实验3栈实验

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 张渊 | 学号 | 2022210000 | 专业班级 | 计科22-0 |
| 指导教师 | 张先宜 | 实验时间 | 2023.4.5 | 实验地点 |  |

**实验名称：实验三 栈实验**

**3.1 实验目的**

1. 熟练掌握栈的顺序存储结构和链式存储结构。
2. 熟练掌握栈的有关算法设计，并在顺序栈和链栈上实现。
3. 根据具体给定的需求，合理设计并实现相关结构和算法。

**3.2 实验要求**

3.2.1 顺序栈的实验要求

1. 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；
2. 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；
3. 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。

**3.3 实验任务**

3.3.1 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。

3.4\* 栈的扩展实验

非必做内容，有兴趣的同学选做。自行选择栈的存储结构。

1. 假设栈的输入序列为1、2、3、...、n，设计算法实现对给定的一个序列，判定其是否是此栈合法的输出序列。比如输入1、2、3、4、5序列，13452为合法出栈序列；15234是不合法输出序列。
2. 利用栈求解算术表达式的值。

**3.4 数据结构设计**

typedef struct seqStack//顺序栈

{

char data[MAXSIZE];//因为要用到字母所以用char来存储数据

int top;

}seqStack;

typedef struct LinkNode//链栈

{

char data;//因为要用到字母所以用char来存储数据

LinkNode\* next;

}LinkNode, \* LinkStack;

**3.5算法设计**

1【算法思想】让数字num不断除以需要转化的进制并且不断取余，将余数存入创建的栈中。利用栈的后进先出原则，这便不用逆序输出。同时将超过10的数据用大写字母存储。所以创建了char类型的存储空间。最终输入创建的栈的每个元素即可。

【算法描述】

seqStack ChangeSystem1(int num, int x)//num 为10进制的数字，x是目标的进制数

{

seqStack S;

initStack1(S);

while (num != 0)

{

int temp = num % x;

if (temp >= 10)

S.data[++S.top] = temp - 10 + 'A';

else

S.data[++S.top] = temp + '0';

num /= x;

}

return S;

}

2【算法思想】传入一个字符串str，将每一个括号存入。对左右括号进行不同的判断。左括号存入，有括号就要判断栈是否为空、是否有对应元素。

注意：要在遍历过程中建立临时的booL类型变量，确保在str遍历完并且栈也为空结束，才能保证结果正确。

【算法描述】

//判断一个数学表达式是否合法

bool bracketMatch1(string str)

{

seqStack S;//定义一个栈

initStack1(S);

bool result, temp = true;

int i = 0;

char x;

while (i < str.length() && temp)

{

switch (str[i])

{

//所有左括号入栈

case '(':

case '[':

case '{':

if (S.top <= 1000)

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

else

{

cout << "栈满！" << endl;

temp = false;

}

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)

{

temp = false;

break;

}

x = S.data[S.top--];//得到栈顶元素，并出栈

if (x == '(')

break;

else

{

temp = false;

break;

}

case ']':

if (S.top == -1)

{

temp = false;

break;

}

x = S.data[S.top--];//得到栈顶元素，并出栈

if (x == '[')

break;

else

{

temp = false;

break;

}

case '}':

if (S.top == -1)

{

temp = false;

break;

}

x = S.data[S.top--];//得到栈顶元素，并出栈

if (x == '{')

break;

else

{

temp = false;

break;

}

default:

break;

}

i++;

}

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

result = true;

else

result = false;

return result;

}

3【算法思想】因为一个数只有入栈和出栈两种行为，所以分别对其进行讨论。

出栈、入栈递归返回后要恢复递归前状态。通过递归以及不断地入栈出栈，最终输出符合要求的结果。

【算法描述】

void back\_track1(vector<int> ans, stack<int> stk, stack<int> input, vector<vector<int>>& result, int n)//判断栈的输出是否合适

{

if (ans.size() == n)

{

result.push\_back(ans);

return;

}

stack<int> stk\_tmp = stk;

vector<int> ans\_tmp = ans;

//输入直接出栈

if (!stk.empty())

{

ans.push\_back(stk.top());

stk.pop();

back\_track1(ans, stk, input, result, n);

}

//输入直接入栈

if (!input.empty())

{

stk\_tmp.push(input.top());

input.pop();

back\_track1(ans\_tmp, stk\_tmp, input, result, n);

}

}

4【算法思想】让数字num不断除以需要转化的进制并且不断取余，将余数存入创建的栈中。利用栈的后进先出原则，这便不用逆序输出。同时将超过10的数据用大写字母存储。所以创建了char类型的存储空间。最终输入创建的栈的每个元素即可。

【算法描述】

LinkNode\* ChangeSystem2(int num, int x)//num 为10进制的数字，x是目标的进制数

{

LinkNode\* top;

initStack2(top);

while (num != 0)

{

int temp = num % x;

if (temp >= 10)

{

LinkNode\* s;

s = (LinkNode\*)malloc(sizeof(LinkNode));

s->data = temp - 10 + 'A';

s->next = top;

top = s;

}

else

{

LinkNode\* s;

s = (LinkNode\*)malloc(sizeof(LinkNode));

s->data = temp + '0';

s->next = top;

top = s;

}

num /= x;

}

return top;

}

5【算法思想】传入一个字符串str，将每一个括号存入。对左右括号进行不同的判断。左括号存入，有括号就要判断栈是否为空、是否有对应元素。

注意：要在遍历过程中建立临时的booL类型变量，确保在str遍历完并且栈也为空结束，才能保证结果正确。

【算法描述】

//判断一个数学表达式是否合法

bool bracketMatch2(string str)

