重庆大学本科学生实验项目任务书

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验题目 | 栈的实现及其应用 | | | |
| 实验时间 |  | 实验地点 | |  |
| 实验性质 | □验证性 √设计性 □综合性 | | | |
| 实验目的  1. 掌握栈“后进先出”的特点；  2. 熟练掌握栈的基本操作及其应用；  3. 掌握栈的顺序存储实现；  4. 掌握表达式求值的实现。 | | | | |
| 实验内容：   1. 实现顺序栈； 2. 用键盘输入一个包括+、-、\*、/、正整数和圆括号的合法算术表达式； 3. 先将算术表达式转换成后缀表达式，然后对后缀表达式求值； 4. 要求输出：输入的算术表达式、转化后的后缀表达式、计算结果；   5. 最后提交完整的实验报告和源程序。 | | | | |
| 参考资料：   * + 《数据结构（ C 语言版）》，严蔚敏，吴伟民编著，清华大学出版社，2007年第1版 | | | | |
| 任务下达日期 2020 年 10月 26 日 | | | 完成日期 年 月 日 | |

说明：学院、专业、年级均填全称，如：计算机学院、计算机科学与技术、2019

1. 顺序栈的实现：

#include <stdio.h>

#include <stdlib.h>

/\* 宏定义 \*/

#define STACK\_INIT\_SIZE 100 // 初始分配量

#define STACKINCREMENT 10 // 分配增量

/\* 状态码 \*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int Status;

typedef int SElemType;

/\* 结构描述 \*/

typedef struct{

SElemType \*base; // 在栈构造之前和销毁之后, base的值为NULL

SElemType \*top; // 栈顶指针

int stacksize; // 当前已分配的存储空间, 以元素为单位

}SqStack;

/\* 初始化顺序栈 \*/

Status InitStack(SqStack \*S){

// 构造一个空栈S

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

if (!S->base) exit(OVERFLOW); // 存储分配失败

S->top = S->base;

S->stacksize = STACK\_INIT\_SIZE;

return OK;

}// InitStack

/\* 销毁顺序栈 \*/

Status DestoryStack(SqStack \*S){

// 销毁栈S, S不再存在

if ((\*S).base) free((\*S).base);

(\*S).base = NULL;

(\*S).stacksize = 0;

return OK;

}// DestoryStack

/\* 清空栈S \*/

Status ClearStack(SqStack \*S){

// 把S置为空栈

(\*S).base = (\*S).top;

return OK;

}// ClearStack

/\* 判断栈S是否为空 \*/

Status StackEmpty(SqStack S){

if (S.base == S.top) return TRUE;

else return FALSE;

}// StackEmpty

/\* 返回栈的长度 \*/

Status StackLength(SqStack S){

if (S.base) return S.top - S.base;

}// StackLength

/\* 取栈顶元素 \*/

Status GetTop(SqStack S, SElemType \*e){

// 若栈不空, 则用e返回S的栈顶元素, 并返回OK; 否则返回ERROR

if (S.top == S.base) return ERROR;

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

return OK;

}// GetTop

/\* 压栈 \*/

Status Push(SqStack \*S, SElemType e){

// 插入元素e为新的栈顶元素

if ((\*S).top - (\*S).base >= (\*S).stacksize){ // 栈满, 追加存储空间

(\*S).base = (SElemType \* )realloc((\*S).base, ((\*S).stacksize + STACKINCREMENT) \* sizeof(SElemType));

if (!(\*S).base) exit(OVERFLOW); // 存储分配失败

(\*S).top = (\*S).base + (\*S).stacksize;

(\*S).stacksize += STACKINCREMENT;

}

\*(\*S).top++ = e;

return OK;

}// Push

/\* 弹栈 \*/

Status Pop(SqStack \*S, SElemType \*e){

// 若栈不空, 则删除S的栈顶元素, 用e返回其值, 并返回OK; 否则返回ERROR

if ((\*S).top == (\*S).base) return ERROR;

\*e = \*(--(\*S).top);

return OK;

}// Pop

/\* 打印栈S \*/

Status ShowStack(SqStack S){

// 遍历栈S并打印

for (int i=0; i<StackLength(S); ++i){

printf("%d ", S.base[i]);

}

return OK;

}// ShowStack

int main()

