

**实验报告**



**题目： 实验一：链表-栈-队列综合应用实验**

**班 级： 2023211314**

**学 号： 2023211304**

**姓 名： 唐术洋**

**学 院： 计算机学院**

**2024年 10 月 24 日**

1. 实验目的
2. 练习链表的实现及操作，学习用链表解决实际问题；
3. 练习栈与队列的实现及操作，学习用栈与队列解决实际问题；
4. 锻炼程序编写及调试的能力；
5. 学习自己查找相关资料以解决实际问题的能力。

二、实验环境

简述使用的工具（操作系统+c环境+编译器）

Windows，vscode，gcc

三、实验内容

**场景设定：**

假设有一副扑克牌，用两位字符来表示扑克牌的牌面，第一位字符用A/B/C/D代表四种花色，第二位字符用1……9、0(代表10)、J、Q、K表示数值，用XX和YY代表大小王。初始扑克牌的序列为：A1、A2、……、AK、B1、B2……、DK、XX、YY。

**要求：**

* 将代表54张扑克牌的初始序列放入到顺序表中。
* 洗牌：将54张扑克牌打乱顺序，可以调研学习一下洗牌算法（Fisher-Yates Shuffle算法或Knuth-Durstenfeld Shuffle算法）
* 插入循环链表：依次将洗牌后的元素从顺序表中取出，插入到一个循环链表中。
* 定位：在循环链表中定位到第一个出现的大王或小王牌。
* 报数+出牌：从上一步定位到牌开始从1计数，当计数到M（预先设定的一个数字）时，将牌打出。然后从下一张牌重新开始从1计数，直至剩余一张牌时游戏结束，请输出最后剩余的一张牌。（约瑟夫环问题）
* 到游戏结束时，最后被打出的五张A开头的牌是什么？（可使用队列记录最多五张打出的A开头的牌）
* **附加要求**：假如当打出的牌是大小王时，可以取回最多五张（不足五张时全部取走，超过五张时取五张）最近打出的牌并插入到循环链表中，那么最后剩余的一张牌以及最后被打出的五张A开头的牌又是什么。（可使用栈记录打出的牌）

