

《数据结构》

实验报告

姓名： 庄李晨

学号： 10522238

成绩：

电子与计算机工程学院

School of Electronic & Computer Engineering

2022年10月

【实验名称】栈的实现与应用

【实验目的】

一、掌握栈的基本操作，实现栈的入栈、出栈、返回栈顶元素等基本运算。

【实验内容】

1. 实验要求：

1. 仔细阅读顺序栈实验代码，补充下面7个函数的实现过程。补充完成main（）函数的定义，调试顺序栈程序，给出测试用例及测试结果。

（1）初始化栈

Status InitStack( SqStack &S ){}

（2）将元素e入栈

Status Push( SqStack &S, SElemType e){}

（3）出栈，用e返回出栈元素

Status Pop( SqStack &S, SElemType &e){}

（4）返回栈顶元素

Status GetTop( SqStack S, SElemType &e){}

（5）清空栈，变为空栈

void Clear( SqStack &S ){}

（6）遍历栈元素 （从栈底到栈顶）

void Traverse(SqStack S ){}

（7）进制转换函数,将十进制数dec转化为n进制数,并输出转换后结果

void Convert(int dec, int n){ }

2. 仔细阅读链栈实验代码，补充下面6个函数的实现过程。补充完成main（）函数的定义，调试链栈程序，给出测试用例及测试结果。

（1）将元素e入栈

Status Push( LinkStack &S, SElemType e){}

（2）出栈，用e返回出栈元素

Status Pop( LinkStack &S, SElemType &e){}

（3）返回栈顶元素

Status GetTop( LinkStack S, SElemType &e){}

（4）清空栈，变为空栈

void Clear( LinkStack &S ){}

（5）遍历栈元素 （从栈顶到栈底）

void Traverse(LinkStack S ){}

（6）回文是指正读反读均相同的字符序列，如"acdca"、"dceecd"均是回文，但"book"不是回文。试写一个算法，利用栈结构，判定输入的字符串是否为回文，并将结果输出，例如："acdca"是回文

void HuiWen() {}

3. 选作题目（三选一）

题目一：定义表达式求值，利用栈实现表达式求值。

设定两栈 ：OPND-----操作数或运算结果　OPTR------运算符

a) 初始化OPTR栈和OPND栈，将 “=”压入OPTR

b) 依次读入字符ch，循环执行（c）至（e）

c) 取出OPTR的栈顶元素，当OPTR的栈顶元素和ch均为“=”时，表达式求值完毕，OPND栈顶元素为表达式的值

d) if (ch是操作数) 则ch进OPND，读入下一字符ch

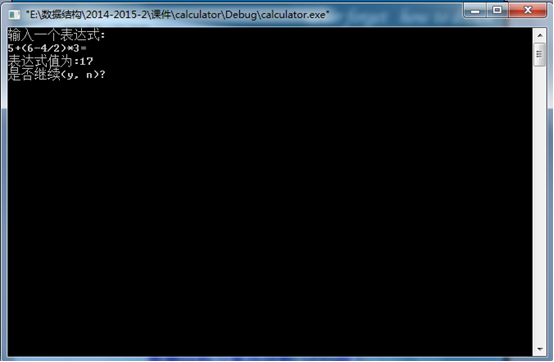
e) else 比较OPTR栈顶元素和ch的优先级

case ‘<’: 运算符ch 进OPTR，读入下一字符ch

case ‘>’: 栈顶运算符退栈、计算，结果进OPND

case‘=’: 脱括号（弹出左括号），读入下一字符ch

实验运行预期效果如下图所示：



题目二：检验含两种括弧的表达式中括弧匹配的正确性。

如表达式（［］（））或［（［ ］［ ］）］为正确的匹配，

［（］）或（［（）] 或 [ ( ) ] )均为不正确的匹配。

算法的设计思想：

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。

题目三：迷宫求解

理解并运行迷宫求解代码，给出代码注释，并画出核心函数的流程图。

findpath(int mg[M+2][N+2],int xb,int yb,int xe, int ye)

1. 程序清单
2. 顺序栈代码

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

#define MAXSIZE 100

using namespace std;

typedef int SElemType;

typedef int Status;

