四川大学

数据结构与算法分析实验报告

实验名称 ：带括号的算术表达式求值

指导老师 ：\_\_\_\_\_\_\_\_孙界平\_\_\_\_\_\_\_\_

学 院 ：\_\_\_\_\_\_\_软件学院\_\_\_\_\_\_\_

专 业 ：\_\_\_\_\_\_\_软件工程\_\_\_\_\_\_\_

姓 名 ：\_\_\_\_\_\_\_\_马 健\_\_\_\_\_\_\_\_

学 号 ：\_\_\_\_\_2013141463026\_\_\_\_

班 级 ：\_\_\_\_\_\_\_\_5 班\_\_\_\_\_\_\_\_

日 期 ：\_\_\_2014年10月24日\_\_\_

目录

[四川大学 1](#_Toc463527519)

[数据结构与算法分析实验报告 1](#_Toc463527520)

[目录 2](#_Toc463527521)

[一、实验题目: 2](#_Toc463527522)

[二、实验目的和要求: 2](#_Toc463527523)

[三、实验的环境: 3](#_Toc463527524)

[四、算法描述: 3](#_Toc463527525)

[五、源程序清单 5](#_Toc463527526)

[六、运行结果 5](#_Toc463527527)

[七、实验运行情况分析 7](#_Toc463527528)

[附录(源程序代码) 8](#_Toc463527529)

一、实验题目:

* 带括号的算术表达式求值

二、实验目的和要求:

* 采用算符优先数算法,能正确求值表达式;
* 熟练掌握栈的应用;
* 熟练掌握计算机系统的基本操作方法,了解如何编辑、编译、链接和运行一个C程序;
* 上机调试程序,掌握查错、排错使程序能正确运行。

三、实验的环境:

* 硬件环境:

联想 笔记本电脑

* 软件环境:

操作系统: windows 7 旗舰版

编译软件:Visual C++ 6.0

四、算法描述:

* 程序框图

否

开始

判断表达式是否正确

从键盘读入算术表

达式存入单链表中

是

用栈计算显示算数表达式

退出程序

是

是否继续计算？

显示错误信息

否

判断表达式是否正确

* 文字解释:

1. 用户从键盘读入算术中缀表达式，以”=”结尾;
2. 程序判断用户输入表达式是否正确;
3. 若表达式正确，则用栈计算算术表达式;
4. 打印输出计算过程和最终结果;
5. 程序询问用户是否继续计算;
6. 若继续，则执行第1步;若否定，则退出程序
7. 若表达式错误，则打印错误信息，提示用户重新输入
8. 返回第1步;

* 函数及结构说明:
* 结构体:

1) 存储算数表达式的单链表:

struct Expression{

char sign;

struct Expression \*next;

};

2) 操作符和操作数栈:

typedef struct{

char \*top;

char \*bottom;

int stack\_size;

}Stack;

* 构造函数:

1) 栈相关操作函数:

① 初始化空栈 int StackCreate(Stack \*s);

② 入栈操作 int PUSH(Stack \*s, char c);

③ 出栈操作 int POP(Stack \*s, char c);

④ 获取栈顶元素 char GetTop(Stack \*s);

2) 计算相关操作函数:

①利用栈计算算术表达式 int Calculate(struct Expression \*exp);

②子式的值的计算 int Count(char num1, char sign, char num2);

③判断字符是否为运算符 int IsOpOrNum(char c);

④判断运算符优先级 char JudgeLevel(char c1, char c2);

⑤判断表达式正确性 int IsExpresiion(struct Expression \*exp);

⑥输出计算结果 void PrintResult(struct Expression \*exp,int result);

3) 算术表达式输入函数:

①键盘读入存于单链表 struct Expression \*GetExp();

* 构造操作符优先级表

符号2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 符号1  比较 | + | - | \* | / | ( | ) | =(#) |
| + | > | > | < | < | < | > | > |
| - | > | > | < | < | < | > | > |
| \* | > | > | > | > | < | > | > |
| / | > | > | > | > | < | > | > |
| ( | < | < | < | < | < | = |  |
| ) | > | > | > | > |  | > | > |
| =(#) | < | < | < | < | < |  | = |

