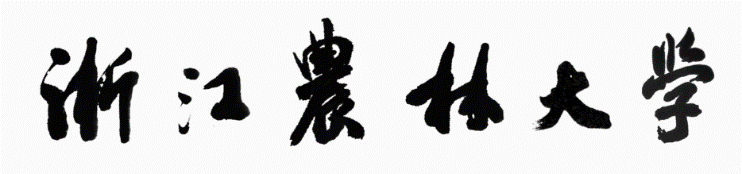
**** 

**《操作系统》期末课程实验报告**

**（2020-2021学年第一学期）**

**Name（姓名）：周炜翔、张金浩**

**Student ID（学号）：201805120532、201805120629**

**Department（学院）：信息工程学院**

**Major（专业）：计算机科学与技术**

**Class（班级）：183班**

**Date（日期）：2021/01/05**

目录

[一、课题设计的目的 3](#_Toc60854536)

[二、课程设计的问题描述 3](#_Toc60854537)

[2.1 问题描述 3](#_Toc60854538)

[2.2 问题要求 3](#_Toc60854539)

[三、详细设计 4](#_Toc60854540)

[3.1 算法流程图 4](#_Toc60854541)

[3.2 流程描述 4](#_Toc60854542)

[3.3.1 数据初始化 4](#_Toc60854543)

[3.3.2 进入模拟FIFO运算过程 5](#_Toc60854544)

[3.3.3 显示所有结果，程序结束 5](#_Toc60854545)

[四、源代码 5](#_Toc60854546)

[五、测试结果 9](#_Toc60854547)

[5.1 测试样例表格一览 9](#_Toc60854548)

[5.2 测试样例对应截图 10](#_Toc60854549)

[六、总结 22](#_Toc60854550)

# 一、课题设计的目的

理解以及实现FIFO(先进先出)算法，了解Belady现象。

# 二、课程设计的问题描述

## 2.1 问题描述

先进先出（FIFO）页面置换算法程序设计，通过对FIFO算法的模拟，进一步理解进程的基本概念，加深对进程运行状态和进程调度过程、调度算法的理解。

## 2.2 问题要求

用C++语言编程实现对FIFO算法的模拟。

# 三、详细设计

## 3.1 算法流程图

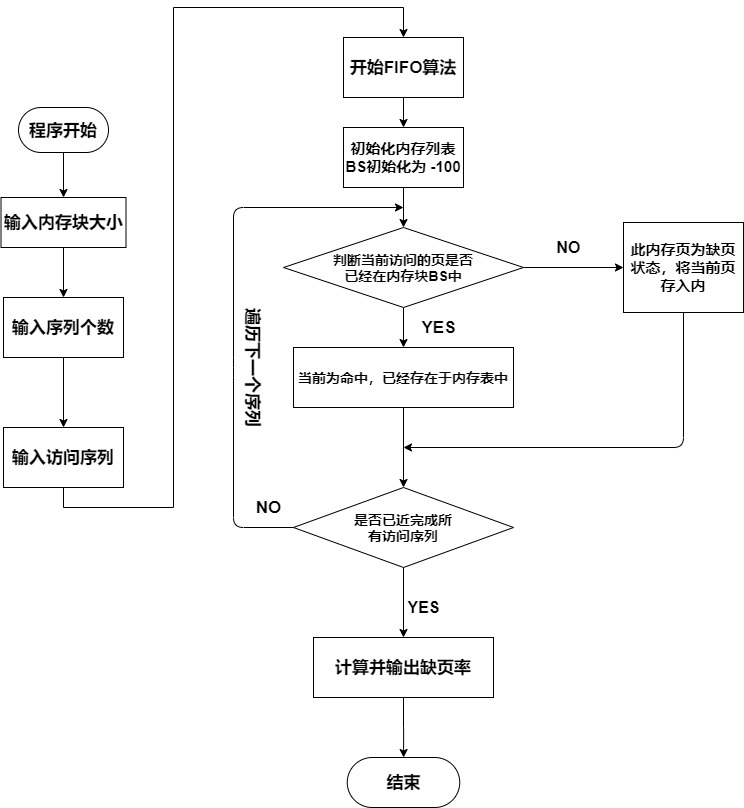


图3-1 算法流程图

## 3.2 流程描述

### 3.3.1 数据初始化

1. 数据初始化，主要包含，访问序列以及内存块大小的读取；
2. 主要实现：通过C语言的标准流式输入。

### 3.3.2 进入模拟FIFO运算过程

1. 循环初始化访问内存表（BS）为-100，主要目的是为了表示内存页的一个空闲状态。
2. 循环遍历我们的访问序列，从第一个开始一次判断，如果当前这个访问数在内存页列表内已近存在表示改值已经命中，进行下一次判断，如果当前的下标小于内存块数目则直接存入记为缺页标记值（bno）加一，否则记为命中未缺页，标记值（bno）加一。如果，当前的访问值不在内存块序列中flag记为0，则记为缺页将当前的值存入内存块，并且标记值加一（bno）
3. 根据上一步中我们流下的标记值统计缺页的操作次数，循环显示每个序列号的缺页情况，并且计算缺页率。

### 3.3.3 显示所有结果，程序结束

# 四、源代码

1. #include<iostream>
2. **using** **namespace** std;
4. #define Max1 25 //访问序列数组大小
5. #define Max2 15  //内存块表数组大小
7. **int** saveCount[Max1][Max1];
8. **int** x\_save, y\_save;
9. **int** now\_couont = 0;
11. **struct** pt {
12. **int** pno;    //页号
13. **int** bno;    //块号
14. **int** flag;   //状态位，为0时在不内存，为1时在内存
15. **int** order;  //优先序列
16. };
18. // 输入
19. **void** input(**int** \*a, **int** n)
20. {
21. **for** (**int** i = 0; i < n; i++) {
22. cin >> \*a;
23. a++;
24. }
25. }
27. // 输出序列
28. **void** output(**int** \*a, **int** n)
29. {
30. **for** (**int** i = 0; i < n; i++) {
31. cout << \*a << ' ';
32. a++;
33. }
34. cout << '\n';
35. }
37. // 算法本体
38. **void** fifo(**int**\*List\_pages, **int**\*bs, **int** n, **int** m)//n 序列个数  m 块个数
39. {
40. pt ptlist[Max1];//定义结构数组

