作业 6 参考答案（by 薛正康）

P184-12

（1）

```
//pushNode
template<class T>
void linkedStack<T>::pushNode(chainNode<T>* theNode) {
	theNode->next = stackTop;//头插法，直接将当前节点的下一个变为原来的头节点
	stackTop = theNode;//将当前节点设置为新的头节点
	stacksize++;//插入了一个元素，所以数量+1
}

//popNode
template<class T>
void linkedStack<T>::popNode() {
	if (stackTop == NULL) {//如果栈顶为空，说明栈内无元素，返回
		cout << "Stack is empty!" << "\n";
		return;
	}
	chainNode<T>* deleteNode = stackTop;//否则将要删掉栈顶
	stackTop = stackTop->next;//此时新的栈顶为原来栈顶的后一个节点
	delete deleteNode;//删除栈顶
	stacksize--;//删除一个元素，数量-1
}
```

（2）（3）

提示：使用 rank 等方法生成多组大批量随机数据（操作序列），然后每一次分别用链表实现的栈（pushNode，popNode）和非链表实现的栈（push，pop）运行相同的一组大批量随机数据，执行多次，统计运行时间，并进行对比。

P211-5

（1）

```
//此处使用了循环队列，如果不使用循环队列，需要对 head 和 tail 做处理
```

i.输入队列

```
template<class T>
void roundArrayQueue<T>::input() {
	//此处可以考虑实现一个 clear 函数，清空当前元素
	//this->clear();
	cout << "Input the element num of new queue" << "\n";
	int n;
```

```
    cin >> n;//输入个数
    cout << "Input the elements" << "\n";
    T element;
    for (int i = 0; i < n; i++) {
        //如果 element 无法直接输入，需要重载“>>”
        cin >> element;//输入元素
        this->push(element);//放入队列
    }
}
```

ii.输出队列

```
template<class T>
void roundArrayQueue<T>::output() {
    if (this->empty()) {
        cout << "Queue is empty" << "\n";
        return;
    }
    T element;
    for (int i = 0; i < this->size(); i++) {
        //从头开始往后依次寻找元素
        element = queue[(queueFront + i) % arrayLength];
        //如果 element 无法直接输出，需要重载“<<”
        cout << element << " ";//输出元素
    }
    cout << "\n";
}
```

iii.队列分解

```
template<class T>
void roundArrayQueue<T>::split(roundArrayQueue<T>* q1, roundArrayQueue<T>* q2) {
    //此处如果之前实现了 clear 函数，可以直接使用
    //q1->clear(); q2->clear();
    //目的是清空队列
    while (!q1->empty())
        q1->pop();
    while (!q2->empty())
        q2->pop();
    T element;
    for (int i = 0; i < this->size(); i++) {
        //从头开始往后依次寻找元素
        element = queue[(queueFront + i) % arrayLength];
        //这里有歧义，如果下标从 1 开始，按照 135，246 分的话，需要使用给出的写法
        //如果下标从 0 开始，按照 024，135 分的话，可以直接 if（i % 2）
        //此处不做严格要求，只要把奇偶分离就行
```

```
            if((i + 1) % 2)
                q1->push(element);
            else
                q2->push(element);
        }
}
```

iv.队列合并

```
template<class T>
void roundArrayQueue<T>::merge(roundArrayQueue<T>* q1, roundArrayQueue<T>* q2)
{
    //此处如果之前实现了 clear 函数，可以直接使用
    //this->clear();
    //目的是清空队列
    while (!this->empty())
        this->pop();
    //记录多少元素可以交替放，这样可以将其中 1 个队列元素全部放完
    int size = min(q1->size(), q2->size()) * 2;
    T element;
    for (int i = 0; i < size; i++) {
        //此处 size % 2 和(size + 1) % 2 都行
        if (size % 2) {
            element = q1->front();//取 q1 队头元素
            //下面两行可以使 q1 最终保持不变，且能依次取到 q1 的每个元素
            //其他方法能依次取到每个元素也可以
            q1->pop();//弹出 q1 队头
            q1->push(element);//将元素插入 q1 队尾
            this->push(element);//将元素插入总队列的队尾
        }
        else {
            element = q2->front();
            q2->pop();
            q2->push(element);
            this->push(element);
        }
}
```

