第3次作业

## 第3章习题

3.1对一个栈的输入序列a1,a2,a3,… ,an,，称由此栈依次出栈后所得到的元素序列为栈的合法输出序列。例如，假设栈S的一个输入序列为1,2,3,4,5，则可得到多个输出序列，例如，1,2,3,4,5就是一个合法的输出序列，同理，5,4,3,2,1和3,2,1,4,5也分别是其合法的输出序列。分别求解下列问题：

（1）判断序列1,3,4,5,2是否是合法的输出序列。

（2）对输入序列1,2,3,4,5，求出其所有的合法的输出序列。

（3\*）设计算法以判断对输入序列1,2,3,… ,n，序列a1,a2,a3,… ,an是否是该栈的合法的输出序列（假设输出序列在数组A中）。

3.4对一个合法的数学表达式来说，其中的各大小括号“{”，“}”，“[”，“]”，“(”和“)”应是相互匹配的。设计算法对以字符串形式读入的表达式S，判断其中的各括号是否是匹配的。

3.5对表达式5+3\*(12+4)/4-8，依次画出在求解过程中的各步骤中的栈的状态。

3.9对教材中所讨论的循环队列及其约定，给出求解队列中元素个数的表达式。

3.11如果采用带尾指针的单循环链表（带头结点）作为队列的存储结构，设计算法以实现队列的各运算。

3.1(1)合理

（2）利用卡特兰数求出有42中

#include <iostream>

#include <stack>

#include <vector>

using namespace std;

int number = 0;

void func(vector<int>kind, int count[], int n, int A[])

{

if (count[0] >= 1)

{

kind.push\_back(1);

count[0]--;

func(kind, count, n, A);

count[0]++;

kind.pop\_back();

}

if ((count[1] >= 1) && (count[1] > count[0]))

{

kind.push\_back(0);

count[1]--;

func(kind, count, n, A);

count[1]++;

kind.pop\_back();

}

if (kind.size() == 2 \* n)

{

vector<int>::iterator iter;

stack<int>stk;

int j = 0;

for (iter = kind.begin(); iter != kind.end(); iter++)

{

//cout<<(\*iter)<<" ";

if (1 == (\*iter))

{

stk.push(A[j]);

j++;

}

else

{

cout << stk.top() << " ";

stk.pop();

}

}

number++;

cout << endl;

}

}

int main()

{

int n, i;

cout << "please input the number:" << endl;

cin >> n;

int A[100];

cout << "please input the push sequence:" << endl;

for (i = 0; i < n; i++)

{

cin >> A[i];

}

int count[2] = { n - 1,n };

vector<int>kind;

kind.push\_back(1);

cout << "the result is:" << endl;

func(kind, count, n, A);

cout << "total:" << number << endl;

return 0;

}

12345 13245 14325 21354 23154 24351 32415 34521 45321 12354 13254

14352 21435 23415 24531 32451 35421 54321 12435 13425 14532 21453

23451 25431 32541 43215 12453 13452 15432 21543 23541 32145 34215

43251 12543 13542 21345 23145 24315 32154 34251 43521

（3）

#include <iostream>

#include <stack>

using namespace std;

bool outresult(int n)

{

cout << "请输入n的值";

while (cin >> n, n != 0)

{

int\* a = (int\*)malloc(sizeof(int) \* n);

int b[100];

int flag = 1;

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

{

cin >> a[i];

}

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

{

int c = 0;

for (int j = i + 1; j < n; j++)

{

if (a[i] > a[j])

{

b[c++] = a[j];

}

}

for (int k = 0; k < c - 1; k++)

{

if (b[k] < b[k + 1])

{

flag = 0;

break;

}

}

if (!flag)

{

break;

}

}

if (flag)

{

return true;

}

}

}

int main()

{

int n=0;

int flag = outresult(n);

if(flag)cout << "Yes" << endl;

else cout << "No" << endl;

}

3.4

#include<iostream>

using namespace std;

#include<stack>

bool JudgeExpression(const char\* expression, int length)

{

bool isLegal = false;

stack<char> bracketsStack;

//如果是左括号，则进栈；如果是右括号，则与栈顶元素比较，匹配则出栈，否则返回false

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

{

if (expression[i] == '(' || expression[i] == '[' || expression[i] == '{')

{

bracketsStack.push(expression[i]);

