**第三章 栈和队列实验**

**（说明：3.1和3.2选做1题，3.3和3.4选做1题）**

**3.1栈的应用—括号匹配**

**一、实验目的**  
  1．掌握堆栈特殊线性表的存储方式的基本操作方法。

2．掌握堆栈后进先出运算原则在解决实际问题中的应用。

3．掌握使用栈的原理来解决表达式中的括号配对问题。

**二、实验内容**

 假设一个算术表达式中包含圆括弧、方括弧三种类型的括弧，编写一个程序用于判别表达式中括弧是否正确配对。

说明：检验表达式中的括号匹配情况：假设在一个算术表达式中，可以包含三种括号：圆括号"（"和"）"，方括号"["和"]"，并且这两种括号可以按任意的次序嵌套使用。比如，...[......[...]...]...(...)..。现在需要设计一个算法，用来检验在输入的算术表达式中所使用括号的合法性。

提示：

算术表达式中各种括号的使用规则为：出现左括号，必有相应的右括号与之匹配，并且每对括号之间可以嵌套，但不能出现交叉情况。我们可以利用一个栈结构保存每个出现的左括号，当遇到右括号时，从栈中弹出左括号，检验匹配情况。在检验过程中，若遇到以下几种情况之一，就可以得出括号不匹配的结论。

（1）当遇到某一个右括号时，栈已空，说明到目前为止，右括号多于左括号；

（2）从栈中弹出的左括号与当前检验的右括号类型不同，说明出现了括号交叉情况；

（3）算术表达式输入完毕，但栈中还有没有匹配的左括号，说明左括号多于右括号。

算法步骤：扫描表达式，

1）凡出现左括弧，则进栈；

2）凡出现右括弧，首先检查栈是否空。

若栈空，则表明该“右括弧”多余

否则和栈顶元素比较，

若相匹配，则“左括弧出栈”

否则表明不匹配

3）表达式检验结束时，

若栈空，则表明表达式中匹配正确

否则表明“左括弧”有余

栈的类型定义及基本操作代码可参考教材相关定义和程序。

三、实验源代码

四、实验结果

**3.2栈的应用--数制转换**

一、实验目的

1．掌握堆栈的存储方式和基本操作。

2．掌握堆栈后进先出运算原则在解决实际问题中的应用。

3．掌握使用栈的原理来解决数制转换问题。

二、实验内容

利用栈结构，编写程序将十进制数转换成二进制数或八进制数。

提示：十进制数值转换成二进制使用辗转相除法将一个十进制数值转换成二进制数值。即用该十进制数值除以2，并保留其余数；重复此操作，直到该十进制数值为0为止。最后将所有的余数反向输出就是所对应的二进制数值。十进制数值转换成八进制算法类似。转换算法要求用一个函数完成。

顺序栈的基本操作的代码可参考如下：

typedef struct

{

DataType stack[MaxStackSize];

int top;

} SeqStack;

void StackInitiate(SeqStack \*S) /\*初始化顺序堆栈S\*/

{ S->top = 0; /\*定义初始栈顶下标值\*/

}

int StackNotEmpty(SeqStack S)

/\*判顺序堆栈S非空否，非空则返回1，否则返回0\*/

{ if(S.top <= 0) return 0;

else return 1;

}

int StackPush(SeqStack \*S, DataType x)

/\*把数据元素值x压入顺序堆栈S，入栈成功则返回1，否则返回0 \*/

{ if(S->top >= MaxStackSize)

{ printf("堆栈已满无法插入! \n");

return 0;

}

else

{ S->stack[S->top] = x;

S->top ++;

return 1;

}

}

int StackPop(SeqStack \*S, DataType \*d)

/\*弹出顺序堆栈S的栈顶数据元素值到参数d ，出栈成功则返回1，否则返回0\*/

{ if(S->top <= 0)

{

printf("堆栈已空无数据元素出栈! \n");

return 0;

}

else

{

S->top --;

\*d = S->stack[S->top];

return 1;

}

}

int StackTop(SeqStack S, DataType \*d)

/\*取顺序堆栈S的当前栈顶数据元素值到参数d ，成功则返回1，否则返回0\*/

{

if(S.top <= 0)

{ printf("堆栈已空! \n");

return 0;

}

else

{ \*d = S.stack[S.top - 1];

return 1;

}

}

三、实验源代码

四、实验结果

**3.3 队列的基本操作**

**一、实验目的**

1．掌握队列的顺序存储结构。

2．掌握队列先进先出运算原则在解决实际问题中的应用。

3. 掌握自定义数据类型的用法。

**二、实验内容**

仿照资料中顺序循环队列的例子，设计一个只使用队头指针和计数器的顺序循环队列抽象数据类型。其中操作包括：初始化、入队列、出队列、判断队列是否非空。编写主函数，验证所设计的顺序循环队列的正确性。

