

数据结构项目文档

题目：银行业务

指导教师：张颖

姓名：王亮

学号：1653340

数据结构项目文档

一、题目分析

- 项目简介
- 功能需求
- 设计思路

二、设计

- 数据结构设计
- 类的设计
 - LinkNode
 - LinkedQueue
- 函数设计
 - 分配顾客
 - 处理业务

三、实现

- 分配顾客
- 处理业务
 - 流程图
 - 相关代码

四、测试

- 功能测试（包括各种边界情况）
 - 正常测试，A窗口人多
 - 正常测试，B窗口人多
 - 最小N
 - 只有奇数编号顾客
 - 只有偶数编号顾客

一、题目分析

1. 项目简介

设某银行有A, B两个业务窗口，且处理业务的速度不一样，其中A窗口处理速度是B窗口的2倍----即当A窗口每处理完2个顾客是，B窗口处理完1个顾客。给定到达银行的顾客序列，请按照业务完成的顺序输出顾客序列。假定不考虑顾客信后到达的时间间隔，并且当不同窗口同时处理完2个顾客时，A窗口的顾客优先输出。

2. 功能需求

1. 输入说明：输入为一行正整数，其中第一数字N（ $N \leq 1000$ ）为顾客总数，后面跟着N位顾客的编号。编号为奇数的顾客需要到A窗口办理业务，为偶数的顾客则去B窗口。数字间以空格分隔。
2. 输出说明：按照业务处理完成的顺序输出顾客的编号。数字键以空格分隔，但是最后一个编号不能有多余的空格。
3. 测试用例：

序号	输入	输出	说明
1	8 2 1 3 9 4 11 13 15	1 3 2 9 11 4 13 15	正常测试，A窗口人多
2	8 2 1 3 9 4 11 12 16	1 3 2 9 11 4 12 16	正常测试，B窗口人多
3	1 6	6	最小N

3. 设计思路

首先确定项目采用链式队列作为数据结构，定义类的成员变量和成员函数；然后实现处理顾客业务的函数；最后完成主函数以验证程序的功能并得到运行结果。

二、设计

1. 数据结构设计

问题情景是银行需要处理排队顾客的业务，对于经典的排队问题，适合选择队列作为抽象数据结构，发挥队列先进先出的特性。

2. 类的设计

使用链表实现一个队列，即链式队列。需要定义LinkNode和LindedQueue两个类，按照复合的方式，协同实现链式队列。

采用模板类的定义方式，提高实现的类的灵活性和可复用性。

2.1 LinkNode

包括一个数据域和一个指针域。

```

template <typename T>struct LinkNode{
    T data;
    LinkNode<T> *link;
    LinkNode(LinkNode<T> *ptr = NULL){
        link = ptr;
    }
    LinkNode(const T &item, LinkNode<T> *ptr = NULL){
        data = item;
        link = ptr;
    }
};

```

2.2 LinkedQueue

包含一个头指针和一个尾指针，共两个数据成员。

提供基本的队列公共函数。

```

template <typename T>class LinkedQueue{//无头结点
public:
    LinkedQueue(){
        rear = NULL;
        front = NULL;
    }
    ~LinkedQueue(){
        makeEmpty();
    }
    bool EnQueue(const T &x);
    bool DeQueue(T &x);
    bool getFront(T &x)const;
    void makeEmpty();
    bool IsEmpty()const{
        return front == NULL;
    }
    int getSize()const;
    friend ostream& operator << (ostream &os, LinkedQueue<T> &Q){
        LinkNode<T> *p = Q.front;
        int i = 0;
        while (p){
            os << "#" << ++i << ": " << p->data << endl;
            p = p->link;
        }
        os << "Queue Size: " << Q.getSize() << endl;
        return os;
    }
    void output();
protected:
    LinkNode<T> *front, *rear;
};

```

3. 函数设计

解决了数据结构的问题，接下来要解决功能需求。即题目中所述的

- 编号为奇数的顾客需要到A窗口办理业务，为偶数的顾客则去B窗口。
- A窗口处理速度是B窗口的2倍----即当A窗口每处理完2个顾客是，B窗口处理完1个顾客。给定到达银行的顾客序列，请按照业务完成的顺序输出顾客序列。假定不考虑顾客信后到达的时间间隔，并且当不同窗口同时处理完2个顾客时，A窗口的顾客优先输出。

需要两个功能函数，一个函数按照编号的奇偶将顾客分配给A、B两个窗口，另一个函数用来处理业务，并按照完成顺序输出顾客序列。

3.1 分配顾客

读取所有顾客数据，并按照顺序将奇数编号的顾客存入窗口A的顾客队列，偶数编号的顾客存入窗口B的顾客队列。

3.2 处理业务

通过分析题意简化算法。首先按照三个一组的顺序输出顾客，输出两个窗口A中的顾客，然后输出一个窗口B中的顾客。直至某个窗口的顾客全部处理完，然后直接输出另一个窗口剩余的所有顾客。

