

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 数据结构 | | | | |
| 实验编号： | 实验六 | | | | |
| 实验名称： | 表达式求值 | | | | |
| 实验人员： | 学号 | 18111303044 | | | |
| 姓名 | 邵一波 | | | |
| 班级 | 2018级计算机类一班 | | | |
| 实验日期： | 2019-11-07 | | | | |
| 实验室： | 2号实验楼202室 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

|  |
| --- |
| **注：具体内容可根据专业特点和实验性质略作调整，页面不够可附页。**   1. **需求分析** 2. 描述：合法数学表达式的求值 3. 输入的形式和输入值的范围   一个包含 ’+’、’-’ 、’\*’ 、’/’ 、’^’、整数和圆括号的合法数学表达式字符串，  并以’#’结尾。（输入的整数范围为0~9）     1. 输出的形式   数学表达值的结果（结果保留两位小数）     1. 程序所能达到的功能   输入一个包含 ’+’、’-’ 、’\*’ 、’/’ 、’^’、整数和圆括号的合法数学表达式,计算该表达式的运算结果。   1. 测试数据   3\*(7-2)#  2-5\*(4-8)#  (4-7)^3+5#  (3+5)-4/5#   1. **概要设计** 2. 抽象数据类型的定义   **CommonDef.h**  #include<stdio.h>  #include<math.h>  #include<stdlib.h>  #define STACK\_INIT\_SIZE 100  #define STACKINCREMENT 10  typedef int Status;  typedef char RElemtype;  typedef double DElemtype;  typedef struct{  RElemtype \*base;  RElemtype \*top;  int stacksize;  }SqStack1; //运算符栈类型定义  typedef struct{  DElemtype \*base;  DElemtype \*top;  int stacksize;  }SqStack2; //操作数栈类型定义   1. 函数的定义   **EoEDef.h**  #include "CommonDef.h"  //运算符栈的初始化  Status OPTR\_InitStack(SqStack1 \*s);  //操作数栈的初始化  Status OPND\_InitStack(SqStack2 \*s);  //取运算符栈的栈顶元素  RElemtype OPTR\_GetTop(SqStack1 \*s);  //取操作数栈的栈顶元素  DElemtype OPND\_GetTop(SqStack2 \*s);  //压栈  Status OPTR\_Push(SqStack1 \*s, RElemtype e);  //压栈  Status OPND\_Push(SqStack2 \*s, DElemtype e);  //出栈  Status OPTR\_Pop(SqStack1 \*s, RElemtype \*e);  //出栈  Status OPND\_Pop(SqStack2 \*s, DElemtype \*e);  //比较运算符的优先级  char Precede(char a,char b);  DElemtype Operate( DElemtype d, char e,DElemtype f);  //判断读入字符是否为运算符  int In(char e);  //计算表达式值  DElemtype EvaluateExpression();   1. 主程序的流程图      1. **详细设计**   **EoETestApp.c**  #include "EoEDef.h"  /\*主函数\*/  int main()  {  printf("请输入算术表达式,以#结束\n");  printf("结果是: %.2lf\n",EvaluateExpression());  return 0;  }  **EoEApp.c**  #include "EoEDef.h"  //运算符栈的初始化  Status OPTR\_InitStack(SqStack1 \*s)  {  s->base=(RElemtype\*)malloc(STACK\_INIT\_SIZE\* sizeof(RElemtype));  if(!s->base)  {  return 0;  }  s->top=s->base;  s->stacksize=STACK\_INIT\_SIZE;  return 1;  }  //操作数栈的初始化  Status OPND\_InitStack(SqStack2 \*s)  {  s->base=(DElemtype\*)malloc(STACK\_INIT\_SIZE\* sizeof(DElemtype));  if(!s->base)  {  return 0;  }  s->top=s->base;  s->stacksize=STACK\_INIT\_SIZE;  return 1;  }  //取运算符栈的栈顶元素  RElemtype OPTR\_GetTop(SqStack1 \*s)  {  if(s->top==s->base)  {  return -1;  }  return \*(s->top-1);  }  //取操作数栈的栈顶元素  DElemtype OPND\_GetTop(SqStack2 \*s)  {  if(s->top==s->base)  {  return -1;  }  return \*(s->top-1);  }  //压栈  Status OPTR\_Push(SqStack1 \*s, RElemtype e)  {  