{

LinkNode\* u, \* top;//定义一个栈

top = NULL;

bool temp = true;

bool result;

int i = 0;

char x;

while (i < str.length() && temp == true)

{

switch (str[i])

{

//所有左括号入栈

case '(':

case '[':

case '{':

LinkNode \* s;

s = (LinkNode\*)malloc(sizeof(LinkNode));

s->data = str[i];

s->next = top;

top = s;

break;

case ')':

if (top == NULL)//扫描到右括号时，如果当前栈空，右括号多于左括号

{

temp = false;

break;

}

x = top->data;//得到栈顶元素，并出栈

u = top;

top = top->next;

free(u);

if (x == '(')

break;

else

{

temp = false;

break;

}

case ']':

if (top == NULL)

{

temp = false;

break;

}

x = top->data;//得到栈顶元素，并出栈

u = top;

top = top->next;

free(u);

if (x == '[')

break;

else

{

temp = false;

break;

}

case '}':

if (top == NULL)

{

temp = false;

break;

}

x = top->data;//得到栈顶元素，并出栈

u = top;

top = top->next;

delete u;

if (x == '{')

break;

else

{

temp = false;

break;

}

default:

break;

}

i++;

}

if (top == NULL && temp == true)

result = true;

else

result = false;

return result;

}

6【算法思想】因为一个数只有入栈和出栈两种行为，所以分别对其进行讨论。

出栈、入栈递归返回后要恢复递归前状态。通过递归以及不断地入栈出栈，最终输出符合要求的结果。

【算法描述】

void back\_track2(vector<int> ans, stack<int> stk, stack<int> input, vector<vector<int>>& result, int n)//判断栈的输出是否合适

{

if (ans.size() == n)

{

result.push\_back(ans);

return;

}

stack<int> stk\_tmp = stk;

vector<int> ans\_tmp = ans;

//输入直接出栈

if (!stk.empty())

{

ans.push\_back(stk.top());

stk.pop();

back\_track1(ans, stk, input, result, n);

}

//输入直接入栈

if (!input.empty())

{

stk\_tmp.push(input.top());

input.pop();

back\_track1(ans\_tmp, stk\_tmp, input, result, n);

}

}

**3.6运行和测试**

1.1（测试中的是转化为20进制）

第一组数据：4



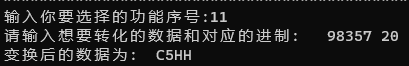
第二组数据：311



第三组数据：7254

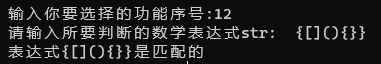


第四组数据：98357



1.2

{[](){}}

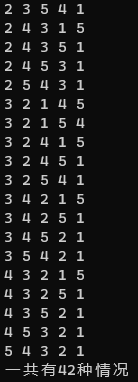
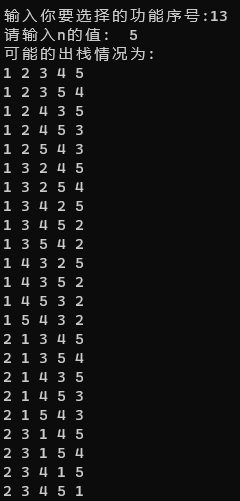


{[(})]



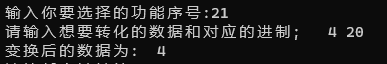
1.3

1、2、3、4、5 (n)

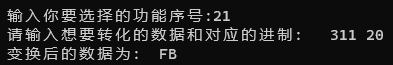


2.1（测试中的是转化为20进制）

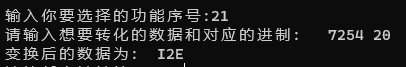
第一组数据：4



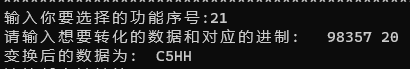
第二组数据：311



第三组数据：7254



第四组数据：98357

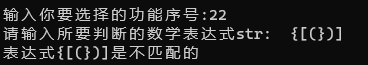


2.2

{[](){}}

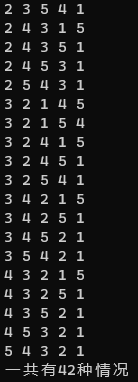
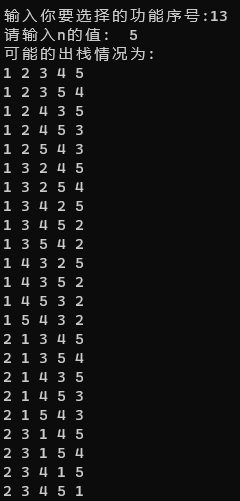


{[(})]



2.3

1、2、3、4、5 (n)



**3.7总结、心得和建议**

本次栈实验收获颇多。顺序栈和链栈各3个题，而且题目的内容相同。在代码实现的过程中对顺序栈和链栈有了更深刻的理解。进制转化主要是理解转化的过程，并灵活运用不同类型数据存储即可。而判断括号是否正确，则需要充分理解栈的后进先出原则。此处对于顺序栈和链栈的实现差别较大，链栈的实现更复杂一些。可以类比链表的插入过程。同时为了便于显示，我将顺序栈和链栈分别进行实现。第三题我觉得难度很大，要理解递归的过程，但此题我仍然是不明白，以往等以后能力更强了回来再看。

总体来说本次实验花费时间较长，特别是在对第三题的查找和和理解过程花费了很多时间。虽然最终还是没有取得好的结果但是对自己的能力有了更清楚的认识，也加深了对栈知识的理解。希望在以后的学习中不断反思，提高能力。

**[3.8 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）

