

# 浙江大学实验报告

专业: 混合班  
姓名: 徐圣泽  
学号: 3190102721  
日期: 2020.5.9

课程名称: C 程序设计专题 指导老师: 翁恺 实验名称: 表达式计算

## 一、实验要求

编写一个程序，该程序读取一行中的表达式作为输入并打印出结果。表达式中仅允许使用以下整数和运算符：

+ - \* / % ( )

输入格式 | 输入样例

A line of expression. | (2+32)/2-6

输出格式 | 输出样例

The result. | 11

## 二、实验思路与过程描述

在这个程序中，主要部分是三个功能函数：

```
int addup(); //处理加减运算
int plus(); //处理乘除取余运算
int brackets(); //括号运算
```

首先需要在 main 函数和三个功能函数外面定义一个全局变量 char op，用来记录每次读入的字符，判断字符是运算符或是数字，并进行相应的操作。

## 三、实验代码解释

### (一) main 函数

```
int main(){
    int sum=0;
    op=getchar();
    sum=addup();
    printf("%d",sum);
}
```

定义一个整型变量 sum 用来存储表达式运算的结果。首先用 op=getchar(); 读取一个字符并直接调用 addup 函数得到结果。本程序的主要计算过程都在三个功能函数之间的互相调用中实现，故 main 函数十分简短。

### (二) 功能函数

### 3.1 加减运算

```
int addup(){
    int sum=0;
    sum=plus();
    while(op=='+'||op=='-'){
        switch(op){
            case '+': op=getchar(); sum+=plus(); break;
            case '-': op=getchar(); sum-=plus(); break;
        }
    }
    return sum;
}
```

在 `addup` 函数中，判断运算符后调用 `plus` 函数（因为乘除和取余的优先级都比加减法高），在求得乘除运算式的结果后，判断此时的符号是加或减，进而再进行加减操作。在这步加减操作中，对运算号后面的式子再次调用 `plus` 函数，同时需要在调用前再次用 `op=getchar()` 读取下一个字符。

上述过程大致可以用 `expression(1)±expression(2)` 的式子来表现，在求得加减号左右两边的值后再进行最简单的加减操作求得运算的结果，即此函数的 `return` 返回值。

### 3.2 乘除取余运算

```
int plus(){
    int sum=0;
    sum=brackets();
    while(op=='*'||op=='/'||op=='%'){
        switch(op){
            case '*': op=getchar(); sum*=brackets(); break;
            case '/': op=getchar(); sum/=brackets(); break;
            case '%': op=getchar(); sum%=brackets(); break;
        }
    }
    return sum;
}
```

乘除法和取余这三种运算中，运算号左右或者是一个带括号的表达式，或者是一个确切的数字。

因此在 `plus` 函数中，首先定义一个 `sum` 变量存储调用 `brackets` 函数得到运算符左边的表达式的值（认为一个数字也是一个表达式），再判断运算符符号，并在各情况中再次调用 `brackets` 函数得到运算符右边的表达式的值，将两个值作运算即得到了运算结果，即此函数的 `return` 返回值。

### 3.3 括号运算

```
int brackets(){
    int sum=0;
    if(op=='('){
        op=getchar();
        sum=addup();
        if(op==')') op=getchar();
    }
    else if((op>='0'&&op<='9')||op=='-'){
        if(op=='-'){
            op=getchar();
            while(op>='0'&&op<='9'){
                sum=sum*10+op-'0';
            }
        }
    }
}
```

```

        op=getchar();
    }
    sum=-1*sum;
}else{
    while(op>='0'&&op<='9'){
        sum=sum*10+op-'0';
        op=getchar();
    }
}
return sum;
}

```

`brackets` 这个函数的功能，主要是计算乘除取余号左右的表达式的值。这个函数也是递归的具体体现。


当读取的字符为 `(` 时，需要计算这个括号中的值，此时调用 `addup` 函数即可，并且在计算结束后找到 `)` 并读取后面的字符。

当读取的字符不是括号时，由题意，乘除号两边的负数不需要带括号，故此时又分为两种情况：正数和负数。

①正数。因为本题中是逐个输入的字符，故需要将连续的几个代表数字的字符转化为整型变量。利用 `op-'0'` 得到该字符代表的数字，此时定义的整型变量 `sum` 的初始值为 `0`，故每读入一个字符，利用 `sum*10+op-'0'` 可得到多位的整数。

②负数。此时读入的字符是 `-`，故我们只要类似地重复上述操作，再将得到的整数乘 `-1` 即得到了我们想要的数，只不过在重复之前需要再进行一步 `op=getchar()`。


### (三) 结果样例

 C:\Users\DELL\Desktop\P3专题\简单计算.exe

```

2+1-5
-2
-----
Process exited after 3.999 seconds with return value 0
请按任意键继续. . .
    
```


---

 C:\Users\DELL\Desktop\P3专题\简单计算.exe

```

3*12/6
6
-----
Process exited after 8.134 seconds with return value 0
请按任意键继续. . .
    
```

---

 C:\Users\DELL\Desktop\P3专题\简单计算.exe

```

(3+2*7)*(29+17*-1)+32*(-1+4*3+5*-1)/(2-1)
396
-----
Process exited after 104.6 seconds with return value 0
请按任意键继续. . .
    
```

## 四、实验体会和心得

在本次专题实验中，主要利用了三个函数的互相调用，其中 `addup` 函数调用 `plus` 函数，`plus` 函数调用 `brackets` 函数，而 `brackets` 函数再次调用 `addup` 函数，这是整个实验递归的主要体现之处。

如果根据运算符之间的优先级理清了各个函数之间的关系，整个实验的思路其实是比较顺畅的。根据优先级的关系，加减法优先级最低，故调用优先级更高的乘除取余函数，再在此函数中调用优先级最高的括号运算函数，最后得到了表达式的值。

在推荐作业中，我发现第三个函数中 `else if((op>='0'&&op<='9')||op=='-')` 这个条件中复杂的代码部分可以用很简单的 `ungetc` 函数代替，具体实现方式为 `ungetc(op,stdin);`  
`scanf("%d",&sum);` 直接省去了复杂的判断和将字符转化成数字的部分。

C 库函数 `int ungetc(int char, FILE *stream)` 把字符 `char`（一个无符号字符）推入到指定的流 `stream` 中，以便它是下一个被读取到的字符。其中 `char` 是要被推入的字符，该字符以其对应的 `int` 值进行传递。`stream` 是指向 FILE 对象的指针，该 FILE 对象标识了输入流。如果成功，则返回被推入的字符，否则返回 `EOF`，且流 `stream` 保持不变。

通过本次实验对递归和迭代有了进一步的认识，也能更熟练地在各个函数之间进行反复操作和调用。