五、源程序清单

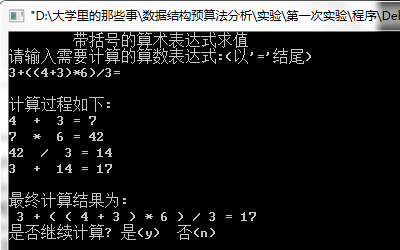
* 见附录

六、运行结果

* 测试表
* 本次测试采用用户从键盘输入算数表达式，共进行16组测试

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 测试功能 | 测试内容 | 输入项 | 预期输出 | 实际输出 | 结果 |
| 1 | 基本计算操作 | 单括号运算 | 2\*(3+4)+5\*3= | 29 | 29 | 通过 |
| 2 | 基本计算操作 | 多括号运算 | 3+((4+3)\*6)/3= | 17 | 17 | 通过 |
| 3 | 基本计算操作 | 不能除整运算 | 6\*(2+(3/2))= | 21 | 18 | 有误差 |
| 4 | 基本计算操作 | 不能除整运算 | (1+2+3)/4= | 1.5 | 1 | 有误差 |
| 5 | 式子正误判断 | 括号不匹配 | 1+(1+3\*2= | 输出错误信息 | 输出错误信息 | 通过 |
| 6 | 式子正误判断 | 计算符多余 | 1++2\*3+6= | 输出错误信息 | 输出错误信息 | 通过 |
| 7 | 式子正误判断 | 含非计算符 | x+y\*z+w= | 输出错误信息 | 输出错误信息 | 通过 |
| 8 | 式子正误判断 | 未输入”=” | 1+2\*3 | 7 | 7 | 通过 |
| 9 | 容错能力 | 除数为0 | 2+3/0= | 输出除数为0 | 输出除数为0 | 通过 |
| 10 | 容错能力 | 自动去空格 | 1+ 2 \* ( 3 + 4)= | 15 | 15 | 通过 |
| 11 | 容错能力 | “=”输为”#” | 2\*(3+2)# | 10 | 10 | 通过 |
| 12 | 拓展功能 | 多位正数计算 | 10\*(2+13)= | 150 | 输出错误信息 | 未通过 |
| 13 | 拓展功能 | 负数计算 | 4+3\*(-4)+5= | -3 | 输出错误信息 | 未通过 |
| 15 | 拓展功能 | 小数计算 | 1+10\*0.1/2= | 6 | 输出错误信息 | 未通过 |
| 16 | 全体测试 | 最终测试 | 2 \*(4 –(3+3) )+3 # | -1 | -1 | 通过 |

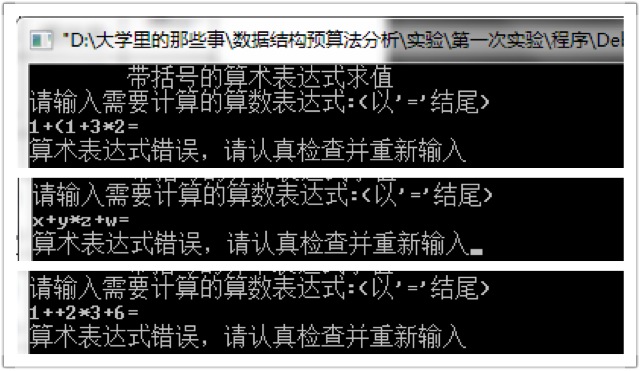
* 部分运行截图
* 基本计算操作



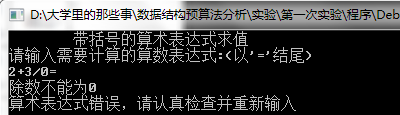
不能整除运算(3)

多括号运算(2)

