第 5 章

常见错误分析

C 语言的功能强，方便灵活，所以得到广泛应用，它使程序设计人员有发挥聪明才智、显示编程技巧的机会。一个有经验的 C 程序设计人员可以编写出能解决复杂问题、可靠性好、运行效率高、通用性强、容易维护的高质量程序。

C 程序是由函数构成的，利用标准库函数和自己设计的函数可以完成许多功能。善于利用函数，可以实现程序的模块化，将许多函数组织成一个大的程序。正因如此，C 语言受到越来越广泛的重视，从初学者到高级软件人员，都在学习 C 语言、使用 C 语言。

但是要真正学好 C 语言、用好 C 语言，并不容易，“灵活”固然是好事，但也使人难以掌握，尤其是初学者往往出了错还不知怎么回事。C 编译程序对语法的检查不如其他高级语言那样严格（这是为了给程序人员留下“灵活”的余地）。因此，往往要由程序设计者自己设法保证程序的正确性，需要不断积累经验，提高程序设计和调试程序的水平。

笔者根据多年来从事 C 程序设计教学的经验，将初学者在学习和使用 C 语言时容易犯的错误总结归纳如下，以帮助读者尽量避免重犯这些错误。这些内容其实在教材的各章中大多都曾提到过，为便于编程和调试程序时查阅，在这里集中列举，供初学者参考。

（1）忘记定义变量。

例如：

```
int main()
{
  x=3
  y=6;
  printf("%d\n",x+y);
  return 0;
}
```

C 语言要求对程序中用到的每一个变量都必须先定义，在程序编译时对已定义的变量进行存储空间的分配。上面程序中没有对 x、y 进行定义。应在函数体的开头加写：

```
int  x,y;
```

C 语言要求对用到的每一个变量进行强制定义（在本函数中定义或定义为外部变量）。

(2) 输入输出的数据类型与所用格式说明符不一致。

例如,若 a 已定义为 int 型,b 已定义为 float 型:

```
int a=3;
float b=4.5;
printf("%f%d\n",a,b);
```

编译时不给出出错信息,但运行结果将与原意不符,在 Visual C++ 6.0 环境中运行的结果为

```
0.000000 1074921472
```

在 Turbo C 2.0 环境中运行的结果为

```
0.000000  16402
```

它们并不是按照赋值的规则进行转换(如把实数 4.5 转换成整数 4),而是将数据在存储单元中的形式按格式符的要求组织输出(如 b 在内存中占 4 字节,按浮点数方式存储,今将其在内存中的二进制存储形式按整数格式组织输出。用 Turbo C 时,由于整数只占 2 字节,所以只把变量 b 在内存中最后 2 字节中的二进制数按%d 要求作为整数输出)。

这种情况下的输出结果往往是不可预测的。在调试程序时,如遇到输出的结果是莫名其妙的,应首先考虑是否输出格式符有问题。

(3) 未注意整型数据的数值范围。Turbo C 等编译系统,对一个整型数据分配 2 字节。因此一个整数的范围为$-2^{15}\sim2^{15}-1$,即$-32\ 768\sim32\ 767$。常见这样的程序段:

```
int num;
num=89101;
printf("%d",num);
```

在 Turbo C 中得到的却是 23 565,原因是 89 101 已超过 32 767。2 字节容纳不下 89 101,则将高位截去,见图 5.1,即将超过低 16 位的数截去,也即将 89 101 减去 2^{16}(即 16 位二进制所形成的模):89 101－65 536＝23 565。

89 101:	00 00 00	00 00 01	01 01 11 00	00 00 11 01
23 565:			01 01 11 00	00 00 11 01

图 5.1

如果用 Visual C++ 6.0,把 num 定义成 short 类型(占 2 字节)时,也会出现以上情况。

有时明明是个正数,却输出一个负数。例如:

```
num=196607;
```

输出得－1。因为 196 607 的二进制形式为

0000000000000010	11111111111111

去掉高字节的16位(即舍弃了10),低16位的值是-1(-1的补码是111111111111 1111)。

对于超过整数范围的数,要用long型,即在用Turbo C时要改为

```
long int num;
num=89101;
printf("%ld",num);
```

