**栈的应用实例**

#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;

}

**//栈的应用一：进制转换**

void Convert\_8(int value) {

SeqStack st;

InitStack(&st);

int v;

while (value) {

Push(&st, value % 8);

value /= 8;

}

Show(&st);

//方法二

/\*while (!IsEmpty(&st)) {

GetTop(&st, &v);

Pop(&st);

printf("%d", v);

}

printf("\n");\*/

}

void main() {

int value = 47183;

Convert\_8(value); //-->8

}

**//栈的应用二：括号的匹配**

bool Check(string str) {

SeqStack st;

InitStack(&st);

char v;

for (int i = 0; i < size(str); i++) {

if (str[i] == '[' || str[i] == '(') {

Push(&st,str[i]);

}

else if (str[i] == ']') {

GetTop(&st, &v);

if (v != '[') {

return false;

}

Pop(&st);

}

else if (str[i] == ')') {

GetTop(&st, &v);

if (v != '(') {

return false;

}

Pop(&st);

}

}

return IsEmpty(&st); //返回栈空不空 来判断是否都匹配

}

void main() {

string str = "[([][])]";

bool flag = Check(str);

if (flag) {

printf("OK!\n");

}

else {

printf("Error!\n");

}

}

**//栈的应用三：行编辑程序**

void Print(SeqStack\* s) {

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

printf("%c", s->base[i]);

}

printf("\n");

}

//void LineEdit() {

// SeqStack st;

// InitStack(&st);

//

// //这段程序，一次只能修正一个，如何不停修正呢？ 借助一个外层的循环

// char ch = getchar();

// while (ch != '$' && ch != '\n') {

// switch (ch) {

// case '#':

// Pop(&st);

// break;

// case'@':

// Clear(&st);

// break;

// default: //表示其他情况

// Push(&st, ch);

// break;

// }

//

// ch = getchar();

// }

// //Show(&st); //这句是不对的，因为这样打印出来是逆序

// Print(& st);

// Destroy(&st);

//}

//本段程序相当精彩 要好好研究

void LineEdit() {

SeqStack st;

InitStack(&st);

//这段程序，一次只能修正一个，如何不停修正呢？ 借助一个外层的循环

char ch = getchar(); //获取一个字串

while (ch != '$') {

while (ch != '$' && ch != '\n') { //字串里每一个字符的判断条件

switch (ch) {

case '#':

Pop(&st);

break;

case'@':

Clear(&st);

break;

default: //表示其他情况

Push(&st, ch);

break;

}

ch = getchar(); //获取字串里的下一个字符

}

Print(&st);

ch = getchar(); //重新获取一个字串

}

Destroy(&st);

}

void main() {

LineEdit();

}