if(s->top-s->base>=s->stacksize) //栈满追加储存空间  {  s->base=(RElemtype\*)realloc(s->base,(s->stacksize+STACKINCREMENT)\*sizeof(RElemtype));  if(!s->base)  {  return 0;  }  s->top=s->base+s->stacksize;  s->stacksize+=STACKINCREMENT;  }  \*(s->top)=e;  s->top++;  return 1;  }  //压栈  Status OPND\_Push(SqStack2 \*s, DElemtype e)  {  if(s->top-s->base>=s->stacksize) //栈满追加储存空间  {  s->base=(DElemtype\*)realloc(s->base,(s->stacksize+STACKINCREMENT)\*sizeof(DElemtype));  if(!s->base)  {  return 0;  }  s->top=s->base+s->stacksize;  s->stacksize+=STACKINCREMENT;  }  \*(s->top)=e;  s->top++;  return 1;  }  //出栈  Status OPTR\_Pop(SqStack1 \*s, RElemtype \*e)  {  if(s->top==s->base)  return 0;  \*e=\*(--(s->top));//先运算再赋值  return 1;  }  //出栈  Status OPND\_Pop(SqStack2 \*s, DElemtype \*e)  {  if(s->top==s->base)  return 0;  \*e=\*(--(s->top));//先运算再赋值  return 1;  }  //比较运算符的优先级  char Precede(char a,char b)  {  char f;  if(a=='+'||a=='-')  {  if(b=='+'||b=='-'||b==')'||b=='#')  f='>';  else if(b=='\*'||b=='/'||b=='('||b=='^')  f='<';  }  else if(a=='\*'||a=='/'||a=='^')  {  if(b=='+'||b=='-'||b=='\*'||b=='/'||b==')'||b=='#'||b=='^')  f='>';  else if(b=='(')  f='<';  }  else if(a=='(')  {  if(b=='+'||b=='-'||b=='\*'||b=='/'||b=='('||b=='^')  f='<';  else if(b==')')  f='=';  }  else if(a==')')  {  if(b=='+'||b=='-'||b=='\*'||b=='/'||b==')'||b=='#'||b=='^')  f='>';  }  else if(a=='#')  {  if(b=='+'||b=='-'||b=='\*'||b=='/'||b=='('||b=='^')  f='<';  else if(b=='#')  f='=';  }  return f;  }  DElemtype Operate( DElemtype d, char e,DElemtype f)  {  switch(e)  {  case '+' :return d+f;  case '-' :return d-f;  case '\*' :return d\*f;  case '/' :return d/f;  case '^' :return pow(d,f);  default :printf("Error!\n");return 0;  }  }  //判断读入字符是否为运算符  int In(char e)  {  if(e=='+'||e=='-'||e=='\*'||e=='/'||e=='('||e==')'||e=='#'||e=='^')  return 1;//是  else  return 0; //不是  }  //计算表达式值  DElemtype EvaluateExpression()  {  SqStack1 OPTR;  SqStack2 OPND;  RElemtype ch,theta,x;  DElemtype a,b;    OPND\_InitStack(&OPND);//寄存操作数和运算结果  OPTR\_InitStack(&OPTR);//寄存运算符    OPTR\_Push(&OPTR,'#');    ch=getchar();  while(ch!='#'||OPTR\_GetTop(&OPTR)!='#')  {  if(!In(ch))  {  OPND\_Push(&OPND,ch-'0');  ch=getchar();  }  else  {  switch(Precede(OPTR\_GetTop(&OPTR),ch))  {  case '<':  OPTR\_Push(&OPTR,ch);  ch=getchar();  break;  case '>':  OPTR\_Pop(&OPTR,&theta);  OPND\_Pop(&OPND,&b);  OPND\_Pop(&OPND,&a);  OPND\_Push(&OPND,Operate(a,theta,b));  break;  case '=':  OPTR\_Pop(&OPTR,&x);  ch=getchar();  break;  }  }  }  return OPND\_GetTop(&OPND);  }   1. **调试分析** 2. 调试过程中遇到的问题是如何解决的以及对设计与实现的回顾讨论和分析   运算结果出错，经检查是符号的优先级未设置好   1. 算法的时空分析，改进设想   该算法比较高效，之后会尝试逆波兰算法对比分析   1. **测试数据与结果** |