# 实验3 栈的应用

班级： 2020 级 计算机科学与技术 专业 2 班 学号 姓名

**实验日期**： 2021 年 4 月 2日---- 4月 2 日 **报告日期**： 2021 年 4 月 2日

### 实验目的

掌握栈的结构特性及其入栈，出栈操作；会利用栈实现各种算法。

### 实验预习

复习以下概念

顺序栈：顺序栈是栈的顺序实现。顺序栈是指利用顺序存储结构实现的栈。

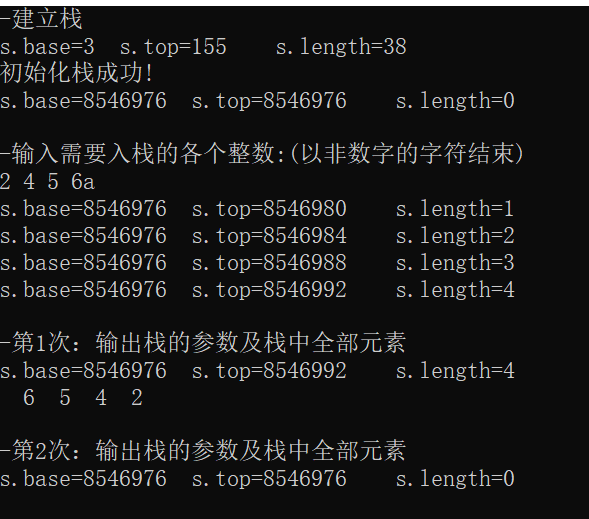
链栈：链式栈是一种数据存储结构，可以通过单链表的方式来实现

循环队列：循环队列就是将队列存储空间的最后一个位置绕到第一个位置，形成逻辑上的环状空间，供队列循环使用。

链队：用链表表示的队列简称为链队

### 实验内容和要求

1.阅读下面程序，将函数Push、函数Pop、及其他函数补充完整。若输入元素序列2456a，运行结果如下所示。



（1）头文件**intstack.h**

/\*顺序栈的实现

文件名：**intstack.h**

\*/

#include<iostream>

#include<fstream>

using namespace std;

//顺序栈定义

#define OK 1

#define ERROR 0

#define OVERFLOW -2

#define MAXSIZE 100//顺序栈存储空间的初始分配量

typedef int Status;

typedef int SElemType;

typedef struct {

SElemType \*base;//栈底指针

SElemType \*top;//栈顶指针

int stacksize;//栈可用的最大容量

} SqStack;

//算法3.1　顺序栈的初始化

Status InitStack(SqStack &S) {

//构造一个空栈S

S.base = new SElemType[MAXSIZE];//为顺序栈动态分配一个最大容量为MAXSIZE的数组空间

if (!S.base)

exit(OVERFLOW); //存储分配失败

S.top = ; //top初始为base，空栈

S.stacksize = MAXSIZE; //stacksize置为栈的最大容量MAXSIZE

return OK;

}

//算法3.2　顺序栈的入栈

Status Push(SqStack &S, SElemType e) {

// 插入元素e为新的栈顶元素

//待补充

}

//算法3.3　顺序栈的出栈

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

//删除S的栈顶元素，用e返回其值

//待补充

}

//算法3.4　取顺序栈的栈顶元素

char GetTop(SqStack S) {//返回S的栈顶元素，不修改栈顶指针

if (S.top != S.base) //栈非空

//返回栈顶元素的值，栈顶指针不变

}

////算法3.5　判断栈是否为空

int StackEmpty(SqStack S)

//若栈为空，返回1 ，否则返回0

{if ()

return 1;

else

return 0;

}

void prt\_base\_top\_length(SqStack s)

//输出栈的参数

{

printf("s.base=%d s.top=%d s.length=%d\n",s.base,s.top,s.top-s.base);

}

（2）CPP文件：（请命名为： **学号姓名实验3-1.cpp**）

#include<stdio.h>

#include "intstack.h"

void PrintStack(SqStack &S){

SElemType e;

while(!StackEmpty(S))

{ Pop(S,e);

printf("%3d",e);

}

printf("\n");

}/\*Pop\_and\_Print\*/

int CreateStack(SqStack &S){

int e;

if(InitStack(S))

{printf("初始化栈成功!\n");

prt\_base\_top\_length(S);

}

else{

printf("初始化失败!\n");

return ERROR;

}

printf("\n输入需要入栈的各个整数:(以非数字的字符结束)\n");

while(scanf("%d",&e))

{ Push(S,e);

prt\_base\_top\_length(S);

};

return OK;

}/\*CreateStack\*/

int main(){

SqStack ss;

printf("建立栈\n");

prt\_base\_top\_length(ss);

CreateStack(ss);

printf("\n-第1次：输出栈的参数及栈中全部元素\n");

prt\_base\_top\_length(ss);

PrintStack(ss);

printf("\n-第2次：输出栈的参数及栈中全部元素\n");

prt\_base\_top\_length(ss);

