

**课程 实 验 报 告**

**课程名称： C语言程序设计课程实验**

**更改部分用红字表示**

**专业班级： 自实1901**

**学 号： U201916457**

**姓 名： 张皓然**

**指导教师：**

**报告日期： 2020年4月13日**

**人工智能与自动化学院**

# 实验1 用2个实例详细说明，在调试过程中，如何分析查找程序设计中逻辑错误的过程（F4，F7，F8，Alt\_F4的使用）

### 实验目的

（1）复习F4，F7，F8，Alt\_F4的使用。

（2）熟悉分析程序中逻辑错误的方法。

### 实验内容

（1）打开TEST1(QQ群)，编译运行并分析改正错误。

（2）打开TEST2(QQ群)，编译运行并分析改正错误。

### 程序

TEST1（三个浮点数比大小）

1. #include <stdio.h>
2. **void** main()
3. {
4. **float** x,y,z;
5. **int** t;
7. printf("please input three float datas:\n");
8. scanf("%f%f%f",x,y,z);
10. **if**(x>y)
11. {
12. t=x;
13. x=y;
14. y=t;
15. }
16. **if**(x>z)
17. {
18. t=x;
19. x=z;
20. z=t;
21. }
22. **if**(y>z)
23. {
24. t=y;
25. y=z;
26. z=t;
27. }
29. printf("%f %f %f",x,y,z);
30. }

TEST2（输入字符，统计各类字符出现的次数）

1. #include <stdio.h>
2. **void** main()
3. {
4. **char** cc;
5. **int** a,b,c,d;
7. printf("please input a string:\n");
8. **while**((cc=getchar())!='\n')
9. {
10. **if**((cc>='A'&&cc<='Z')||(cc>='a'&&cc<='z'))
11. a++;
12. **else** **if**(cc==' ')
13. b++;
14. **else** **if**(cc>=0&&cc<=9)
15. c++;
16. **else**
17. d++;
18. }
19. printf("letter:%d,space:%d,digit:%d,other letter:%d\n",a,b,c,d);
20. }

### TEST1实验过程

1.运行程序并查看结果

在BC中打开TEST1，按Ctrl\_F9编译运行，编译未报错，可是按Alt\_F5打开用户界面后发现程序并未正常运行，程序界面显示:

please input three float datas:

scanf : floating point formats not linked

Abnormal program termination

2.分析程序的逻辑错误

解读输出的错误信息，理解用户界面的提示。用户界面第一行的输出的句子是没问题的，第二行的大致意思是“scanf：浮点型数据未连接”，第三行的大致意思是“不正常的程序结束”。

依此，我们可以大致推测出程序的问题可能与scanf有关。

3.进行调试

首先将光标移动到第8行（scanf所在的一行），按F4，使程序运行到该行。

然后将光标分别移动到x,y,z（第4行），按Alt\_F4,从而显示x,y,z的值。由于这三个变量没有被人为赋予初值，所以显示出的值都是内存中随机的值。

第三步，按F8让程序运行到下一行，发现程序运行第8行之后直接就结束了。

发现scanf中未加地址符&,于是在源程序中添加&。

再次编译运行，输入数据

2.5 3.4 0.9

输出结果为

0.900000 2.000000 3.000000

再次出现问题，于是我们进行第二轮调试。重复上述类似的操作，先用F4让程序运行到第10行，然后移动光标配合Alt\_F4,观察x,y,z,t的值。

按F8逐步运行，观察各个变量值的变化。函数进入第16行的if语句内部，if内第一步是让t=x，我们期望t得到2.5的值，但却发现执行第18行的赋值操作后t=2.

