

数据结构课程设计
项目说明文档

银行业务

| | |
|---------|------------------|
| 作者姓名: | <u>汪明杰</u> |
| 学 号: | <u>1851055</u> |
| 指导教师: | <u>张 颖</u> |
| 学院专业: | <u>软件学院 软件工程</u> |



同济大学
Tongji University

目录

| | |
|----------------------------|----|
| 1 项目分析 | 3 |
| 1.1 项目背景 | 3 |
| 1.2 项目需求分析 | 3 |
| 1.3 项目要求 | 4 |
| 1.3.1 功能要求 | 4 |
| 1.3.2 输入格式 | 4 |
| 1.3.3 输出格式 | 4 |
| 1.3.4 项目示例 | 4 |
| 2 项目设计 | 5 |
| 2.1 数据结构设计 | 5 |
| 2.2 类设计 | 5 |
| 2.2.1 结点类 (ListNode) | 5 |
| 2.2.2 双向链表类 (List) | 5 |
| 2.2.3 队列类 (Queue) | 7 |
| 3 项目实施 | 9 |
| 3.1 项目主体功能 | 9 |
| 3.1.1 项目主体功能流程图 | 9 |
| 3.1.2 项目主体功能代码 | 10 |
| 3.2 队列操作 | 11 |
| 3.2.1 入队操作 | 11 |
| 3.2.2 出队操作 | 12 |
| 4 项目测试 | 13 |
| 4.1 A窗口人多的情况测试 | 13 |
| 4.2 B窗口人多的情况测试 | 13 |
| 4.3 A窗口和B窗口人数相同的情况测试 | 14 |
| 4.4 最小人数的情况 | 14 |
| 4.5 非法输入测试 | 15 |

1 项目分析

1.1 项目背景

排队是现实生活中最为常见的一个场景。作为学生，在食堂打饭的时候我们需要排队；领取十大歌手的票的时候，我们需要排队；在银行的时候，我们需要排队办理业务。

排队的重要性也是毋庸置疑的，大家自觉的排队实际上也能够规避很多潜在的安全隐患。此外，排队是现代文明的一种很重要的表现，并体现了人与人之间的相互关怀，营造出很好的社会氛围。国家甚至制定了排队日，旨在希望人们在排队的一点一滴中形成一种习惯和意识，让更多的人不自觉的履行这一公民的基本义务。

因此，研究排队问题具有极高的现实应用价值。

1.2 项目需求分析

针对于银行业务这一项目，本项目在实现的过程中，应当能够满足以下的需求：

- ✓ **功能完善**

系统应当能够满足基本需求，即能够正确的按照完成顺序输出顾客的编号。

- ✓ **执行效率高**

针对数据量比较大的情况，本系统也应该具有在较短时间内求解出正确答案的能力。

- ✓ **健壮性**

当用户输入的数据非法时，系统应当能够识别并处理错误，而非直接崩溃退出。

1.3 项目要求

1.3.1 功能要求

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。

1.3.2 输入格式

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。

1.3.3 输出格式

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。

1.3.4 项目示例

| 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 |

2 项目设计

2.1 数据结构设计

如上述功能分析所示，本项目需要实现一种先进先出的数据结构，也即队列。因此，本项目需要使用队列这一数据结构，从而满足所需要项目需求。

2.2 类设计

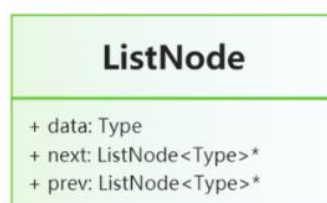
本项目中使用到队列（Queue）这一数据结构，而队列的底层数据类型则采用了链表进行实现。经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（ListNode）与链表类（List），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。

为了实现代码的复用性，本系统实现了一个**链表**。采用 struct 描述链表结点类（ListNode），这样使得链表结点类（List）可以直接访问链表结点而不需要定义友元关系。本系统实现的链表结构各种操作的时间复杂度如下：