43. **int** k = 0, flag, cn = 0, i, j;//cn——统计缺页数
44. **for** (j = 0; j < m; j++)//赋初值 m 块个数
45. {
46. bs[j] = -100;
47. }
49. **for** (i = 0; i < n; i++)// 访问序列循环  n 序列个数
50. {
51. flag = 0;
52. // 判断当前的值是否已经在 内存块中 ，如果在 flag 直接 = 1
53. **for** (j = 0; j < m; j++)
54. **if** (List\_pages[i] == bs[j]) {
55. flag = 1;
56. **break**;
57. }
59. **if** (flag == 1)//命中
60. {
61. **if** (i >= m) {
62. ptlist[i].bno = j + 1;
63. ptlist[i].flag = 1; //命中，已存入，未缺页
64. ptlist[i].pno = List\_pages[i];//ye
65. }
66. **else** { // 前 m 页直接载入就好了
67. ptlist[i].flag = 0;//缺页，还未存入
68. ptlist[i].pno = List\_pages[i];
69. bs[k] = List\_pages[i];//将当前页存入内存块
70. ptlist[i].bno = k + 1;
71. k = (k + 1) % m;//循环队列
72. cn++;
73. }
74. }
75. **else** { // 因为 目标值不在块中
76. ptlist[i].flag = 0;//缺页，还未存入
77. ptlist[i].pno = List\_pages[i];
79. bs[k] = List\_pages[i];//将当前页存入内存块
80. ptlist[i].bno = k + 1;
81. k = (k + 1) % m;//循环队列
82. cn++;
83. }
85. cout << "第 " << i << "次进入 \n";
86. **for** (**int** i = 0; i < m; i++)
87. {
88. saveCount[now\_couont][i] = bs[i];
89. }
90. now\_couont++;
91. }
92. cout << "计算结果：\n";
93. cout << "---------------------------------------------------------------------\n";
94. cout << "缺页个数：" << '\t' << cn << '\n';
95. cout << "---------------------------------------------------------------------\n";
96. cout << "访问次数：" << '\t' << n << '\n';
97. cout << "---------------------------------------------------------------------\n";
98. cout << "缺页率为：" << '\t' << (**float**)cn / n << '\n';
99. cout << "---------------------------------------------------------------------\n";
100. cout << "操作顺序：\n";
101. cout << "---------------------------------------------------------------------\n";
102. **for** (i = 0; i < m; i++)
103. {
104. cout << List\_pages[i] << "\t!!缺页,还未存入!!\t" << "现已直接存入内存块!\t" << ptlist[i].bno << '\n';
105. cout << "本次序列： >>>>>  ";
106. **for** (**int** j = 0; j < y\_save; j++)
107. {
108. **if** (saveCount[i][j] != -1 && saveCount[i][j] != -100)
109. cout << saveCount[i][j] << ' ';
110. }
111. cout << endl;
112. cout << "---------------------------------------------------------------------\n";
113. }
114. **for** (i = m; i < n; i++)
115. {
116. **if** (ptlist[i].flag == 0)
117. cout << List\_pages[i] << "\t!!缺页,还未存入!!\t" << "操作：调出块号为：" << ptlist[i].bno << "--页号为" << ptlist[i].pno << "的进程" << '\n';
118. **else**
119. cout << List\_pages[i] << "\t!!命中,已经存入!!\t" << "操作：查询块号为：" << ptlist[i].bno << "--页号为" << ptlist[i].pno << "的进程" << '\n';
120. cout << "本次序列： >>>>>  ";
121. **for** (**int** j = 0; j < y\_save; j++)
122. {
123. **if** (saveCount[i][j] != -1 && saveCount[i][j] != -100)
124. cout << saveCount[i][j] << ' ';
125. }
126. cout << endl;
127. cout << "---------------------------------------------------------------------\n";
128. }
130. }
132. **void** initGroup(**int** x , **int** y)
133. {
134. x\_save = x;
135. y\_save = y;
136. **for** (**int** i = 0; i < Max1; i++)
137. **for** (**int** j = 0; j < Max1; j++)
138. saveCount[i][j] = -1;
139. }
141. **void** main()
142. {
143. **int** List\_pages[Max1], bs[Max1];
144. **int** n, m;
146. cout << "输入内存块大小：\n";
147. cin >> m;
148. cout << "输入序列个数：\n";
149. cin >> n;
150. initGroup(n, m);
152. cout << "请输入访问序列：\n";
153. input(List\_pages, n);
154. cout << "访问序列：" << endl;
155. output(List\_pages, n);
156. cout << "---------------------------------------------------------------------\n";
157. cout << '\n';
158. fifo(List\_pages, bs, n, m);
159. }

# 五、测试结果

## 5.1 测试样例表格一览

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 实验数据 | 理想结果 | 预期结果 | 是否正确 | 对应图片 |
| 01 | 输入内存块大小-1 | 报错，并要求重新输入 | 报错，并要求重新输入 | 是 | 5-2-01 |
| 02 | 输入内存块大小0 | 报错，并要求重新输入 | 报错，并要求重新输入 | 是 | 5-2-02 |
| 03 | 输入内存块大小3 | 正确，输入序列个数 | 正确，输入序列个数 | 是 | 5-2-03 |
| 04 | 输入序列个数-1 | 报错，并要求重新输入 | 报错，并要求重新输入 | 是 | 5-2-04 |
| 05 | 输入序列个数0 | 报错，并要求重新输入 | 报错，并要求重新输入 | 是 | 5-2-05 |
| 06 | 输入序列个数17 | 正确，输入序列 | 正确，输入序列 | 是 | 5-2-06 |
| 07 | 70120304230321201 | 正确，输出结果 | 正确，输出结果 | 是 | 5-2-07 |
| 08 | 输入内存块大小4 | 正确，输入序列个数 | 正确，输入序列个数 | 是 | 5-2-08 |
| 09 | 输入序列个数17 | 正确，输入序列 | 正确，输入序列 | 是 | 5-2-09 |
| 10 | 70120304230321201 | 正确，输出结果 | 正确，输出结果 | 是 | 5-2-10 |
| 11 | 输入内存块大小3 | 正确，输入序列个数 | 正确，输入序列个数 | 是 | 5-2-11 |
| 12 | 输入序列个数12 | 正确，输入序列 | 正确，输入序列 | 是 | 5-2-12 |
| 13 | 123412512345 | 正确，输出结果 | 正确，输出结果 | 是 | 5-2-13 |
| 14 | 输入内存块大小4 | 正确，输入序列个数 | 正确，输入序列个数 | 是 | 5-2-14 |
| 15 | 输入序列个数12 | 正确，输入序列 | 正确，输入序列 | 是 | 5-2-15 |
| 16 | 123412512345 | 正确，输出结果 | 正确，输出结果 | 是 | 5-2-16 |
| 17 | 输入内存块大小1 | 正确，输入序列个数 | 正确，输入序列个数 | 是 | 5-2-17 |
| 18 | 输入序列个数1 | 正确，输入序列 | 正确，输入序列 | 是 | 5-2-18 |
| 19 | 0 | 错误 | 错误 | 是 | 5-2-19 |
| 20 | 输入内存块大小1 | 正确，输入序列个数 | 正确，输入序列个数 | 是 | 5-2-20 |
| 21 | 输入序列个数1 | 正确，输入序列 | 正确，输入序列 | 是 | 5-2-21 |
| 22 | 1 0 | 错误 | 错误 | 是 | 5-2-22 |