三、实现

1. 分配顾客

定义两个队列，分别用来记录两个银行窗口的排队顾客队伍。

按照顾客编号的奇偶，将每个顾客分配到两个队列中。

相关代码如下：

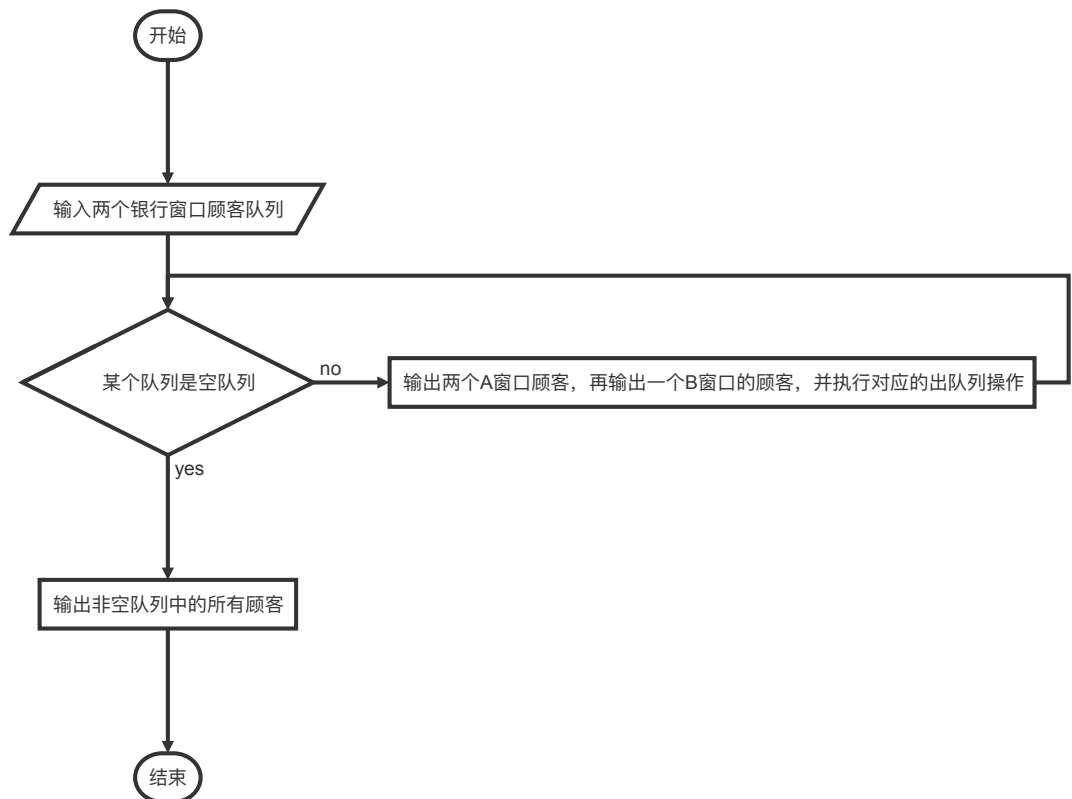
```
/*
 *@brief: 读取所有顾客数据，并按照顺序将奇数编号的顾客存入窗口A的顾客队列，偶数编号的顾客存入窗口B的顾客队列。
 */
void readData(int sum, LinkQueue<int>& win_1, LinkQueue<int>& win_2)
{
    int cnt;
    int tempCustomer;
    for (cnt = 0; cnt < sum; cnt++)
    {
        cin >> tempCustomer;
        if (tempCustomer % 2)
            win_1.Enqueue(tempCustomer);
        else
            win_2.Enqueue(tempCustomer);
    }
}
```

```
}
```