{

SqStack S; // 创建顺序栈S

// 判断初始化

if (InitStack(&S) == OK)

printf("顺序栈初始化完成.\n");

else{

printf("初始化失败，请重试.\n");

return 0;}

// 向栈中输入值, 利用循环压栈的方法

int n; // n为顺序栈长度

int e = 0;

printf("请输入顺序栈的长度.\n");

scanf("%d", &n);

printf("你将得到一个长度为%d的顺序栈.\n", n);

printf("请输入%d个数据元素, 以回车为间隔.\n", n);

for (int i=0; i<n; ++i){ // 输入数据元素

scanf("%d", &e);

Push(&S, e);

}

// 打印顺序栈

printf("输入完成, 你的顺序栈为:\n");

ShowStack(S);

// 打印栈顶元素

int t = 0; // t存储栈顶元素值

GetTop(S, &t);

printf("栈顶元素值为: %d.\n", t);

// 弹栈

int p = 0; // p存储弹栈出的值

Pop(&S, &p);

printf("弹栈操作完成, 弹栈后的顺序栈如下:\n");

ShowStack(S);

// 清空顺序栈

printf("操作完成, 清空顺序栈.\n");

if (ClearStack(&S) == OK)

printf("已清空顺序栈.\n");

else{

printf("清空顺序栈失败, 请检查.\n");

return 0;

}

// 销毁顺序栈

printf("操作完成, 销毁顺序栈.\n");

if (DestoryStack(&S) == OK)

printf("已销毁顺序栈.\n");

else{

printf("销毁顺序栈失败, 请检查.\n");

return 0;

}

system("pause");

return 0;

}

1. 表达式求值：

#include <stdio.h>

#include <stdlib.h>

/\* 宏定义 \*/

#define STACK\_INIT\_SIZE 100 // 初始分配量

#define STACKINCREMENT 10 // 分配增量

/\* 状态码 \*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int Status;

typedef char SElemType;

/\* 结构描述 \*/

typedef struct{

SElemType \*base; // 在栈构造之前和销毁之后, base的值为NULL

SElemType \*top; // 栈顶指针

int stacksize; // 当前已分配的存储空间, 以元素为单位

}SqStack;

/\* 初始化顺序栈 \*/

Status InitStack(SqStack \*S){

// 构造一个空栈S

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

if (!S->base) exit(OVERFLOW); // 存储分配失败

S->top = S->base;

S->stacksize = STACK\_INIT\_SIZE;

return OK;

}// InitStack

/\* 销毁顺序栈 \*/

Status DestoryStack(SqStack \*S){

// 销毁栈S, S不再存在

if ((\*S).base) free((\*S).base);

(\*S).base = NULL;

(\*S).stacksize = 0;

return OK;

}// DestoryStack

/\* 清空栈S \*/

Status ClearStack(SqStack \*S){

// 把S置为空栈

(\*S).base = (\*S).top;

return OK;

}// ClearStack

/\* 判断栈S是否为空 \*/

Status StackEmpty(SqStack S){

if (S.base == S.top) return TRUE;

else return FALSE;

}// StackEmpty

/\* 返回栈的长度 \*/

Status StackLength(SqStack S){

if (S.base) return S.top - S.base;

}// StackLength

/\* 取栈顶元素 \*/

Status GetTop(SqStack S){

// 若栈不空, 则用e返回S的栈顶元素, 并返回OK; 否则返回ERROR

SElemType e;

if (S.top == S.base) return ERROR;

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

return e;

}// GetTop

/\* 压栈 \*/

Status Push(SqStack \*S, SElemType e){

// 插入元素e为新的栈顶元素

if ((\*S).top - (\*S).base >= (\*S).stacksize){ // 栈满, 追加存储空间

(\*S).base = (SElemType \* )realloc((\*S).base, ((\*S).stacksize + STACKINCREMENT) \* sizeof(SElemType));

if (!(\*S).base) exit(OVERFLOW); // 存储分配失败

(\*S).top = (\*S).base + (\*S).stacksize;

(\*S).stacksize += STACKINCREMENT;

}

\*(\*S).top++ = e;

return OK;

}// Push

/\* 弹栈 \*/

Status Pop(SqStack \*S, SElemType \*e){

// 若栈不空, 则删除S的栈顶元素, 用e返回其值, 并返回OK; 否则返回ERROR

if ((\*S).top == (\*S).base) return ERROR;

\*e = \*(--(\*S).top);

return OK;

}// Pop

/\* 打印栈S \*/

Status ShowStack(SqStack S){

// 遍历栈S并打印

for (int i=0; i<StackLength(S); ++i){

printf("%c ", S.base[i]);

}

return OK;