//顺序栈结构体定义

typedef struct {

SElemType\* base;

SElemType\* top;

int stacksize;

}SqStack;

//初始化栈

Status InitStack(SqStack& S) {

S.base = new SElemType[MAXSIZE];

if (!S.base) exit(0);

S.top = S.base;

S.stacksize = MAXSIZE;

}

//将元素 e 入栈

Status Push(SqStack& S, SElemType e) {

if (S.top - S.base == S.stacksize) return 0; //栈满

\*S.top = e;

S.top++;

return 1;

}

//出栈，用 e 返回出栈元素

Status Pop(SqStack& S, SElemType& e) {

if (S.top == S.base) {

cout << "栈为空" << endl;

return 0; //栈空

}

S.top--;

e = \*S.top;

return 1;

}

//返回栈顶元素

Status GetTop(SqStack S, SElemType& e) {

if (S.top != S.base) { //非栈空时返回

cout<<"栈顶元素为" << \*(S.top - 1)<<endl;

return \*(S.top - 1); //++会改变指针自身值，而-1不会改变

}

}

//清空栈，变为空栈

void Clear(SqStack\* S) {

free(S->base);

S->base = S->top = NULL;

S->stacksize = 0;

}

// 判断栈是否为空，只需要判断栈顶指针与栈底指针是否相同即可

Status EmptyStack(SqStack\* S) {

return S->base == S->top;

}

//遍历栈元素 （从栈底到栈顶）

void Traverse(SqStack S) {

printf("栈底：");

SElemType\* p = S.base;

while (p != S.top)

{

printf("%d ", \*p);

p++;

}

printf("\n");

}

//进制转换函数,将十进制数 dec 转化为 n 进制数,并输出转换后结果

void Convert(int dec, int n) {

SElemType e;

SqStack S;

InitStack(S); // 构造空栈

cout << "转换后的结果为" << endl;

while (dec) {

Push(S, dec % n);

dec = dec / n;

}

while (!EmptyStack(&S)) {

Pop(S, e);

printf("%d", e);

}

printf("\n");

}

int main(void)

{

SqStack st;

InitStack(st);

int c = 0;

while (c != 7)

{

cout << endl << "1. 入栈";

cout << endl << "2. 出栈";

cout << endl << "3. 读栈顶";

cout << endl << "4. 遍历栈";

cout << endl << "5. 清空顺序栈";

cout << endl << "6. 进制转换";

cout << endl << "7. 退出";

cout << endl << "选择功能(1~6):";

cin >> c;

int e;

switch (c)

{

case 1: {

cin >> e;

Push(st,e);

break;

}

case 2: {

Pop(st,e);

cout << "栈元素" << e << "出栈" << endl;

break;

}

case 3: {

GetTop(st,e);

break;

}

case 4: {

Traverse(st);

break;

}

case 5: {

Clear(&st);

break;

}

case 6: {

int a, b;

cout << "请输入数字a的b进制转换" << endl;

cout << "输入a的值：" << endl;

cin >> a;

cout << "输入b的值" << endl;

cin >> b;

Convert(a, b);

break;

}

case 7: {

system("pause");

}

}

}

}

1. 链栈代码

#include<iostream>

#include <stdlib.h>

using namespace std;

typedef int SElemType;

typedef int Status;

//链栈结构体定义

typedef struct SNode

{

SElemType date;

SNode\* next=NULL;

}SNode, \* LinkStack;

//初始化

LinkStack InitStack(LinkStack S) {

S = (LinkStack)malloc(sizeof(struct SNode));

if (!S) {

cout << "分配空间失败！程序退出！" << endl;

exit(-1);

}

else

cout << "初始化成功！成功构造一个栈！" << endl;

S->next = NULL;

return S;

}

//将元素 e 入栈

Status Push(LinkStack& S, SElemType e) {

LinkStack t;

t = (LinkStack)malloc(sizeof(struct SNode));

if (!t) {

cout << "分配空间失败！" << endl;

return 0;

}

t->date = e;

t->next = S;

S= t;

cout << "成功入栈！" << endl;

cout << "您已插入一个元素" << e << endl;

