

顺序栈实现

```
#include <stdio.h>
#include <stdlib.h>

//函数结果状态代码
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -1

typedef int Status; //用作函数值类型，表示函数结果状态
typedef char ElemType; //统一用ElemType表示数据元素类型，视情况而改

typedef struct{
    ElemType *elem; //存储空间的基址
    int top; //栈顶位标（栈顶元素的下一个位置）
    int size; //当前分配的存储容量
    int increment; //扩容时，增加的容量
}SqStack, *pSqStack;

//operation
Status InitStack(pSqStack S, int size, int inc); //初始化顺序栈

Status DestroyStack(pSqStack S); //销毁栈

/*判断栈是否为空，若空返回TRUE；若非空返回FALSE */
Status StackEmpty(pSqStack S);

void ClearStack(pSqStack S); //清空栈S

Status Push(pSqStack S, ElemType e); //元素e压入栈

Status Pop(pSqStack S, ElemType *e); //栈顶元素出栈，用e返回

Status GetTop(pSqStack S, ElemType *e); //获取栈顶元素

//初始化顺序栈
Status InitStack(pSqStack S, int size, int inc)
{
    if(size <= 0 || inc < 0) return ERROR; //参数不合理
    S->elem = (ElemType *)malloc(size * sizeof(ElemType));
    if(NULL == S->elem)
    {
        return OVERFLOW;
    }
    S->top = 0; //用0表示不存在元素
    S->size = size;
    S->increment = inc;
    return OK;
}
```

```

}
//销毁栈
Status DestroyStack(pSqStack S)
{
    ClearStack(S);
    free(S->elem); //释放elem空间
    free(S);
    return OK;
}
/*判断栈是否为空，若空返回TRUE；若非空返回FALSE */
Status StackEmpty(pSqStack S)
{
    /*S不会为空的，因为会先定义一个结构体类型，然后取其地址赋值给S
    if(S == NULL){
        return ERROR;
    }
    */
    if(S->top != 0){
        return FALSE;
    }else{
        return TRUE;
    }
}
//清空栈S
void ClearStack(pSqStack S)
{
    //这里直接把栈顶位标设为0就可以了吧？毕竟释放空间是销毁操作的事
    S->top = 0;
}
//元素e压入栈
Status Push(pSqStack S, ElemType e)
{
    if(S->top == S->size){ //栈满，扩容
        ElemType *newbase;
        newbase = (ElemType *)realloc(S->elem, (S->size + S->increment) *
sizeof(ElemType));
        if(newbase == NULL){
            return OVERFLOW;
        }
        S->elem = newbase;
        S->size += S->increment;
        newbase = NULL;
    }
    S->elem[S->top++] = e; //e入栈，栈顶位标自增
    return OK;
}
//栈顶元素出栈，用e返回
Status Pop(pSqStack S, ElemType *e)
{
    //判断是否为空
    Status flag = StackEmpty(S);
    if(flag == TRUE){
        e = NULL;
        return ERROR;
    }
    //由于是定义一个e然后传递其地址过来，所以e一定不为NULL
    *e = S->elem[--S->top]; //注意，这里是--top，因为对于数组来说，top多一位
    return OK;
}

```

```

}

//获取栈顶元素
Status GetTop(pSqStack S,ElemType *e)
{
    //判断是否为空
    Status flag = StackEmpty(S);
    if(flag == TRUE){
        e = NULL;
        return ERROR;
    }
    //由于是定义一个e然后传递其地址过来，所以e一定不为NULL
    *e = S->elem[S->top - 1];
    return OK;
}

```

顺序栈测试1：进制转换

```

#include <stdio.h>
#include "stack_sq.c"
#include <windows.h>

/*
    进制转换为不断把数除以转换进制的基数
*/

void Conversion(int num,int base);

int main()
{
    system("color 4F");//控制小窗口颜色
    printf("*****\n\n");
    printf("    *欢迎来到顺序栈测试页面*\n\n");
    printf("    *这是进制转换测试，步骤为:*\n\n");
    printf("    ***1、输入想转换的数***\n\n");
    printf("    ***2、输入转换后的进制***\n\n");
    printf("*****\n\n");

    int num,base;//num为原数，base为转换后的进制
    printf("\n请输入想转换的数：");
    scanf("%d",&num);
    printf("\n请输入转换后的进制：");
    scanf("%d",&base);
    printf("\n%d的%d进制为：",num,base);
    Conversion(num,base);

    system("pause");
    return 0;
}

void Conversion(int num,int base)
{
    Status status;
    //这里有一个问题，e不要设为指针，要不然地址不确定
    ElemType e;
    SqStack S;

