实验 3-1　栈的动态顺序存储实现

一、实验目标

通过本实验，你将能够在以下几个方面得到锻炼：

1. 用 C 语言熟练编写程序和调试程序；
2. 掌握和运用 C 语言的高级语法特性，包括：结构体、指针、函数指针、typedef、动态内存分配和递归函数，等。
3. 掌握从分析问题到编写伪代码、编写程序、调试程序和测试程序的思路；
4. 能够根据给定的 ADT 规格说明熟练的编写代码；
5. 理解并熟练掌握栈动态顺序存储结构的 C 语言描述；
6. 掌握线栈的动态顺序存储结构的算法实现。

二、ADT 规格说明

栈的 ADT 规格说明请参考教材 45 页。

三、实验要求

用动态顺序存储的方式实现栈的 ADT 规格说明。

四、实验步骤

1. 建立本次实验的文件夹 Lab3-1。
2. 从 Moodle 网络课堂下载实验文件 Lab3-1.rar，包含五个代码文件：ElemType.h、ElemType.cpp、DynaSeqStack.h、DynaSeqStack.cpp 和 Lab.cpp。各代码文件的作用如下表所示：

文件	作用
ElemType.h	定义 ElemType 数据类型
ElemType.cpp	实现 ElemType 数据类型的 Compare 和 visit 两个操作
DynaSeqStack.h	动态顺序栈 ADT 的定义

DynaSeqStack.cpp	**动态顺序栈 ADT 的实现（待完成）**
Lab.cpp	**测试动态顺序栈的主程序（待完成）**

3. ElemType.h 文件定义了 ElemType 类型，ElemType.cpp 文件针对 ElemType 类型实现了 Compare 和 visit 函数，这样栈中就可以存放 ElemType 这种类型的数据了。这两个文件的代码如下。

ElemType.h

```
#ifndef ELEMTYPE_H
#define ELEMTYPE_H

typedef int ElemType;

int  compare(ElemType x, ElemType y);
void visit(ElemType e);

#endif /* ELEMTYPE_H */
```

ElemType.cpp

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

int compare(ElemType x, ElemType y)
{
   return(x-y);
}

void visit(ElemType e)
{
   printf("%d\n", e);
}
```

4. DynaSeqStack.h 文件是动态顺序栈的数据结构储存定义和基本操作的声明。DynaSeqStack.cpp 是动态顺序栈基本操作的实现。**本次实验的主要内容就是编写动态顺序栈所有操作函数的实现代码，也就是完成 DynaSeqStack.cpp 代码文件的编写。**注意每个操作函数前面都有函数头注释，包括：操作目的、初始条件、操作结果、函数参数和返回值。这些信息对于保证函数代码的逻辑正确性以及对函数进行测试是非常有用的。这两个文件的代码如下。

DynaSeqStack.h

```
#if !defined(DYNASEQSTACK_H)
#define DYNASEQSTACK_H

#include "ElemType.h"
```

```
/*-------------------------------------------------------------
// 栈结构的定义
-------------------------------------------------------------*/
typedef struct Stack
{
   ElemType *base;         // 栈基址
   ElemType *top;          // 栈顶
   int stacksize;          // 栈存储空间的尺寸
} SqStack;

/*-------------------------------------------------------------
// 栈的基本操作
-------------------------------------------------------------*/

bool InitStack(SqStack *S);
void DestroyStack(SqStack *S);
bool StackEmpty(SqStack S);
int  StackLength(SqStack S);
bool GetTop(SqStack S, ElemType *e);
void StackTraverse(SqStack S, void (*fp)(ElemType));
bool Push(SqStack *S, ElemType e);
bool Pop(SqStack *S, ElemType *e);

#endif /* DYNASEQSTACK_H */
```

DynaSeqStack.cpp

```
/***
*DynaSeqStack.cpp - 动态顺序栈，即栈的动态顺序存储实现
*
*
*题目：实验3-1 栈的动态顺序存储实现
*
*班级：
*
*姓名：
*
*学号：
*
****/

#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <assert.h>
```

```
#include "DynaSeqStack.h"

const int STACK_INIT_SIZE = 100;    // 初始分配的长度
const int STACKINCREMENT  = 10;     // 分配内存的增量

