

**数据结构实验报告（三）**

实验课题：栈的基本操作

学 院：理学院

专 业：数学系

组 员：190110304王君涛

190110312战祥贤

190110318王 鹏

190110302王 琪

190110331陈 立

组 长：王君涛

2020年10月17日

顺序栈的实现及应用

顺序栈的基本操作：

* 栈的初始化
* 入栈
* 出栈并返回栈顶
* 返回栈顶元素
* 打印栈内全部元素

具体实例：

* 用顺序栈实现进制转换

顺序栈的基本操作具体实现

准备部分：

用Stack实现顺序表并在Stack类内定义 栈底指针(int\* base;)栈顶指针(int\* top;)最大容量(int stacksize;)。类内定义实现函数 初始化（void initStack();）入栈（void push(int e);）出栈并返回栈顶元素（int pop();）返回栈顶元素（int getTop();）打印栈内元素（void show();）

class Stack

{private:

int\* base;//栈底指针

int\* top;//栈顶指针

int stacksize;//最大容量

public:

void initStack();//初始化

void push(int e);//入栈

int pop();//出栈并返回栈顶元素

int getTop();//返回栈顶元素

void show();//打印栈内元素但会导致全部出栈};

1. 栈的初始化

* 说明：

首先让栈底指针指向栈底（base =new int[MAXSIZE];）让栈头指针指向栈底（top = base;）给定栈的大小（stacksize = MAXSIZE; ） 初始化结束。

* 实现函数：

void Stack::initStack() //初始化

{

base = new int[MAXSIZE];

top = base;

stacksize = MAXSIZE;

}

1. 入栈

* 说明：

首先通过栈顶指针与栈底指针的差与栈的大小比较，判断栈是否已满（if (top - base == stacksize) {cout << "栈已满，操作失败。。"；）如果栈未满则让栈顶指针指向要压入栈内的元素（\*top = e;） 最后栈顶指针上移完成入栈（top++;）

* 实现函数：

void Stack::push(int e)入栈

{if (top - base == stacksize)

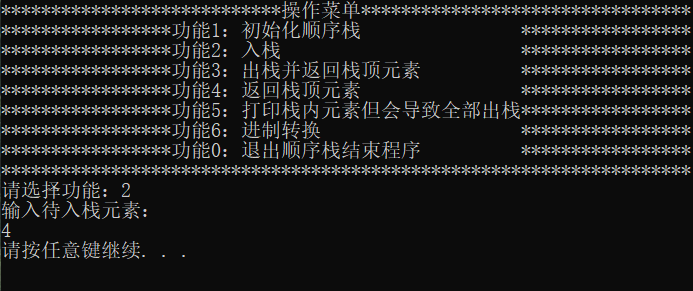
{cout << "栈已满，操作失败。。" << endl;

return;}

\*top = e;

top++;

}

运行结果：

1. 出栈并返回栈顶元素

* 说明：

首先通过比较栈顶指针与栈底指针是否相同判断栈内是否有元素（if (top == base){cout << "栈为空，出栈失败。。" ）如果栈内有元素返回栈顶元素并让栈顶指针减一完成出栈出栈（return \*--top;）

* 实现函数：

int Stack::pop()出栈

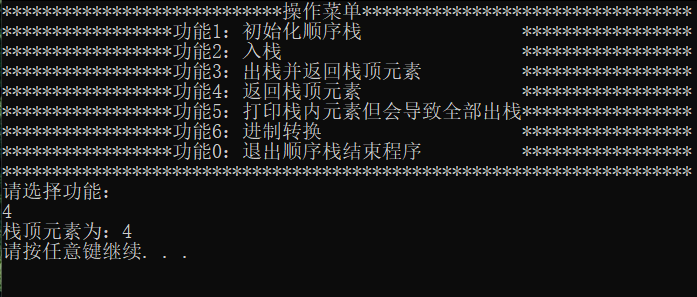
{if (top == base)

{cout << "栈为空，出栈失败。。" << endl;

return -10000;//避免和0混淆

}return \*--top

}

运行结果：

1. 返回栈顶元素

* 说明：

首先通过栈顶指针与栈底指针不相同判断栈非空（if (top != base)）如果非空返回栈顶元素(return \*(top - 1);) （即指针指向前一个地址存的元素）

