

Project documentation

# Data structure course design

——Bank business

Author name: Xuedi Liu

Number: 1752985

instructor: Ying Zhang

College/Major: School of Software Engineering /Software Engineering

## 1. Analysis

### 1.1 Project background analysis

### 1.2 Project function requirements

1. Input description:

2. Output description:

## 2. Design

### 2.1 Data structure design

### 2.2 Algorithms analysis and design

### 2.3 Class member and operational design

1. Node

2. Queue

### 2.4 System design

## 3. Realization and Test

### 3.1 General condition 1

Normal test, where window  $A$  has more people than window  $B$

### 3.2 General condition 2

Normal test, where  $B$  window has more people than  $A$  window

### 3.3 Border test

Boundary test, with  $N$  being the minimum 1

### 3.4 Error test

The number of people entered is not more than 1

## ■ Operating Environment:

- Unix executables: running on **Unix** platforms
- Linux executables: running on **Linux** platforms
- exe executable file: Windows Console Application, running on 64-bit Windows platform

## ■ Code hosting platform: Github

# 1. Analysis

## *1.1 Project background analysis*

Suppose a bank has two business windows, A and B, and the speed of processing business is different. The processing speed of window A is twice that of window B. That is, when window A has processed 2 customers, window B processes Finish 1 customer. Given the sequence of customers arriving at the bank, output the sequence of customers in the order in which the business was completed. It is assumed that the time interval after the arrival of the customer letter is not taken into account, and when two customers are processed in different windows at the same time, the customer in window A has priority to output.

## *1.2 Project function requirements*

### **1. Input description:**

Enter a line of positive integers, where the first number  $N$  ( $N \leq 1000$ ) is the total number of customers, followed by the number of  $N$  customers. Customers with odd numbers need to go to window A for transactions, and customers with even numbers go to window B. Numbers are separated by spaces.

### **2. Output description:**

The customer numbers are output in the order in which business processes are completed. The number keys are separated by spaces, but the last number must not have extra spaces.

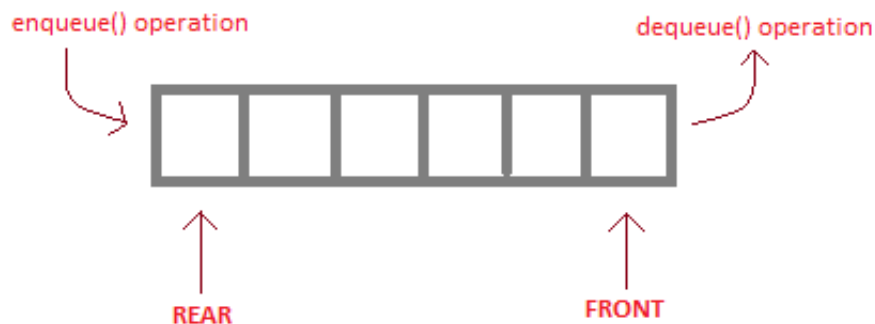
## 2. Design

### *2.1 Data structure design*

Bank business are a typical queuing problem. Such problems follow the "first in, first out" principle and can be implemented using data structures such as queues.

**Queue** is an abstract data type or a linear data structure, just like stack data structure, in which the first element is inserted from one end called the **REAR**(also called **tail**), and the removal of existing element takes place from the other end called as **FRONT**(also called **head**).

This makes queue as **FIFO**(First in First Out) data structure, which means that element inserted first will be removed first.

