学号 WA2214014 专业 人工智能 姓名 杨跃浙
实验日期 **5 月 27 号** 教师签字 成绩

实验报告

【实验名称】 栈和队列

【实验目的】

掌握顺序栈和链栈的进栈和出栈算法，明确栈空和顺序栈栈满的条件。

掌握循环队列和链队列的进队和出队算法，明确队空和循环队列队满的条件。

掌握栈和队列的特点，能够在相应的应用问题中正确选用不同的数据结构，能够借助栈和队列的基本操作来解决某些实际应用问题，如括号的匹配问题。

【实验原理】

初始化一个数据元素为整型的顺序栈和链栈，并实现对应的进栈、出栈、获得栈顶元素等操作通过控制台 scanf 函数将 1、2、3、4、5 进顺序栈，然后出顺序栈并将出栈元素入链栈，直到顺序栈为空获得链栈栈顶元素并输出，打印链栈内的所有元素;
初始化一个数据元素为整型的顺序循环队列和链队列,通过控制台 scanf 函数将 1、2、3、4、5 进顺序循环队列，出顺序循环队列两次，并将出队元素入链队列，在从链队列中出队两次，输出队列中的元素;
通过栈判断一个表达式的门是否匹配

【实验内容】

初始化一个数据元素为整型的顺序栈和链栈，并实现对应的进栈、出栈、获得栈顶元素等操作通过控制台 scanf 函数将 1、2、3、4、5 进顺序栈，然后出顺序栈并将出栈元素入链栈，直到顺序栈为空获得链栈栈顶元素并输出，打印链栈内的所有元素;
初始化一个数据元素为整型的顺序循环队列和链队列,通过控制台 scanf 函数将 1、2、3、4、5 进顺序循环队列，出顺序循环队列两次，并将出队元素入链队列，在从链队列中出队两次，输出队列中的元素;

