数据结构实验报告

实验成绩：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 |  | 学号 |  | 专业班级 |  |
| 指导教师 | 张先宜 | 实验时间 | 4月5号 | 实验地点 | 4号机房 |

**实验名称： \_\_**栈实验\_\_\_

1. **实验目标**
2. 熟练掌握栈的顺序存储结构和链式存储结构。
3. 熟练掌握栈的有关算法设计，并在顺序栈和链栈上实现。
4. 根据具体给定的需求，合理设计并实现相关结构和算法。
5. **实验要求3.2.1 顺序栈的实验要求**
6. 顺序栈结构和运算定义，算法的实现以库文件方式实现，比如seqStack.h，不得在测试主程序中直接实现；
7. 实验程序有较好可读性，各运算和变量的命名直观易懂，符合软件工程要求；
8. 程序有适当的注释。

**3.2.2 链栈实验要求**

1. 本次实验中的链栈结构指带头结点的单链表；
2. 链栈结构和运算定义，算法的实现以库文件方式实现，比如linkedStack.h，不得在测试主程序中直接实现；
3. 实验程序有较好可读性，各运算和变量的命名直观易懂，符合软件工程要求；
4. 程序有适当的注释。
5. **实验内容 顺序栈实验任务**

设计并实现一个顺序栈，编写算法实现下列问题的求解。

1. 利用顺序栈实现将10进制数转换为x进制数，2<=x<=36，除了阿拉伯数字字符，不够字符使用大写英文字符。要求键盘输入10进制数和转换的目标进制数。比如：37转换为20进制数为1H。

第一组数据：4

第二组数据：311

第三组数据：7254

第四组数据：98357

1. 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。比如：“{[](){}}”是匹配的，“{[(})]”就是不匹配的。
2. 假设栈的输入序列为1、2、3、...、n，设计算法求出所有可能的出栈序列。比如输入1、2、3、4、5，可能出栈的序列为12345、13452等42个。

3.3.2 链栈实验任务

以带头结点的单链表表示链栈，编写算法实现下列问题的求解。

1. 利用链栈实现将10进制数转换为x进制数，2<=x<=36，除了阿拉伯数字字符，不够字符使用大写英文字符。要求键盘输入10进制数和转换的目标进制数。比如：37转换为20进制数为1H。

第一组数据：4

第二组数据：311

第三组数据：7254

第四组数据：98357

1. 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。比如：“{[](){}}”是匹配的，“{[(})]”就是不匹配的。
2. 假设栈的输入序列为1、2、3、...、n，设计算法求出所有可能的出栈序列。比如输入1、2、3、4、5，可能出栈的序列为12345、13452等42个。
3. **数据结构设计**

（所有存储结构的封装描述）

顺序栈的储存结构：

typedef struct sStack

{

elementType data[MAXLEN];

int top;

}seqStack;

链栈的储存结构：

Typedef struct lsNode

{

elementType data;

struct lsNode \*next;

}sNode,\*linkedStack;

1. **算法设计**

（除书上给出的基本运算（这部分不必给出设计思想），其它实验内容要给出算法设计思想）

顺序栈与链栈算法思想相似，就不再一一赘述。

(1)“除x取余”方法。设十进制数为N，循环除x，取出 余数，商的整数部分重新赋给N，直至N==0； 再将最后的余数作为x进制数的最高位，第一个余数 作为x进制的最低位，得到x进制数。即：余数逆 （反）序输出。利用栈，每次余数入栈，结束时余数全部出栈即可 得到x进制数。

void convert(int x, int m){

int stack[MAXLEN];

int top = -1;

int y = x;

while(x > 0){

top = push(stack,top,x%m);

x = x/m;

}

cout<<"十进制数"<<y<< "转化为"<<m<<"进制数为：";

while(top >=0){

if(peek(stack,top) < 10){

printf("%d",peek(stack,top));

}else{

printf("%c",peek(stack,top)-10+'A');

}

top = pop(stack,top);

}

putchar('\n');