请注意:如果只定义num为long型,而在输出时仍用"%d"说明符,也会出现以上错误。

(4) 在输入语句scanf中忘记使用变量的地址符。

例如:

```
scanf("%d%d",a,b);
```

这是许多初学者刚学习C语言时一个常见的疏忽,或者说是习惯性的错误,因为在其他语言中在输入时只需写出变量名即可,而C语言要求指明"向哪个地址标识的单元送值"。应写成

```
scanf("%d%d",&a,&b);
```

(5) 输入数据的形式与要求不符。

用scanf函数输入数据,应注意如何组织输入数据。假如有以下scanf函数:

```
scanf("%d%d",&a,&b);
```

有人按下面的方法输入数据:

```
3,4
```

这是错的。数据间应该用空格(或Tab键,或回车符)来分隔。读者可以用

```
printf("%d%d",a,b);
```

来验证一下。应该用以下方法输入:

```
3 4
```

如果scanf函数为

```
scanf("%d,%d",&a,&b);
```

对scanf函数中格式字符串中除了格式说明符外,对其他字符必须按原样输入。因此,应按以下方法输入:

```
3,4
```

此时如果用"3 4"反而错了。还应注意,不能企图用

```
scanf("input a&b: %d,%d",&a,&b);
```

想在屏幕上显示一行信息：

```
input a&b:
```

然后在其后输入 a 和 b 的值，这是不行的。如果想在屏幕上得到所需的提示信息，可以另加一个 printf 函数语句：

```
printf("input a&b:");
scanf("%d,%d",&a,&b);
```

(6) 误把“＝”作为“等于”运算符。

许多人习惯性地用数学上的等于号“＝”作为 C 程序中的关系运算符“等于”。而在 C 语言中，“＝＝”才是关系运算符“等于”。有人写出如下的 if 语句：

```
if(score=100)  n++;
```

本意是想统计 score 为 100 分的人数，当 score 等于 100 时就使 n 加 1。但 C 编译系统将“＝”作为赋值运算符，将“score＝100”作为赋值表达式处理，把 100 赋给 score，作为 score 的新值。if 语句检查 score 是否为零。若为非零，则作为“真”；若为零，则作为“假”。今 score 经过赋值之后显然不等于 0，因此总执行 n＋＋，不论 score 的原值是什么，都使 n 的值加 1。

这种错误在编译时是检查不出来的，但运行结果往往是错的。而且由于习惯的影响，在检查源程序时，往往设计者自己是不易发觉的。

(7) 语句后面漏分号。

C 语言规定语句末尾必须有分号，分号是 C 语句不可缺少的一部分，这也是和其他语言不同的。有的初学者往往忘记写这一分号。例如：

```
a=3
b=4;
```

在程序编译时，编译系统在“a＝3”后面未发现分号，就接着检查下一行有无分号。“b＝4”也作为上一行的语句的一部分，这就出现语法错误。由于在第 2 行才能判断语句有错，所以编译系统指出“在第 2 行有错”，但用户在第 2 行却未发现错误。这时应该检查上一行是否漏了分号。

如果用复合语句，有的学过 Pascal 语言的读者往往漏写最后一个语句的分号，例如：

```
{t=a;
 a=b;
 b=t
}
```

在 Pascal 中分号是两个语句间的分隔符而不是语句的一部分，而在 C 语言中，没有分号的就不是语句。

(8) 在预编译指令的末尾多加了一个分号。

预编译指令不是 C 语句，末尾不需要加分号，但是有人习惯于在每行末尾都加分号。例如：

```
#include <stdio.h>;
#define N 10;
```

这就错了。

(9) 语句未结束就加分号。

例如：

```
if(a>b);
  printf("a  is  larger  than b\n");
```

本意是：当 a>b 时输出“a is larger than b”的信息。但由于在 if (a>b)后加了分号，因此 if 语句到此结束，即当 a>b 为真时，执行一个空语句。本来想 a≤b 时不输出上述信息，但现在 printf 函数语句并不从属于 if 语句，而是与 if 语句平行的语句。不论 a>b 还是 a≤b，都输出“a is larger than b”。

又如：

```
for(i=0;i<10;i++);
 { scanf("%d",&x);
   printf("%d\n",x*x);
 }
```

本意为先后输入 10 个数，每输入一个数后输出它的平方值。由于在 for()后不经意地加了一个分号，使循环体变成了空语句。执行 for 语句的效果只是使变量 i 的值由 0 变到 10。然后输入一个整数并输出它的平方值。