通过栈判断一个表达式的门是否匹配

```
#include <iostream>
using namespace std;
#define OK 1
#define ERROR 0
#define OVERFLOW -2
#define MAXSIZE 100
#define MAXQSIZE 100
typedef int Status;
typedef int ElemType;
typedef int SElemType;
typedef int QElemType;
typedef struct
{
	SElemType* base;
	SElemType* top;
	int stacksize;
}SqStack;
typedef struct StackNode
{
	ElemType data;
	struct StackNode* next;
}StackNode,*LinkStack;
typedef struct QNode
{
	QElemType data;
	struct QNode* next;
}QNode,*QueuePtr;
typedef struct
{
	QueuePtr front;
	QueuePtr rear;
}LinkQueue;
Status Init_SqStack(SqStack &S)
{
	S.base = new SElemType[MAXSIZE];
	if (!S.base) return(OVERFLOW);
	S.top = S.base;
	S.stacksize = MAXSIZE;
	return OK;
}
Status Push_SqStack(SqStack *S, SElemType e)
{
	if (S->top - S->base == MAXSIZE) return ERROR;
```

```
        *(S->top++) = e;
        return OK;
}
Status Pop_SqStack(SqStack* S,SElemType &e)
{
        if (S->top == S->base) return ERROR;
        e = *--S->top;
        return OK;
}
Status GetTop_SqStack(SqStack* S)
{
        if (S->top == S->base) return ERROR;
        return *(S->top - 1);
}
Status Init_LinkStack(LinkStack& S)
{
        S = NULL;
        return OK;
}
Status Push_LinkStack(LinkStack& S, SElemType e)
{
        StackNode* p = new StackNode;
        p->data = e;
        p->next = S;
        S = p;
        return OK;
}
Status Pop_LinkStack(LinkStack & S, SElemType& e)
{
        if (S==NULL) return ERROR;
        e = S->data;
        StackNode* p = S;
        S = S->next;
        delete p;
        return OK;
}
SElemType GetTop_LinkStack(LinkStack S)
{
        if (S != NULL) return S->data;
}
void solve_SqStack(SqStack& S)
{
        for (int i = 1; i <= 5; i++)
        {
                int n;
```

```
            cin >> n;
            Push_SqStack(&S, n);
        }
}
void solve_Sq_to_Link(SqStack& S, LinkStack& L)
{
        int e;
        while (Pop_SqStack(&S,e)) Push_LinkStack(L, e);
}
void solve_LinkStack(LinkStack& L)
{
        int e;
        while (Pop_LinkStack(L, e)) cout << e << "\t";
        cout << endl;
}
typedef struct
{
        QElemType* base;
        int front;
        int rear;
}SqQueue;
Status Init_SqQueue(SqQueue& Q)
{
        Q.base = new QElemType[MAXQSIZE];
        if (!Q.base) return(OVERFLOW);
        Q.front = Q.rear = 0;
        return OK;
}
Status En_SqQueue(SqQueue& Q, QElemType e)
{
        if ((Q.rear + 1) % MAXQSIZE == Q.front) return ERROR;
        Q.base[Q.rear] = e;
        Q.rear = (Q.rear + 1) % MAXQSIZE;
        return OK;
}
Status De_SqQueue(SqQueue& Q, QElemType& e)
{
        if (Q.front == Q.rear) return ERROR;
        e = Q.base[Q.front];
        Q.front = (Q.front + 1) % MAXQSIZE;
        return OK;
}
Status Init_LinkQueue(LinkQueue &Q)
{
        Q.front = Q.rear = new QNode;
```

```
    Q.front->next = NULL;
    return OK;
}
Status En_LinkQueue(LinkQueue& Q, QElemType e)
{
    QNode* p = new QNode;
    p->data = e;
    p->next = NULL;
    Q.rear->next = p;
    Q.rear = p;
    return OK;
}
Status De_LinkQueue(LinkQueue& Q, QElemType& e)
{
    if (Q.front == Q.rear) return ERROR;
    QNode* p = Q.front->next;
    e = p->data;
    Q.front->next = p->next;
    if (Q.front->next==NULL) Q.rear = Q.front;
    delete p;
    return OK;
}
void solve_SqQueue(SqQueue &Q)
{
    int n;
    for (int i = 1; i <= 5; i++)
    {
        cin >> n;
        En_SqQueue(Q, n);
    }
}
void solve_Sq_to_Link_Q(SqQueue& Q, LinkQueue& L)
{
    int e;
    for (int i = 1; i <= 2; i++)
    {
        De_SqQueue(Q, e);
        En_LinkQueue(L, e);
    }
}
void solve_LinkQueue(LinkQueue& L)
{
    int e;
    for (int i = 1; i <= 2; i++)
    {
```

```
            De_LinkQueue(L, e);
            cout << e << "\t";
        }
        cout << endl;
}
bool Matching()
{
        LinkStack S;
        Init_LinkStack(S);
        int flag = 1;
        int flag_c;
        char ch;
        cin >> ch;
        while (ch != '#' && flag)
        {
            if (ch == '[')
            {
                flag_c = 1;
                Push_LinkStack(S, flag_c);
            }
            if (ch == ']')
            {
                if (Pop_LinkStack(S, flag_c)) void;    else flag = 0;
            }
            cin >> ch;
        }
        if ((S == NULL) && flag) return true; else return false;
}
int main()
{
        SqStack Sq;
        LinkStack Link;
        SqQueue Q;
        LinkQueue LinkQ;
        Init_SqStack(Sq);
        Init_LinkStack(Link);
        solve_SqStack(Sq);
        solve_Sq_to_Link(Sq, Link);
        cout<<GetTop_LinkStack(Link)<<endl;
        solve_LinkStack(Link);
        Init_SqQueue(Q);
        Init_LinkQueue(LinkQ);
        solve_SqQueue(Q);
        solve_Sq_to_Link_Q(Q, LinkQ);
        solve_LinkQueue(LinkQ);
```

```
    if (Matching()) cout << "Yes"; else cout << "No";
    return 0;
}
```

【小结或讨论】

通过该次实验我掌握了顺序栈和链栈的初始化、进栈、出栈和获得栈顶元素等基本操作以及循环队列和链队列的初始化、入队、出队获得队首元素等基本操作，并能够应用栈解决实际问题，如括号的匹配等。