}// ShowStack

/\* 判断运算符之间的优先级 \*/

char Precede(char a, char b){

// 判断运算符之间的优先级并返回运算符的优先级

int i, j;

char pre[][7] = {

{'>', '>', '<', '<', '<', '>', '>'},

{'>', '>', '<', '<', '<', '>', '>'},

{'>', '>', '>', '>', '<', '>', '>'},

{'>', '>', '>', '>', '<', '>', '>'},

{'<', '<', '<', '<', '<', '=', 0},

{'>', '>', '>', '>', 0, '>', '>'},

{'<', '<', '<', '<', '<', 0, '='}

};

switch(a){

case '+': i = 0; break;

case '-': i = 1; break;

case '\*': i = 2; break;

case '/': i = 3; break;

case '(': i = 4; break;

case ')': i = 5; break;

case '#': i = 6; break;

}

switch(b){

case '+': j = 0; break;

case '-': j = 1; break;

case '\*': j = 2; break;

case '/': j = 3; break;

case '(': j = 4; break;

case ')': j = 5; break;

case '#': j = 6; break;

}

return pre[i][j];

}// Precede

/\* 运算操作实现 \*/

char Operate(char a, char theta, char b){

// 定义表达式求值过程中可能用到的操作

int c; // c为表达式的值

int a1 = atoi(&a);

int b1 = atoi(&b);

switch(theta){

case '+': c = a1 + b1; break;

case '-': c = a1 - b1; break;

case '\*': c = a1 \* b1; break;

case '/': c = a1 / b1; break;

}

char \* c1 = (char\*)malloc(20);

itoa(c, c1, 10);

return \*c1;

}// Operate

/\* 中缀表达式转化为后缀表达式 \*/

SqStack ChangeExpression(){

// 首先将中缀表达式转化为后缀表达式, 随后再对表达式求值

// 使用Num存储运算数, 使用Opera存储运算符

// 栈Opera中, 越往栈顶优先级越高

SqStack NUM; SqStack OPERA;

InitStack(&NUM); InitStack(&OPERA);

char c; char x; char theta;

Push(&NUM, '#'); Push(&OPERA, '#');

printf("请输入一个中缀表达式, 并以\'#\'字符结尾.\n");

c = getchar();

while(c!='#' || GetTop(OPERA)!='#'){

if (c>='0' && c<='9') {Push(&NUM, c); c = getchar();}

else{

switch(Precede(GetTop(OPERA), c)){

case '<': Push(&OPERA, c); c = getchar(); break;

// 优先级大的运算符填入栈中

case '=': Pop(&OPERA, &x);

// 将与右括号配对的左括号一起配对消除

while(x!='('){

Push(&NUM, x); Pop(&OPERA, &x);

}// while

c = getchar(); break;

case '>': Pop(&OPERA, &theta); Push(&NUM, theta);

// 保证运算符中的优先次序是由高到低

break;

}// switch

}// else

}// while

printf("转换完成, 对应的后缀表达式如下:\n");

ShowStack(NUM);

printf("\n");

char e;

SqStack S1;

InitStack(&S1);

while(!StackEmpty(NUM)){

Pop(&NUM, &e);

Push(&S1, e);

}// while

Pop(&S1, &e);

return S1;

}// ChangeExpression

/\* 后缀表达式求值 \*/

char ExpressionEvaluation(SqStack \*PostfixExpression){

// 实现后缀表达式求值

// NUM栈暂存计算中的运算数, OPERA栈暂存计算中的运算符

SqStack NUM, OPERA;

InitStack(&NUM);

InitStack(&OPERA);

char top;

char e; char a, b, theta; char c;

Pop(PostfixExpression, &top); Push(PostfixExpression, top);

while(!StackEmpty(\*PostfixExpression)){

if (GetTop(\*PostfixExpression)>='0' && GetTop(\*PostfixExpression)<='9')

// 将运算数存储至NUM栈中

{Pop(PostfixExpression, &e); Push(&NUM, e);}

else{

// 遇到运算符, 连续弹栈两次取出最近的两个运算数并计算

// 将运算结果继续存入NUM栈中

Pop(PostfixExpression, &theta);

Pop(&NUM, &b); Pop(&NUM, &a);

c = Operate(a, theta, b);

Push(&NUM, c);

}// else

}// while

char answer;

Pop(&NUM, &answer);

return answer;

}// ExpressionEvaluation

int main()

{

SqStack PostfixExpression;

InitStack(&PostfixExpression);

PostfixExpression = ChangeExpression();

printf("下面进行后缀表达式的运算:\n");

char answer;

answer = ExpressionEvaluation(&PostfixExpression);

printf("计算结果为: %c\n", answer);

return 0;

}

3.实验结果：

以此为两个程序的实验结果：