2. 处理业务

按照三个一组的顺序输出顾客，输出两个窗口A中的顾客，然后输出一个窗口B中的顾客。直至某个窗口的顾客全部处理完，然后直接输出另一个窗口剩余的所有顾客。

2.1 流程图



2.2 相关代码

```
/*
```

```
 *@brief: 通过分析题意简化算法。首先按照三个一组的顺序输出顾客，输出两个窗口A中的顾客，  
 然后输出一个窗口B中的顾客。直至某个窗口的顾客全部处理完，然后直接输出另一个窗口剩余的所有  
 顾客。
```

```

*/
void handle(int sum, LinkedQueue<int>& win_1, LinkedQueue<int>& win_2)
{
    int cntCustomers;           //已经处理的顾客计数
    int temp = 0;               //暂时存放待输出数据
    bool continueHandle = true; //是否继续按照三个一组的方式进行处理
    for (cntCustomers = 1; cntCustomers <= sum && continueHandle;
        cntCustomers++)
    {
        switch (cntCustomers % 3) {
            case 1:
            case 2:
                if (!win_1.IsEmpty())
                {
                    win_1.DeQueue(temp);
                    std::cout << temp;

                    if (!win_1.IsEmpty() || !win_2.IsEmpty())
                        std::cout << " ";

                }
                else
                {
                    continueHandle = false;
                }
                break;

            case 0:
                if (!win_2.IsEmpty())
                {
                    win_2.DeQueue(temp);
                    std::cout << temp;

                    if (!win_1.IsEmpty() || !win_2.IsEmpty())
                        std::cout << " ";

                }
                else
                {
                    continueHandle = false;
                }
                break;

            default:
                break;
        }
    }

    while (!win_1.IsEmpty()) {
        win_1.DeQueue(temp);
        std::cout << temp;
    }
}

```

```

        if(!win_1.IsEmpty())
            std::cout << " ";

    }

    while (!win_2.IsEmpty()) {
        win_2.DeQueue(temp);
        std::cout << temp;
        if(!win_2.IsEmpty())
            std::cout << " ";
    }

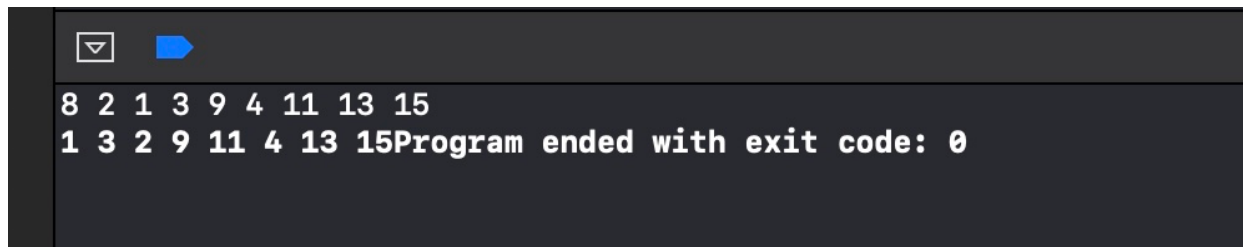
}

```

四、测试

1. 功能测试（包括各种边界情况）

1.1 正常测试，A窗口人多



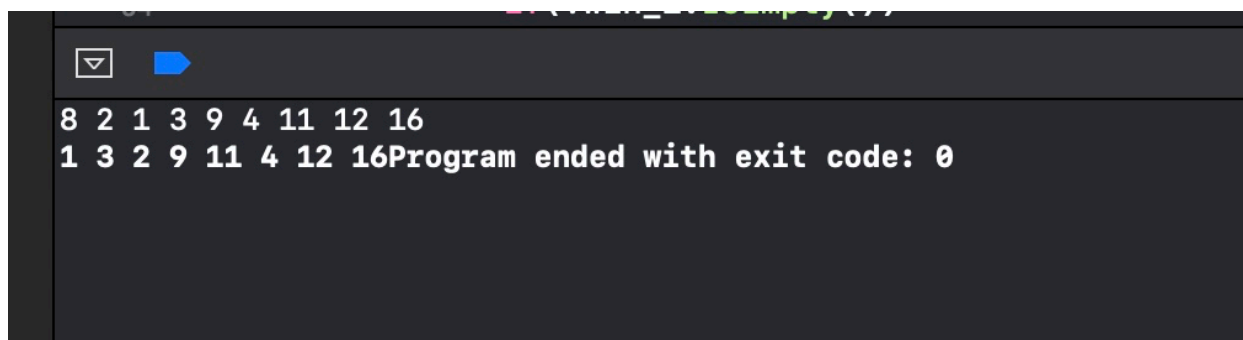
```

8 2 1 3 9 4 11 13 15
1 3 2 9 11 4 13 15Program ended with exit code: 0

```

测试结果正确。

1.2 正常测试，B窗口人多



```

8 2 1 3 9 4 11 12 16
1 3 2 9 11 4 12 16Program ended with exit code: 0

```

测试结果正确。

1.3 最小N

```
1 6
6Program ended with exit code: 0
```

测试结果正确。

1.4 只有奇数编号顾客

```
54 if (!win_1.IsEmpty())
3 1 5 7
1 5 7Program ended with exit code: 0
```

测试结果正确。

1.5 只有偶数编号顾客

```
54 if (!win_1.IsEmpty())
3 2 4 6
2 4 6Program ended with exit code: 0
```

测试结果正确。