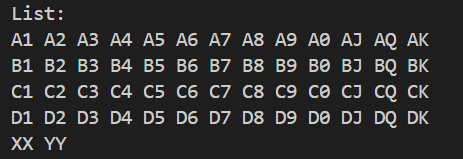
1. 实验步骤（基础80分+附加10分）

整体步骤：编写程序，然后运行程序得出结果并截图。（为了防止结果截图之间相互割裂，故选择一次性输出所有要求的部分并截图）

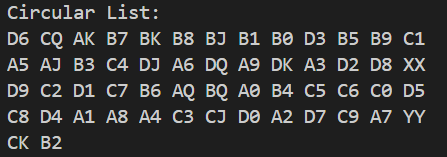
**提供的截图和源代码中M的值设置为10；**

1. 基础部分

构建扑克牌序列并输出（数据结构为单链表）：尾插法构建单链表



调用Fisher-Yates Shuffle算法打乱牌序并输出（数据结构为单向循环链表）：

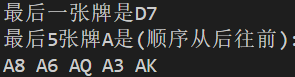


找到第一个大王/小王的位置：通过循环判断结点数据域字符数组的ch[0]是否为X或Y，如果是，则停止并返回位置。

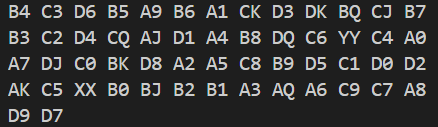


通过核心算法，得到最后一张牌（该项是通过函数返回的指针内数据部分输出），并通过栈从后往前输出最后五张A。

**核心算法：**整个函数有一个变量check2来记录打出的牌数，当check2为53时退出并返回剩下的单个节点。每当计数达到规定值M，计数从下一个结点开始归1并将当前节点通过指针方法从循环链表当中移除，并将值深拷贝到pend数组里。

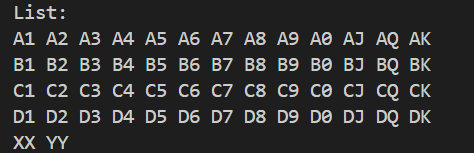


（以下为牌堆内牌的顺序，pend数组从头顺序输出）：

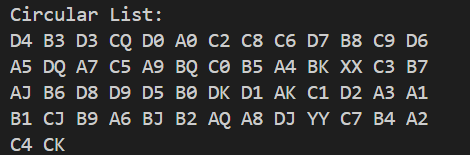


1. 附加部分

构建扑克牌序列并输出（数据结构为单链表）：



调用Fisher-Yates Shuffle算法打乱牌序并输出（数据结构为单向循环链表）：

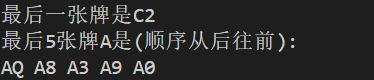


找到第一个大王/小王的位置：

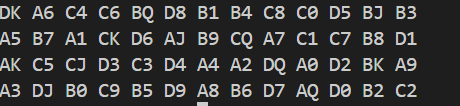


通过核心算法，得到最后一张牌（该项是通过指针内数据部分输出），并通过栈从后往前输出最后五张A：

**核心算法：**整个函数有一个变量check2来记录打出的牌数，当check2为53时退出并返回剩下的单个节点。每当计数达到规定值M，计数从下一个结点开始归1并将当前节点通过指针方法从循环链表当中移除，并将值深拷贝到pend数组里。如果该节点为大/小王，则从pend数组末依次读取n个节点的值（当pend数组内元素大于5则n为5，否则为数组内元素个数）构建出一个单链表并按照循环链表遍历顺序插入到原链表中，并进行指针复位。最后将pend数组的元素计数减n，末尾n个元素的数据置0.



（以下为牌堆内的牌的顺序，pend数组从头顺序输出，因为大小王被丢弃，所以只会有52个元素）



五、实验分析和总结（10分）

实验整体过程比较简单，基本要求的程序部分是一遍过，但是在编写附加要求的源代码时因为链表指针操作不太熟练导致有一个指针复位的问题我花了很久才发现（之前一直认为是循环链表断开的问题）。后面通过bug的一些特性我自行设计了2套测试方案，在第二套方案测试时发现了这个因指针未复位导致程序逻辑混乱的bug（也就差了一行代码）。

这个实验让我对栈，链表等数据结构的使用更为熟练，提高了我程序编写的水准以及debug的能力。

1. 程序源代码（10分）

**PS：有两份源代码。因为两份代码的差异只有CheckTheLastNode一个函数内执行逻辑的不同，所以附加要求的源代码只提交该函数。**

1. **不包含附加要求的源代码：**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

typedef struct node

{

char ch[3];

struct node \*next;

} element;

element \*CreateInitialList()// 创建顺序表（带有空头节点）

{

element \*tail = NULL;

element \*L = (element \*)malloc(sizeof(element));

for (int i = 0; i <= 2; i++)

L->ch[i] = '\0';

L->next = NULL;

tail = L;

// 插入扑克牌 A-D, 1-13

for (char i = 'A'; i <= 'D'; i++)

{

for (int h = 1; h <= 13; h++)

{

element \*p = (element \*)malloc(sizeof(element));

p->ch[2] = '\0';

p->ch[0] = i;

if (h <= 9)

p->ch[1] = h + '0';

else

{

switch (h)

{

case 10:

p->ch[1] = '0';

break;

case 11:

p->ch[1] = 'J';

break;

case 12:

p->ch[1] = 'Q';

break;

case 13:

p->ch[1] = 'K';

break;

}

}

tail->next = p;

tail = p;

}

}

// 插入大小王 'X', 'Y'

for (char c = 'X'; c <= 'Y'; c++)

{

element \*p = (element \*)malloc(sizeof(element));

p->ch[0] = c;

p->ch[1] = c;

p->ch[2] = '\0';

p->next = NULL;

tail->next = p; // 连接新节点

tail = p; // 更新尾指针

}

return L->next;

}

void ShuffleList(element \*\*L) // 洗牌函数

{

if (\*L == NULL || (\*L)->next == NULL)

return;

// 计算链表长度

int length = 0;

element \*p = \*L;

while (p != NULL)

{

length++;

p = p->next;

}

// 将链表元素存入数组

element \*\*array = (element \*\*)malloc(length \* sizeof(element \*));

p = \*L;

for (int i = 0; i < length; i++)