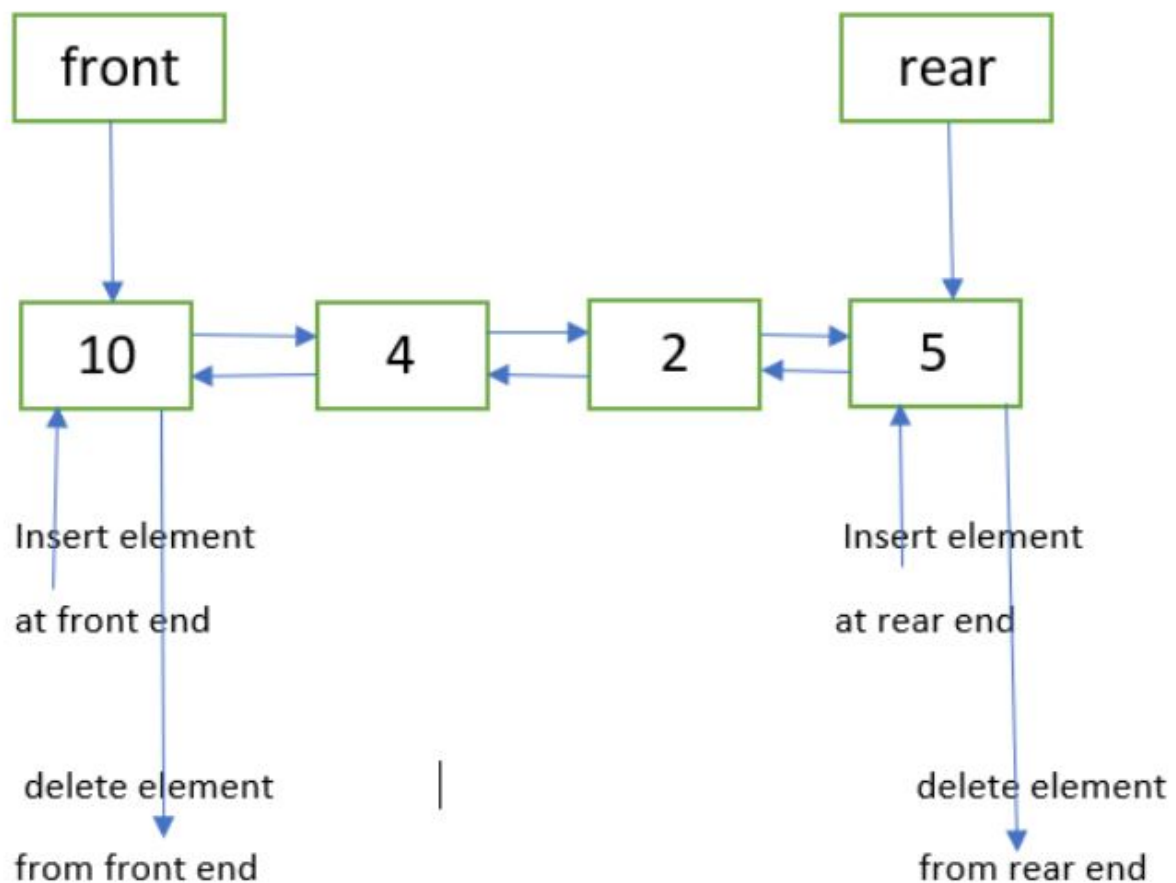


**enqueue( )** is the operation for adding an element into Queue.

**dequeue( )** is the operation for removing an element from Queue .

#### **QUEUE DATA STRUCTURE**

In order to facilitate coding, this question uses a doubly linked list storage structure to implement the queue. At the same time, a head sentinel node and a tail sentinel node are maintained at the head and tail of the queue, respectively.



## 2.2 Algorithms analysis and design

In this question, the bank has two windows. The business of the two windows does not interfere with each other and is independent of each other. Therefore, you can consider using two queues to handle this problem.

### Algorithm Description:

1. First add the customer numbers to the  $A$  and  $B$  queues according to the parity.
2. Because the processing speeds of the two teams are different, we define a loop as processing two customers of  $A$  window and one customer of  $B$  window. This cycle will end when any of the  $A$  and  $B$  queues is empty.
3. If one of the  $A$  and  $B$  queues is empty, the remaining elements of the other queue are output.
4. End if  $A$  and  $B$  queues are empty.