/*----------------------------------------------------------
操作目的：初始化栈
初始条件：无
操作结果：构造一个空的栈
函数参数：
      SqStack *S  待初始化的栈
返回值：
      bool        操作是否成功
----------------------------------------------------------*/
bool InitStack(SqStack *S)
{
}

/*----------------------------------------------------------
操作目的：销毁栈
初始条件：栈S已存在
操作结果：销毁栈S
函数参数：
      SqStack *S  待销毁的栈
返回值：
      无
----------------------------------------------------------*/
void DestroyStack(SqStack *S)
{
}

/*----------------------------------------------------------
操作目的：判断栈是否为空
初始条件：栈S已存在
操作结果：若S为空栈，则返回true，否则返回false
函数参数：
      SqStack S   待判断的栈
返回值：
      bool        是否为空
----------------------------------------------------------*/
bool StackEmpty(SqStack S)
{
}

/*----------------------------------------------------------
```

```
操作目的：得到栈的长度
初始条件：栈S已存在
操作结果：返回S中数据元素的个数
函数参数：
      SqStack S   栈S
返回值：
      int         数据元素的个数
----------------------------------------------------------------*/
int StackLength(SqStack S)
{
}

/*----------------------------------------------------------------
操作目的：得到栈顶元素
初始条件：栈S已存在
操作结果：用e返回栈顶元素
函数参数：
      SqStack S   栈S
      ElemType *e 栈顶元素的值
返回值：
      bool        操作是否成功
----------------------------------------------------------------*/
bool GetTop(SqStack S, ElemType *e)
{
}

/*----------------------------------------------------------------
操作目的：遍历栈
初始条件：栈S已存在
操作结果：依次对S的每个元素调用函数fp
函数参数：
      SqStack S      栈S
      void (*fp)()   访问每个数据元素的函数指针
返回值：
      无
----------------------------------------------------------------*/
void StackTraverse(SqStack S, void (*fp)(ElemType))
{
}

/*----------------------------------------------------------------
操作目的：压栈——插入元素e为新的栈顶元素
初始条件：栈S已存在
操作结果：插入数据元素e作为新的栈顶
函数参数：
```

```
      SqStack *S  栈S
      ElemType e  待插入的数据元素
返回值:
      bool        操作是否成功
-------------------------------------------------------------*/
bool Push(SqStack *S, ElemType e)
{
}

/*-------------------------------------------------------------
操作目的: 弹栈—删除栈顶元素
初始条件: 栈S已存在且非空
操作结果: 删除S的栈顶元素，并用e返回其值
函数参数:
      SqStack *S  栈S
      ElemType *e 被删除的数据元素值
返回值:
      bool        操作是否成功
-------------------------------------------------------------
*/
bool Pop(SqStack *S, ElemType *e)
{
}
```

5. Lab.cpp 文件包含程序入口函数 main 用来声明和定义栈对象，并使用栈 ADT 提供的操作。在这里主要写测试代码，用来保证栈 ADT 实现的正确性。初始代码如下。

Lab.cpp

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

#include "DynaSeqStack.h"

int main()
{
   // TODO: place your test code here

   return 0;
}
```

6. 使用 VC++ 6.0 建立一个空的控制台工程，把上面的五个代码文件加入到工程中，进行代码的编写和调试。

五、思考问题

1. SqStack 数据结构定义中 stacksize 的作用是什么？
2. 如何判断 SqStack 为空栈？
3. 压栈时栈顶指针如何变化？
4. 出栈时栈顶指针如何变化？
5. SqStack 数据结构的 8 个操作中，哪些操作需要传递 SqStack 对象的指针，哪些操作不需要传递 SqStack 对象的指针。这里面有什么规律？为什么某些操作需要传递 SqStack 对象的指针？
6. SqStack 中的元素类型为 ElemType，有什么好处？
7. 自己定义一个 student 类型（包括：name，age，gender，三个字段），再定义一个 book 类型（包括：name, author, publisher, datetime，四个字段），请问能用本实验完成的 SqStack ADT，在不改动 DynaSeqStack.h 和 DynaSeqStack.cpp 代码的前提下，能对这两种不同类型的数据进行栈操作吗？请写这样的测试程序？如果可以实现，请问这其中的原理何在？