西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

编写程序语句

1、编写简单语句和复合语句

2、编写注释

3、语句的写法

- 语句（statement）是C程序的最小单位。
- 程序由一条一条语句组成，语句执行的次序则称为流程。

3.1.1 简单语句

- 1．表达式语句
- 在任何表达式后面加上一个分号（；）就构成了一个表达式语句（expression statement），语句形式为：

```
表达式; //用分号结束
```

- 示例

```
x=a+b; //赋值语句
t=a,a=b,b=t; //a和b交换
a+b+c; //运算但无实际意义
```

3.1.1 简单语句

- 2. 函数调用语句
- 函数调用语句是由函数调用加分号（；）形成的，语句形式为：

```
函数调用(实参); //用分号结束
```

- 示例

```
printf("a+b=%d",a+b); //输出函数调用语句
```

3.1.1 简单语句

- 3．空语句
- 仅有一个分号就形成了空语句（null statement），它什么也不做，语句形式为：

```
; //单个分号
```

- 示例

```
while(getchar()!='\n') ; //使用空语句
```

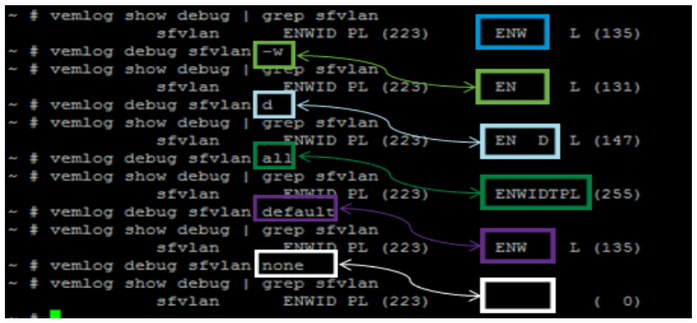
3.1.1 简单语句

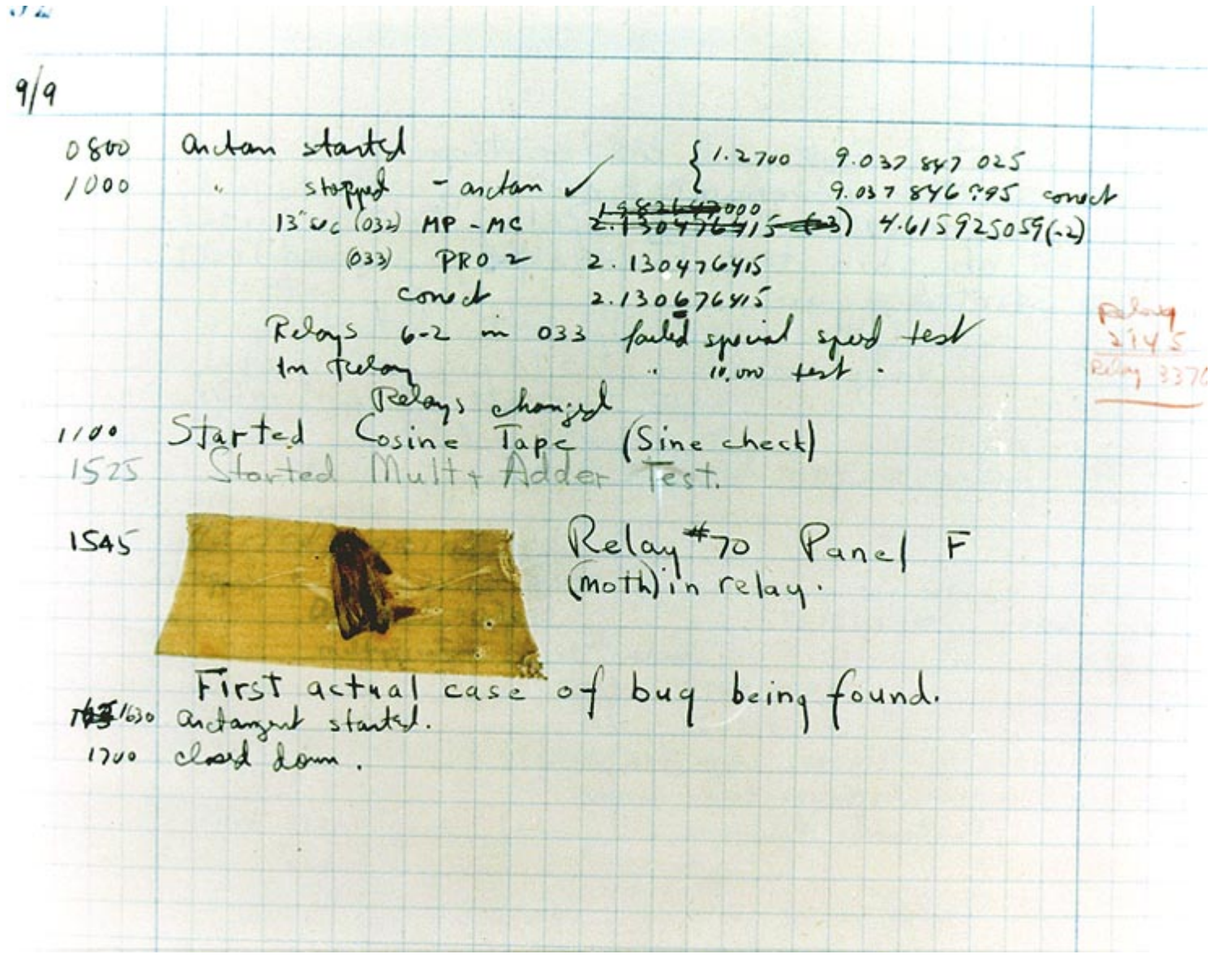
- 由于空语句是一个语句，因此可用在任何允许使用语句的地方。而意外出现的多余分号（即空语句）不是语法错误，不会由编译器报告出来，因而有时容易让程序员忽视而产生难以消除的程序漏洞（BUG）。

```
printf(“a+b=%d”,a+b); ; //第2个即为空语句
```

3.1.1 简单语句

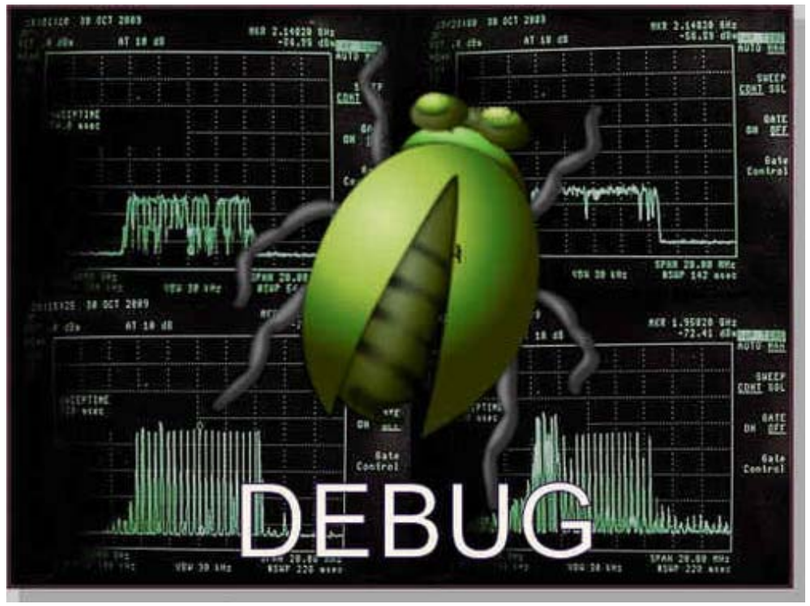
- 第一个电脑上的bug





3.1.1 简单语句

- Debugging / Debug，调试，又称除错，是发现和减少计算机程序或电子仪器设备中程序错误的一个过程。
- 调试的基本步骤
  - 发现程序错误的存在
  - 以隔离、消除的方式对错误进行定位
  - 确定错误产生的原因
  - 提出纠正错误的解决办法
  - 对程序错误予以改正，重新测试



