**c语言综合研究十五报告**

1. **研究过程展示**

**1>.程序1部分：**

将综合研究15中的main.c程序与研究12和研究13的作对比,可以发现非常明显的差别有：

“extern char \* code;”

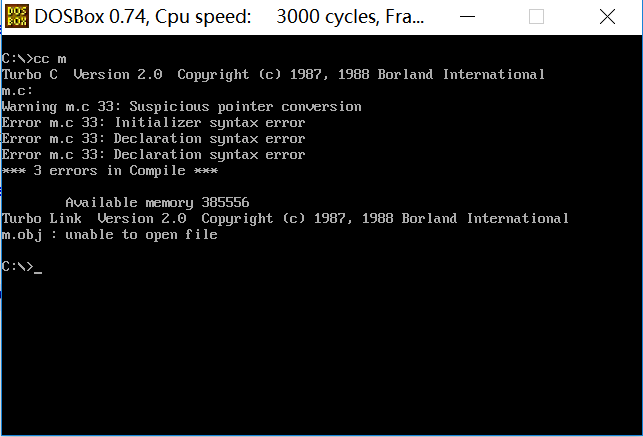
“extern int (\*\*f)(int,int);”

其中extern这个外部声明也就提醒了我们在m.c中应该存放什么。但是这里还有一个与研究12的b.c程序不同的地方：

这里的f声明为一个指向函数指针的指针，而研究12中存在的是一个函数指针数组。

开始，在m.c中直接“int (\*\*f)(int,int) = {add,sub,mul,div};”

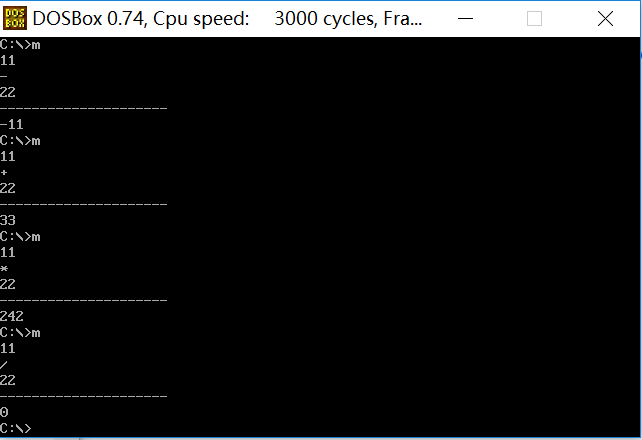
结果编译出现错误：



这样不可以，那么就把两个都用一下：

“int (\*func[4])(int,int) = {add,sub,mul,div};”

“int (\*\*f)(int,int) = func;”



程序m.c:

char\* code = "+-\*/";

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)

{

return a/b;

}

int (\*func[])(int,int) = {add,sub,mul,div};

int (\*\*f)(int,int) = func;

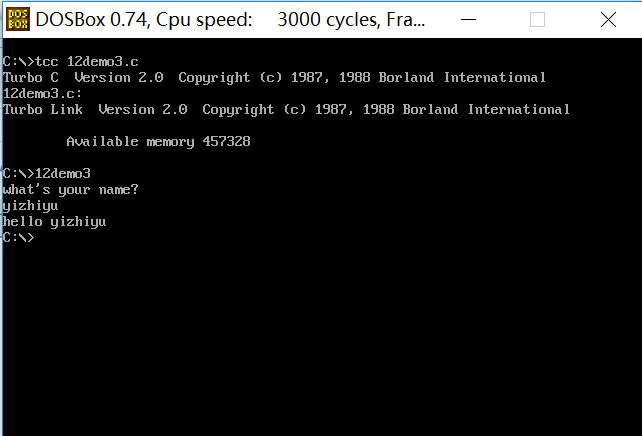
开发工具cc默认链接main.obj，而main.obj中的内容是由我们自定义的，所以就像是此处实现了一个自定义的封装库，封装的不仅有共性元素，还有main()函数，而研究13中将共性元素都放在\*.h文件中，将个性元素放在\*.c文件中，拓展个性时，只需要修改\*.c文件中的内容，而main（）函数也在其中。与之不同，研究15 将main（）函数封装起来，就好像拿一个黑盒子将其套了起来，如果我们不是亲自参与这个cc工具的制作过程，可能main.obj文件中的内容是一丁点都不知道的，但是从分工合作的角度，main.obj文件的实现是没必要让编写m.c文件的人知道的，而且，如果编写m.c文件的工作人员可以根本不用知道一点main.obj的事情，就可以把m.c的事情做好，进而实现整个程序的功能，这样的封装就是省事的，分工性很强，个人负责个人的事，有点像车间的流水线，关心的事情少了，抗干扰性也强，或许，万一整体功能实现不了，这样也很容易揪出问题到底出现在哪一环节。