出现错误，进行分析。发现这原来是因为t为整型变量，所以在赋值时直接抹去了浮点型变量的小数位。于是我们将第五行改为

float t;

再次编译运行，输入数据：

2.5 3.4 0.9

打开用户界面，发现程序运行正常。

### TEST2实验过程

1.运行程序并查看结果

编译运行TEST2，输入数据:

abc123,./

打开用户界面观察输出的结果：

letter:883,space:886,digit:264,other letter:1501

显然有问题，进入调试环节。

2.调试（与TEST1的实验过程类似）

按F4运行到第8行，（while所在行）。用Alt\_F4观察变量a,b,c,d以及字符型变量cc的值。

发现a,b,c,d都是较大的随机数，进而明确程序的错误之一是：没有对a,b,c,d进行初始化。于是将第5行改为

int a=0,b=0,c=0,d=0;

尝试重新编译运行程序，输入数据：

abc123,./

输出结果为：

letter:3,space:0,digit:0,other letter:6

分析结果，按理说应该是digit:3,other letter:3,如今的结果说明数字字符被误判为其他型的字符了。即数字字符的判断出现了问题。

于是开启第二轮调试，与第一轮调试类似，不过这次让程序直接运行到第14行（数字字符在此行开始进行判断），同时观察c和cc的值。

第一次运行到14行时，cc的值为’1’ 49,c的值为1.

按F8让程序逐行运行，发现程序运行至第17行，即程序判断此时c为other letter。

分析后发现，‘1’作为字符的ASCII码值为49，而在14行判断数字型字符时，所用的条件是

else if(cc>=0&&cc<=9)

这是将字符的1和1对应的ASCII码搞错了，于是可将程序第14行改为

else if(cc>=’0’&&cc<=’9’)

尝试再次编译运行，输入数据

abc123,./

输出结果：

letter:3,space:0,digit:3,other letter:3

程序运行成功。

### 心得与体会

关于两个程序的具体错误，TEST1中的错误一个是scanf内未加地址符&，一个是数据类型的选择错误。TEST2中的错误一个是未对变量赋初值，一个是混淆了数字字符的ASCII码值与数字本身值得关系。这都是我们也非常容易犯的错误，在实际编写程序时一定要留意。同时注意到，scanf中所需的变量地址实际上是一个整数，所以TEST1中如果x,y,z是int型变量的话，可以编译通过，但由于程序把x,y,z的值看成了他们的地址，所以依旧无法得到正确的结果。

关于如何分析程序中的逻辑错误。首先我们要分析运行得到的错误信息，我们期望程序输出什么？程序实际输出了什么？通过比较这两个问题的答案，我们就可以大致推测出程序的哪一部分出现了错误。然后再针对可能出错的部分进行单步调试，同时比较各个变量的预期值与实际值之间的差别，以此来确定问题所在。

在验证程序是否错误时，应当确保测试集的完备性，同时应当将程序分块化设计，在调试时，逐块分析。

# 实验2 用一个实例，熟悉并说明函数的定义，函数的原型，以及函数调用时实参与形参的关系

### 实验目的

（1）熟悉并说明函数的定义，函数的原型。

（2）熟悉并说明函数调用时实参与形参的关系。

### 实验内容

编写一个简单的程序，用这个程序解释“函数的定义”“函数的原型”，以及实参和形参的关系。

### 程序

1. #include<stdio.h>
2. **void** funA(**int** a,**int** b,**int** c);
3. **void** main()
4. {
5. **void** funB();
6. **int** a=1,b=2,c=3;
7. funA(c,b,a);
8. funB();
9. }
10. **void** funA(**int** a,**int** b,**int** c)
11. {
12. printf("first is %d\nsecond is %d\nthird is %d\n",a,b,c);
13. }
14. **void** funB()
15. {
16. printf("funB\n");
17. }