## 5.2 测试样例对应截图

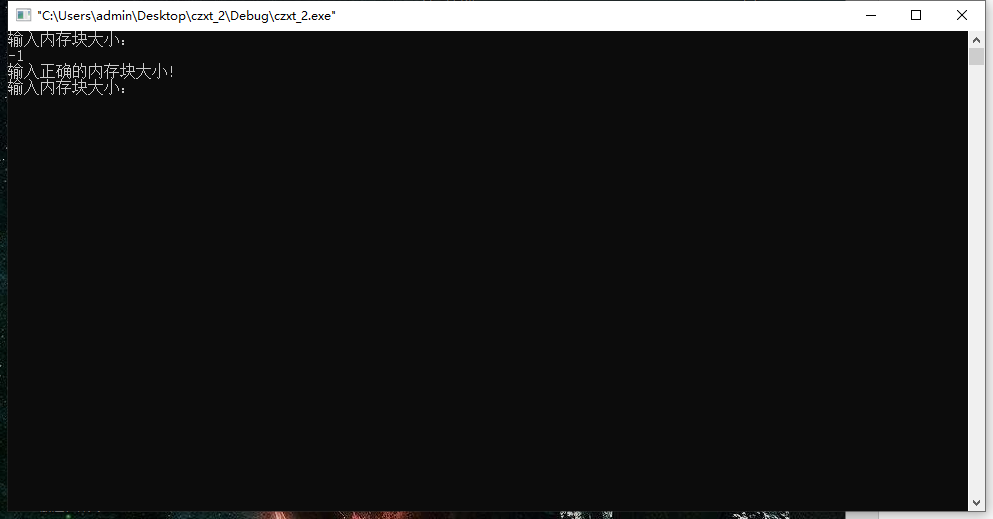


图 5-2-01

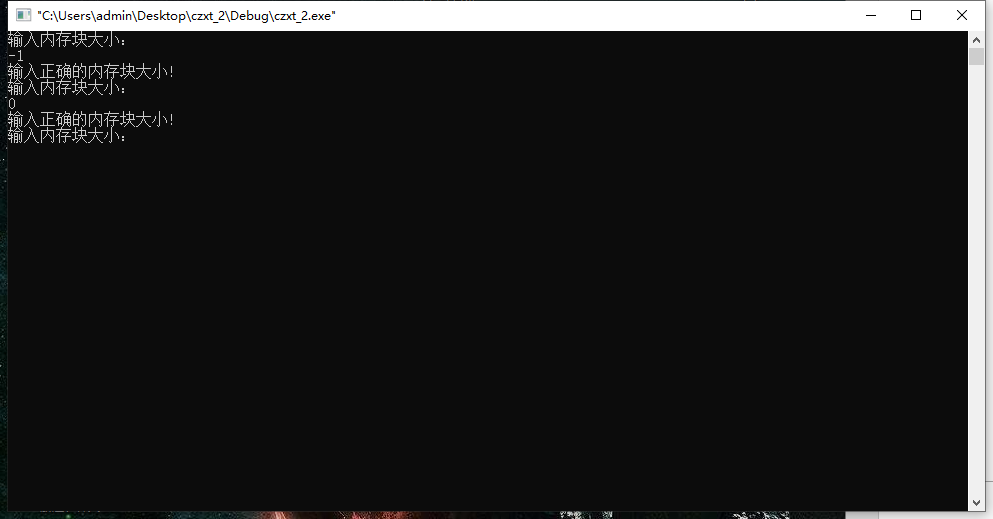


图 5-2-02

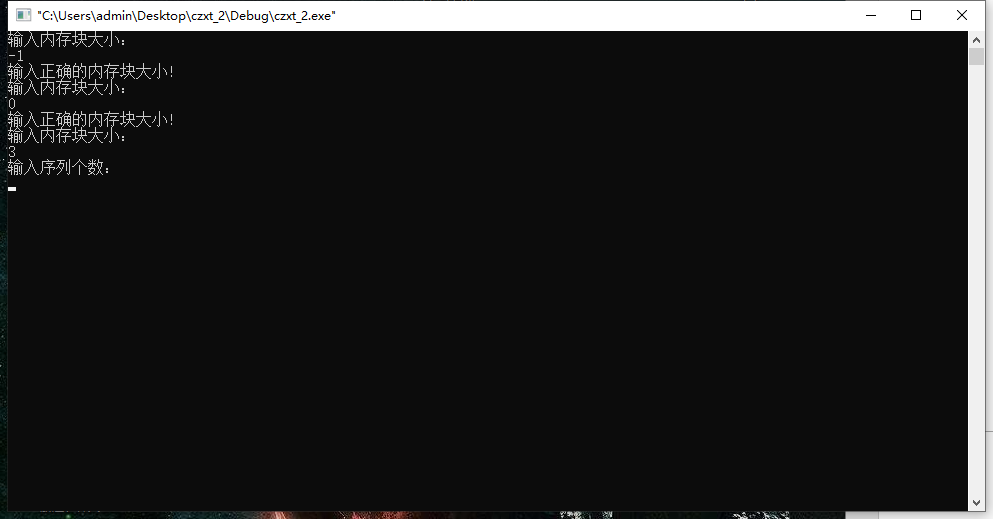


图 5-2-03

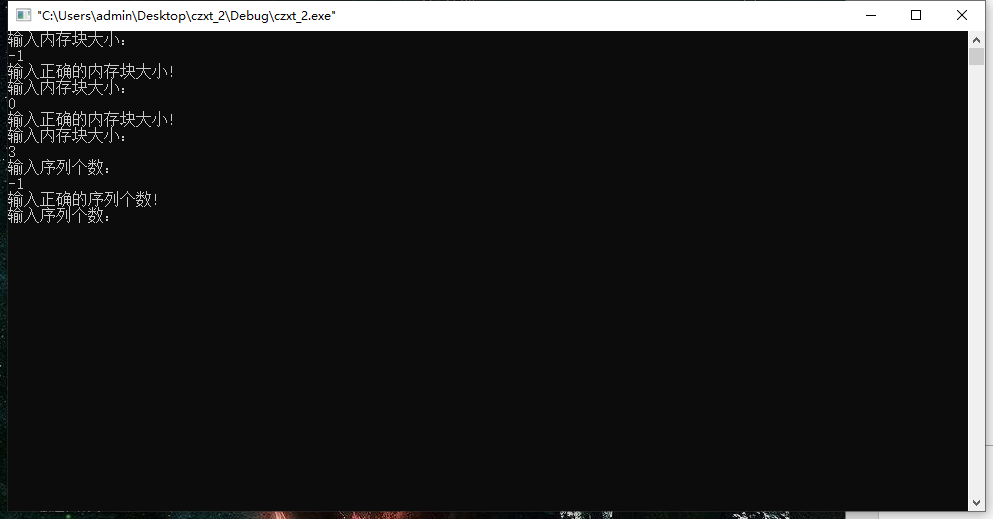


图 5-2-04

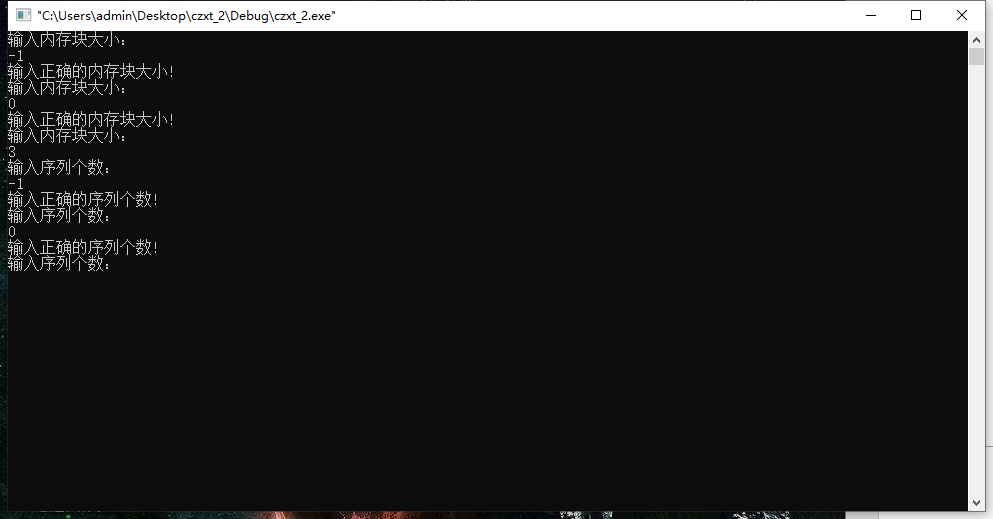


图 5-2-05