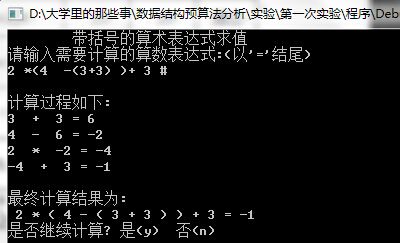
* 式子正误判断



* 容错能力



* 全体测试



七、实验运行情况分析

* 优点:
* 用户输入格式较为自由。增添了一些能想到的容错系统，如用户未输入”=”，或是用户将”=”错输成”#”，或是用户在字符间输入空格，程序都会智能识别出正确的算数表达式，计算正确答案。
* 防错报错系统较为完善。多方面考虑输入方面的错误，如括号不匹配、计算符少输多输、输入非计算符等方面均考虑，并提示用户错误信息，便于用户检查输入状况。
* 存储方式较为规范。采用单链表存储用户输入的算术表达式，更加清晰简单,头结点存储表达式中符号的个数，方便在必要的时候统计对照最终结果的正确性。
* 缺点:
* 只能实现数字之间的四则运算，不能实现其他算数功能如平方，开方等。
* 仅仅局限于个位数之间的运算，无法运算两位数或两位以上数字的运算。
* 计算数字只能为0-9的整数，无法对负数或者小数进行计算。
* 在除法中若遇到无法除尽的结果，只能保留其整数部分，造成最终结果跟预期结果存在误差
* 由于控制台系统的限制，与用户交互性较差，界面太过简单单调。
* 感受:

通过本次“带括号算数表达式的计算”实验，我不光复习了之前学习的单链表的相关知识并加以采用，而且还巩固了有关于栈的相关操作；在实现题目要求基本功能的基础上，还增加拓展了一些内容。在为期两周的实验中，从刚开始的整理思路，到接下里的编写代码，到最后的填写实验报告，都是我自己一步步完成。唯独可惜的是，由于知识掌握不够全面，导致只能实现最基本的功能，无法进行进一步的扩展和完善，致使用户输入一些运算式未得到预期的结构，这是比较遗憾的方面。

附录(源程序代码)

/\*

\*实验一：带括号的算数表达式求值

\*实现：栈

\*语言：C

\*作者：马健

\*/

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#define STACK\_SIZE\_FIRST 100 //栈的初始空间大小

#define STACK\_SIZE\_INCREASE 20 //栈的增加空间大小

/\*---------------------------------算术表达式单链表----------------------------------\*/

struct Expression{

char sign;

struct Expression \*next;

};

/\*---------------------------------操作符和操作数栈----------------------------------\*/

typedef struct{

char \*top;

char \*bottom;

int stack\_size; //栈的空间大小

}Stack;

struct Expression \*GetExp(); //键盘读入表达式存入单链表

/\*---------------------------------栈相关的操作函数-----------------------------------\*/

int StackCreate(Stack \*s); //初始化一个空栈函数

int PUSH(Stack \*s, char c); //入栈函数

int POP(Stack \*s, char c); //出栈函数

char GetTop(Stack \*s); //取出栈顶元素

/\*---------------------------------计算相关操作函数----------------------------------\*/

int Calculate(struct Expression \*exp); //带括号的算数表达式计算函数

int Count(char num1, char sign, char num2); //两个数的计算

int IsOpOrNum(char c); //判断一个字符是不是运算符

char JudgeLevel(char c1, char c2); //判断两运算符优先级

void PrintResult(struct Expression \*exp,int result); //打印最终结果

int IsExpresiion(struct Expression \*exp); //判断是否为正确的算术表达式

void main()

{

struct Expression \*head;

int Result = 0; //定义最终计算结果

int temp = 0; //定义循环参数

char select;

while(temp == 0)

{

head = GetExp(); //创建算术表达式单链表

if(IsExpresiion(head) == 0 || head == NULL) //算术表达式错误，重新输入

{

printf("算术表达式错误，请认真检查并重新输入");

getch();

system("cls");

}

else

{

Result = Calculate(head); //计算算术表达式

PrintResult(head,Result); //输出最终计算结果

printf("是否继续计算? 是(y) 否(n)\n"); //是否继续

select = getch();

if(select == 'n' || select == 'N')

temp = 1;

else

system("cls");

}

}

}

//读入算术表达式并存储于链表

struct Expression \*GetExp()