}

（2）考虑到表达式的匹配有左右括号配对次序不正确、右括号多于左括号、左括号多于右括号、左右括号匹配正确四种情况，具体操作如下：

①我们顺序读入算数表达式，让左括号表达式进栈；

②当读到右括号时，比较当前栈顶元素是否匹配，若匹配，退栈继续判断；若当前栈顶元素与当前扫描的括号不匹配，则左右括号配对次序不正确；

③若当前为右括号而栈已经空，则右括号多于左括号；读入结束时，若堆栈非空（即堆栈尚有某种类型的左括号），则说明左括号多于右括号；否则，括号配对正确。

bool brackeMatch2(string str)

{

seqStack S; //定义一个栈

S.top=-1;

int len=str.length();

bool tag=true;

bool result=false;

int i=0;

char x;

while(i<len&&tag==true)

{

switch(str[i]){

//所有括号入栈

case'(':

case'[':

case'{':

if(S.top<=1000)

{

S.top++;

S.data[S.top]=str[i];

}

else{

cout<<"栈满！"<<endl;

}

break;

case')':

//扫描到右括号时，如果当前栈空，右括号多于左括号

if(S.top==-1){

tag=false;

result=false;

break;

}

//得到栈顶元素，并出栈

x=S.data[S.top];

S.top--;

if(x='('){

break;

}

else

{

tag=false;

result=false;

break;

}

case']':

if(S.top==-1)

{

tag=false;

result=false;

break;

}

//得到栈顶元素，并出栈

x=S.data[S.top];

S.top--;

if(x=='[')

{

break;

}

else

{

tag=false;

result=false;

break;

}

case'}':

if(S.top==-1)

{

tag=false;

result=false;

break;

}

//得到栈顶元素，并出栈

x=S.data[S.top];

S.top--;

if(x=='{')

{

break;

}

else

{

tag=false;

result=false;

break;

}

default:

break;

}

i++;

}

if(S.top==-1&&tag==true)

{

result=true;

cout<<"各括号匹配"<<endl;

}

else

{

result=false;

cout<<"各括号不匹配"<<endl;

}

return result;

}

(6) 每次操作有可能有 2 中操作，要么出栈，要么入栈，且 2 种操作是或的关系。但， 出栈、入栈递归返回后要恢复递归前状态。

1.栈不空，入栈序列中还有数据，出栈递归，入栈递归。

2. 栈不空，入栈序列数据已经处理结束，此时只能选择出栈操作

3. 栈空，入栈序列未处理结束。此时，只能选择入栈操作

每次进出栈的过程，栈中元素和出栈元素初始都在参数中。所以当遇到既能出栈又能进栈的过程时，需要将放在后面函数还原回初始过程的参数。

void print\_all\_Stack(char a[], int l, int r) {

int i, j, k;

bool tag;

if (l == r) {

tag = true;

for (i = 0; i < r - 1; i++) {

for (j = i + 1; j < r; j++) {

for (k = j + 1; k < r + 1; k++) {

if (a[j] < a[k] && a[k] < a[i]) {

tag = false;

}

}

}

}

if (tag) {

cout << a << endl;

}

}

for (int x = l; x <= r; x++)

{

char temp = a[x];

a[x] = a[l];

a[l] = temp;

print\_all\_Stack(a, l + 1, r);

a[l] = a[x];

a[x] = temp;