## 2.3 Class member and operational design

### 1. Node

```
struct Node {  
    int data;  
    Node *next;  
    Node *pre;  
};
```

## 2. Queue

Private members:

```
private:  
    int _size;//length  
    Node *head;  
    Node *trail;
```

Public operation:

```
public:  
    Queue();  
  
    int size() { return _size; }  
  
    int front();//Returns the value of the head node  
    void pop();//Head element out of the team  
    void push(int t);//Elements enqueued  
    bool empty() { return !_size; };
```

Core function:

Constructor :

```
Queue::Queue() {  
    _size = 0;  
    head = new Node;  
    trail = new Node;  
    head->pre = nullptr;  
    head->next = trail;  
    trail->pre = head;  
    trail->next = nullptr;  
}
```

Insert in the tail of the team:

```

void Queue::push(int t) {
    auto p = new Node;
    p->data = t;
    p->pre = trail->pre;
    p->next = trail;
    p->pre->next = p;
    trail->pre = p;
    _size++;
}

```

Return the head of the queue :

```

int Queue::front() {
    return head->next->data;
}

```

Pop the head element of the queue:

```

void Queue::pop() {
    auto p = head->next;
    head->next = p->next;
    p->next->pre = head;
    _size--;
    delete p;
}

```

## 2.4 System design

```

int main() {
    int time;//人数
    cin >> time;
    while (time < 1) {
        cout << "您输入的数字应大于或等于1，请重新输入： ";
        cin >> time;
    }
    Queue bankA, bankB;
    while (time-->0) {
        int num;
        cin >> num;
        if (num % 2) { //奇数
            bankA.push(num);
        } else { //偶数
            bankB.push(num);
        }
    }
}

```

```
while (!bankA.empty() && !bankB.empty()) {  
    cout << bankA.front() << ' ';  
    bankA.pop();  
    if (!bankA.empty()) {  
        cout << bankA.front() << ' ';  
    }  
    bankA.pop();  
    cout << bankB.front() << ' ';  
    bankB.pop();  
}  
  
while (!bankA.empty()) {  
    cout << bankA.front() << ' ';  
    bankA.pop();  
}  
while (!bankB.empty()) {  
    cout << bankB.front() << ' ';  
    bankB.pop();  
}  
return 0;  
}
```



### 3. Realization and Test

#### 3.1 General condition 1

**Normal test, where window  $A$  has more people than window  $B$**

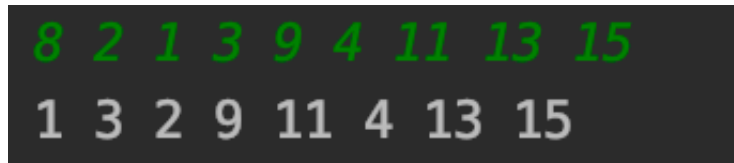
**input:**

8 2 1 3 9 4 11 13 15

**Expected outcome:**

1 3 2 9 11 4 13 15

**Experimental result:**



```
8 2 1 3 9 4 11 13 15
1 3 2 9 11 4 13 15
```

#### 3.2 General condition 2

**Normal test, where  $B$  window has more people than  $A$  window**

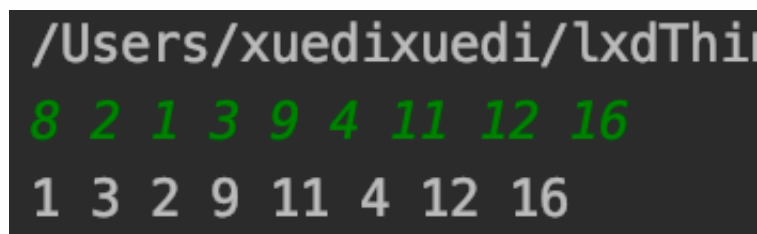
**input:**

8 2 1 3 9 4 11 12 16

**Expected outcome:**

1 3 2 9 11 4 12 16

**Experimental result:**



```
/Users/xuedixuedi/lxdThi
8 2 1 3 9 4 11 12 16
1 3 2 9 11 4 12 16
```

#### 3.3 Border test

## Boundary test, with $N$ being the minimum 1

input:

1 6

Expected outcome:

6

Experimental result:

```
/Users/xuedixuedi/lxdTh  
1 6  
6
```

## 3.4 Error test

### The number of people entered is not more than 1

input:

-1

Expected outcome:

Program output error prompt, normal operation does not crash.

Experimental result:

```
/Users/xuedixuedi/lxdThings/Code/q  
-1  
您输入的数字应大于或等于1，请重新输入：
```