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;
    }
```

```
14      printf("------------------\n");
15      switch (ch) {
16      case '+':
17          printf("%d", atoi(a) + atoi(b));
18          break;
19      case '-':
20          printf("%d", atoi(a) - atoi(b));
21          break;
22      case '*':
23          printf("%d", atoi(a) * atoi(b));
24          break;
25      case '/':
26          if (atoi(b) == 0) {
27              printf("error!");
28              break;
29          }
30          printf("%d", atoi(a) / atoi(b));
31          break;
32
33      default:
34          break;
35      }
36  }
```

- 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`

```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`

```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;

```

```
18      int n;
19
20      gets(a);
21      printf("%c\n", ch = getch());
22      gets(b);
23
24      for (n = 0; codes[n] && codes[n] != ch; n++)
25          ;
26      if (!codes[n]) {
27          printf("error!");
28          return;
29      }
30
31      printf("--------------\n");
32
33      printf("%d", func[n](atoi(a), atoi(b)));
34  }
```

- 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