}

if (expression[i] == ')')

{

if (!bracketsStack.empty() && bracketsStack.top() == '(')

{

bracketsStack.pop();

}

else

{

isLegal = false;

break;

}

}

else if (expression[i] == ']')

{

if (!bracketsStack.empty() && bracketsStack.top() == '[')

{

bracketsStack.pop();

}

else

{

isLegal = false;

break;

}

}

else if (expression[i] == '}')

{

if (!bracketsStack.empty() && bracketsStack.top() == '{')

{

bracketsStack.pop();

}

else

{

isLegal = false;

break;

}

}

}

//如果栈为空，说明匹配成功了

if (bracketsStack.empty())

{

isLegal = true;

}

return isLegal;

}

//测试代码

//int main()

//{

// string expression1 = "(5-4)+{[2\*(5-3\*4)-8]\*2}";

// bool isLegal = JudgeExpression(expression1.c\_str(), expression1.length());

//

// cout << "(5-4)+{[2\*(5-3\*4)-8]\*2} isLegal = " << isLegal << endl;

//

// string expression2 = "(5-4)+{[2\*(5-3\*4]-8)\*2}";

// isLegal = JudgeExpression(expression2.c\_str(), expression2.length());

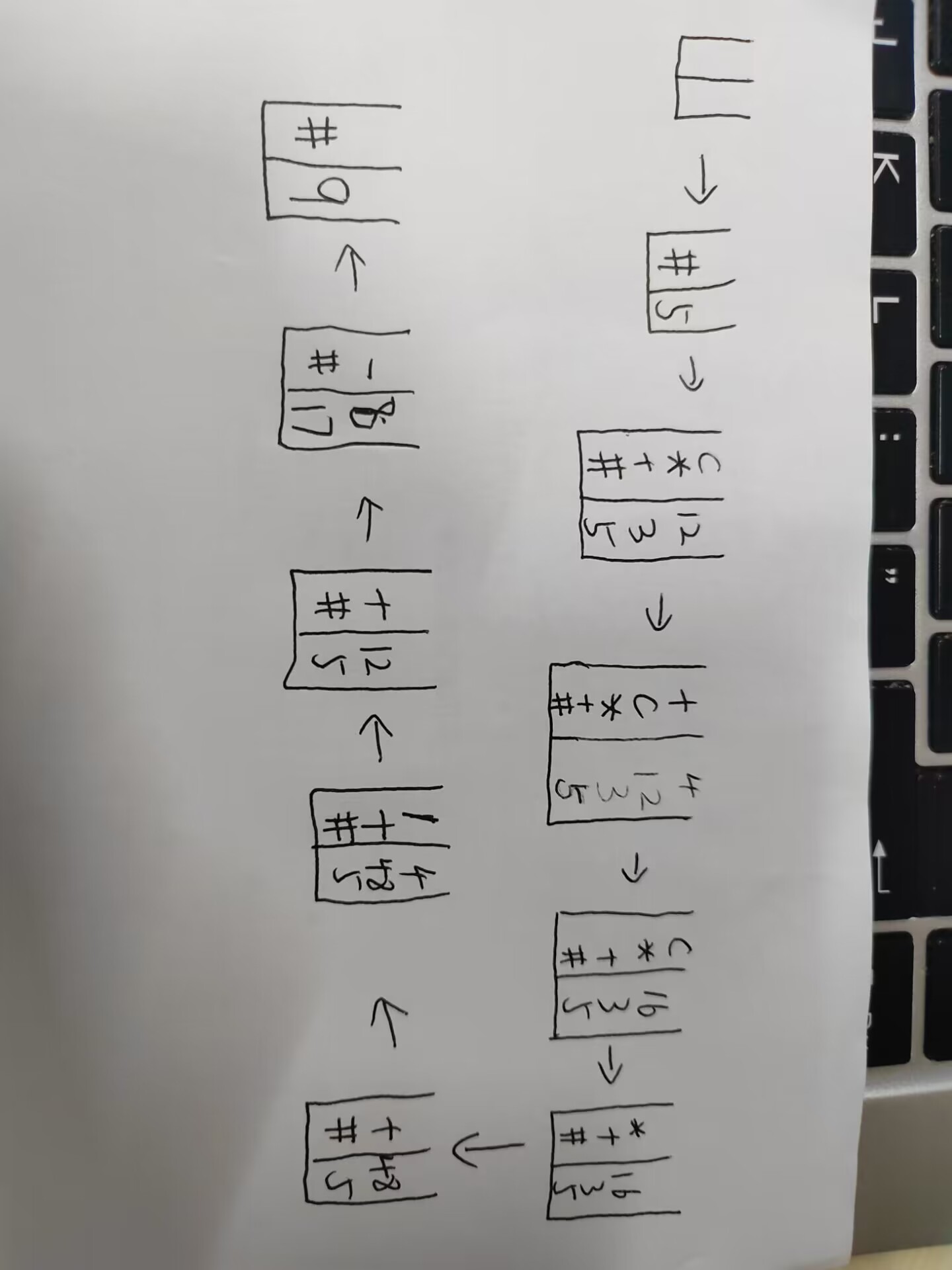
//

// cout << "(5-4)+{[2\*(5-3\*4]-8)\*2} isLegal = " << isLegal << endl;

// return 0;

//}

3.5



3.9

设循环队列为Q,数组最大长度为MaxLen,头指针和尾指针分别为front和rear元素个数=(Q. rear-Q. front+MaxLen) % MaxLen

3.11

# include <iostream>

using namespace std;

typedef int DataType;

typedef struct Node

{

DataType data;

struct Node\* next;

}LinkQueue, \* PLinkQueue;

//由于仅带尾指针的单循环链表尾指针不断变化， 所以只能传递地址，通过地址改变实参的值；

PLinkQueue CreateLinkQueue(void);

void DestroyLinkQueue(PLinkQueue\* pR);

bool EmptyLinkQueue(PLinkQueue\* pR);

bool InLinkQueue(PLinkQueue\* pR, DataType x);

bool OutLinkQueue(PLinkQueue\* pR, DataType\* val);

bool FrontLinkQueue(PLinkQueue\* pR, DataType\* val);

PLinkQueue CreateLinkQueue(void)

{