3.1.1 简单语句

- 4．声明部分
- 在C语言中变量的定义或类型的声明称为声明部分，不能视作语句，尽管看起来它也有分号像语句。C语言规定声明部分必须出现在所有可执行语句的前面，即在函数或语句块的开头位置定义变量或进行类型声明，一般形式为：

```
声明部分.....;
执行语句.....;
```

```
int a,b,t; //定义整型变量
```

3.1.1 简单语句

- 下面的程序代码会产生编译错误：

```
1 int a,b; //正确，定义变量放在所有可执行语句前面
2 a=10,b=20;
3 int t; //错误，声明和定义不能放在语句中间
4 t=a,a=b,b=t;
```

3.1.1 简单语句

- 请注意，目前许多编译器同时支持C和C++，而C++将变量的定义或类型的声明当作是语句，允许放在程序任意位置上，所以上述代码在这些编译器中是合法的（按C++处理）。
- 建议养成将声明部分放在函数或语句块开头位置的习惯。

3.1.2 复合语句

- 将多个语句组成的语句序列用一对大括号｛｝括起来组成的语句称为复合语句（compound statement），又称语句块，简称块（block）。语句形式为：

```
{
   [局部声明部分.....;]
   语句序列.....;
}
```

- 其中，“语句序列.....”表示任意数目的语句，方括号内的局部声明部分是可选的。

3.1.2 复合语句

- 例如：

```
{ //复合语句
    double s, a=5, b=10, h=8; //局部声明
    s=(a+b)*h/2.0;
    printf("area=%lf\n", s);
} //复合语句不需要分号结尾
```

- 复合语句内的每条语句必须以分号（；）结尾，但复合语句右大括号（}）已表示结尾，因此其后不需要分号。如果在后面添加分号，意思变为一个复合语句与一个空语句。

3.1.2 复合语句

- 复合语句内部可以进行变量定义或类型声明，这些定义或声明仅在复合语句内部可以使用，称为块的局部作用域。
- 示例

```
{ //块的局部作用域
    int t,a=10,b=7; //定义局部变量t、a、b
    t=a,a=b,b=t; //仅在这个复合语句里使用
}
```

3.1.2 复合语句

- 复合语句允许嵌套，即在复合语句里还可以包含复合语句。
- 示例

```
{ //复合语句
    double v1,r=5; //局部声明
    v1=4*3.1415926*r*r*r/3;
    { //嵌套的复合语句
        double v2,h=12; //嵌套的局部声明
        v2=3.1415926*r*r*h;
        printf("%lf,%lf\n",v1,v2);
    } //嵌套的复合语句结尾
} //复合语句结尾
```

3.1.2 复合语句

- 如果复合语句中没有任何内容，如｛｝称为空复合语句，空复合语句与空语句等价，它为空语句提供了一种替代语法。

3.1.2 复合语句

- 使用复合语句嵌套，程序有了更大能力应付复杂的流程处理。

- 使用复合语句的目的是描述长而复杂的语句序列，利于将复杂的语句形式简单化和结构化。

3.1.3 注释

- 可以在程序中编写注释（comments），有两种形式：
- ①/*……*/块注释语法形式：

```
/*
   ……注释内容
*/
```

3.1.3 注释

- ②//行注释语法形式：

```
//.....注释内容
```

3.1.3 注释

- （1）注释仅是对源程序的说明文字，它不是程序代码，对程序运行没有任何影响。实际上，在编译程序时所有注释内容将被忽略。
- （2）/*……*/块注释允许多行注释，以（/*）开头，以（*/）结尾，这中间的任何内容均是注释内容。注释可以是任何来自于字符集的字符组合，包括换行符，也允许中文等非ASCII字符。/*……*/不允许嵌套。