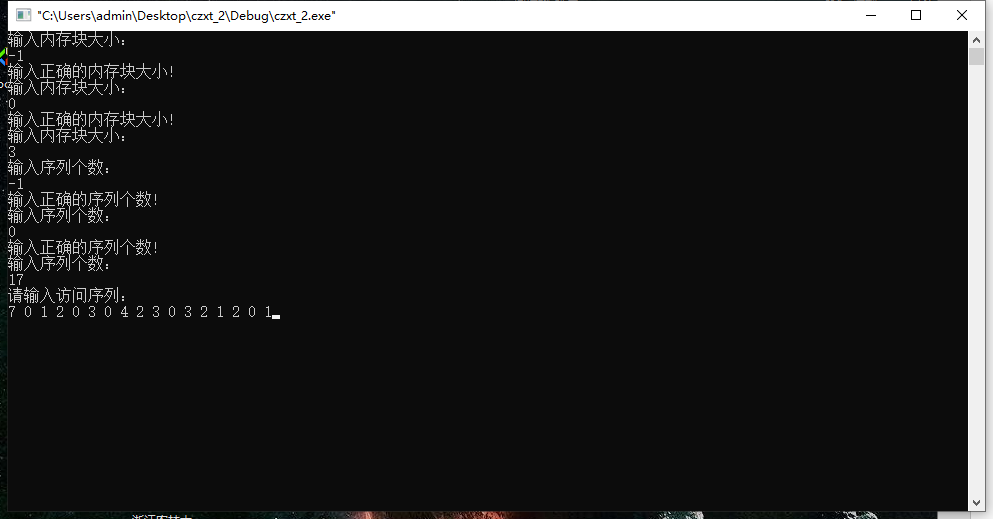


图 5-2-06

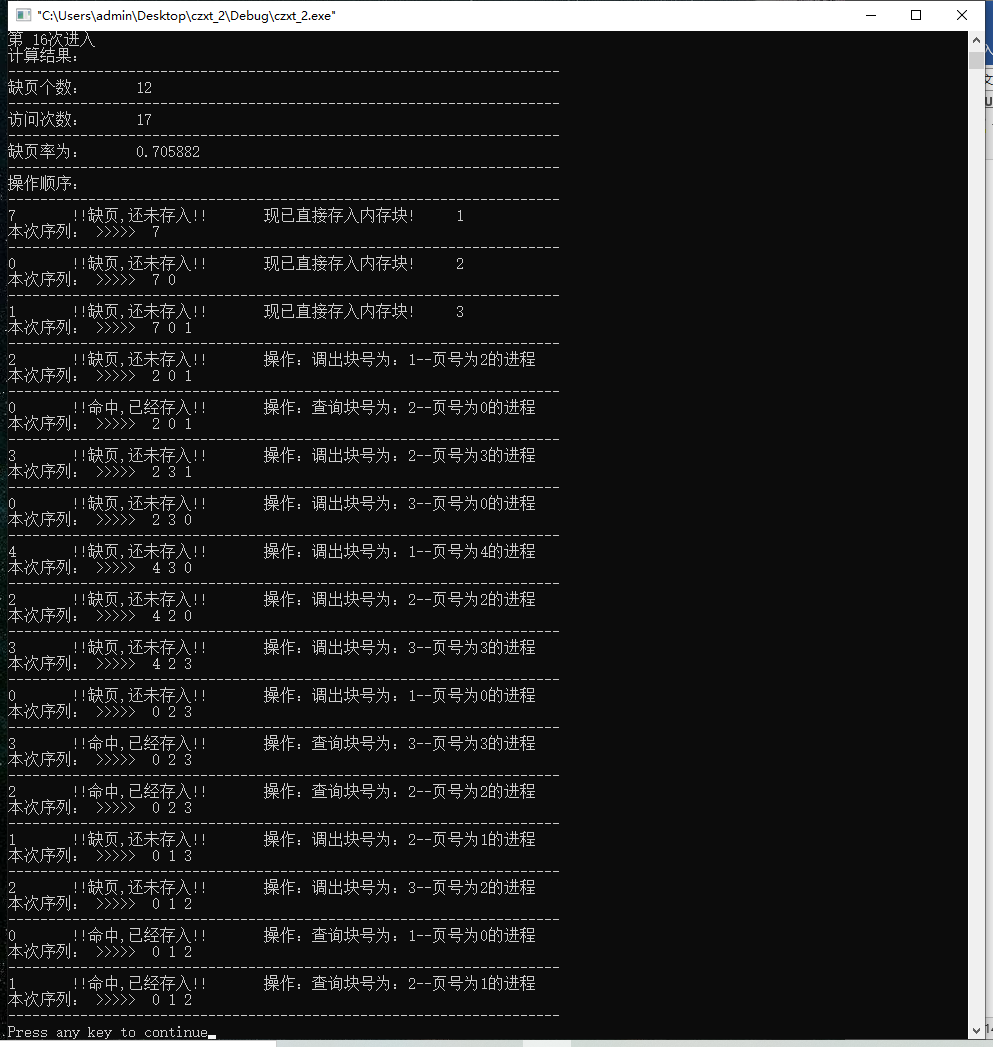


图 5-2-07

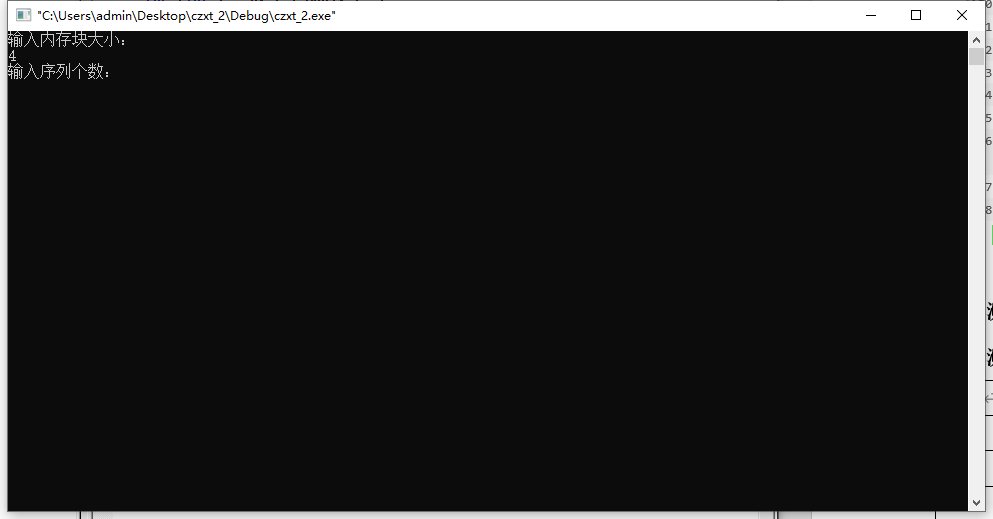


图 5-2-08

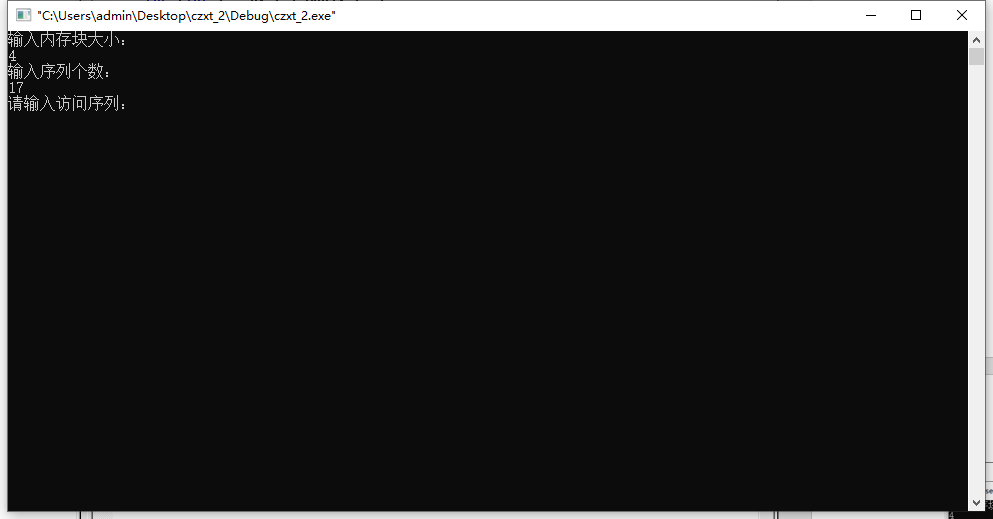


图 5-2-09

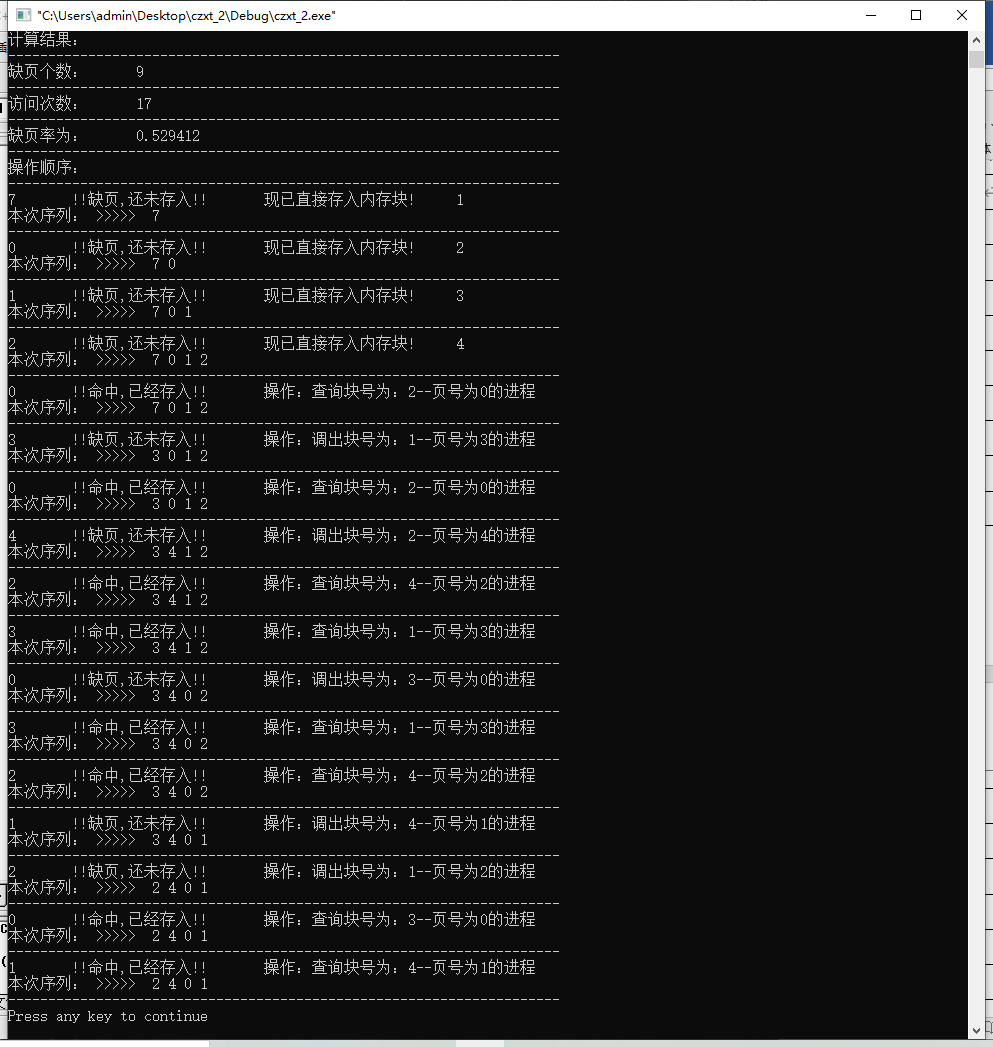


图 5-2-10