PrintStack(ss);

return 0;

}

**算法分析：**

1. 输入元素序列1 2 3 4，为什么输出序列为4 3 2 1？体现了栈的什么特性？

因为栈的性质是先进后出

1先进 4最后进

4先出 1最后出

1. **为什么第二次输出栈时，栈为空栈？ 如果需要保持栈的原样，应该如何修改程序？**

**复杂的解释**：执行第二次输出时会执行PrintStack(ss);

PrintStack函数会通过循环的方式输出栈

循环条件是栈不为空，符合条件则执行

Pop(S, e);

printf("%3d", e);

其中，Pop函数将会执行

if (S.top == S.base) return ERROR; //extra 2

e = \*(--S.top); //m3

return OK; //m8

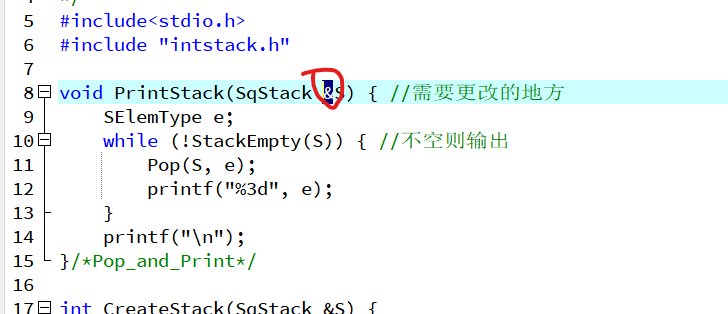
将S.top指针前移，再赋值\*S.top给e并输出

最后S.top等于S.base

在此过程中，对S.top的修改会真实修改在S中，第二次输出时则会输出空栈

**简单的解释**：传入的是地址，一趟输出下来栈已经被清空了

**修改方法**：



移除&符号即可

2.参考教材算法3.20，编写一个十进制转换为八进制的数制转换程序（要求利用栈来实现），并验证其正确性。如输入1346 8，输出2502

（请命名为： **学号姓名实验3-2.cpp**）

提示：可使用第1题建立的intstack.h

参考代码：

#include<stdio.h>

#include "intstack.h"

Status conversion(int N,int R) {

//对于任意一个十进制数，打印输出与其等值的R进制数,2<=R<=16

int e;

SqStack S;

if () return ERROR;

InitStack(S); //初始化空栈S

if(N<0)

{ N=-N; cout<<'-'; }

while (N) //当N非零时,循环

{

Push(); //把N与8求余得到的八进制数压入栈S

; //N更新为N与8的商

}

while (!StackEmpty(S)) //当栈S非空时，循环

{

; //弹出栈顶元素

printf("%X",e); //输出e

}

return OK;

}

int main(){

;

while(1)

{ printf("\n请输入一个十进制整数及需要转换的进制(如45 2),输入0 0结束\n");

scanf("%d%d",&n,&r);

if(r<2) ;

printf("十进制数%4d 对应的%2d 进制数为: ",n,r);

if(conversion(n,r))

printf("\n转换成功！\n");

else

printf("\n参数错误！\n");

}

return 0;

}

验证（可自行设计测试用例，尽量覆盖各种可能情况）：

|  |  |
| --- | --- |
| 输入数据 | 运行结果 |
| 159 8 | 237 |
| -159 8 | -237 |
| 159 16 | 9F |
| 159 2 | 10011111 |
| 159 20 | 参数错误 |
| 159 0 | 【无输出】 |

3.(选做)参考教材算法3.21，设计一个判断表达式括号是否匹配的算法，并编程实现。

（请命名为： **学号姓名实验3-3.cpp**）

如： 输入：2+[(c-d)\*6-(f-7)\*a)]/6 输出：不匹配

输入：a-[(c-d)\*6-(s/3-x)]/2 输出：匹配

代码：

/\*参考教材算法3.21，设计一个判断表达式括号是否匹配的算法，并编程实现。

文件名：\*\_实验3-3.cpp

by Chenx221 (https://chenx221.cyou)

2.0

\*/

#include<stdio.h>

#include"intstack.h"

int main() {

    SqStack S;

    char key;

    int l,flag=0;

    InitStack(S);

    cout<<"pls type:";

    while ((key=getchar())!='\n') {

        switch (key) {

            case ')':{

                if (GetTop(S)!='(') flag=1;

                Pop(S,l);

                break;

            }

            case ']':{

                if (GetTop(S)!='[') flag=1;

                Pop(S,l);

                break;

            }

            case '}':{

                if (GetTop(S)!='{') flag=1;

                Pop(S,l);

                break;

            }

            case '(':case '[':case '{':{

                Push(S,key);

                break;

            }

        }

    }

    if (flag || !StackEmpty(S)) cout<<"not match";

    else cout<<"match";

}

验证：

### 实验小结

栈的性质是先进后出

选做题一开始写的程序少了对

一上来就是)]

多了)]

两种情况的判断

后面发现了问题所在

\*重置了选做题的代码