- ✓ 插入操作： $O(n)$
- ✓ 删除操作： $O(n)$
- ✓ 查询操作： $O(n)$
- ✓ 遍历操作： $O(n)$

2.2.1 结点类（ListNode）

链表的结点中存储的数据有链表数据、后继结点和前驱结点。其 UML 图如下所示：

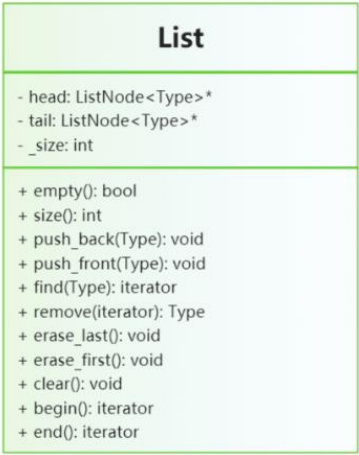


2.2.2 双向链表类（List）

链表的实现原理大同小异，不同之处在于：是否带头结点、是否带尾结点、每一个结点是否带前驱结点等。为了使得链表中各种操作的时间复杂度都尽可能

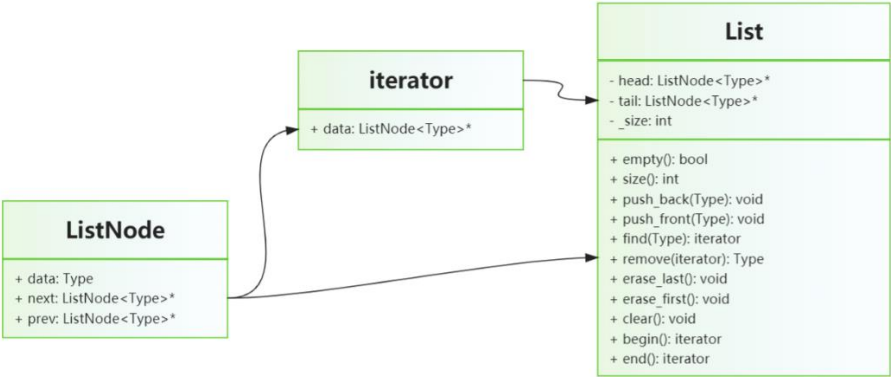
低，因此这里选择了带头结点和尾结点的双向链表来实现链表中的各种操作。也即：选择了牺牲空间来达到降低时间复杂度的效果。

链表的 UML 图如下所示：



为了便于对链表进行遍历、插入、删除、查找等操作，增加了一个 `iterator` 类。`iterator` 类内部存储一个链表节点指针，同时，通过运算符重载相应的自增、自减、判等等操作。

`iterator` 类、`ListNode` 类和 `List` 类的关系如下图所示：



其中，`List` 类中的主要函数如下所示：

- ◆ `inline int size()const`
返回链表中结点的个数，不包括头结点。
- ◆ `inline bool empty()const`
判断链表是否为空，也即链表中结点的个数是否为 0。
- ◆ `void push_back(Type data)`
在链表尾部插入新的数据，也即新增一个结点并且加入到链表末端。
- ◆ `void push_front(Type data)`
在链表头部插入新的数据，也即新增一个结点并且加入到链表的头部。
- ◆ `iterator find(const Type& data)const`
在链表中查找值为 `data` 的元素是否存在，返回该位置的迭代器，若查找失败返回空指针对应的迭代器。