```

```

int i = 0; //每三位输出空格
status = InitStack(&S, 10, 5);
if(status == OVERFLOW){
    printf("内存分配失败! \n");
}
while(num != 0){ //不断把余数存到栈中
    Push(&S, num%base);
    num /= base;
}
while(StackEmpty(&S) != TRUE){
    i++;
    Pop(&S, &e);
    printf("%d", e);
    if(i % 3 == 0){
        printf(" ");
    }
}
printf("\n\n");
DestroyStack(&S); //销毁栈
}

```

顺序栈测试2：括号匹配

```

#include <stdio.h>
#include "stack_sq.c"
#include <string.h>
#include <windows.h>

/*
    括号匹配有三种不匹配情况：1) 右括号非所期待；
    2) 右括号多余；3) 左括号多余
*/

Status Matching(char *exp, int n);

int main()
{
    system("color 4F"); //控制小窗口颜色
    printf("*****\n\n");
    printf("    欢迎来到顺序栈测试页面*\n\n");
    printf("    *这是括号匹配测试，步骤为:*\n\n");
    printf("    ***1、想输入多少个括号***\n\n");
    printf("    ***2、输入想匹配的括号集***\n\n");
    printf("*****\n\n");

    int n;
    printf("\n请输入想匹配的括号数: ");
    scanf("%d", &n);
    char *exp = (char *)malloc(sizeof(char)*(n+1));
    if(NULL == exp){
        printf("开辟空间失败!\n");
        return ;
    }
    printf("\n请输入想匹配的括号集: ");
    fflush(stdin); //清空输入缓冲区

```

```

scanf("%s",exp);
printf("\n括号集%s的匹配检查结果: ",exp);
if(strlen(exp) < n){
    char *temp = (char *)realloc(exp,strlen(exp)+1);
    if(temp == NULL) printf("内存重新分配失败! \n");
    else exp = temp;
    temp = NULL;
}
Status status = Matching(exp,strlen(exp));
if(status == OK){
    printf("匹配! \n");
}else{
    printf("不匹配! \n");
}
free(exp);//释放内存
system("pause");
}

Status Matching(char *exp,int n){
    int i;
    ElemType e;
    SqStack S;
    Status status = InitStack(&S,n,5);
    if(status == OVERFLOW){
        printf("内存分配失败! \n");
    }
    /*
    1、一次扫描exp，每读入一个括号：
        1) 如果是左括号，入栈；
        2) 如果是右括号，检查栈是否空，若是，则表示"右括号多余"，不匹配，结束；
        否则与栈顶元素比较，若匹配，栈顶元素出栈；否则，属"右括号非所期待"，不匹配，结束；
    2、若已经扫描完，判断栈是否空了，若是，则匹配；否则，"左括号多余"，不匹配，结束
    */
    for(i = 0;i < n;i ++){
        switch(exp[i]){
            case '(':
            case '[':
            case '{':
                {
                    Push(&S,exp[i]);
                    break;
                }
            case ')':
            case ']':
            case '}':
                {
                    if(StackEmpty(&S) == TRUE){//空栈
                        return ERROR;
                    }else{
                        GetTop(&S,&e);
                        if((e == '(' && exp[i] == ')') || (e == '[' && exp[i] ==
']') ||
                            (e == '{' && exp[i] == '}'))//匹配，则出栈
                        {
                            Pop(&S,&e);
                            break;
                        }
                    }
                }
            }
    }

```

```
        else{//右括号多余
            return ERROR;
        }
    }
}
default:
{
    if(exp[i] < 0 || exp[i] > 255)
        printf("\n不是ASCII码! \n");
    break;
}
}
}
if(StackEmpty(&S) != TRUE){
    return ERROR;//左括号多余
}else{
    return OK;//匹配
}
DestroyStack(&S);
}
```