{

array[i] = p;

p = p->next;

}

// Fisher-Yates 洗牌算法

srand(time(NULL)); // 随机数种子

for (int i = length - 1; i > 0; i--)

{

int j = rand() % (i + 1); // 随机选择一个索引

// 交换 array[i] 和 array[j]

element \*temp = array[i];

array[i] = array[j];

array[j] = temp;

}

// 将打乱顺序的元素重新链接回链表

for (int i = 0; i < length - 1; i++)

{

array[i]->next = array[i + 1];

}

array[length - 1]->next = NULL; // 最后一个节点的 next 设为 NULL

\*L = array[0]; // 更新链表头指针

free(array); // 释放数组内存

}

void PrintList(element \*L)//打印链表

{

int i = 0;

element \*p = L;

printf("List: \n");

while (p != NULL)

{

printf("%s ", p->ch);

i++;

if (i % 13 == 0)

printf("\n");

p = p->next;

}

printf("\n");

}

element \*FindtheXY(element \*L, int \*h) // 表示大王/小王在链表中是第h个

{

element \*p = L;

while (p != NULL && p->next != L)

{

if (p->ch[0] == 'X' || p->ch[0] == 'Y')

{

(\*h)++;

return p;

}

p = p->next;

(\*h)++;

}

}

element \*InsertToCircularList(element \*L) // 将洗牌后的链表元素插入单循环链表

{

if (L == NULL)

return NULL;

element \*head = NULL; // 新的循环链表头指针

element \*tail = NULL; // 循环链表尾指针

element \*p = L; // 遍历洗牌后的链表

// 遍历链表并将节点插入循环链表中

while (p != NULL)

{

element \*newNode = (element \*)malloc(sizeof(element));

newNode->ch[0] = p->ch[0];

newNode->ch[1] = p->ch[1];

newNode->ch[2] = '\0';

newNode->next = NULL;

if (head == NULL)

{ // 第一个节点

head = newNode;

tail = newNode;

}

else

{ // 尾插结点

tail->next = newNode;

tail = newNode;

}

p = p->next;

}

// 将尾节点连接到头节点

if (tail != NULL)

{

tail->next = head;

}

return head;

}

void PrintCircularList(element \*L)

{ // 打印单循环链表

if (L == NULL)

{

printf("Empty Circular List\n");

return;

}

element \*p = L;

int i = 0;

printf("Circular List: \n");

do

{

printf("%s ", p->ch);

i++;

if (i % 13 == 0)

printf("\n");

p = p->next;

} while (p != L); // 当再次到达头节点时停止

printf("\n");

}

element \*CheckTheLastNode(element \*\*target, element \*pend, int h, int M, int \*pe1) //得到最后一张牌

{

element \*p = \*target;

int k = 1, check = 2, check2 = 0;

// 将指针移动到第1个大王/小王的位置

while (k < h && p != NULL) {

p = p->next;

k++;

}

element \*p2 = p->next;

while (p2 != NULL && check2 < 53) //操作53次，直接返回最后一个结点

{

if (check % M == 0) {

pend[\*pe1].ch[0] = p2->ch[0];

pend[\*pe1].ch[1] = p2->ch[1];

pend[\*pe1].ch[2] = '\0';

if(\*pe1!=0)

pend[\*pe1].next = &pend[(\*pe1)-1]; //得到一个由后往前指的链表

else

pend[\*pe1].next = NULL;//最先打出的牌不指向任何节点

(\*pe1)++;

// 移除 p2 节点并调整链表结构

p->next = p2->next;

free(p2); // 释放 p2 节点

p2 = p->next; // 更新 p2 指向 p 的下一个节点

check = 1; // 重置检查计数

check2++;

} else

{

p = p->next;

p2 = p2->next;

check++;

}

}

pend[\*pe1].ch[0] = p->ch[0];//对栈的最后一位进行赋值

pend[\*pe1].ch[1] = p->ch[1];

pend[\*pe1].ch[2] = '\0';

pend[\*pe1].next = NULL;

return p;//返回最后一个结点用于输出

}

void FindTheLast5Card(element\* pend)