1. **已思考研究并已解决问题汇总**
2. 之前对scanf研究还不够，在这里补充一些。

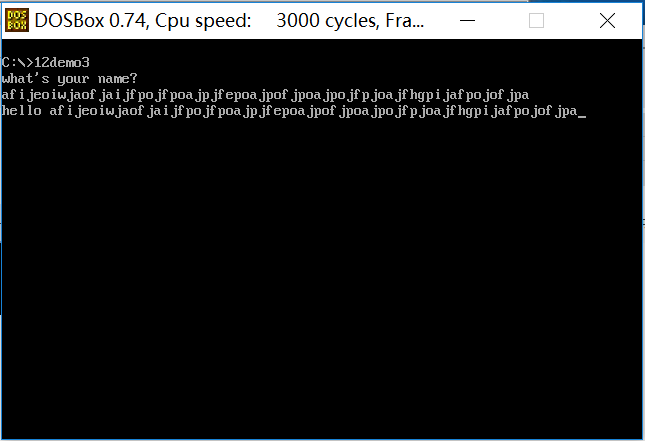
Scanf()函数的坏毛病：

定义一个“char name[5];”

输入大于它长度的字符串“yizhiyu”:



正常显示。如果名字再长点呢？



程序就奔溃了。

这是为什么呢？

此处的程序：

main()

{

char name[5];

printf("what's your name?\n");

scanf("%s",name);

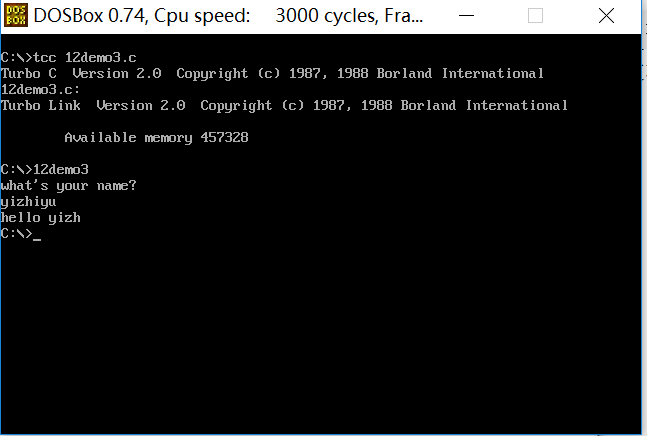
printf("hello %s",name);

}

用户可以输入远远超出程序空间的数据，多余的数据会写到计算机还没有分配好的存储器中。如果运气好，数据不但可以保存，而且不会有任何问题。但缓冲区溢出很有可能导致程序出错，这种情况通常称为断错误或abort trap,这样程序就会崩溃。

但是scanf也有自己的对策，那就是利用自己的格式化输入的特点：

定义的char name[5]，因为字符串末尾需要“\0”，那就用 %4s：



此处程序：

main()

{

char name[5];

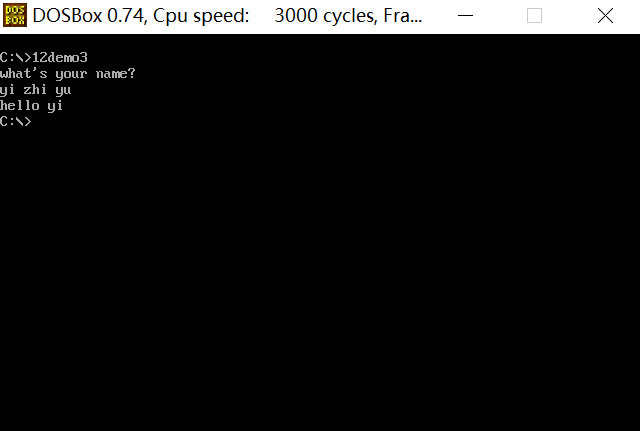
printf("what's your name?\n");

scanf("%4s",name);

printf("hello %s",name);