}

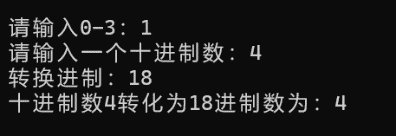
}

1. **运行和测试**

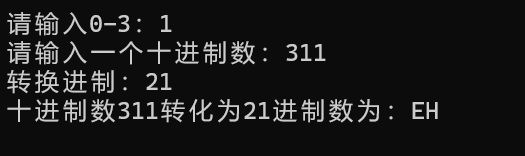
**顺序栈：**

**1）**

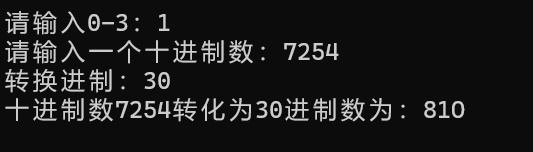
第一组数据：4



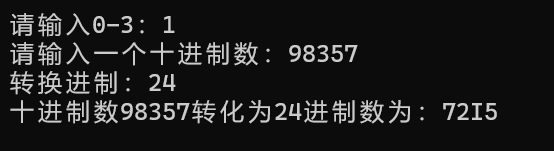
第二组数据：311



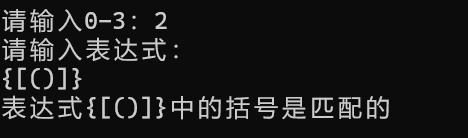
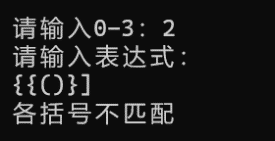
第三组数据：7254



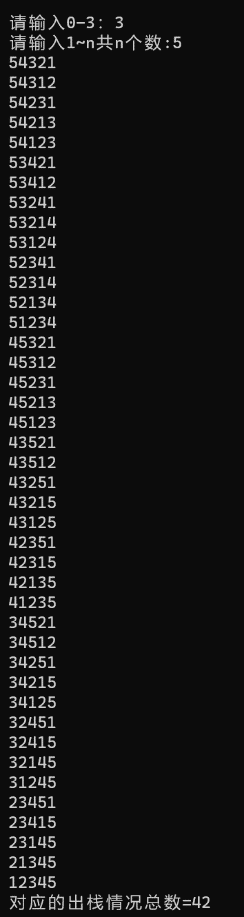
第四组数据：98357



**2）**

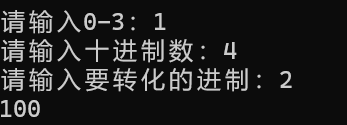
**3)**



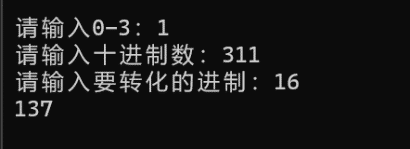
**链栈：**

**1）**

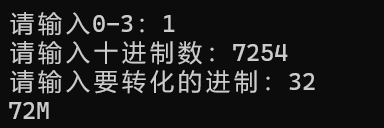
第一组数据：4



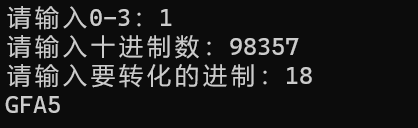
第二组数据：311



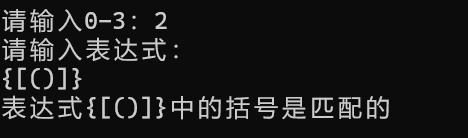
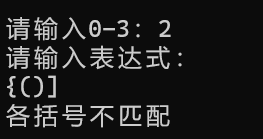
第三组数据：7254



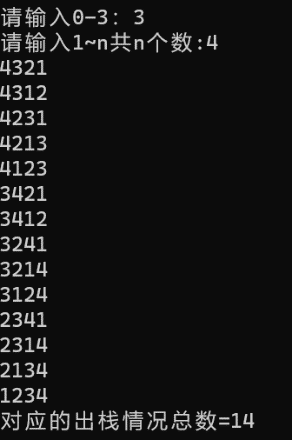
第四组数据：98357



**2）**

**3)**



1. **总结、心得和建议**

这部分主要涉及栈的知识，因为要学习的太多，导致这部分还没有学的太好，写的时候遇见很多bug，不知道该怎么办。例如进制转换的字符的转换，可以用辗转相除法进行计算，后来通过查阅资料了解了如何去写。

对于输入的括号是否匹配，刚开始遇见很多困难，花了不少时间，还有最后一题，也由于递归和链栈学不是很好，导致花了好长时间。通过这次实验，我深刻认识到了我存在的问题，以后会更加注意，不断地提升自己。

**[8. 附录]**

（源代码清单。纸质报告不做要求。电子版报告，可直接附源文件，删除编译生成的所有文件）