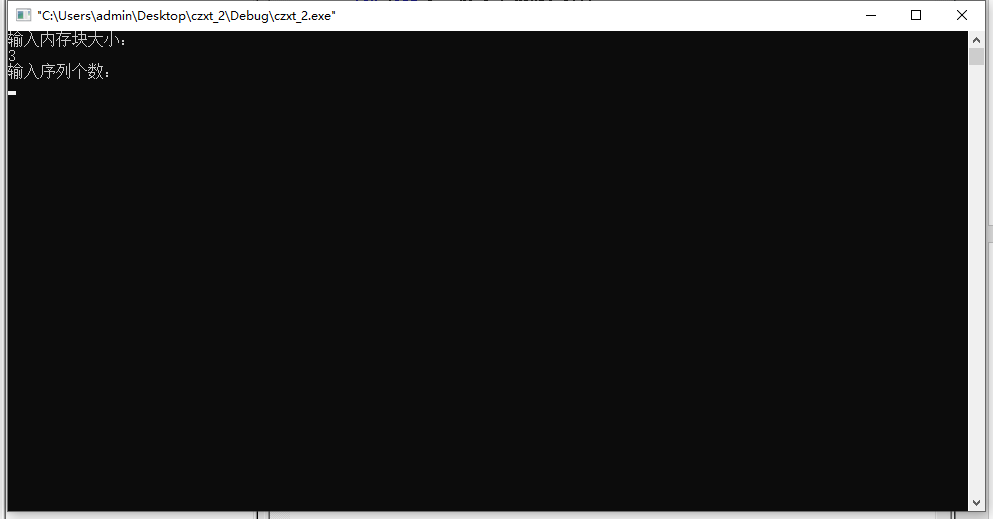


图 5-2-11

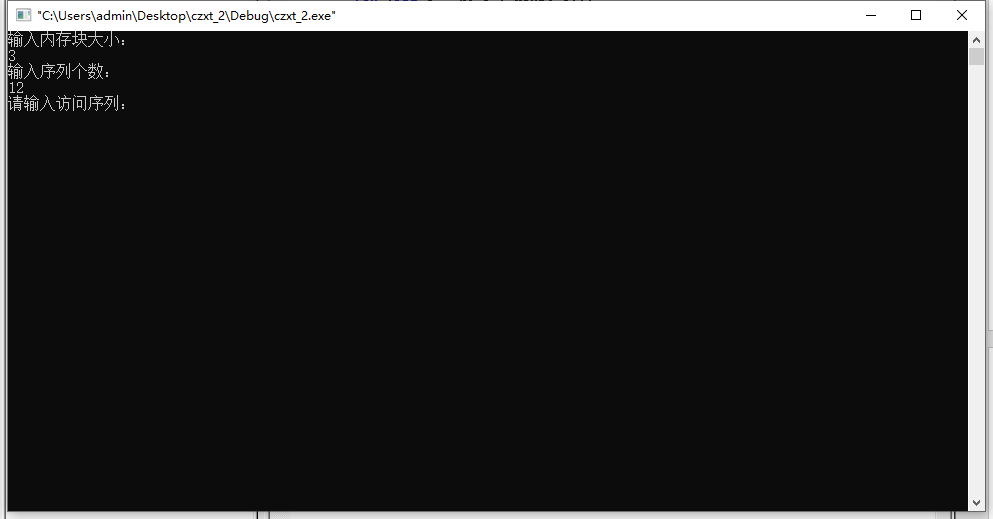


图 5-2-12

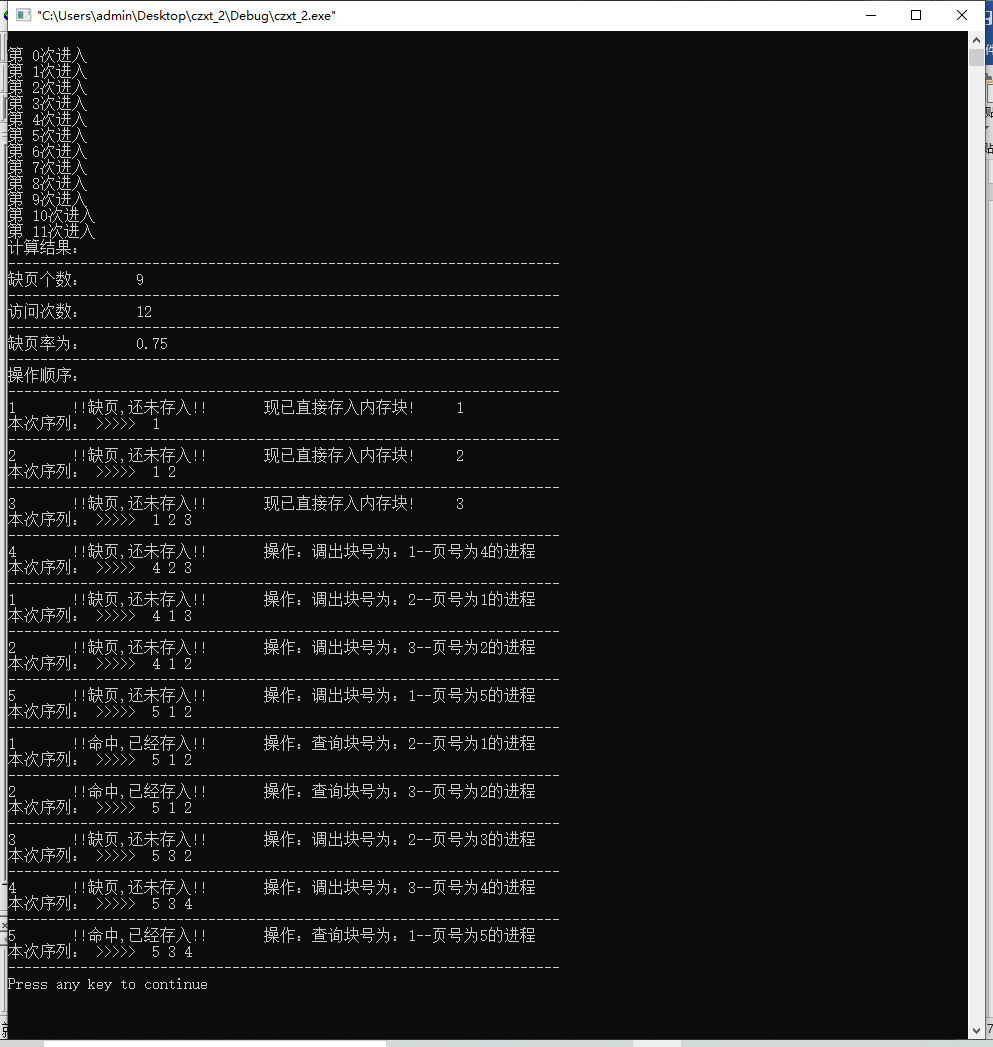


图 5-2-13

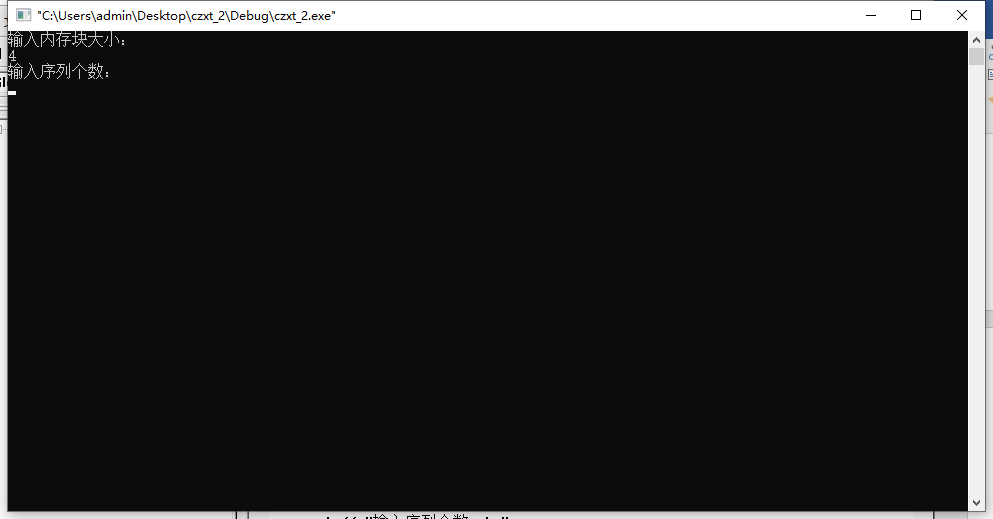


图 5-2-14

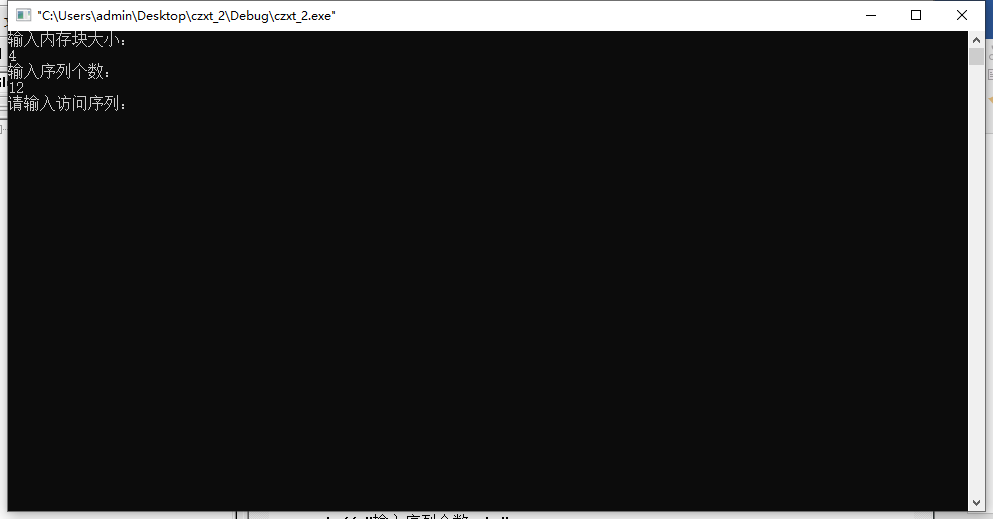


图 5-2-15

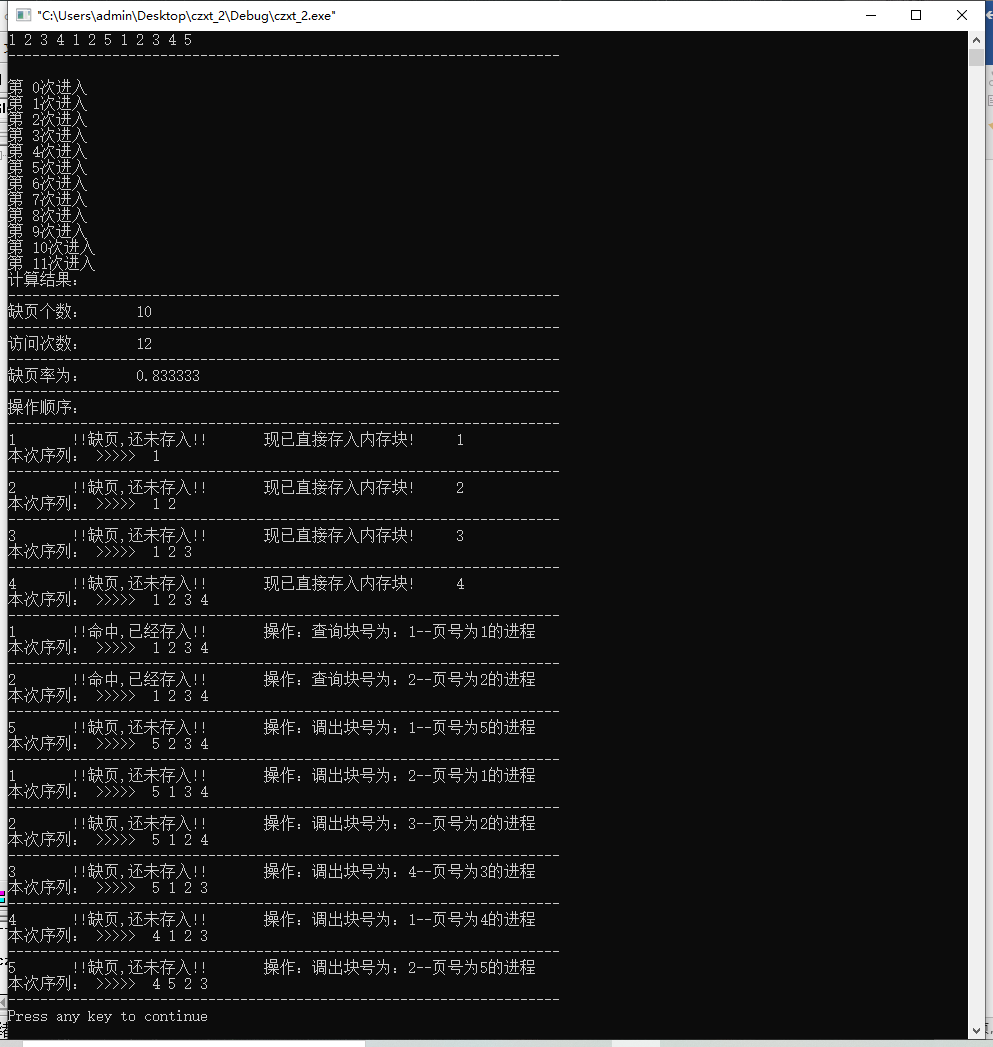


