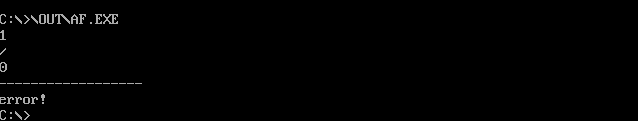
# C语言综合研究与高强度程序设计训练12

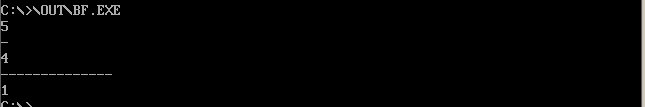
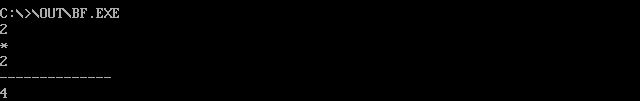
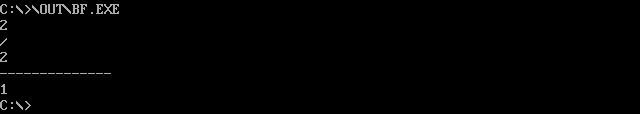
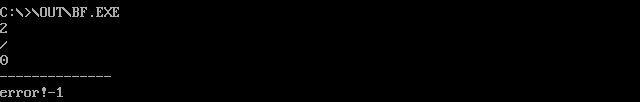
**1**

1. 写一个程序ac，功能如下:  
   可以接收用户依次输入:一个字符串a、一个字符ch、一个字符串b这些内容分别描述一个整型数据（如:10）、一个运算符（如:＋）、一整型数据（如:20），然后根据字符ch所描述的运算符如:“＋”-，对字符串a和b的描述数据进行运算，将结果显示出来  
   对程序a.c进行研究，然后在ac的逻辑的基础上将这个程序扩充为识别“＊“”，可以乘、除运算的程序af.c

* a.c
* main() {  
   char a[20];  
   char b[20];  
   char ch;  
    
   gets(a);  
   printf("%c\n", ch = getch());  
   gets(b);  
    
   if (ch != '+' && ch != '-') {  
   printf("error!");  
   return;  
   }  
   printf("------------------\n");  
   if (ch == '+')  
   printf("%d", atoi(a) + atoi(b));  
   if (ch == '-')  
   printf("%d", atoi(a) - atoi(b));  
  }
* af.c
* main() {  
   char a[20];  
   char b[20];  
   char ch;  
    
   gets(a);  
   printf("%c\n", ch = getch());  
   gets(b);  
    
   if (ch != '+' && ch != '-' && ch != '\*' && ch != '/') {  
   printf("error!");  
   return;  
   }  
   printf("------------------\n");  
   switch (ch) {  
   case '+':  
   printf("%d", atoi(a) + atoi(b));  
   break;  
   case '-':  
   printf("%d", atoi(a) - atoi(b));  
   break;  
   case '\*':  
   printf("%d", atoi(a) \* atoi(b));  
   break;  
   case '/':  
   if (atoi(b) == 0) {  
   printf("error!");  
   break;  
   }  
   printf("%d", atoi(a) / atoi(b));  
   break;  
    
   default:  
   break;  
   }  
  }
  + 结果验证
  + 
  + 
  + 
  + 
  + 

## 2

写一个与程序a.c功能相同的程序b.c，程序如下  
对程序b.c进行研究，对比b.c与a.c在程序设计思想上的不同然后在b.c的逻辑的基础上将这个程序扩充为识别“\*\*”、“/”，可以乘、除运算的程序b.c

* b.c
* char \*codes = "+-";  
  int add(int a, int b) { return a + b; }  
  int sub(int a, int b) { return a - b; }  
  int (\*func[2])(int, int) = {add, sub};  
  main() {  
   char a[20];  
   char b[20];  
   char ch;  
    
   int n;  
    
   gets(a);  
   printf("%c\n", ch = getch());  
   gets(b);  
    
   for (n = 0; codes[n] && codes[n] != ch; n++)  
   ;  
   if (!codes[n]) {  
   printf("error!");  
   return;  
   }  
    
   printf("--------------\n");  
    
   printf("%d", func[n](atoi(a), atoi(b)));  
  }
* 通过分析可以看到b.c通过函数指针数组来存储函数指针以此方便统一函数的逻辑性操作
* bf.c
* char \*codes = "+-\*/";  
  int add(int a, int b) { return a + b; }  
  int sub(int a, int b) { return a - b; }  
  int mul(int a, int b) { return a \* b; }  
  int div(int a, int b) {  
   if (b == 0) {  
   printf("error!");  
   return -1;  
   } else  
   return a / b;  
  }  
  int (\*func[4])(int, int) = {add, sub, mul, div};  
  main() {  
   char a[20];  
   char b[20];  
   char ch;  
    
   int n;  
    
   gets(a);  
   printf("%c\n", ch = getch());  
   gets(b);  
    
   for (n = 0; codes[n] && codes[n] != ch; n++)  
   ;  
   if (!codes[n]) {  
   printf("error!");  
   return;  
   }  
    
   printf("--------------\n");  
    
   printf("%d", func[n](atoi(a), atoi(b)));  
  }
  + 结果测试
    - 加法
    - 
    - 减法
    - 
    - 乘法
    - 
    - 除法
    - 
    - 

## 3

思考以下几个问题:

b.c中，程序的共性实现在什么函数里？

个性实现在什么函数里？

你认为b.c的程序设计思想的普遍意义是什么？

* 程序的共性实现在main函数中.
* 个性实现在各个子函数（add，sub，mul，div）中
* 简化对细节的操作，减少代码量提高编写质量