项目说明文档

数据结构课程设计

——银行业务

作 者 姓 名： 郑柯凡

学 号： 1950072

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目录

[1 分析 2](#_Toc23390)

[1.1背景分析 2](#_Toc15966)

[1.2功能分析 3](#_Toc25046)

[2 设计 3](#_Toc32505)

[2.1思路设计 3](#_Toc2586)

[2.2数据结构的选择 3](#_Toc5511)

[2.3类的设计 3](#_Toc12157)

[3 处理业务功能实现 6](#_Toc9428)

[3.1处理业务功能流程图 6](#_Toc21336)

[3.2处理业务功能核心代码 6](#_Toc21414)

[3.3处理业务功能截屏示例 7](#_Toc18650)

[4测试 8](#_Toc31183)

[4.1 A窗口无人 8](#_Toc7282)

[4.2 B窗口无人 8](#_Toc7801)

[4.3正常测试，A窗口人多 8](#_Toc31554)

[4.4正常测试，B窗口人多 8](#_Toc6715)

**1 分析**

**1.1背景分析**

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。假定不考虑顾客先后到达的时间间隔，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。

**1.2功能分析**

输入一行正整数，数字间以空格分隔，其中第一个数字N（N<=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。程序需要按照业务处理完成的顺序输出顾客的编号。数字间以空格分隔，但是最后一个编号不能有多余的空格。

**2 设计**

**2.1思路设计**

银行A和B两个窗口可以抽象成两个队列模型。各个顾客按照自己编号的奇数偶数分别选择A窗口或B窗口，每个窗口前的顾客又根据先后关系依次排成一个队列。每个时刻只需要判断队头的人是否完成业务即可，若队头的人完成了业务，就将它输出，若队头的人没有完成业务，就将它处理业务的时间加1，等到下个时刻再看。

**2.2数据结构的选择**

由于顾客需要按照先来后到的顺序依次处理，因此采用先进先出的队列结构进行存储顾客序列。

**2.3类的设计**

顾客结点类：其私有成员包括num，储存顾客的编号；time，储存顾客处理业务的时间。

1. //节点类
2. **class** Customer {
3. **friend** **class** Bank;
4. **private**:
5. //存储的数据
6. **int** num;
7. **int** time;
8. **public**:
9. Customer() { num = 0; time = 0; }
10. ~Customer() {};
11. };

（以下代码包含在MySTL.h头文件中）

队列结点类：其私有成员包括一个未定类型的数据变量，以及指向下一个结点地址的指针变量。其公有成员包括获取数据，获取next地址，修改数据，修改next地址四个接口函数。

1. //结点
2. **template** <**class** Type>
3. **class** Node {
4. **private**:
5. Type data;
6. Node<Type>\* next;
7. **public**:
8. Node() { next = NULL; }
9. ~Node() {};
10. //获取结点信息
11. Type GetData() { **return** data; }
12. //获取下个结点地址
13. Node<Type>\* GetNext() { **return** next; }
14. //修改数据
15. **void** ModifyData(Type tmp)
16. {
17. data = tmp;
18. };
19. //修改地址
20. **void** ModifyNext(Node<Type>\*tmp)
21. {
22. next = tmp;
23. }
24. };

链式队列类，储存排队顾客的信息：其私有成员包括队头队尾元素的地址，便于插入删除操作；以及size记录队列的长度。其公有成员包括判空，入队，出队等功能函数，其实现与链表的插入删除操作别无二致。

