/\*顺序栈的C语言定义与实现\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>//malloc,realloc函数

/\*预定义常量类型\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define LOVERFLOW -2

typedef int Status;

/\*结构体定义\*/

#define STACK\_INIT\_SIZE 100 //初始大小为100

#define STACKINCREMENT 10 //若空间不够，每次增长10

typedef int SElemType;

typedef struct {

SElemType \*base; //存放动态申请空间的首地址（栈底）

SElemType \*top; //栈顶指针

int stacksize; //当前分配的元素个数

} SqStack;

SqStack s;

/\*初始化栈\*/

Status InitStack(SqStack \*s)

{

s->base = (SElemType\*)malloc(STACK\_INIT\_SIZE \* sizeof(SElemType));

if (!s->base) exit(LOVERFLOW);

s->top = s->base;

s->stacksize = STACK\_INIT\_SIZE;

return OK;

}

Status DestroyStack(SqStack \*s)

{

if (s->base)

free(s->base);

s->top = NULL;

s->stacksize = 0;

return OK;

}

Status ClearStack(SqStack \*s)

{

if (s->stacksize > STACK\_INIT\_SIZE)

{/\*如果栈扩展过，恢复初始大小\*/

s->base = (SElemType \*)realloc(s->base, STACK\_INIT\_SIZE \* sizeof(SElemType));

if (s->base == NULL)

exit(LOVERFLOW);

s->stacksize = STACK\_INIT\_SIZE;

}

s->top = s->base;

return OK;

}

Status StackEmpty(SqStack s)

{

if (s.top == s.base)

return TRUE;

else

return FALSE;

}

int StackLength(SqStack s)

{

return s.top - s.base;

}

Status GetTop(SqStack s,SElemType \*e)

{

if (s.top==s.base)

return ERROR;

\*e=\*(s.top - 1);

return OK;

}

Status PushStack(SqStack \*s, SElemType e)//\*s或者&s都可

{

/\*如果栈已满则扩充空间\*/

if (s->top - s->base >= s->stacksize)

{

s->base = (SElemType\*)realloc(s->base, (s->stacksize + STACKINCREMENT) \* sizeof(SElemType));

if (!s->base)

return LOVERFLOW;

/\*因为s->base可能会变，因此要修正s->top的值\*/

s->top = s->base + s->stacksize;

s->stacksize += STACKINCREMENT;

}

\*s->top++ = e;

return OK;

}

Status PopStack(SqStack &s, SElemType &e)

{

if (s.top == s.base)

return ERROR;

e = \*--s.top;

return OK;

}

/\*顺序栈的C++语言定义与实现\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

/\*预定义常量类型\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define LOVERFLOW -2

typedef int Status;

typedef int SElemType;

/\*结构体定义\*/

#define STACK\_INIT\_SIZE 100 //初始大小为100

#define STACKINCREMENT 10 //若空间不够，每次增长10

class SqStack

{

protected:

SElemType \*base;

SElemType \*top;

int stacksize;

public:

SqStack();

~SqStack();

Status ClearStack();

Status StackEmpty();

int StackLength();

Status GetTop(SElemType&e);

Status Push(SElemType e);

Status Pop(SElemType&e);

};

SqStack::SqStack()

{

base = new SElemType[STACK\_INIT\_SIZE];

if (base == NULL)

exit(LOVERFLOW);

top = base;

stacksize = STACK\_INIT\_SIZE;

}

SqStack::~SqStack()

{

if (base)

delete base;

top = NULL;

stacksize = 0;

}

Status SqStack::ClearStack()

{

if (stacksize > STACK\_INIT\_SIZE)

{/\*如果栈扩展过，恢复初始大小\*/

delete base;

base = new SElemType[STACK\_INIT\_SIZE];

if (base == NULL)

exit(LOVERFLOW);

stacksize = STACK\_INIT\_SIZE;

}

top = base;

return OK;

}

Status SqStack::StackEmpty()

{

if (top == base)

return TRUE;

else

return FALSE;

}

int SqStack::StackLength()

{

return top - base;

}

Status SqStack::GetTop(SElemType &e)

{

if (top ==base)

return ERROR;

e = \*(top - 1);

return OK;

}

Status SqStack::Push(SElemType e)

{

/\*如果栈已满则扩充空间\*/

if (top - base >= stacksize)

{

SElemType \*newbase;

newbase = new SElemType[stacksize + STACK\_INIT\_SIZE];

if (!newbase)

return LOVERFLOW;

memcpy(newbase, base, stacksize \* sizeof(SElemType));

delete base;

base = newbase;

top = base + stacksize;

stacksize += STACKINCREMENT;

}

\*top++ = e;

return OK;

}

Status SqStack::Pop(SElemType &e)

{

if (top == base)

return ERROR;

e = \*--top;

return OK;

}