◆ **Type remove(iterator index)**

移除迭代器所处位置的元素，返回移除位置元素的值。

◆ **void erase_last()**

移除链表末端的元素，即最后的结点。

◆ **void erase_first()**

移除链表首端的元素，即第一个结点。

◆ **void clear()**

清空链表，即删除链表中所有的结点。

◆ **iterator begin()**

返回链表第一个元素所在位置的迭代器。

◆ **iterator end()**

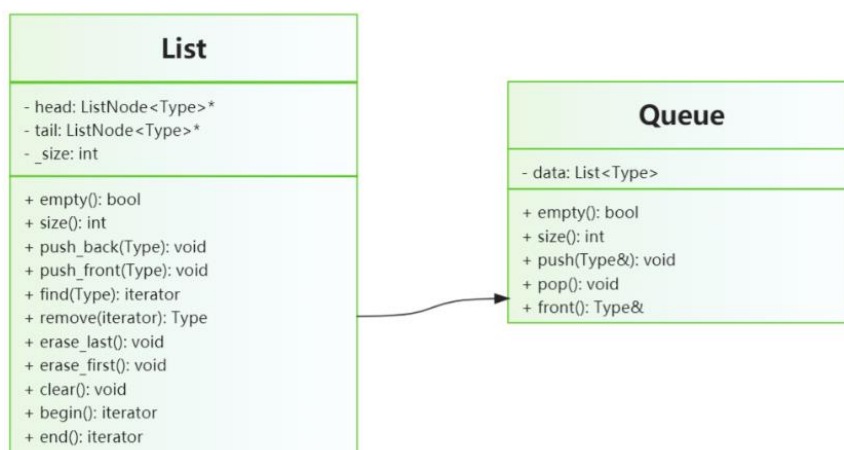
返回链表尾结点后的空结点的迭代器。

2.2.3 队列类（Queue）

队列是计算机程序中常用的数据结构，常常用于计算机模拟现实事务，如：排队等。队列是一种特殊的线性表，特殊之处在于它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作，因此是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为队头。队列中没有元素时，被称为空队列。

队列的数据元素又称为队列元素。在队列中插入一个队列元素称为入队，从队列中删除一个队列元素称为出队。因为队列只允许在一端插入，在另一端删除，所以只有最早进入队列的元素才能最先从队列中删除，故队列又称为先进先出（FIFO—first in first out）线性表。

队列的包括了顺序队列和循环队列，实现存储的底层数据可以通过向量或者链表完成。本项目中的队列以链表作为底层数据结构，实现了顺序队列。其 UML 图如下所示：



队列中主要函数如下所示：

◆ `inline bool empty()const`

判断队列是否为空，也即队列内部链表是否为空。

◆ `inline int size()const`

返回队列中链表节点的个数。

◆ `void push(const Type& i)`

在队列尾部加入一个元素，也即入队。

◆ `void pop()`

删除队列头部的元素，也即出队。

◆ `const Type& front()const`

获取队首的元素值。

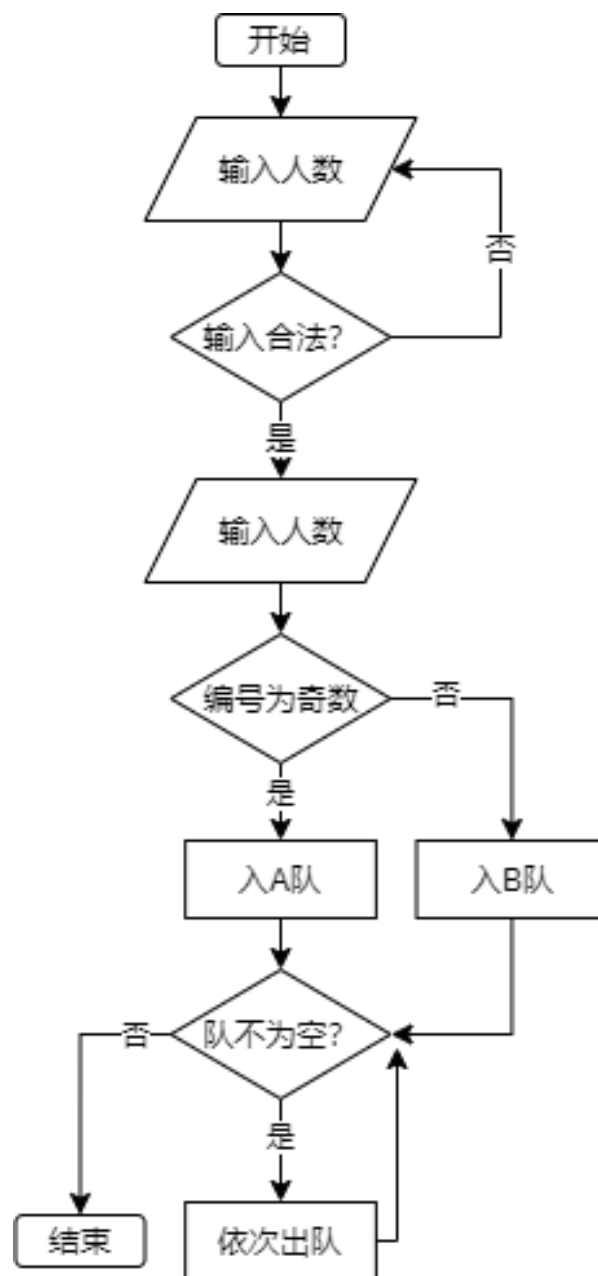
通过上述函数操作，即完成了一个队列所需要的最基本操作。其中，队的各个操作的时间复杂度如下所示：