这种错误往往发生在不熟悉 C 语法的初学者身上。

总之，在 if、for、while 语句中，不要画蛇添足，多加分号。

(10) 对应该有花括号的复合语句，忘记加花括号。例如：

```
sum=00;
i=1;
while(i<=100)
   sum=sum+i;
   i++;
```

本意是实现 1+2+…+100，即 $\sum_{i=0}^{100} i$ ，但上面的语句只是重复了 sum+i 的操作，而且循环永

不终止,因为 i 的值始终没有改变。错误在于没有写成复合语句形式。因此,while 语句的范围到其后第一个分号为止。语句“i++;”不属于循环体范围之内。应改为

```
while(i<=100)
 { sum=sum+i;
   i++;
 }
```

(11) 括号不配对。

当一个语句中使用多层括号时常出现这类错误,纯属粗心所致。例如:

```
while((c=getchar()!='#')
  putchar(c);
```

少了一个右括号。

(12) 在用标识符时,混淆了大写字母和小写字母的区别。

例如:

```
#define PI 3.1416926
int main()
 ( float .area,r=2.5;
   area=pi*r*r;
   return 0;
}
```

编译时出错。编译程序把 PI 和 pi 认作两个不同的标识符处理,所以认为“变量 pi 未经定义”,出错。

(13) 引用数组元素时误用了圆括号。

例如:

```
int main()
{ int i,a(10);
  for(i=0;i<10;i++)
    scanf("%d",&a(i));
  return 0;
}
```

C 语言中对数组的定义或引用数组元素时必须用方括号。

(14) 在定义数组时,将定义的“元素个数”误认为是“可使用的最大下标值”。

例如:

```
int main()
{ int a[10]={1,2,3,4,5,6,7,8,9,10};
  int i:
```

```
  for(i=1; i<=10; i++)
    printf("%d",a[i]);
  return 0;
}
```

编程者想输出 a[1]～a[10]这 10 个元素是不可能的。C 语言规定在定义数组时用 a[10],表示 a 数组有 10 个元素,而不是可以用的最大下标值为 10。在 C 语言中数组的下标是从 0 开始的,因此,数组 a 只包括 a[0]～a[9]这 10 个元素,想引用 a[10]就超出 a 数组的范围了。值得注意的是,在程序编译时,C 编译系统对此并不报错,编译能通过,但运行结果不对。系统把 a[9]后面的存储单元作为 a[10]输出,这显然不是编程者的原意。由于编译系统不报错,有时编程者难以发现这类错误。要注意仔细分析运行结果。这是一些初学者常犯的错误。

(15) 对二维或多维数组的定义和引用的方法不对。

例如:

```
int main()
{ int a[5,4];
  printf("%d",a[1+2,2+2]);
  return 0;
}
```

对二维数组和多维数组在定义和引用时必须将每一维的数据分别用方括号括起来。上面 a[5,4]应改为 a[5][4],a[1+2,2+2]应改为 a[1+2][2+2]。根据 C 语言的语法规则,在一个方括号中的是一个维的下标表达式,a[1+2,2+2]方括号中的"1+2,2+2"是一个逗号表达式,它的值是第二个数值表达式的值,即 2+2 的值为 4。所以 a[1+2,2+2]相当于 a[4],而 a[4]是 a 数组的第 4 行的首地址。因此执行 printf 函数输出的结果并不是 a[3][4]的值,而是 a 数组第 4 行的首地址。

(16) 误以为数组名代表数组中全部元素。

例如:

```
int main()
  {int a[10]={1,3,5,7,9,11,13,15,17,19},b[10];
   b=a;
   return 0;
  }
```

用这种方法将 a 数组的全部元素赋给 b 数组的相应元素是做不到的。在 C 语言中,数组名代表数组首地址,不代表全部元素。

(17) 混淆字符数组与字符指针的区别。

例如:

```
int main()
  { char str[4];
   str="Computer and c";
   printf("%s\n%",str);
   return 0;
  }
```

编译出错。str是数组名,在编译时对str数组分配了一段内存单元,数组名代表数组首地址。因此在程序运行期间str是一个常量,不能再被赋值。所以,str="Computer and c"是错误的。