}

//出栈，用 e 返回出栈元素

Status Pop(LinkStack& S, SElemType& e) {

LinkStack t;

if (S == NULL) {

cout << "栈空，无法弹出！！！" << endl;

return 0;

}

else {

e = S->date;

t = S;

S = S->next;

delete t;

cout << "元素" << e << "已出栈" << endl;

return e;

}

}

//返回栈顶元素

Status GetTop(LinkStack S, SElemType& e) {

if (S== NULL) {

cout << " 栈空!" << endl;

return 0;

}

else {

e = S->date;

cout << "栈顶元素为" << e << endl;

return e;

}

}

//判断是否为空栈

int Stackempty(LinkStack S) {

if (S == NULL) {

cout << "该栈为空" << endl;

return 0;

}

else {

cout << "该栈非空" << endl;

return 1;

}

}

//清空栈，变为空栈

void Clear(LinkStack& S) {

LinkStack t;

while (S) {

t = S;

S = S->next;

free(t);

}

cout << "栈已经清空" << endl;

}

//遍历栈元素 （从栈顶到栈底）

void Traverse(LinkStack S) {

int i; int s[10005];

memset(s, 0, sizeof(s));

if (S == NULL) {

cout << " 栈空！ " << endl;

return;

}

else {

for (i = 1; S != NULL; i++) {

//cout << "该栈第 " << i << "个元素是：" << S->date << endl;

s[i] = S->date;

S = S->next;

}

for (int j = i-1; j > 0; j--) {

cout << "该栈第 " << i-j << "个元素是：" <<s[j]<< endl;

}

cout << "栈内有： " << i - 1 << " 个元素，已经全部输出完毕！" << endl;

}

}

//回文是指正读反读均相同的字符序列，如"acdca"、"dceecd"均是回文，但"book"不是回文。

//试写一个算法判定输入的字符串是否为回文，并将结果输出，例如："acdca"是回文

void HuiWen()

{

cout << "输入需要判断的字符串" <<endl;

string s;

bool deng=false;

cin >> s;

int l = s.length();

for (int i = 0; i < l / 2; i++) {

if (s[i] != s[l - i - 1]) {

deng = true;

cout << "字符串" << s << "不是回文" << endl;

break;

}

continue;

}

if(!deng)cout<< "字符串" << s << "是回文" << endl;

}

int main(void)

{

LinkStack st = NULL;

int c = 0;

while (c != 7)

{

cout << endl << "1. 入栈";

cout << endl << "2. 出栈";

cout << endl << "3. 读栈顶";

cout << endl << "4. 遍历栈";

cout << endl << "5. 清空顺序栈";

cout << endl << "6. 回文判断";

cout << endl << "7. 退出";

cout << endl << "选择功能(1~6):";

cin >> c;

switch (c)

{

case 1: {

cout << "请输入一个数插入栈";

SElemType e;

cin >> e;

Push(st, e);

break;

}

case 2: {

cout << "出栈一个元素" << endl;

SElemType e;

Pop(st, e);

break;

}

case 3: {

SElemType e;

GetTop(st, e);

break;

}

case 4: {

Traverse(st);

break;

}

case 5: {

Clear(st);

break;

}

case 6: {

HuiWen();

break;

}

case 7: {

system("pause");

break;

}

}

}

}

1. 选做题代码

括号匹配

#include <iostream>

#include <stack>

#include <string>

#define MAXSIZE 100

#define ERROR 0

#define OK 1

#define EMPTY '\0' // 定义一个特殊值表示栈为空

using namespace std;

typedef int SElemType;

typedef int Status;

typedef struct {

char\* base;

char\* top;

int stacksize;

} SqStack;

Status InitStack(SqStack& stack) {

stack.base = new char[MAXSIZE];

if (!stack.base) exit(ERROR);

stack.top = stack.base;

stack.stacksize = MAXSIZE;

return OK;

}

Status Push(SqStack& stack, char element) { //入栈函数

if (stack.top - stack.base == stack.stacksize) {

return ERROR;

}

\*stack.top++ = element;

return OK;

}

Status Pop(SqStack& stack, char& element) { //出栈函数

if (stack.top == stack.base) {

return ERROR;