{

printf("\t带括号的算术表达式求值\n");

printf("请输入需要计算的算数表达式:(以'='结尾)\n");

char c;

int number = 0;

struct Expression \*head = NULL,\*p1,\*p2,\*first = NULL; //定义指针变量

head = (struct Expression \*)malloc(sizeof(struct Expression));

while((c = getchar()) != '\n')

{

if(c != ' ') //若出现空格,自动删除

{

p1 = (struct Expression \*)malloc(sizeof(struct Expression));

if(c == '=')

c = '#';

p1->sign = c;

if(first == NULL)

first = p1;

else

p2->next = p1;

p2 = p1;

number++;

}

}

if(p2->sign != '#') //若未输入'=',则自动补齐

{

p1 = (struct Expression \*)malloc(sizeof(struct Expression));

p1->sign = '#';

p2->next = p1;

p2 = p1;

number++;

}

head->next = first;

head->sign = number + '0'; //头结点存储表达式中字符个数

if(head->next != NULL)

p2->next = NULL;

return head;

}

//创建空栈

int StackCreate(Stack \*s)

{

s->bottom = (char \*)malloc(STACK\_SIZE\_FIRST \* sizeof(char));

if (s->bottom == NULL)

{

printf("栈初始化失败!\n");

exit(0);

}

else

{

s->top = s->bottom;

s->stack\_size = STACK\_SIZE\_FIRST;

}

return 1;

}

//入栈

int PUSH(Stack \*s, char c)

{

if(s->top - s->bottom >= STACK\_SIZE\_FIRST)

{

s->bottom = (char \*)realloc(s->bottom,(STACK\_SIZE\_FIRST+STACK\_SIZE\_INCREASE) \* sizeof(char));

if(s->bottom == NULL)

{

printf("增加栈空间失败!\n");

exit(0);

}

s->stack\_size = s->stack\_size + STACK\_SIZE\_INCREASE;

}

\*(s->top) = c; //赋值需要入栈的元素

s->top ++; //栈顶指针上移

return 1;

}

//出栈

int POP(Stack \*s, char c)

{

if(s->top == s->bottom)

{

printf("栈为空!出栈失败!\n");

exit(0);

}

else

{

c = \*(s->top);

s->top --;

}

return 1;

}

//获取栈顶元素

char GetTop(Stack \*s)

{

char c;

if(s->top == s->bottom)

printf("栈空，无法获取栈顶元素!\n");

else

{

c = \* (s->top - 1);

}

return c;

}

//计算算术表达式

int Calculate(struct Expression \*exp)

{

exp = exp->next; //取表达式开始

char OpSign =' '; //存储出栈操作符

char NumSign1=' ',NumSign2= ' '; //存储出栈数字符

char result; //存储部分运算结果

char temp = ' '; //接受出栈操作数

Stack s\_operator, s\_number;

StackCreate(&s\_operator); //创建操作符栈

StackCreate(&s\_number); //创建数字栈

PUSH(&s\_operator,'#'); //先在操作栈底压入'#'

printf("\n计算过程如下:\n");

while(exp->sign != '#' || GetTop(&s\_operator) != '#')

{

//操作数存入操作数栈

if(IsOpOrNum(exp->sign) == 0)

{

PUSH(&s\_number,exp->sign);

exp = exp->next;

}

//操作符存入操作符栈

else if(IsOpOrNum(exp->sign) == 1)

{

OpSign = GetTop(&s\_operator); //获取栈顶元素

switch(JudgeLevel(OpSign,exp->sign)) //比较栈顶元素和运算符的优先级

{

case '<':PUSH(&s\_operator,exp->sign); exp = exp->next;break;

case '=':POP(&s\_operator,OpSign); exp = exp->next; break;

case '>':

POP(&s\_operator,OpSign);

NumSign1 = GetTop(&s\_number);

POP(&s\_number,temp);

NumSign2 = GetTop(&s\_number);

POP(&s\_number,temp);

result = Count(NumSign2,OpSign,NumSign1);

PUSH(&s\_number,result);

break;

default: break;

}

}

}

result = GetTop(&s\_number); //获取最终计算结果

return result - '0';