如果改成

```
char *str;                    //改为指针变量
str="Computer and c";
printf("%s\n",str);
```

该程序正确。此时str是指向字符数据的指针变量,str="Computer and c"是合法的,它将字符串的首地址赋给指针变量str,然后在printf函数语句中输出字符串“Computer and c”。

因此应当弄清楚字符数组与字符指针变量用法的区别。

(18) 在引用指针变量之前没有对它赋予确定的值。例如:

```
int main()
{ char *p;
  scanf("%s",p);
  return 0;
}
```

没有给指针变量p赋值就引用它,编译时给出警告信息。其实指针变量p中不是空的,而是存放了一个不可预测的值,即p指向地址为此值的存储单元,而这个存储单元中可能是存放了有用的数据的。如果执行上面的scanf语句,就将一个字符串输入到此存储单元开始的一段存储空间,这就改变了这段存储空间的原有状况,有可能破坏了系统的工作环境,产生灾难性的后果,十分危险。应当改为

```
char *p,c[20];
p=c;
scanf("%s",p);
```

即先根据需要定义一个大小合适的字符数组c,然后将c数组的首地址赋给指针变量p,此时p有确定的值,指向数组c的首元素。再执行scanf函数就没有问题了,把从键盘输入的字符串存放到字符数组c中。

(19) switch语句的各分支中漏写break语句。

例如:

```
switch(score)
{case 5:printf("Very Good!");
 case 4:printf("Good!");
 case 3:printf("Pass!");
 case 2:printf("Fail!");
 default:printf("data error!");
}
```

写上述 switch 语句的原意是希望根据 score(成绩)输出评语。但当 score 的值为 5 时，输出为

```
Very Good! Good! Pass!Fail!data error!
```

原因是漏写了 break 语句。case 只起标号的作用，而不起判断作用，因此在执行完第一个 printf 函数语句后接着执行第 2～5 个 printf 函数语句。应改为

```
switch(score)
{case 5:printf("Very good!");      break;
 case 4:printf("Good!");           break;
 case 3:printf("Pass!");           break;
 case 2:printf("Fail!");           break;
 default:printf("data error!");
}
```

(20) 混淆字符和字符串的表示形式。

例如：

```
char sex;
sex="M";
⋮
```

sex 是字符变量，只能存放一个字符。而字符常量的形式是用单撇号括起来的，"M"是用双撇号括起来的字符串，它包括两个字符：'M'和'\0'，无法存放到字符变量 sex 中。应改为

```
sex='M';
```

(21) 使用自加(++)和自减(--)运算符时容易出的错误。

例如：

```
int main()
{ int *p,a[5]={1,3,5,7,9};
 p=a;
 printf("%d",*p++);
 return 0;
```

```
}
```

不少人认为“*p++”的作用是先使p加1,即指向第1个元素a[1]处,然后输出第一个元素a[1]的值3。其实应该是先执行p++,而p++的作用是先用p的原值,用完后使p加1。今p的原值指向数组a的第0个元素a[0],因此*p就是第0个元素a[0]的值1。结论是先输出a[0]的值,然后再使p加1。如果是*(++p),则先使p指向a[1],然后输出a[1]的值。

在使用++和--运算符时,一定要避免歧义性,如无把握,宁可多加括号,如上面的*p++可改写为*(p++)。

(22) 所调用的函数在调用语句之后才定义,而又在调用前未声明。

例如:

```
int main()
 {float x,y,z;
  x=3.5;y=-7.6;
  z=max(x,y);
  printf("%f\n",z);
  return 0;
 }
 float max(float x,float y)
   {
      return (z=x>y?x:y):
   }
```

这个程序在编译时有出错信息。原因是max函数是在main函数之后才定义,也就是max函数的定义位置在main函数调用max函数之后。改错的方法可以用以下二者之一。

① 在main函数中增加一个对max函数的声明,即函数的原型:

```
int main()
  {float max(float,float);        //声明将要用到的max函数为实型
   float x,y,z;
   x=3.5;y=-7.6;
   z=max(x,y);
   printf("%f\n",z);
  }
```

② 将max函数的定义位置调到main函数之前,即

```
float max(float x,float y)
  {return (z=x>y?x:y);}

void main()
```