}

element = \*--stack.top;

return OK;

}

char GetTop(SqStack& stack) { //取栈顶元素

if (stack.top != stack.base) {

return \*(stack.top - 1);

}

else {

return EMPTY; // 返回特殊值表示栈为空

}

}

int main() {

SqStack stack;

InitStack(stack);

cout << "输入要检验的式子：";

string input;

getline(cin, input); // 读取整行输入，包括空格和换行符

bool valid = false; // 标记输入是否有效

while (!valid) { // 循环直到用户输入有效的括号序列为止

for (char c : input) { // 逐个检查输入的字符

if (c == '(' || c == '[') { // 如果遇到左括号，将其推入堆栈

Push(stack, c);

}

else if (c == ')' || c == ']') { // 如果遇到右括号，检查堆栈顶元素是否匹配并弹出堆栈顶元素（如果匹配）

char top = GetTop(stack); // 获取堆栈顶元素但不弹出堆栈顶元素（保持原状）

if ((c == ')' && top == '(') || (c == ']' && top == '[')) { // 检查右括号和左括号是否匹配（右括号和左括号的数量是否相等）并弹出堆栈顶元素（如果匹配）

Pop(stack, top); // 弹出堆栈顶元素并更新堆栈顶指针

valid = true; // 如果括号匹配成功，则将valid标记设置为true，退出循环

}

}

}

if (!valid) { // 如果括号匹配不成功，则需要提示用户重新输入

cout << "输入的括号序列不合法，请重新运行程序！" << endl;

exit(0); // 退出程序

}

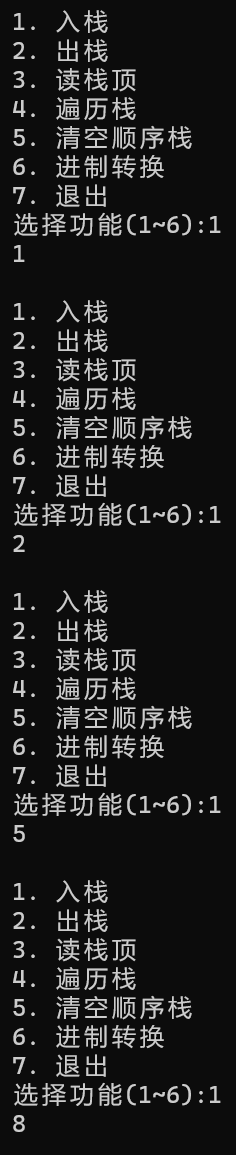
}

cout << "输入的括号序列合法！" << endl;

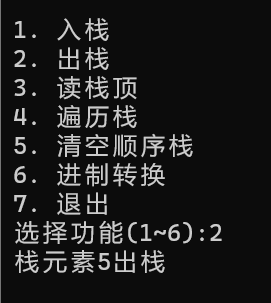
return 0;