### 实验过程

编写，编译运行程序，得到用户界面的结果为：

first is 3

second is 2

third is 1

funB

### 分析

本实验中用到的程序一共定义了3个函数，main函数，funA函数和funB函数。就“函数的定义”而言，3-9行是main函数的函数定义，10-13行是funA函数的函数定义，14-17行是funB函数的函数定义。函数定义就是告诉计算机，这个函数具体有什么样的含义，当遇到这个函数时，计算机应该进行怎样的操作。以funA函数为例，函数的定义包括两部分：函数头（第10行），函数体（11-13行）。函数头规定了该函数的存储类型，包括extern和static，当存储类型缺省时，定义函数为外部函数，可以被任何源文件调用。此外函数头还规定了函数的数据类型，这个数据类型指的是函数返回值的数据类型，如果不需要返回值时，需要写上void，如果什么都不写，默认函数返回值为int型。函数头的括号内是函数的形式参数表，这个参数表规定了函数运行时从外部获得的参数的顺序，数据类型，以及参数的个数。当参数的个数不确定时，可以用类似void funA(int a,…)来表示。

至于“函数的原型”，笔者认为其意义在于提前给计算机“打个招呼”，以funB为例，当函数的使用先于函数的定义时，计算机可以通过函数原型提前得知这个函数使用时具体的格式，进而正常运行。同时需要注意，就像本实验中程序显示的那样，函数原型声明可以放在函数外，也可以放在main函数内部。当放在函数外时，声明这个函数是外部函数；当放在函数内时，声明这个函数是局部函数。

对于形式参数与实际参数的区分，以funA为例。

第10行void funA(int a,int b,int c)，a,b,c这三个就是形式参数。

第7行funA(c,b,a)，c,b,a这三个变量所代表的值就是实际参数。

可以理解为：实参的数值（3,2,1），按函数中参数的顺序（3在第一位，2在第二位，1在第三位），依次赋给形参作为他们的初值（funA中a=3,b=2,c=1）。

### 心得与体会

一个程序要求有且仅有一个main函数，函数的定义应当注意存储的类型。在仅有一个c文件时，存储为外部函数并不会产生什么影响，但当一个工程中有多个c文件时，使用外部函数就有可能导致不同文件之间的串用，此时用static存储类型。

函数的声明要注意句尾的分号。

形参和实参一定要一一对应，形参只继承了实参的值，但却不需要考虑实参原先属于哪个变量。还有，在定义形参时应当选择合适的变量名，这样在阅读时就会方便很多。

# 实验3 用程序验证自动型变量（含寄存器变量）与静态变量（含一般全局变量）存储特点（初始化，存储记忆等）

### 实验目的

（1）区分，熟悉自动型变量（auto以及register），静态变量（一般全局变量以及static）

### 实验内容

编写程序，编译运行，并分析这几种变量的特点。

### 程序

1. #include<stdio.h>
2. #include<time.h>
3. **int** all;
4. **void** funA();
5. **void** funC();
6. **void** main()
7. {
8. auto **int** i;
9. **register** **int** j;
10. **static** **int** k;
12. printf("i=%d,j=%d,k=%d,all=%d\n",i,j,k,all);
14. **for**(i=0;i<4;i++)
15. {
16. funA();
17. }
19. **for**(j=0;j<4;j++)
20. {
21. funC();
22. }
23. }
24. **void** funA()
25. {
26. auto **int** p1=0;
27. **register** **int** p2=0;
28. p1++;
29. p2++;
30. printf("%d%d\n",p1,p2);
31. }
32. **void** funC()
33. {
34. **static** **int** q;
35. q++;
36. all++;
37. }

### 实验过程

输入，编译运行程序，得到结果：

i=852,j=846,k=0,all=0

11

11

11

11

11

22

33

44

### 分析

第一行的输出内容显示，在不人为赋初值的情况下，自动型变量和寄存器型变量的初值是随机给定的；而一般全局变量和静态变量在这种情况下的初值就是0。

之后程序执行4次funA()函数，funA()函数会定义一个auto变量和一个register变量，两个变量的初值为1，并在加1后输出两者的值。从用户界面第2行到第5行的内容可以验证，auto和register型变量的寿命与函数相同，即函数执行完毕，函数内的这两种变量也将不再被储存在相应的位置中。