图 5-2-16

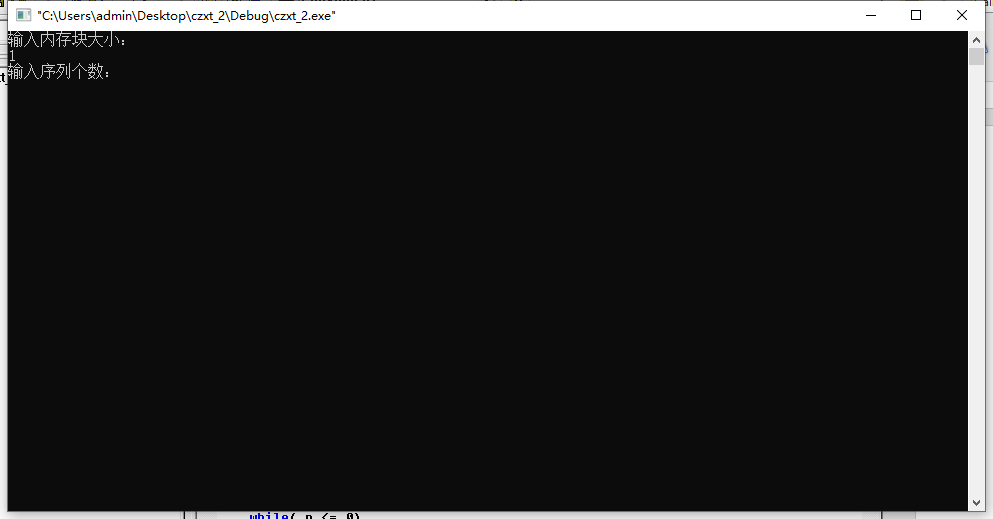


图 5-2-17

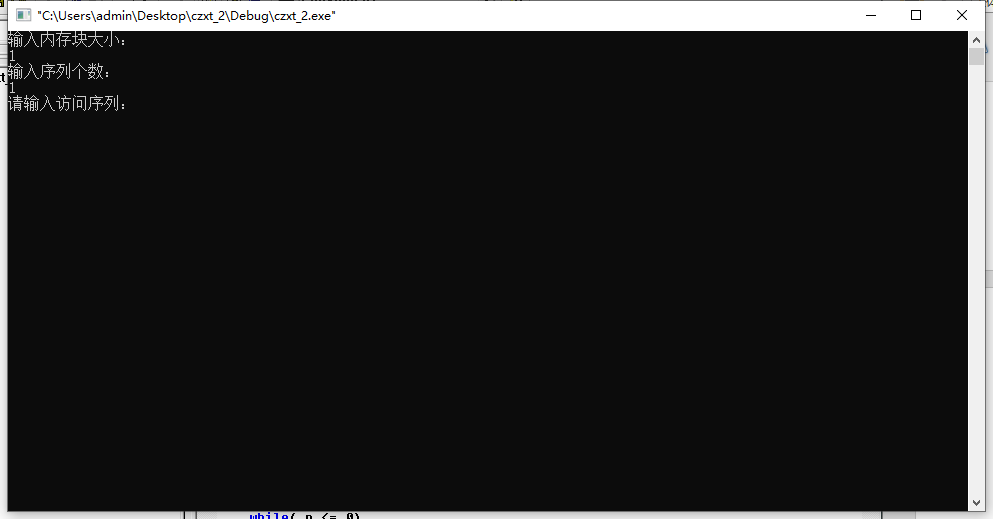


图 5-2-18



图 5-2-19

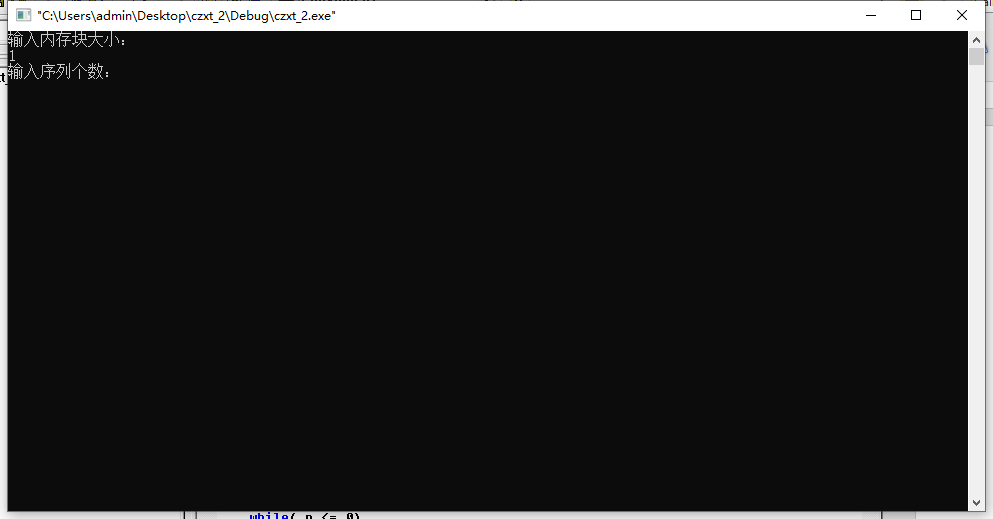


图 5-2-20

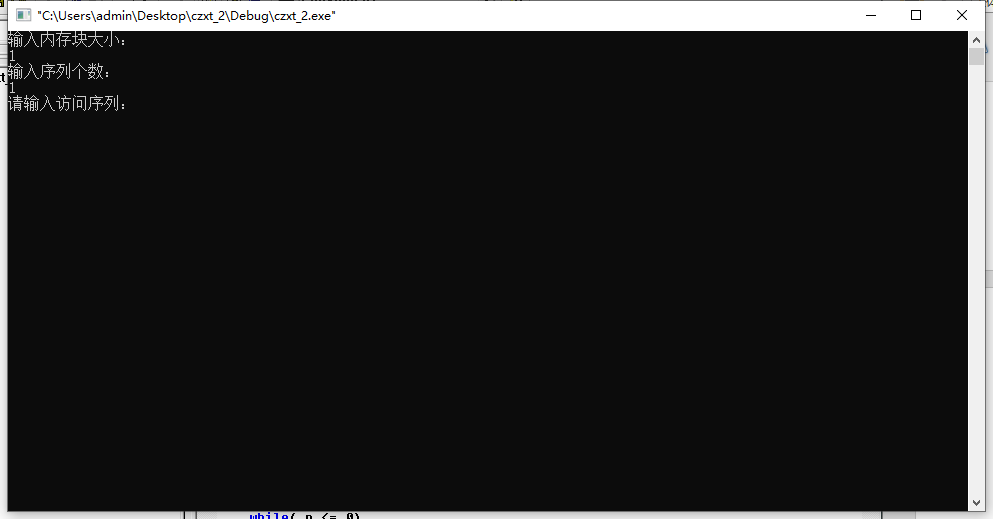


图 5-2-21



图 5-2-22

