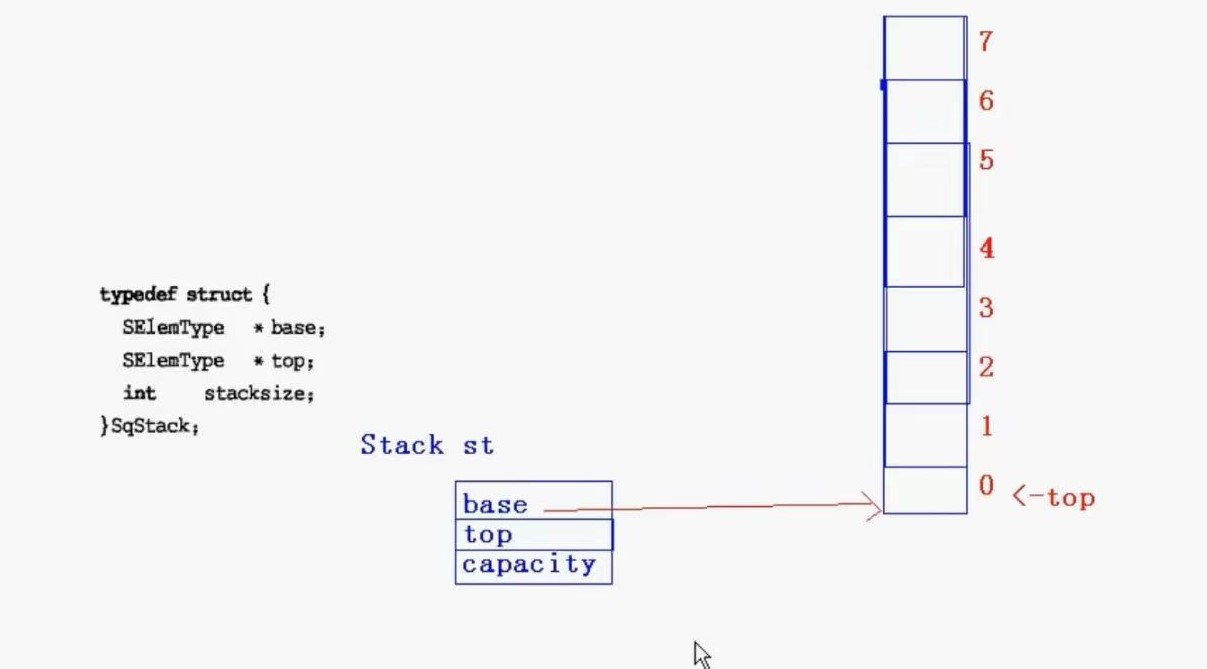
**栈结构之栈的两种实现方式**



top指针可以从0开始也可以从-1开始，每放入一个数据后top++。二者唯一的区别就是在取数据的时候，从0开始的top要减去1之后再取数。

**一、顺序栈的实现**

//栈结构之顺序栈的实现

#include <stdio.h> //标准输入输出的头文件

#include <malloc.h>

#include <assert.h>

typedef int ElemType;

#define STACK\_INIT\_SIZE 8 //初始栈的空间大小

#define STACK\_INC\_SIZE 3 //增加空间的大小

typedef struct SeqStack {

ElemType \*base; //指向整个栈的空间的指针

int capacity;

int top;

}SeqStack;

void InitStack(SeqStack \*s) {

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

assert(s->base != NULL);

s->capacity = STACK\_INIT\_SIZE;

s->top = 0;

}

bool Inc(SeqStack\* s) {

//重新开辟空间的代码 一定要熟悉

ElemType\* newbase = (ElemType\*)realloc(s->base, sizeof(ElemType) \* (s->capacity + STACK\_INC\_SIZE));

if (newbase == NULL) {

printf("内存不足，无法申请空间.\n");

return false;

}

s->base = newbase; //更改地址指针的指向

s->capacity += STACK\_INC\_SIZE; //更改容量大小

return true;

}

bool IsFull(SeqStack\* s) { //压栈判断满不满

return s->top >= s->capacity;

}

bool IsEmpty(SeqStack\* s) { //出栈判断空不空

return s->top == 0;

}

void Push(SeqStack\* s, ElemType x) {

//不用单独写一句Inc(s);是因为在下面的if语句里会执行

if (IsFull(s) && !Inc(s)) {

printf("栈空间已满，%d不能入栈.\n", x);

return;

}

s->base[s->top++] = x;

//s->top++;

}

void Show(SeqStack\* s) {

for (int i = s->top - 1; i >= 0; i--) {

printf("%d\n", s->base[i]);

}

printf("\n");

}

void Pop(SeqStack\* s) {

if (IsEmpty(s)) {

printf("栈空间已空，不能出栈.\n");

return;

}

s->top--; //直接将top--，后续如果需要有值入栈，会直接将top所指向的数值覆盖

}

//该函数是为了获取栈顶元素

bool GetTop(SeqStack\* s, ElemType \*v) {

if (IsEmpty(s)) {

printf("栈空间已空，不能取栈顶元素.\n");

return false;

}

\*v = s->base[s->top - 1]; //\*v=s->base[--s->top]

//这是一个不对的操作，因为除了获取了栈顶元素，还进行了出栈操作

return true;

}

int Length(SeqStack\* s) {

return s->top;

}

void Clear(SeqStack\* s) {

s->top = 0;

}

void Destroy(SeqStack\* s) {

free(s->base);

s->base = NULL; //千万要记得赋空 预防遗指针

s->capacity = 0;

s->top = 0;

}

void main() {

SeqStack st;

InitStack(&st);

ElemType v;

for (int i = 1; i <= 5; i++) {

Push(&st, i);

}

Show(&st);

GetTop(&st, &v); //通过这个函数可以给v赋上值

printf("%d 出栈.\n", v);

Pop(&st);

Show(&st);

//Clear(&st);

//Show(&st);

//Destroy(&st);

}

**二、链栈的实现**

//栈结构之链栈的实现

//头部插入必须进行头部删除

#include <stdio.h> //标准输入输出的头文件

#include <malloc.h>

#include <assert.h>

typedef int ElemType;

typedef struct StackNode {

ElemType data;

struct StackNode\* next;

}SeqStack,\*LinkStack; //LinkStack 表示栈结点类型的指针

void InitStack(LinkStack\* s) { //括号里的是二级指针

\*s = NULL; //头指针赋值为空

}

void Push(LinkStack\* s, ElemType x) {

LinkStack t = (LinkStack)malloc(sizeof(StackNode));

assert(t != NULL);

t->data = x;

if (\*s == NULL) { //为第一个结点的情况

\*s = t; //头指针指向新开辟的t结点

t->next = NULL;

}

else { //进行头插

t->next = \*s;

\*s = t;

}

}

void Show(LinkStack\* s) {

LinkStack p = \*s;

while (p != NULL) {

printf("%d\n", p->data);

p = p->next;

}

printf("\n");

}

void Pop(LinkStack\* s) {

LinkStack p = \*s;

\*s = p->next;

free(p);

p = NULL; //记得将辅助指针用完后悬挂

}

int main() {

LinkStack st;

InitStack(&st);

for (int i = 1; i <= 5; i++) {

Push(&st, i);

}

Show(&st);

Pop(&st);

Show(&st);

return 0;

}