然后程序执行4次funB()函数，funB()函数定义了一个static变量，然后让之前在程序开头定义的一般全局变量，以及funB()函数内定义的static变量各自加1，然后输出各自的值。从用户界面第6行到第9行的内容可以验证，一般全局变量和static型变量在程序运行的过程中会一直存在，即使funB()函数执行完了，funB()内定义的static型变量依旧储存在相应的位置。而当第2次执行funB()函数时，static int q; 实际上不再执行，即给一般全局变量和静态变量赋初值的操作只进行一次，之后这两种变量的值就会一直储存下去。

### 心得与体会

变量的存储类型有四种：auto,register,extern,static.其中auto和register算作自动型变量，auto储存在内存的动态储存区，register储存在CPU的通用寄存器中，相较于储存在内存中的变量，register型变量使用的速度会快上一些。extern和static变量储存在内存的静态储存区中。个人感觉extern就类似于函数的原型声明，用上extern，就可以使用在其他地方定义的变量了（其他文件中，或者本文件计算机尚未执行到的地方），而static变量则只能在本文件中使用，可以在函数内部定义，也可以在函数外部定义。如果一般全局变量不是在程序开头定义的，那就只能在本文件中使用。做一个表格可以让他们的关系更加清楚。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 储存类型 | 在哪定义？ | 在哪储存？ | 寿命？ | 使用范围 | 注意 |
| auto | 函数内或复合语句内 | 内存动态储存区 | 从定义开始，到函数运行结束终止。 | 本文件，局部变量 | auto可缺省 |
| register | 函数内或复合语句内 | CPU通用寄存器 | 从定义开始，到函数运行结束终止。 | 本文件，局部变量 | 1.运行速度快，但也要省着用2。不能取地址 |
| extern | 函数外 | 内存静态储存区 | 从定义开始，到程序运行结束终止。 | 本文件以及其他文件，全局变量 | 定义一般全局变量不需要加extern，引用时才需要加extern。另外，如果一般全局变量的定义不在程序开头，则该变量只能在本文件使用。 |
| static | 函数外，函数内或复合语句内 | 内存静态储存区 | 从定义开始到程序运行结束终止。 | 本文件，局部变量或者全局变量 | 可局部可全局，但仅限于本文件，默认初始化为0. |

|  |  |  |
| --- | --- | --- |
|  | 在哪？ | 注意事项 |
| 函数的定义  例如：  static int abc(int a){…} | 函数外 | 无 |
| 函数的声明  例如：  int abc(int);  **本身就不需要写存储类型！！！** | 函数内，或者函数外 | 声明决定了函数的作用域，比如，如果funA的声明在main函数里面，那么我们在定义funB的时候，就无法使用funA函数。还有一定要记得加分号 |

函数定义中的存储类型

|  |  |  |
| --- | --- | --- |
|  | 作用范围 | 注意 |
| extern | 本文件和其他文件 | 可缺省 |
| static | 仅本文件 | 无 |

# 实验4 通过例4.3，例4.9，例4.6进一步熟悉单步调试的功能键的使用，写出关键的观测点和调试过程

### 实验目的

（1）进一步熟悉单步调试的的功能键的使用。

### 实验内容

输入程序，选取关键点进行单步调试。

### 程序

例4.3

1. #include<stdio.h>
2. **int** n;
3. **void** show( );
4. **void** show( )
5. {
6. auto **int** i=3;
7. n++;
8. i++;
9. printf("input the value: n=%d  i=%d\n", n, i);
10. {
11. auto **int**  i=10;
12. i++;
13. printf("now the value i=%d \n",i);
14. }
15. printf("then the value i=%d\n",i);
16. }
17. **void** main( )
18. {
19. auto **int** i;
20. auto **int** n=1;
21. printf("at first n=%d\n",n);
22. **for**(i=1 ; i<3 ; i++)
23. {
24. show( );
25. }
26. printf("at last n=%d",n);
27. }