PLinkQueue R = (PLinkQueue)malloc(sizeof(LinkQueue));

if (NULL != R)

{

R->data = 0;

R->next = R;

return R;

}

else

{

cout << "Memory allocate is errir! " << endl;

system("pause");

exit(0);

}

}

void DestroyLinkQueue(PLinkQueue\* pR)

{

PLinkQueue H = \*pR;

PLinkQueue p = H->next;

PLinkQueue q = NULL;

while (p != H)

{

q = p;

p = p->next;

free(q);

q = NULL;

}

free(H);

\*pR = NULL;

return;

}

bool EmptyLinkQueue(PLinkQueue\* pR)

{

PLinkQueue R = \*pR;

if (R->next == R)

{

return true;

}

else

{

return false;

}

}

bool InLinkQueue(PLinkQueue\* pR, DataType x)

{

PLinkQueue R = \*pR;

PLinkQueue p = (PLinkQueue)malloc(sizeof(LinkQueue));

if (NULL == p)

{

return false;

}

else

{

p->data = x;

p->next = NULL;

p->next = R->next;

R->next = p;

\*pR = p;

return true;

}

}

bool OutLinkQueue(PLinkQueue\* pR, DataType\* val)

{

PLinkQueue R = \*pR;

PLinkQueue H = R->next;

if (EmptyLinkQueue(pR))

{

return false;

}

else

{

PLinkQueue p = H->next;

H->next = p->next;

\*val = p->data;

if (p == R)

{

\*pR = H;

}

free(p);

p = NULL;

return true;

}

}

bool FrontLinkQueue(PLinkQueue\* pR, DataType\* val)

{

PLinkQueue R = \*pR;

PLinkQueue H = R->next;

if (EmptyLinkQueue(pR))

{

return false;

}

else

{

return H->next->data;

}

}

//int main(int argc, char\*\* argv)

//{

// DataType val = 0;

//

// PLinkQueue R = CreateLinkQueue();

// for (int i = 0; i < 10; i++)

// {

// InLinkQueue(&R, i + 1);

// }

//

// for (int i = 0; i < 10; i++)

// {

// OutLinkQueue(&R, &val);

// cout << val << " ";

// }

//

// DestroyLinkQueue(&R);

// if (NULL == R)

// {

// cout << "test" << endl;

// }

//

// system("pause");

// return 0;

//}