}

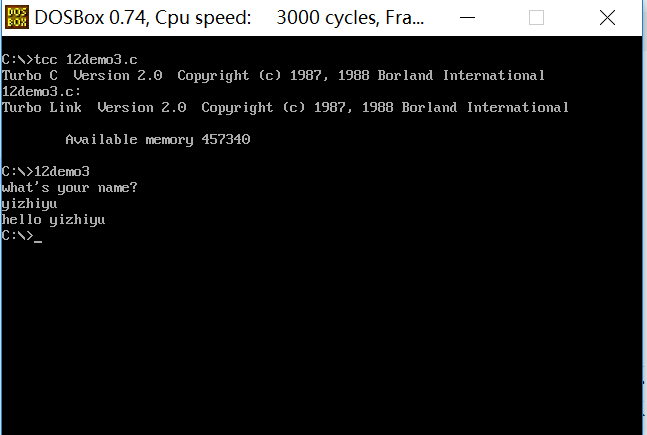
但是这里5啊4的，难道我们每次用，都得记着减一吗？这是它的一个限制。再试试加空格：



这里虽然格式仍是“%4s”，但是，输出效果只是“yi”，不是人们预想的“yi z”;看来用scanf（）函数接收字符串输入时，对于空格，它并没有像接收普通字符接收它，反而对它有点特殊对待。

那么，之前用到的gets()函数呢？

对比着看一下：



此处程序：

main()

{

char name[5];

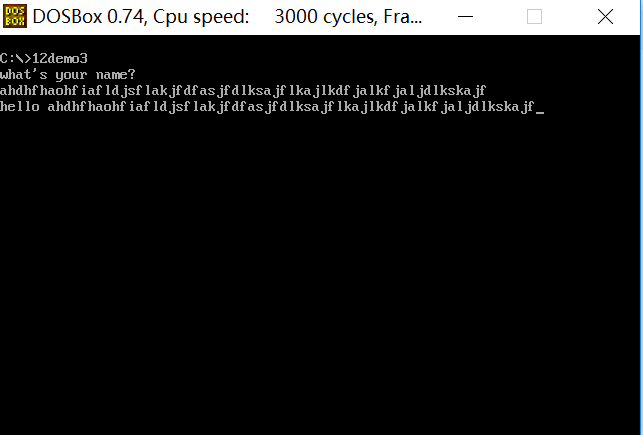
printf("what's your name?\n");

gets(name);

printf("hello %s",name);