以下是队列操作函数的定义：

(1)QueueInitiate(Q) 初始化队列Q

(2)QueueNotEmpty(Q) 队列Q非空否

(3)QueueAppend(Q,x) 入队列，在队列Q的队尾插入数据元素x。

(4)QueueDelete(Q,d) 出队列，把队列Q的队头元素删除并由参数d带回。

 提示：队尾的位置可由队头指针与计数器进行求解，请思考它们之间的关系，同时还要考虑如何实现循环队列（可借助求模运算）。

三、实验源代码

四、实验结果（测试数据）

**3.4栈与队列的应用—回文**

**一、实验目的**  
1、掌握链式堆栈的类型定义方法。  
2、掌握链式堆栈上实现的几种基本操作。

3、掌握链式队列的类型定义方法。

4、掌握链式队列上实现的几种基本操作。

**二、实验内容**  
1、编程序判断一个字符序列是否是回文，要求采用链式队列和链式堆栈。

提示：设字符数组str中存放了要判断的字符串。把字符数组中的字符逐个分别存入队列和堆栈，然后逐个出队列和退栈并比较出队列的字符和退栈的字符是否相等，若全部相等则该字符序列是回文，否则就不是回文。

以下是程序部分代码，请调试并补充使之正确运行。

LinStack.h

typedef struct snode

{

DataType data;

struct snode \*next;

} LSNode;

void StackInitiate(LSNode \*\*head)

/\*初始化带头结点链式堆栈\*/

{

if((\*head = (LSNode \*)malloc(sizeof(LSNode))) == NULL)

{

Printf(“没有空间申请\n”);

Return;

}

(\*head)->next = NULL;

}

int StackNotEmpty(LSNode \*head)

/\*判堆栈是否非空，非空返回1；空返回0\*/

{

if(head->next == NULL) return 0;

else return 1;

}

int StackPush(LSNode \*head, DataType x)

/\*把数据元素x插入链式堆栈head的栈顶作为新的栈顶 \*/

{

LSNode \*p;

if((p = (LSNode \*)malloc(sizeof(LSNode))) == NULL)

{

printf("内存空间不足无法插入! \n");

return 0;

}

p->data = x;

p->next = head->next; /\*新结点链入栈顶\*/

head->next = p; /\*新结点成为新的栈顶\*/

return 1;

}

int StackPop(LSNode \*head, DataType \*d)

/\*出栈并把栈顶元素由参数d带回\*/

{

LSNode \*p = head->next;

if(p == NULL)

{

printf("堆栈已空出错！");

return 0;

}

head->next = p->next; /\*删除原栈顶结点\*/

\*d = p->data; /\*原栈顶结点元素赋予d\*/

free(p); /\*释放原栈顶结点内存空间\*/

return 1;

}

int StackTop(LSNode \*head, DataType \*d)

/\*取栈顶元素并把栈顶元素由参数d带回\*/

{

LSNode \*p = head->next;

if(p == NULL)

{

printf("堆栈已空出错！");

return 0;

}

\*d = p->data;

return 1;

}

void Destroy(LSNode \*head)

{

LSNode \*p, \*p1;

p = head;

while(p != NULL)

{

p1 = p;

p = p->next;

free(p1);

}

}

LinQueue.h

typedef struct qnode

{

DataType data;

struct qnode \*next;

} LQNode;

typedef struct

{

LQNode \*front; /\*队头指针\*/

LQNode \*rear; /\*队尾指针\*/

} LQueue;

void QueueInitiate(LQueue \*Q) /\*初始化链式队列Q\*/

{

Q->rear = NULL; /\*定义初始队尾指针下标值\*/

Q->front = NULL; /\*定义初始队头指针下标值\*/

}

int QueueNotEmpty(LQueue Q)

/\*判链式队列Q非空否，非空则返回1，否则返回0\*/

{

if(Q.front == NULL) return 0;

else return 1;

}

int QueueAppend(LQueue \*Q, DataType x)

/\*把数据元素值x插入链式队列Q的队尾，入队列成功则返回1，否则返回0 \*/

{

LQNode \*p;

if((p = (LQNode \*)malloc(sizeof(LQNode))) == NULL)

{

printf("内存空间不足！");

return 0;

}

p->data = x;

p->next = NULL;

if(Q->rear != NULL) Q->rear->next = p;

Q->rear = p;

if(Q->front == NULL) Q->front = p;

return 1;

}

int QueueDelete(LQueue \*Q, DataType \*d)

/\*删除链式队列Q的队头数据元素值到d ，出队列成功则返回1，否则返回0\*/

{

LQNode \*p;

if(Q->front == NULL)

{

printf("队列已空无数据元素出队列! \n");

return 0;

}

else

{

\*d = Q->front->data;

p = Q->front;

Q->front = Q->front->next;

if(Q->front == NULL) Q->rear = NULL;

free(p);

return 1;

}

}

int QueueGet(LQueue Q, DataType \*d)

/\*取链式队列Q的当前队头数据元素值到d ，成功则返回1，否则返回0\*/

{

if(Q.front == NULL)

{

printf("队列已空无数据元素出队列! \n");

return 0;

}

else

{

\*d = Q.front->data;

return 1;

}

}

void Destroy(LQueue Q)

{

LQNode \*p, \*p1;

p = Q.front;

while(p != NULL)

{

p1 = p;

p = p->next;

free(p1);

}

}

HuiWen(char str[])

{

***自己编写代码***

}

主程序

Void main(void)

{

Char str1[]=”abcdedcba”;

Char str2[]=”ABCDEABCDE”

HuiWen(str1);

HuiWen(str2);

}

三、实验源代码

四、实验结果