- ✓ 入队操作： $O(1)$
- ✓ 出队操作： $O(1)$
- ✓ 读取操作： $O(1)$

3 项目实施

3.1 项目主体功能

3.1.1 项目主体功能流程图



3.1.2 项目主体功能代码

```
int N;//顾客总数
//直至输入正确
while (true)
{
    cin >> N;
    if (N <= 0)
    {
        cin.clear();
        cin.ignore(1024, '\n');
        cout << "Please input a positive." << endl;
    }
    else break;
}
Queue<int> queueA, queueB;
//顾客入队
for (int i = 1; i <= N; ++i)
{
    int temp;
    cin >> temp;
    if (temp % 2 == 0)
        queueB.push(temp);
    else
        queueA.push(temp);
}
cout << "Output:";
//出队
while (!queueA.empty() || !queueB.empty())
{
    for (int i = 0; i < 2; ++i)
        if (!queueA.empty())
        {
            cout << queueA.front() << " ";
            queueA.pop();
        }
    if (!queueB.empty())
    {
        cout << queueB.front() << " ";
        queueB.pop();
    }
}
return 0;
```

3.2 队列操作

本项目中关键操作即为队列的入队和出队操作，因此这两个函数操作需要单独提及。

3.2.1 入队操作

入队操作需要将元素加入到队列末尾，是通过队列类中的 `push` 函数操作完成的，该函数如下：

```
template<class Type>
void Queue<Type>::push(const Type& i)
{
    this->data.push_back(i);
}
```

该函数调用了成员 `data` 的 `push_back` 函数，也即链表类的加入元素到末尾，其定义如下：

```
template<class Type>
void List<Type>::push_back(Type data)
{
    //判断链表是否为空
    if (empty())
    {
        //建立新结点
        ListNode<Type>* newNode = new ListNode<Type>(data, nullptr, head);

        //改变头指针和尾指针
        head->next = newNode;
        tail = newNode;
    }
    else
    {
        //建立新结点
        ListNode<Type>* newNode = new ListNode<Type>(data, nullptr, tail);

        //改变尾指针
        tail->next = newNode;
        tail = newNode;
    }
    ++_size;
}
```

3.2.2 出队操作

出队操作需要删除队列中的首元素,是通过队列类中的 `pop` 函数操作完成的,该函数如下:

```
template<class Type>
void Queue<Type>::pop()
{
    this->data.erase_first();
}
```

该函数调用了成员 `data` 的 `erase_first` 函数,也即删除内部链表的首元素,其定义如下:

```
template<class Type>
void List<Type>::erase_first()
{
    if (!empty())
    {
        //删除tail的后继结点
        ListNode<Type>* delData = head->next;
        //判断尾结点
        if (delData == tail)
        {
            tail = head;
            head->next = nullptr;
            delete delData;
        }
        else
        {
            head->next = delData->next;
            delData->next->prev = head;
            delete delData;
        }
    }
}
```