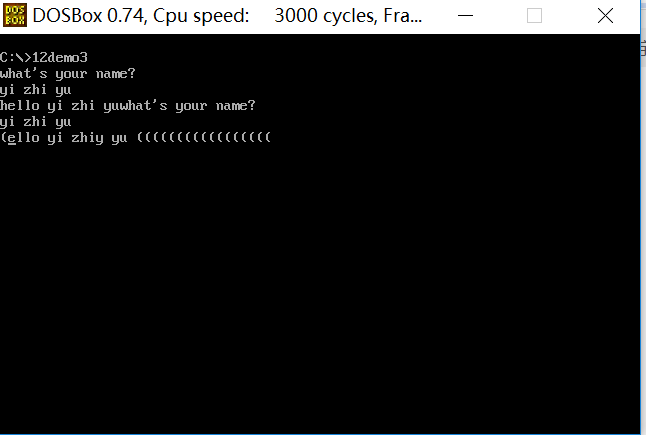
}

再长一点：



得，gets()函数遇到这也崩溃了。

那么空格呢，gets（）咋处理？

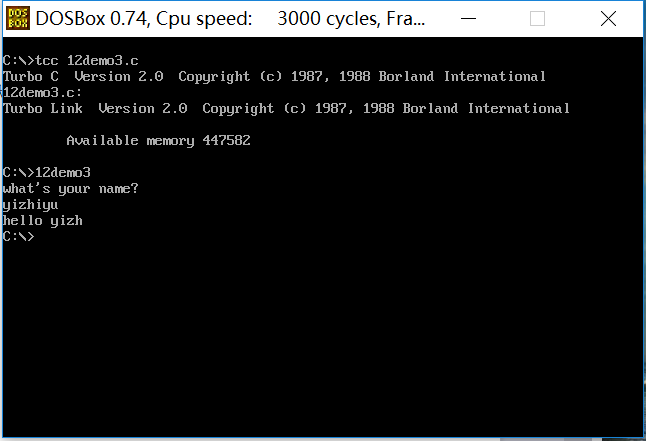


好吧，我得承认，gets()函数你真是吓到我了，你可真是太危险了。有的书上怪不得挂出了你的红牌警告！

哎呀，就不能让我尝一回“小黄鱼炒熊掌”吗？

不过，幸好我还是在书上找到了fgets()函数：

Fgets()函数接收char指针，由我们给出最大长度：

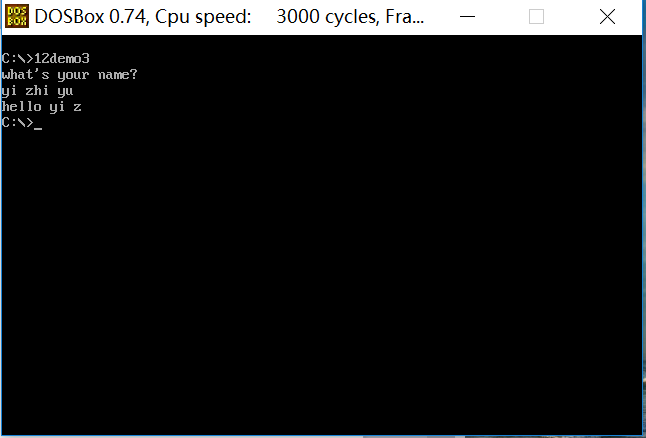


再长一点：



哇，唯一一个没有崩溃的小伙子。

空格呢？



哎呀，fgets()函数，我好中意你呀，而且你都不用像scanf（）函数限制输入字符的时候还得改变和计算什么“%4s”了。很方便，可是空格都被视为字符来对待了，也就是你好像一次只能接收一个字符串了 ，不过这个可以理解。

不过这里fgets()函数是配合sizeof关键字使用的，sizeof是在编译的时候算出具体大小，而sizeof（）内部的计算对象是数组，又有点想起宣讲会上有同学曾说过数组作为函数参数，退化为指针的现象。不过fgets()函数第二个参数也是可以给出具体的数字的，总之这一点还是比较可靠的，关键就看编程人员怎么用了。

贴一段有趣的scanf（）函数与fgets（）函数的对比表格：



1. 对数组和指针一定要有关键的区分点。

计算机为数组分配存储空间，但没有为数组变量分配空间；

计算机为指针是分配了空间的。

由于数组变量没有分配空间，而且它其实在编译的时候就被替换为具体的地址，所以它也无法像一个正经的指针那样指向任何地方。

但是应用方面我觉得最经典的就是，对于字符串常量了：

如果“char\* f = ”hello”;”之后像“f[1] = f[0];”这样想要改变值的情况时，就会出现错误；

如果 “char f1[] = ”hello”;”，之后再改变，就安然无恙。

因为这里位于栈区的f想要改变位于常量区字符串常量，而位于栈区f1想要改变的是位于栈区的字符串常量的副本。

1. **已思考研究并未解决问题汇总**

1.\*f和f[]能够随意互相替换,\*\*f和\*f[]在这里是不一样的，两者发生了什么？那么\*\*\*f和\*\*f[]呢。。。。。。

1. **研究感想（心得体会）**

感觉封装真的是很重要啊，封装做好真的是对自己对整个程序都有很多不同的感受，就好像是让程序整个重新淬炼了一遍筋骨一般，不过感觉自己还需要好好学学。