{

int h=1;

printf("最后5张牌A是(顺序从后往前):\n");

for(int i=53;i>=0&&h<=5;i--)

{

if(pend[i].ch[0]=='A')

{

printf("%c%c ",pend[i].ch[0],pend[i].ch[1]);

h++;

}

}

printf("\n");

}

int main()

{

element \*pend = (element \*)malloc(sizeof(element) \* 54); // 用于存储被打出的牌

int h = 0, M = 10, pe1 = 0;

element \*L = CreateInitialList();

PrintList(L);

ShuffleList(&L);

element \*circularList = InsertToCircularList(L);

PrintCircularList(circularList);

printf("第一个大王/小王在链表的第%d个位置,是%s", h, FindtheXY(circularList, &h));

element\* last = CheckTheLastNode(&circularList, pend, h, M, &pe1);

printf("\n最后一张牌是%c%c",last->ch[0],last->ch[1]);

printf("\n");

FindTheLast5Card(pend);

for(int i=0;i<54;i++)

{

printf("%c%c ",pend[i].ch[0],pend[i].ch[1]);

if((i+1)%13==0)

printf(“\n”);

}

return 0;

}

1. **含附加要求的源代码：**

element \*CheckTheLastNode(element \*\*target, element \*pend, int h, int M, int \*pe1, element \*king, int \*k1) // 得到最后一张牌

{

element \*p = \*target;

int k = 1, check = 1, check2 = 0;

\*k1=0;

// 将指针移动到第1个大王/小王的位置

while (k < h && p != NULL)

{

p = p->next;

k++;

}

element \*p2 = p->next;

while (p2 != NULL && check2 < 53) // 直接返回最后一个结点

{

if (check % M == 0)

{

if (p2->ch[0] == 'X' || p2->ch[0] == 'Y')

{

king[\*k1].ch[0] = p2->ch[0];

king[\*k1].ch[1] = p2->ch[1];

king[(\*k1)++].ch[2] = '\0';

p->next = p2->next;

free(p2); // 释放 p2 节点

p2 = p->next; // 更新 p2 指向 p 的下一个节点

check = 1; // 重置检查计数

check2++;

int count = \*pe1 < 5 ? \*pe1 : 5;

if (count > 0) // pend数组内有元素

{

element \*new\_head = (element \*)malloc(sizeof(element));

new\_head->ch[0] = pend[\*pe1 - 1].ch[0];

new\_head->ch[1] = pend[\*pe1 - 1].ch[1];

new\_head->ch[2] = pend[\*pe1 - 1].ch[2];

new\_head->next = NULL;

element \*current = new\_head;

for (int i = \*pe1 - 2; i >= \*pe1 - 5 && i >= 0; i--)

{

element \*new\_node = (element \*)malloc(sizeof(element));

new\_node->ch[0] = pend[i].ch[0];

new\_node->ch[1] = pend[i].ch[1];

new\_node->ch[2] = pend[i].ch[2];

new\_node->next = NULL;

current->next = new\_node;

current = new\_node;

}

current->next = p->next; // 插回到单循环链表中

p->next = new\_head;

p=current;

for (int i = \*pe1 - count; i < \*pe1; i++)

{

pend[i].ch[0] = '\0';

pend[i].ch[1] = '\0';

pend[i].ch[2] = '\0';

}

\*pe1 -= count;

check2 -= count;

}

}

else // 非大王小王情况

{

pend[\*pe1].ch[0] = p2->ch[0];

pend[\*pe1].ch[1] = p2->ch[1];

pend[\*pe1].ch[2] = '\0';

(\*pe1)++;

// 移除 p2 节点并调整链表结构

p->next = p2->next;

free(p2); // 释放 p2 节点

p2 = p->next; // 更新 p2 指向 p 的下一个节点

check = 1; // 重置检查计数

check2++;

}

}

else

{

p = p->next;

p2 = p2->next; // 出现段错误时p为null

check++;

}

}

pend[\*pe1].ch[0] = p->ch[0]; // 对栈的最后一位进行赋值

pend[\*pe1].ch[1] = p->ch[1];

pend[\*pe1].ch[2] = '\0';

pend[\*pe1].next = NULL;

return p; // 返回最后一个结点用于输出

}