用线性表表示的顺序栈

Aug 29, 2018 | 学习笔记——C数据结构 | 2 阅读 | 1.3k 字 | 6 分钟

文章目录

LinearListStack(线性表栈)

github源码

特点：

1.从数据结构角度看，栈也是线性表，其特殊性在于栈的基本操作是线性表操作的子集，是操作受限的线性表。但是从数据类型角度看，栈是和线性表大不相同的两类抽象数据类型。

2.栈（stack）是限定仅在表尾进行插入或删除操作的线性表，因此，对栈来说，表尾端有其特殊含义，称为栈顶（top），表头端称为栈底（bottom）。

3.栈的修改是按后进先出的原则进行的，因此，栈又称为后进先出（Last in First out）的线性表（简称LIFO结构）。

4.和线性表类似，栈也有两种存储表示方法，顺序栈和链栈。

顺序栈，即栈的顺序存储结构是利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素，同时附设指针top指示栈顶元素在顺序栈中的位置。

5.由于栈在使用过程中所需最大空间的大小很难估计，因此一般情况下，在初始化设置空栈时不应限定栈的最大容量。对于顺序栈，一个较合理的做法是，先为顺序栈分配一个基本容量，然后在应用过程中，当栈的空间不足够使用时再逐步扩大。为此，可设定两个常量：STACK_INIT_SIZE(存储空间初始分配量)和STACKINCREMENT(存储空间分配增量)。

LinearListStack.c文件

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

```c
#include <malloc.h>
#include "LinearListStack.h"
//线性表栈

static void clear(LinearListStack *This);
static int isEmpty(LinearListStack *This);
static int length(LinearListStack *This);
static void risePrint(LinearListStack *This);
static void downPrint(LinearListStack *This);
static int getTop(LinearListStack *This,ElemType* e);
static int push(LinearListStack *This,ElemType* e);
static int pop(LinearListStack *This, ElemType* e);

LinearListStack *InitLinearListStack(){
    LinearListStack *L = (LinearListStack *)malloc(sizeof(LinearListStack));
    LinearListStack_P *p = (LinearListStack_P *)malloc(sizeof(LinearListStack_P));
    p->base = (ElemType *)malloc(STACK_INIT_SIZE * sizeof(ElemType));
    p->top = p->base;
    p->length = 0; //当前长度
    p->size = STACK_INIT_SIZE; //当前分配量
    L->This = p;
    L->clear = clear;
    L->isEmpty = isEmpty;
    L->length = length;
    L->risePrint = risePrint;
    L->downPrint = downPrint;
    L->getTop = getTop;
    L->push = push;
    L->pop = pop;
    return L;
}

void DestroyLinearListStack(LinearListStack *L){
    free(L->This);
    free(L);
    L = NULL;
}

static void clear(LinearListStack *This){
    LinearListStack_P *p = This->This;
    p->top = p->base;
    p->length = 0; //当前长度
}
```

```
static int isEmpty(LinearListStack *This){
    LinearListStack_P *p = This->This;
    return (p->length == 0);
}

static int length(LinearListStack *This){
    LinearListStack_P *p = This->This;
    return p->length;
}

static void risePrint(LinearListStack *This){
    LinearListStack_P *p = This->This;
    int i;
    for (i=0; i < p->length; i++){
        printf("%c", *(p->base + i));
    }
    printf("\n");
}

static void downPrint(LinearListStack *This){
    LinearListStack_P *p = This->This;
    int i;
    for (i=0; i < p->length; i++){
        printf("%c", *(p->top - 1 - i));
    }
    printf("\n");
}

static int getTop(LinearListStack *This,ElemType* e){
    LinearListStack_P *p = This->This;
    if (p->top == p->base) return -1;
    *e = *(p->top-1);
    return 0;
}

static int push(LinearListStack *This,ElemType* e){
    LinearListStack_P *p = This->This;
    if (p->top - p->base >= p->size){ //判断存储空间是否够用
        ElemType *newbase = (ElemType *)realloc(p->base, (p->size + STACKINCREMENT)*sizeof(E
        if (!newbase) return -1;//存储空间分配失败
        p->base = newbase;//新基址
        p->top = p->base + p->size;
```

```
        p->size += STACKINCREMENT;//增加存储容量
    }
    *((p->top)++) = *e;
    ++p->length;
    return 0;
}

static int pop(LinearListStack *This, ElemType* e){
    LinearListStack_P *p = This->This;
    if (p->top == p->base) return -1;
    *e = *(p->top-1);
    p->top--;
    p->length--;
    return 0;
}
```