# 六、总结

本次的操作系统课题在代码实现上还是比较容器的，主要就是一个先进先出的FIFO的算法和Belady现象产生的原因。

拿到课题后我们先去了解了一下关于操作系统内存分配的相关知识，在这块主要有随机淘汰算法、先进先出算法（FIFO）、最近最久未使用算法（LRU）、理想型淘汰算法（OPT）等几个算法,对几个算法的运算流程有一个初步的了解。

在代码设计之初，针对队列的数据结构，我们一开始准备使用数组模拟，实际的使用效果并不太好，由于我们对数据要进行多次遍历以及内存调配（标记值的变化），所以我们最后选用了双端链表，在遍历以及修改数据上都非常方便。在完成初步算法框架后代码依然有很多的bug，在边界情况的处理上还有很多问题。在查阅很多资料以及断点调试后，我们成功的实现了FIFO算法模型。

由于算法本身比较简单，所以实现起来还是很容易的。在学习完成FIFO的空闲时间里我们还学习对比了其他几个算法 。从对比来看，FIFO算法对内存的利用率其实并不高，而且非常容易出现缺页问题，由于不带任何标记只是简单的淘汰进程，在内存分配和合理性上也存在一些问题，LRU算法相对而言更加适合现在的计算机，他在FIFO的基础上增加了一层标记，淘汰的时候标记也作为一个参考依据，提高了内存的利用率。

此次课题让我们对操作系统这么课程又有了更大兴趣和新的体悟，我们后面会继续学习更多的操作系统知识。