4 项目测试

4.1 A 窗口人多的情况测试

测试用例：8 2 1 3 9 4 11 13 15

预期结果：1 3 2 9 11 4 13 15

实验结果：



```
D:\学习文件\数据结构\数据结构课设\5-银行业务\Debug\5-银行业务.exe
8
2 1 3 9 4 11 13 15
Output:1 3 2 9 11 4 13 15
```

4.2 B 窗口人多的情况测试

测试数据：8 2 1 3 9 4 11 12 16

预期结果：1 3 2 9 11 4 12 16

实验结果：



```
D:\学习文件\数据结构\数据结构课设\5-银行业务\Debug\5-银行业务.exe
8
2 1 3 9 4 11 12 16
Output:1 3 2 9 11 4 12 16 _
```

4.3 A 窗口和 B 窗口人数相同的情况测试

测试用例：8 13 3 2 8 7 6 12 1

预期结果：13 3 2 7 1 8 6 12

实验结果：



```
D:\学习文件\数据结构\数据结构课设\5-银行业务\Debug\5-银行业务.exe
8
13 3 2 8 7 6 12 1
Output: 13 3 2 7 1 8 6 12
```

4.4 最小人数的情况

测试用例：1 6

预期结果：6

实验结果：



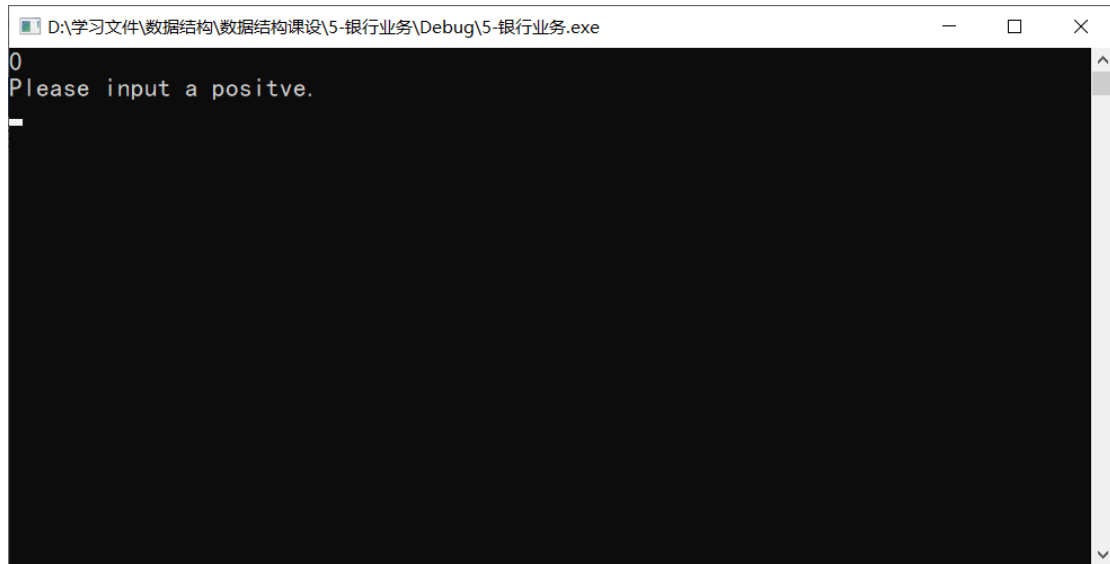
```
D:\学习文件\数据结构\数据结构课设\5-银行业务\Debug\5-银行业务.exe
1
6
Output: 6
```

4.5 非法输入测试

测试用例：0

预期结果：提示输入有误

实验结果：



```
D:\学习文件\数据结构\数据结构课设\5-银行业务\Debug\5-银行业务.exe
0
Please input a positive.
_
```