top1 - 1];
    }
    const E& top2Value() {
        Assert(top2 != maxSize - 1, "Stack2 is empty");
        return listArray[top2 + 1];
    }
    int length() const { return top1 + maxSize - 1 - top2; }
};
```

4.14

```
template <typename E> class AQueue: public Queue<E> {
private:
    int maxSize;
    int front;
    int rear;
    bool empty;
    E* listArray;
public:
    AQueue(int size = defaultSize) {
        maxSize = size + 1;
        rear = 0; front = 1;
        empty = 1;
        listArray = new E[maxSize];
    }
    ~AQueue() { delete [] listArray; }
    void clear() { rear = 0; front = 1; empty = 1; }
    void enqueue(const E& it) {
        Assert(((rear + 2) % maxSize) != front, "Queue is full");
        rear = (rear + 1) % maxSize;
        listArray[rear] = it;
        empty = 0;
    }
    E dequeue() {
        Assert(empty != 1, "Queue is empty");
        E it = listArray[front];
        front = (front + 1) % maxSize;
        if ((rear + 1) % maxSize == front) empty = 1;
        return it;
    }
    const E& frontValue() const {
        Assert(empty != 1, "Queue is empty");
        return listArray[front];
    }
};
```

4.15

```
bool isPalindrome() {
    Stack <char> S;
    Queue <char> Q;
    char c;
    for (; cin >> c; ) {
        S.push(c);
        Q.enqueue(c);
    }
    for (; Q.length(); ) {
        if (S.top() != Q.front()) return false;
        char temp = S.pop();
        temp = Q.dequeue();
    }
    return true;
}
```

4.19

a)

```
bool isBalanced(string str) {
    stack <int> s;
    for (int i = 0; i < str.size(); ++i) {
        if (str[i] == '(') s.push(i);
        else if (str[i] == ')') {
            if (s.empty()) return false;
            else s.pop();
        }
    }
    return s.empty();
}
```

b)

```
int FOP(string str) {
    stack <int> s;
    for (int i = 0; i < str.size(); ++i) {
        if (str[i] == '(') s.push(i);
        else if (str[i] == ')') {
            if (s.empty()) return i;
            else s.pop();
        }
    }
    if (s.empty()) return -1;
    for (int i = s.top(); ; i = s.top(), s.pop())
        if (s.empty()) return i;
}
```