LinearListStack.h文件

```
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __LINEARLISTSTACK_H
#define __LINEARLISTSTACK_H
/* Includes ------------------------------------------------------------------*/
/* Exported types ------------------------------------------------------------*/
typedef char ElemType;

typedef struct LinearListStack_P{
    ElemType *base;
    ElemType *top;   //栈顶指针
    int length;      //当前线性表栈的长度
    int size;    //当前分配的存储容量
}LinearListStack_P;

typedef struct LinearListStack{
    LinearListStack_P *This;
    void (*clear)(struct LinearListStack *This);
    int (*isEmpty)(struct LinearListStack *This);
    int (*length)(struct LinearListStack *This);
    void (*risePrint)(struct LinearListStack *This);
    void (*downPrint)(struct LinearListStack *This);
```

```
    int (*getTop)(struct LinearListStack *This,ElemType* e);
    int (*push)(struct LinearListStack *This,ElemType* e);
    int (*pop)(struct LinearListStack *This, ElemType* e);
}LinearListStack;

/* Exported define -----------------------------------------------------------*/
#define STACK_INIT_SIZE 100 //线性表栈存储空间的初始分配量
#define STACKINCREMENT 10   //线性表栈存储空间的分配增量(当存储空间不够时要用到)
/* Exported macro ------------------------------------------------------------*/
LinearListStack *InitLinearListStack();
void DestroyLinearListStack(LinearListStack *L);

#endif
```

testLinearListStack.c文件

```
#include <stdio.h>
#include <malloc.h>
#include "LinearListStack.h"

int strlen(char *str){
    int i = 0;
    while(*(str+i) != '\0'){
        i++;
    }
    return i;
}

int main(void)
{
    int i;
    char string[] = "abcdefgh";
    int strlength = strlen(string);
    ElemType elem;
    LinearListStack *stack = InitLinearListStack();
    printf("string length = %d\n",strlength);
    printf("stack is empty:%d\n",stack->isEmpty(stack));
    for (i = 0; i < strlength; i++){
        stack->push(stack,string+i);
    }
```

```
    printf("base to top: \n");
    stack->risePrint(stack);
    printf("top to base: \n");
    stack->downPrint(stack);
    printf("stack is empty:%d\n",stack->isEmpty(stack));
    printf("stack length:%d\n",stack->length(stack));
    for(i=0;i < strlength; i++){
        stack->getTop(stack,&elem);
        printf("get top elem:%c\n",elem);
        stack->pop(stack,&elem);
        printf("pop elem:%c\n",elem);
    }
    printf("stack is empty:%d\n",stack->isEmpty(stack));
    printf("stack length:%d\n",stack->length(stack));
    stack->clear(stack);
    DestroyLinearListStack(stack);
    return 0;
}
```

编译：

```
gcc LinearListStack.c LinearListStack.h testLinearListStack.c -o testLinearListStack
```

运行testLinearListStack

输出：

```
string length = 8
stack is empty:1
base to top:
abcdefgh
top to base:
hgfedcba
stack is empty:0
stack length:8
get top elem:h
pop elem:h
get top elem:g
```

```
pop elem:g
get top elem:f
pop elem:f
get top elem:e
pop elem:e
get top elem:d
pop elem:d
get top elem:c
pop elem:c
get top elem:b
pop elem:b
get top elem:a
pop elem:a
stack is empty:1
stack length:0
```


C

Search

分类

学习笔记——C 算法
学习笔记——C数据结构
学习笔记——Python
学习笔记——android
学习笔记——expert c programming
学习笔记——linux
学习笔记——opencv
学习笔记——嵌入式开发
学习笔记——机器学习
学习笔记——网络协议

标签

android C 网络协议 linux 嵌入式开发 Python opencv 机器学习

最近文章

linux解压缩命令
linux查找命令
Little Kernel 04

Little Kernel 04

Little Kernel 03

Little Kernel 02

Little Kernel 01

消息摘要算法

C按位操作实现CRC计算算法

CRC循环冗余校验算法

链表的反转

友情链接

人生的小站