* 实现函数：

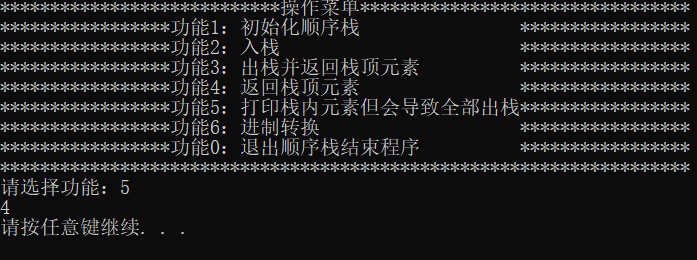
int Stack::getTop()返回栈顶元素

{if (top != base)

return \*(top - 1);

else return 9999;

}

运行结果：

1. 打印栈内全部元素

* 说明

首先判断栈内是否有元素（while (top - base > 0)）当栈内有元素时（ cout << pop() << ' ';）栈顶出栈返回栈顶元素与栈的工作原理相符

* 实现函数

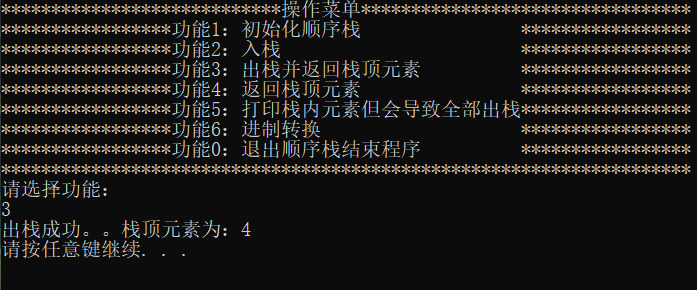
void Stack::show()显示所有元素

{ while (top - base > 0)

cout << pop() << ' ';

cout << endl;

}

运行结果：

实例

（用顺序栈实现进制转换）

* 说明：

初始化一个新的栈以实现该部分功能单独使用（S.initStack();）从键盘输入要进行转换进制的数（cin>>n;）选择进制转换的进制(cin>>b;)通过递归函数进行具体操作：对输入的数n对b求余数t依次插入栈中让n=n/b继续进行递归调用直到n=0完成转换并将转换得到的数存在了栈内通过（S.show();）进行输出。

* 实现函数：

//进制转换函数

void transBase()

{int n,b;

Stack S;

S.initStack();

cout << "请输入待转换的10进制数：";

cin >> n;

cout << "请输入转换的进制：";

cin >> b;

digui(n, b, S);

S.show();

}

//递归函数，用来转换进制

void digui(int n,int b,Stack &S)

{if (n == 0);

else

{int t;

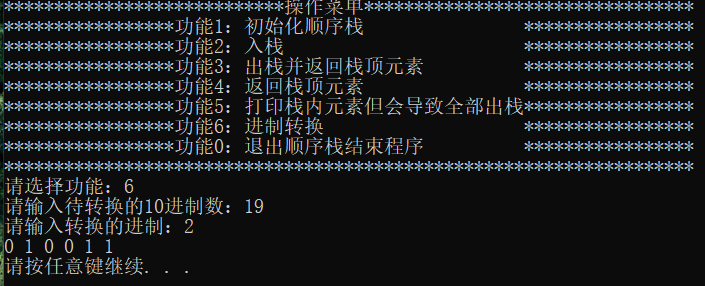
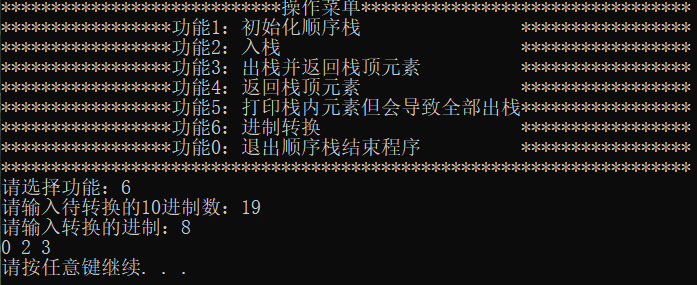
t = n % b;