例4.6

文件file1.c

1. #include <stdio.h>
2. **int** a;
3. **int** m;
4. **int** power( );
5. **void** main( )
6. {
7. **int** b=3,c,d;
8. printf(“input the number a and its power m:\n");
9. scanf("%d,%d",&a,&m);
10. c = a\*b;
11. printf("%d\*%d=%d\n",a,b,c);
12. d = power();
13. printf("%d\*\*%d=%d",a,m,d);
14. }

文件file2.c

1. **extern** **int** a;
2. **extern** **int** m;
3. **int** power( );
4. {
5. **int** i,y=1;
6. **for** ( i=1 ; i<=m ; i++)
7. {
8. y\*=a;
9. }
10. **return**(y);
11. }

例4.9

1. #include<stdio.h>
2. **void** value();
3. **void** value()
4. {
5. **int** au=0;
6. **static** **int** st=0;
7. printf(“au\_variable=%d,st\_variable=%d\n”,au,st);
8. au++;
9. st++;
10. }
11. **void** main()
12. {
13. **int** i;
14. **for**(i=0;i<3;i++)
15. {
16. value();
17. }
18. }

### 例4.3实验过程

1.选取关键的观测点

选取第24行往后的程序重点观测。

2.调试

光标放在24行，按F4使程序运行到该处，用Alt\_F4观测i和n（main函数内部定义的）的值。且地址分别为register(SI)和register(DI)(CPU寄存器内)这是因为两个变量在调试过程中被临时取了出来，被放在寄存器中，但他们并不是寄存器变量。至于为什么这两个变量被取了出来，对i而言，他是for函数中的变量，所以自动变成寄存器变量，对n，就目前所学而言我并不清楚。

按F7进入show()函数内部，发现原来打开的观察窗口依旧可以跟踪到i和n的值，但要注意到，这前后的i实际已经不是同一个i了，**只不过是从内存中临时抽调到寄存器时占用了相同的位置。同时注意，BC的调试系统只考虑了变量名，没有考虑命名空间。**

按F7让函数继续逐行运行，进入复合语句内部后，

auto int i=10;

却发现原先main函数中n的观察窗口变成了10，**这是因为复合语句中定义了新的i，将这个新的i放在了原先寄存器中临时放n的位置。**对复合语句中的i观察，果然这个i临时存放在寄存器中，且地址与原先的n都是register(DI). **这里给出一个建议：观察完一个变量后就把这个窗口关了。**

继续按F8，发现当离开复合语句后，观察的复合语句内的i并没有消失，这还是因为BC的调试没有考虑命名空间。

### 例4.6实验过程

1.选取关键的观测点

从第12行，即power的执行开始观测。

2.调试

与例4.3调试的过程类似，按F7进入power()函数后，程序实际进入了file2.c运行。用Alt\_F4观测file1.c和file2.c中变量a,m的值，发现这两个文件中a，m的值保持一致，且地址一致。

### 例4.9实验过程

1.选取关键的观测点

从第16行，即首次进入value()函数开始观测。

2.调试

与例4.3和4.6的调试过程都非常类似。进入value()函数后，用Alt\_F4观察au与st两个变量的变化。

当离开value()函数时，au会消失，但st依旧存在，而当经过循环再次进入value()函数后，au再次出现，且初值被赋为0，而st则保留上一次离开value()的值。

### 心得体会

在调试过程中，如何选取关键的观测点？哪些语句是关键的？通过三次实验，我总结出了一些自己的经验。关键的观测点，实际上就是最有可能犯错的语句，以此为前提，常见的关键观测点有1.循环语句。2.判断语句。3.用户自定义函数，等等等等。这些关键点的选取，能够极大地简化我们调试的过程，为我们带来很多方便。

注意，F7可以进入用户自定义的函数内部，但F8不行。正常调试用F8，遇到自定义函数，如果想进入其中一探究竟，就要用到F7了。

BC的调试系统没有考虑变量空间，所以才会导致实验的结果和理论推导的结果不全相同，好的做法就是，按理论判断一个变量的作用范围，在合适的时候关掉原先的观察窗口，打开新的观察窗口。