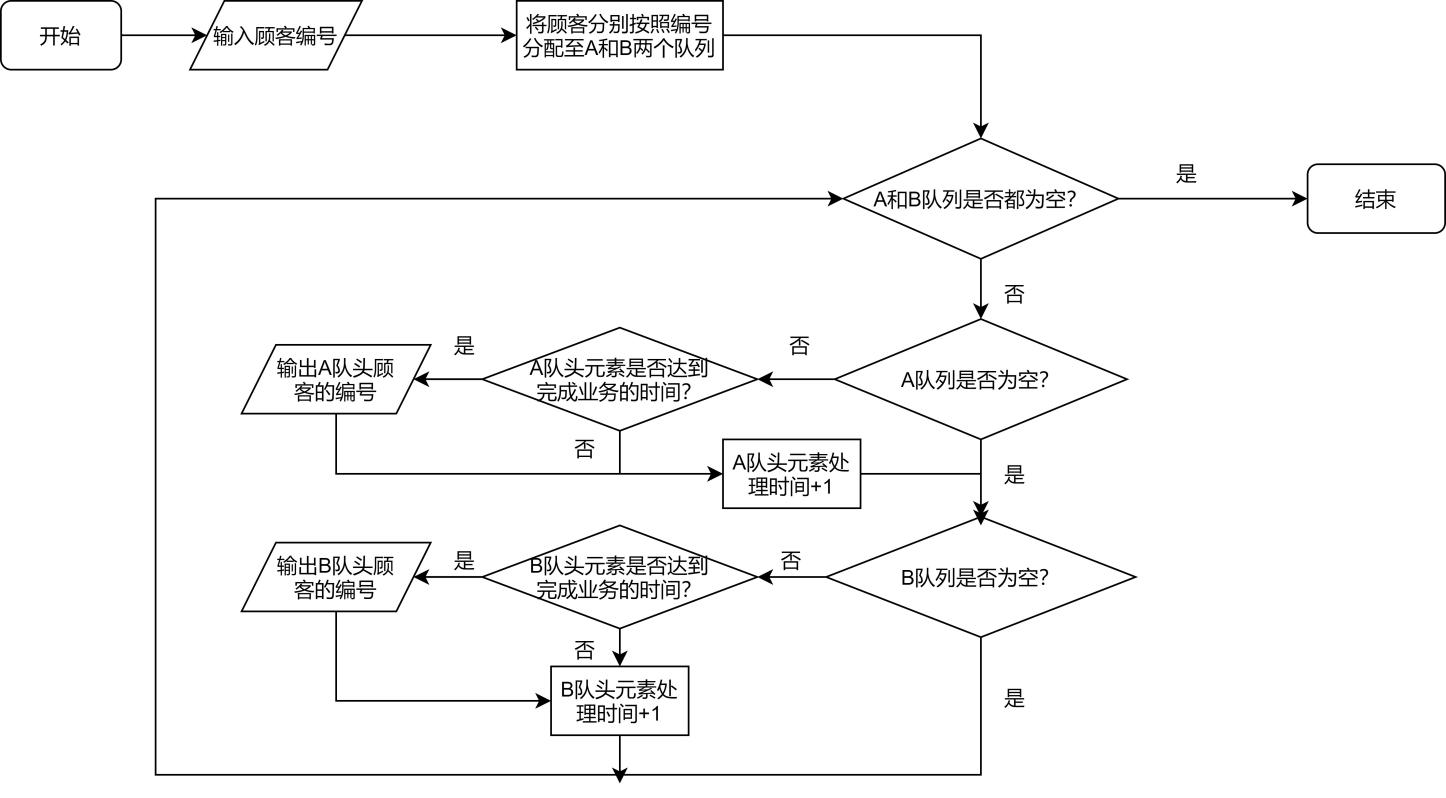
1. //链式队列
2. **template** <**class** Type>
3. **class** MyQueue {
4. **private**:
5. //队列头尾地址
6. Node<Type>\* front;
7. Node<Type>\* rear;
8. //队列的长度
9. **int** size;
10. **public**:
11. //队列的构造函数
12. MyQueue() {
13. front = rear = NULL; size = 0;
14. };
15. //析构函数
16. ~MyQueue() {};
17. //判断队列是否为空
18. **bool** IsEmpty() { **return** (size == 0) ? 1 : 0; }
19. //添加节点
20. **void** EnQueue(Type element)
21. {
22. //队列为空
23. **if** (front == NULL)
24. {
25. front = rear = **new** Node<Type>;
26. rear->ModifyData(element);
27. }
28. //队列非空
29. **else**
30. {
31. Node<Type>\* temp = **new** Node<Type>;
32. rear->ModifyNext(temp);
33. rear = rear->GetNext();
34. rear->ModifyData(element);
35. }
36. size++;
37. };
38. //队头出列
39. **void** DeQueue()
40. {
41. **if** (IsEmpty() == 0)
42. {
43. Node<Type>\* temp = front;
44. front=front->GetNext();
45. **delete** temp;
46. size--;
47. }
48. };
49. //获取队头
50. Node<Type>\* GetFront() { **return** front; }
51. //获取队尾
52. Node<Type>\* GetRear() { **return** rear; }
53. //获取队列长度
54. **int** GetSize() { **return** size; }
55. };