}

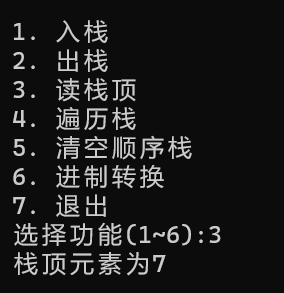
1. 结果截图
2. 顺序栈运行截图



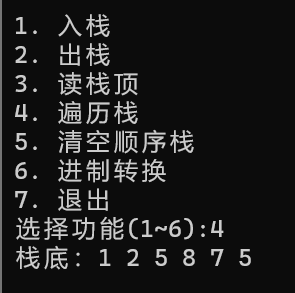
2、出栈



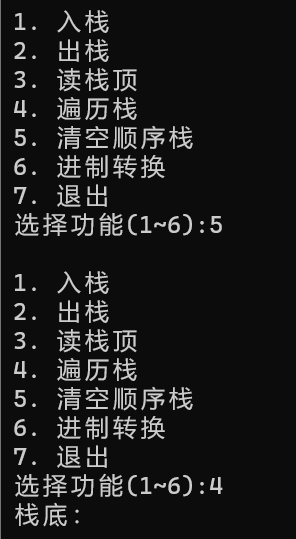
3、读栈顶



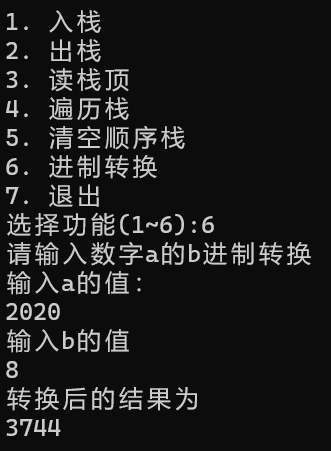
4、遍历栈



5、清空顺序栈

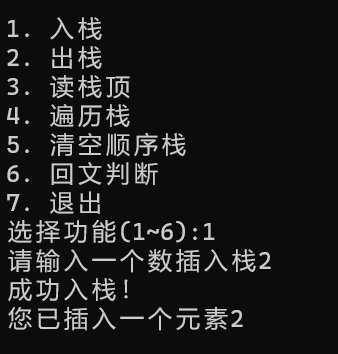


6、进制转换



2.链栈运行截图

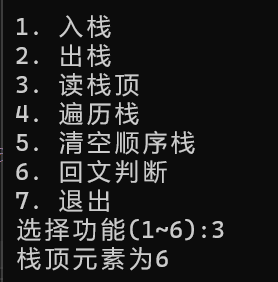
1、入栈



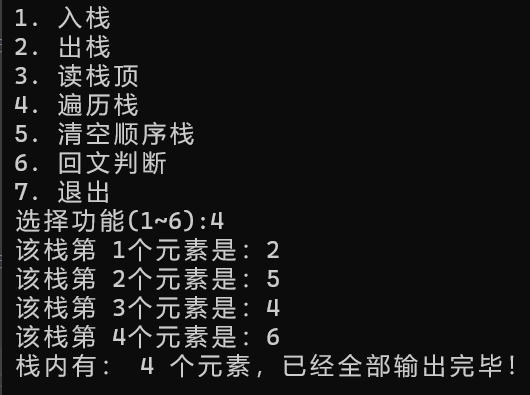
2、出栈



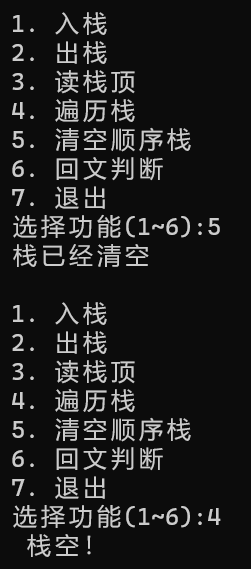
3、读栈顶



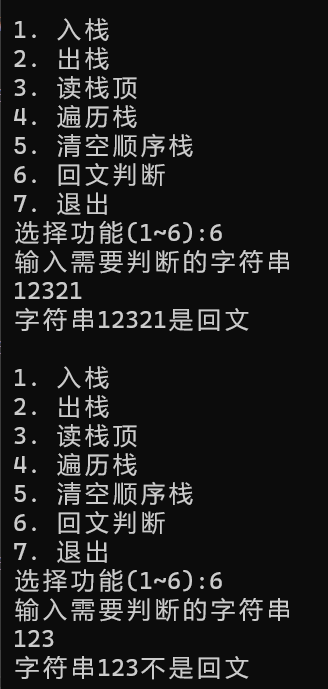
4、遍历栈



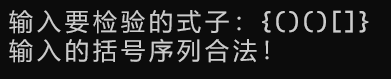
5、清空顺序栈

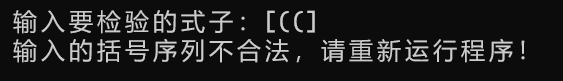


1. 回文数判断



3.选做题运行截图





【实验体会】

在本次实验中，我学习了顺序栈、链栈、3选1的括号匹配，了解栈的基本运行方式，在学习栈的过程中，我深刻理解了栈的基本操作，如压栈（push）、弹栈（pop）、查看栈顶元素（peek）以及检查栈是否为空，以及回文数，进制转换等操作。这些操作都是围绕着栈的特性——后进先出来进行的。