S.push(t);/

digui(n / b, b, S);//递归调用

}

}

运行结果：

总结：

顺序栈通过栈顶指针与栈底指针可实现栈长度判断以及栈空栈满判断，通过栈顶指针的移动进行入栈出栈操作。要注意栈的操作要符合先进先出的原则。

与链表栈比较 顺序表栈的优点是物理地址相邻便于直观理解顺序表可以通过计算地址来定位需要的元素操作方便，链表只能从头到尾来找需要的元素.。缺点是顺序表存储位置是连续的,一般顺序表的长度有限。而链表是通过指针来定位下一个元素的位置,所以没有长度的限制,随时可以增加。

全部程序代码：

#include<iostream>

using namespace std;

#define MAXSIZE 100

//用类stack实现顺序表

class Stack

{

private:

int\* base;//栈底指针

int\* top;//栈顶指针

int stacksize;//最大容量

public:

void initStack();//初始化

void push(int e);//入栈

int pop();//出栈并返回栈顶元素

int getTop();//返回栈顶元素

void show();//打印栈内元素但会导致全部出栈

};

void Stack::push(int e)入栈

{

if (top - base == stacksize)判断栈是否已满

{

cout << "栈已满，操作失败。。" << endl;

return;

}

\*top = e;

让栈顶指针指向要压入栈内的元素

top++;栈顶指针上移完成入栈

}

int Stack::pop()出栈

{

if (top == base)判断栈内是否有元素

{

cout << "栈为空，出栈失败。。" << endl;

return -10000;//避免和0混淆

}

return \*--top;

栈顶指针减一完成出栈

}

int Stack::getTop()返回栈顶元素

{

if (top != base)如果栈内有元素

return \*(top - 1);返回栈顶（即指针指向前 一个地址存的元素）

else return 9999;

}

void Stack::show()显示所有元素

{

while (top - base > 0)当栈内有元素时 cout << pop() << ' ';pop函数栈顶出栈返回栈顶元素与栈的工作原理相符

cout << endl;

}

//递归函数，用来转换进制

void digui(int n,int b,Stack &S)

{

if (n == 0);

else

{

int t;

t = n % b;

S.push(t);//对n求余的值t依次插入栈中

digui(n / b, b, S);//递归调用

}

}

//进制转换函数

void transBase()

{

int n,b;

Stack S;

S.initStack();

cout << "请输入待转换的10进制数：";

cin >> n;

cout << "请输入转换的进制：";

cin >> b;

digui(n, b, S);

S.show();

}

//显示菜单

void showMenu()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*操作菜单\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能1：初始化顺序栈 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能2：入栈 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能3：出栈并返回栈顶元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能4：返回栈顶元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能5：打印栈内元素但会导致全部出栈\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能6：进制转换 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*功能0：退出顺序栈结束程序 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

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

}

int main()

{

Stack S;

char f;

int Choice = 0;

cout << "是否初始化顺序栈？(y/n)" << endl;//初始化顺序栈

cin >> f;

if (f == 'n')

return 0;//若不初始化顺序栈程序结束。。

S.initStack();

cout << "初始化栈成功。。" << endl;

while (true)

{

system("cls");//清屏

showMenu();//显示菜单

cout << "请选择功能：";

while (true)

{

cin >> Choice;

if (Choice >= 0 && Choice <= 6)

break;

else

cout << "输入错误，请重新输入" << endl;

}

switch (Choice)

{

case 1://功能1：初始化顺序栈

S.initStack();

cout << "初始化顺序栈成功。。" << endl;

system("pause");

break;

case 2://功能2：入栈

int e2;

cout << "输入待入栈元素：" << endl;

cin >> e2;

S.push(e2);

system("pause");

break;

case 3://功能3：出栈并返回栈顶元素

int e3;

e3 = S.pop();

if (e3!=-10000)

cout << "出栈成功。。栈顶元素为：" << e3 << endl;

system("pause");

break;

case 4://功能4：返回栈顶元素

cout << "栈顶元素为：" << S.getTop() << endl;

system("pause");

break;

case 5://功能5：//打印栈内元素但会导致全部出栈

S.show();

system("pause");

break;

case 6://功能6：//进制转换

transBase();

system("pause");

break;

case 0://功能6：退出系统

cout << "欢迎下次使用！" << endl;

system("pause");

return 0;

break;

}

}

return 0;

}