# 实验三：栈

#### 实验题目

**顺序栈：**

题目一**：利用顺序栈实现将10进制数转换为16进制数。**

* + 1. **第一组数据：4**
    2. **第二组数据：11**
    3. **第三组数据：254**
    4. **第四组数据：1357**

题目二：**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。**

**链栈：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。**

**题目三：实现将10进制数转换为16进制数。**

1. **第一组数据：4**
2. **第二组数据：11**
3. **第三组数据：254**
4. **第四组数据：1357**

**题目四：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**

**\* 栈扩展实验**

* + **非必做内容，有兴趣的同学选做，**

**<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。**

**<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。**

**<3>利用栈求解算术表达式的值。**

#### （二） 实验源码

## 顺序栈

#pragma once

#include <iostream>

using namespace *std*;

const int max = 10;

class Stack

{

public:

Stack();

~Stack();

bool isFull();//判满

bool isEmpty();//判空

void push(char x);//入栈

char Gettop();//获取栈顶

bool pop();//出栈

void changenum(int x);//进制转换

bool isMatch(*string* str);//判断括号是否匹配

private:

char data[max];

int top;

};

#include "Stack.h"

#include<string>

Stack::Stack()

{

top = -1;

}

Stack::~Stack()

{

}

bool Stack::isEmpty()

{

if (top == -1)

{

return true;

}

else

return false;

}

bool Stack::isFull()

{

if (top + 1 == max)

{

return true;

}

else

return false;

}

void Stack::push(char x)

{

if (isFull())

{

*cout* << "栈满" << *endl*;

}

else

{

top++;

*data*[top] = x;

}

}

char Stack::Gettop()

{

char x;

if (isEmpty())

{

*cout* << "栈空";

return 0;

}

else

{

x = *data*[top];

return x;

}

}

bool Stack::pop()

{

if (isEmpty())

{

return false;

}

else

{

top--;

return true;

}

}

void Stack::changenum(int x)

{

Stack S;

int mod;

char s;

char ch[16] = { '0', '1', '2', '3', '4', '5', '6', '7', '8','9', 'A', 'B', 'C', 'D', 'E', 'F' };//16进制字符

while (x != 0)

{

mod = x % 16;//商余法

S.push( ch[mod]);//按照对应余数的字符入栈

x = x / 16;

}

*cout* <<x<< "的十六进制数为";

while (!S.isEmpty())

{

s=S.Gettop();//不断取栈顶并弹栈

S.pop();

*cout* << s;

}

*cout* << *endl*;

}

bool Stack::isMatch(*string* str)

{

Stack S;

for (unsigned int i = 0; i<str.*length*(); i++) //字符串遍历

{

switch (str[i])//选择带括号的字符串入栈

{

case '(':

S.push('(');

break;

case '[':

S.push('[');

break;

case '{':

S.push('{');

break;

//根据括号的匹配性，判断与右括号相邻的左括号是否匹配，

case ')':

if (S.Gettop() == '(') {

S.pop(); //匹配弹栈

}

else {

return false;//否则判断为不匹配

}

break;

case ']':

if (S.Gettop() == '[') {

S.pop();

}

else {

return false;

}

break;

case '}':

if (S.Gettop()== '{') {

S.pop();

}

else {

return false;

}

break;

}

}

if (!S.isEmpty()) //执行完遍历后如果栈不为空说明存在单个括号不匹配情况

{

return false;

}

else {

return true;

}

}

## 链栈

#pragma once

#include <iostream>

using namespace *std*;

struct MyStruct

{

char data;

MyStruct\* next;

};

class LinkStack

{

public:

LinkStack();

~LinkStack();

void setlinkStack();//链栈初始化

void pushStack( int x);//入栈

bool popStack();//出栈

char GetTOP();//获取栈顶

bool isEmpty();//判空

void printStack();//打印栈

void changeNum(int x);//进制转换

bool isMatch( *string* str);//判断括号匹配

private:

MyStruct\* top;

};

#include "LinkStack.h"

#include <string.h>

LinkStack::LinkStack()

{

}

LinkStack::~LinkStack()

{

MyStruct\* tmp = top;

while (tmp->next)

{

MyStruct\* u = tmp;

tmp = tmp->next;

delete u;

}

}

void LinkStack::setlinkStack()

{

top = new MyStruct;//top代替头节点指针

top->*next* = *NULL*;//栈顶置空

}

void LinkStack::pushStack(int x)

{

MyStruct\* S = new MyStruct;

if (*NULL*==S)

{

*cout* << "false";

}

else

{

S->*data* = x;//入栈元素

S->*next* = top->*next*;//头插法

top->*next* = S;

}

}

char LinkStack::GetTOP()

{

char x;

if (isEmpty())

{

*cout* << "栈空";

return 0;

}

else

{

x = top->*next*->*data*;

return x;

}

}

bool LinkStack::popStack()

{

int x = 0;

MyStruct\* u;

if (isEmpty())

{

*cout* << "栈空" << *endl*;

return false;

}

else

{

x = top->*data*;

u = top;

top=top->*next* ;

delete u;

return true;

}

}

bool LinkStack::isEmpty()

{

if (top ->*next*== *NULL*)

{

return true;

}

else

{

return false;

}

}

void LinkStack::printStack()

{

MyStruct\* s =top->*next*;

while (s!= *NULL*)

{

*cout* << s->data;

s = s->next;

}

*cout* << *endl*;

}

void LinkStack::changeNum(int n)