（2）

```
template<class T>
class extendedQueue:public queue {
    virtual <void><input>() = 0;
    virtual <void><output>() = 0;
    virtual <void><split>(extendedQueue<T>* q1, extendedQueue<T>* q2) = 0;
```

```
    virtual <void><merge>(extendedQueue<T>* q1, extendedQueue<T>* q2) = 0;
};
```

（3）

```
#include <iostream>
#include <vector>
// 抽象基类 - 队列 (Queue)
class Queue {
    public:
        virtual void enqueue(int item) = 0;
        virtual int dequeue() = 0;
        virtual int size() const = 0;
        virtual bool is_empty() const = 0;
};
// 链表节点
class Node {
    public:
        int data;
        Node* next;
        Node(int value) : data(value), next(nullptr) {}
};
// 链表实现的队列 - ExtendedQueue
class ExtendedQueue : public Queue {
    protected:
        Node* front;
        Node* rear;
        int count;
    public:
        ExtendedQueue() : front(nullptr), rear(nullptr), count(0) {}
        void enqueue(int item) override {
            Node* newNode = new Node(item);
            if (is_empty()) {
                front = rear = newNode;
            } else {
                rear->next = newNode;
                rear = newNode;
            }
            count++;
        }
        int dequeue() override {
            if (!is_empty()) {
                int item = front->data;
                Node* temp = front;
```

```
                front = front->next;
                delete temp;
                count--;
                return item;
            } else {
                std::cout << "队列为空，无法出队。" << std::endl;
                return -1;
            }
        }
        int size() const override {
            return count;
        }
        bool is_empty() const override {
            return count == 0;
        }
};
// 循环队列 - ArrayQueue
class ArrayQueue : public Queue {
    private:
        std::vector<int> array;
        int front;
        int rear;
        int capacity;
    public:
        ArrayQueue(int size) : capacity(size), array(size), front(-1), rear(-1) {}
        bool is_full() const {return (rear + 1) % capacity == front; }
        void enqueue(int item) override {
            if (!is_full()) {
                if (front == -1) {
                    front = 0;
                }
                rear = (rear + 1) % capacity;
                array[rear] = item;
            } else {
                std::cout << "队列已满，无法入队。" << std::endl;
            }
        }
        int dequeue() override {
            if (!is_empty()) {
                int item = array[front];
                if (front == rear) {
                    front = rear = -1;
                } else {
                    front = (front + 1) % capacity;
```

```
            }
            return item;
        } else {
            std::cout << "队列为空，无法出队。" << std::endl;
            return -1;
        }
    }
    int size() const override {
        if (is_empty()) {
            return 0;
        } else if (front <= rear) {
            return rear - front + 1;
        } else {
            return capacity - (front - rear - 1);
        }
    }
    bool is_empty() const override {
        return front == -1;
    }
};
// ExtendArrayQueue 继承了 ExtendedQueue 和 ArrayQueue
class ExtendArrayQueue : public ExtendedQueue, public ArrayQueue {
public:
    ExtendArrayQueue(int size) : ArrayQueue(size) {}
    void enqueue(int item) override {
        ArrayQueue::enqueue(item);
    }
    int dequeue() override {
        return ArrayQueue::dequeue();
    }
    int size() const override {
        return ArrayQueue::size();
    }
    bool is_empty() const override {
        return ArrayQueue::is_empty();
    }
};
```

（4）

```
int main() {
    ExtendArrayQueue extendArrayQueue(5);
    // 元素入队
    extendArrayQueue.enqueue(1);
```

```
    extendArrayQueue.enqueue(2);
    extendArrayQueue.enqueue(3);
    extendArrayQueue.enqueue(4);
    extendArrayQueue.enqueue(5);
    // 元素出队并打印
    while (!extendArrayQueue.is_empty()) {
        std::cout << extendArrayQueue.dequeue() << " ";
    }
    std::cout << std::endl;
    return 0;
}
```