}

//判断两个运算符的优先级

char JudgeLevel(char c1, char c2)

{

switch(c1)

{

case '+': switch(c2){

case '\*':

case '/':

case '(': return '<'; break;

default: return '>'; break;

}

break;

case '-': switch(c2){

case '\*':

case '/':

case '(': return '<'; break;

default: return '>'; break;

}

break;

case '\*': switch(c2){

case '(': return '<'; break;

default: return '>'; break;

}

break;

case '/': switch(c2){

case '(': return '<'; break;

default: return '>'; break;

}

break;

case '(': switch(c2){

case ')': return '='; break;

default: return '<'; break;

}

break;

case ')': switch(c2){

case '+': return '>'; break;

default: return '>'; break;

}

break;

case '#': switch(c2){

case '#': return '='; break;

default: return '<'; break;

}

break;

default: return '<'; break;

}

}

//计算一个符号的运算

int Count(char num1, char sign, char num2)

{

int a=0,b=0;

a = num1 - '0'; //取数字字符的值

b = num2 - '0';

int result = 0;

switch(sign)

{

case '+':result = a+b;break;

case '-':result = a-b;break;

case '\*':result = a\*b;break;

case '/':result = a/b;break;

default:break;

}

printf("%d %c %d = %d\n",a,sign,b,result); //输出计算过程

return result + '0';

}

//判断字符是运算符还是数字符

int IsOpOrNum(char c)

{

switch(c)

{

case '+':

case '-':

case '\*':

case '/':

case '(':

case ')':

case '#':

return 1; //操作符

break;

case '0':

case '1':

case '2':

case '3':

case '4':

case '5':

case '6':

case '7':

case '8':

case '9':

return 0; //操作数

break;

default:

return -1; //其他

break;

}

}

//输出最终结果

void PrintResult(struct Expression \*exp,int result)

{

printf("\n最终计算结果为:\n");

exp = exp->next;

while(exp != NULL)

{

if( exp->sign == '#')

exp->sign = '=';

printf(" %c",exp->sign);

exp = exp->next;

}

printf(" %d\n",result);

}

//判断用户输入的算术表达式是否正确

int IsExpresiion(struct Expression \*exp)

{

int parameter = 1; //定义判断表达式正确与否参数

int i=0,j=0; //左右括号数量

if(exp->sign - '0' < 4) //判断表达式字符个数是否正确

parameter = 0;

exp = exp->next;

while((parameter == 1) && (exp != NULL))

{

switch(IsOpOrNum(exp->sign))

{

case 0: //如果是数字,后必须跟操作符,且不能为左括号

exp = exp->next;

if(IsOpOrNum(exp->sign) != 1 || exp->sign == '(')

parameter = 0;

break;

case 1: //如果是操作符，则分情况

switch(exp->sign)

{

case ')': //如果是右括号，则跟计算符

i++;

exp = exp->next;

if(IsOpOrNum(exp->sign) != 1)

parameter = 0;

if(IsOpOrNum(exp->sign) == 1)

if(exp->sign == '(')

{

printf("11111\n");

parameter = 0;

}

break;

case '+': //如果是计算符，则为右括号或数字

case '-':

case '\*':

exp = exp->next;

if(exp->sign == '(')

parameter = 1;

else if(IsOpOrNum(exp->sign) == 0)

parameter = 1;

else

parameter = 0;

break;

case '/':

exp = exp->next;

if(exp->sign == '(')

parameter = 1;

else if(IsOpOrNum(exp->sign) == 0)

{

if(exp->sign == '0') //除数不能为0

{

printf("除数不能为0\n");

parameter = 0;

}

}

else

parameter = 0;

break;

case '(': //若为左括号，则为数字

j++;

exp = exp->next;

if(IsOpOrNum(exp->sign) != 0)

if(exp->sign != '(')

parameter = 0;

break;

case '#': //若为等号，则等号为最后一个字符

if(exp->next != NULL)

parameter = 0;

exp = exp->next;

break;

}

break;

case -1: //如果是非计算符和数字，则错误

parameter = 0;

break;

}

}

if(i != j)

parameter = 0;

return parameter;

}