{

LinkStack S;

S.setlinkStack();

char ch[16] = { '0', '1', '2', '3', '4', '5', '6', '7', '8','9', 'A', 'B', 'C', 'D', 'E', 'F' };

int mod;

while (n != 0)

{

mod = n%16;

S.pushStack(ch[mod]);

n = n/ 16;

}

*cout* <<n<< "的十六进制数为";

S.printStack();

}

bool LinkStack::isMatch(*string* str)

{

LinkStack S;

S.setlinkStack();

for (unsigned int i = 0; i < str.*length*(); i++) {

switch (str[i])

{

case '(':

S.pushStack('(');

break;

case '[':

S.pushStack('[');

break;

case '{':

S.pushStack('{');

break;

case ')':

if (S.GetTOP() == '(') {

S.popStack();

}

else {

return false;

}

break;

case ']':

if (S.GetTOP() == '[') {

S.popStack();

}

else {

return false;

}

break;

case '}':

if (S.GetTOP() == '{') {

S.popStack();

}

else {

return false;

}

break;

}

}

if (!S.isEmpty()) //遍历完后如果栈不为空说明单个括号不匹配

{

return false;

}

else {

return true;

}

}

## Main

#include <iostream>

#include "Stack.h"

#include "LinkStack.h"

int main()

{

Stack A,B;

*string* a = "{[(1+2)\*2]+3}";

A.changenum(4);

A.changenum(11);

A.changenum(254);

A.changenum(1357);

if (B.isMatch(a))

{

*cout* <<"字符串中的括号"<<a<< "不匹配" << *endl*;

}

else

{

*cout* << "字符串中的括号" << a<<"匹配" << *endl*;

}

*cout*<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<*endl*;

LinkStack C, D;

C.setlinkStack();

D.setlinkStack();

C.changeNum(4);

D.changeNum(11);

C.changeNum(254);

C.changeNum(1357);

*string* d = "{[(1+2]\*3]+3}";

if (D.isMatch(d))

{

*cout* << "字符串中的括号" << d<<"不匹配" << *endl*;

}

else

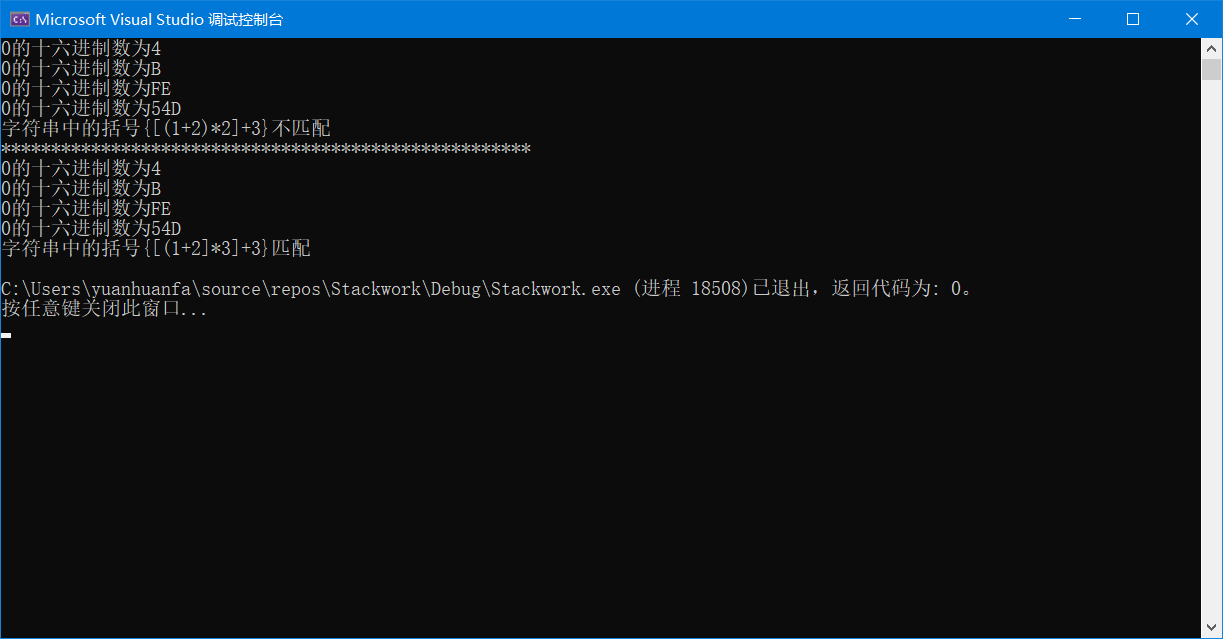
{

*cout* << "字符串中的括号" <<d<< "匹配" << *endl*;

}

}

#### （三） 实验截图



#### （四） 心得体会

这次的顺序栈实验由于上学期接触过一些栈的知识所以可以很快理解，进制转换涉及到了数学上的商余法，也是比较容易理解的，判断括号是否匹配这个问题，一开始采用输入char型数组的方法，后来改用直接输入字符串，避免需要定义长度的问题，对于判断我使用了很多的if else来进行入栈出栈的操作，后来感觉代码很不清晰，自己看着都晕，于是改为选择语句。并直接利用写好的顺序栈简化流程。链栈实验时，我把链栈当做单链表来写，并采用头插法保证先进先出后进后出，在进制计算时由于链表可以一次性输出，又编写了打印函数简化操作。判断括号匹配性时，操作基本和顺序栈的方法相同。