银行业务类：其私有成员包括A，B两个队列，分别对应A，B两个窗口。其公有函数只有一个Run(),用来处理A和B两个窗口的业务，将完成业务的顾客依次输出。

1. //银行业务类
2. **class** Bank {
3. **private**:
4. MyQueue<Customer>queueA;
5. MyQueue<Customer>queueB;
6. **public**:
7. Bank() {}
8. ~Bank() {};
9. //处理业务
10. **void** Run();
11. };

**3 处理业务功能实现**

**3.1处理业务功能流程图**



**3.2处理业务功能核心代码**

先根据奇偶分配各个顾客的队伍。

先判断A，B队列是否全部为空，若空则处理完毕，退出程序；若不空则说明还有顾客未处理完。由于题目中要求，A，B同时时先输出A，因此先对A队列处理。若A空，则直接处理B；若A不空，则判断其当前队头元素是否满足完成业务的时间，若完成了，则将其输出并出队，并给新队头元素的处理时间加1；若没有完成，则将其处理时间加1，下个时刻再看。B队列的处理方式与A相同。

1. **void** Bank::Run()
2. {
3. //输入放进两个队列
4. cout << "请输入顾客人数和各顾客编号：";
5. **int** total = 0; **int** num2;
6. Customer temp ;
7. cin >> total;
8. **for** (**int** i = 0; i < total; i++)
9. {
10. cin >> num2;
11. temp.num = num2;
12. **if** (num2 % 2 == 1)
13. {
14. queueA.EnQueue(temp);
15. }
16. **else** **if** (num2 % 2 == 0)
17. {
18. queueB.EnQueue(temp);
19. }
20. }
21. //根据先后顺序输出
22. **while** (queueA.IsEmpty() == 0 || queueB.IsEmpty() == 0)
23. {
24. **if** (queueA.IsEmpty() == 0)
25. {
26. **if** (queueA.GetFront()->GetData().time == 1)
27. {
28. cout << queueA.GetFront()->GetData().num;
29. queueA.DeQueue();
30. **if** (!queueA.IsEmpty() || !queueB.IsEmpty())
31. {
32. cout << " ";
33. }
34. }
35. **if** (queueA.IsEmpty() == 0)
36. {
37. temp.num = queueA.GetFront()->GetData().num;
38. temp.time = queueA.GetFront()->GetData().time + 1;
39. queueA.GetFront()->ModifyData(temp);
40. }
42. }
43. **if** (queueB.IsEmpty() == 0)
44. {
45. **if** (queueB.GetFront()->GetData().time == 2)
46. {
47. cout << queueB.GetFront()->GetData().num;
48. queueB.DeQueue();
49. **if** (!queueA.IsEmpty() || !queueB.IsEmpty())
50. {
51. cout << " ";
52. }
53. }
54. **if** (queueB.IsEmpty() == 0)
55. {
56. temp.num = queueB.GetFront()->GetData().num;
57. temp.time = queueB.GetFront()->GetData().time + 1;
58. queueB.GetFront()->ModifyData(temp);
59. }
60. }
61. }
62. }

**3.3处理业务功能截屏示例**





**4测试**

**4.1 A窗口无人**

**测试用例：**4 2 4 6 8

**预期结果：**2 4 6 8

**实验结果：**



**4.2 B窗口无人**

**测试用例：**4 1 3 5 7

**预期结果：**1 3 5 7

**实验结果：**



**4.3正常测试，A窗口人多**

**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：**



**4.4正常测试，B窗口人多**

**测试用例：**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：**