# 实验5 P101第一题，程序分析题1、2、3

### 实验目的

（1）解决程序分析题1，2，3

### 实验内容

分析这三道题，得出自己的答案后，利用计算机来验证自己的答案是否正确。

### 题目1

1. #include<stdio.h>
2. **void** func();
3. **int** a;
4. **void** func()
5. {
6. printf("no 1 a=%d",a);
7. }
8. **void** main()
9. {
10. **int** a = 1;
11. printf("no 1 a=%d",a);
12. func();
13. {
14. **int** a=1;
15. printtf("no 1 a=%d",a);
16. func();
17. }
18. }

### 题目1解题思路

程序首先定义了一个全局变量a，然而main函数中开头也定义了一个变量a，于是依就近原则，main函数中的a以main函数中的定义为准。而func函数中为定义a，但也用到了a，所以func中的a以全局变量定义的为准。就此分析程序的结果为：

no 1 a=1no 1 a=0no 1 a=1no 1 a=0

### 验证题目1的答案

输入程序，发现用户界面显示的结果确实如此。

### 题目2

1. #include<stdio.h>
2. **void** func()
3. {
4. **static** **int** a=0;
5. **register** **int** b=0;
6. auto **int** c=0;
7. printf("a=%d\tb=%d\tc=%d\n",a++,b++,c++);
8. }
9. **void** main()
10. {
11. func();
12. func();
13. func();
14. }

### 题目2解题思路

该程序的main函数就是执行三次func函数，于是我们来分析func函数。func中定义了3个变量，a是static型变量，储存在静态储存区，从定义后直至程序运行完毕才会消失；而b，c分别是register和auto变量，他们从定义后直至func函数运行完毕后消失。以此为依据，可以推断程序的运行结果为：

a=0 b=0 c=0

a=1 b=0 c=0

a=2 b=0 c=0

### 验证题目2的答案

输入程序，发现用户界面显示的结果确实如此。

### 题目3

1. #include<stdio.h>
2. **int** n=1;
3. **void** func()
4. {
5. **static** **int** x=4;
6. **int** y=10;
7. x=x+2;
8. n=n+10;
9. y=y+n;
10. printf("func:x=%d,y=%d,n=%d\n",x,y,n);
11. }
12. **void** main()
13. {
14. **static** **int** x=5;
15. **int** y;
17. y=n;
18. printf("main:x=%d,y=%d,n=%d\n",x,y,n);
19. func();
20. printf("main:x=%d,y=%d,n=%d\n",x,y,n);
21. func();
22. }

### 题目3解题思路

程序在开头定义了全局变量n，同时我们可以发现，在main函数和func函数中都不再有n的定义了，所以这个全局变量n适用于整个程序。接下来分析main函数，开头我们定义了一个static变量x=5和自动型变量y。然后令y=n，即y=1.然后输出一次x,y,z的值。随后进入func函数。在func函数中，定义了static变量x和自动型变量y，在经过一番运算并输出x,y,z的结果后，func结束，同时func中定义的自动型变量y的寿命也结束了。等到第二次执行func函数时，x以上次的值为准，而y则会重新定义。以以上分析为依据，可以判断输出的结果为：

main:x=5,y=1,n=1

func:x=6,y=21,n=11

main:x=5,y=1,n=11

func:x=8,y=31,n=21

### 验证题目3的答案

输入程序，发现用户界面显示的结果确实如此。

### 心得与体会

解决这类“读程序，说结果”的题目时，最好的方法也是唯一的方法就是把自己当作电脑去执行程序，从main函数开始逐行执行下去，遇到用户自定义的函数再跳到对应的函数中。不同的函数中可能存在变量名一样的变量，一定要注意区分，必要时，可以在纸上画表格列出各个变量的变化情况，就像是单步调试中观测变量的值那样。