3.1.3 注释

- 例如：

```
/* 第1个注释
    ......
    /* 第2个注释
        ......
        ......
    */
*/
```

  - 编译器将第2个注释的（*/）当作第1个注释的结尾，从而使得后续部分出现编译错误。

3.1.3 注释

- （3）//行注释是C语言新标准允许的另一种注释方法，//注释表示从（//）开始直到本行末尾的所有字符均是注释内容。
- 示例

```
s=3.1415926*r*r*h/3; //计算圆锥体积
```

3.1.3 注释

- //注释只能注释一行，如果要注释多行就要写多次。
- 一般//注释适用于短小精简的注释，/*.....*/注释适用于大段注释。

3.1.3 注释

- （4）编译器将整个注释理解为一个空白字符，相当于一个空格的作用。在编译阶段，所有的注释均被忽略，所以执行程序不包含注释内容；换言之，注释对于程序的执行是没有任何效用的。
- 示例

```
int/*这里有注释*/t, a, b;
//t=a, a=b, b=t;
```

3.1.3 注释

何时使用注释？

（1）注释出现在程序的源文件中，可以对源程序作出说明，从而增加程序的可读性。

（2）可以将一段程序代码用注释临时“屏蔽”起来，即让某段程序“暂时失效”。

3.1.3 注释

```
/* 用注释将下面的程序暂时“屏蔽”不用
  for(a=1;a<8;a++) { //试探商的值
    x1=((a*8+7)*8+1)*8+1;
    x2=(2*a*17+15)*17+4;
    if(x1==x2)
      printf("%d\n",x1);
    else if(a==8)
      printf("不存在这样的数！");
  }
*/
```

3.1.3 注释

如何使用注释？

注释内容应该是那些能够确切描述程序代码功能、目的、接口、概括算法、确认数据对象含义以及阐明难以理解的代码段的说明性文字。

编程时要养成习惯添加注释，但注释也不是越多越好，平衡就好。

3.1.4 语句的写法

- （1）多数情况下，在一个程序行里只写一个语句，这样的程序写法清晰，便于阅读、理解和调试。

- （2）注意使用空格或TAB来作合理的间隔、缩进、对齐，使程序形成逻辑相关的块状结构，养成优美的程序编写风格。

3.1.4 语句的写法

- （3）C语言允许在一行里写多个语句。
- 示例

```
a=i/100; b=i/10%10; c=i%10; //3个语句
```

- 由于行是多数编译器在编译或调试时的基本单位，即使编译器指明了某一行有错误也不能明确判明是哪个语句出错，因此在一行里写多个语句的风格并不好。

3.1.4 语句的写法

- （4）C语言允许将一个语句拆成多行来写。
- 示例

```
printf("a=%f,b=%f,c=%f,d=%f,e=%f,f=%f,g=%f,h=%f\n",
a,b,c,d,e,f,g,h);
```

3.1.4 语句的写法

- 由于计算机屏幕宽度有限，过长的语句拆成多行来写是可能的。但需要注意两点：
  - 一是C语言规定回车换行也是空白符，所以不能在关键字、标识符等中间拆分，否则人为间隔了这些词语，会产生编译错误；
  - 二是在C语言中字符串常量是不能从中间拆分的，因为编译器会认为字符串没有正确结束。

3.1.4 语句的写法

- 示例

```
1   printf("This is a very long
2       string of examples");
```

- 第1行会产生编译错误。

3.1.4 语句的写法

- 解决字符串常量拆分的办法是使用反斜杠（\）行连接符，行连接符的作用是将程序的下一行（从第一列开始）替换当前的行连接符。
- 示例

```
1 "one \
2 two \
3 three"
```

- 上述写法与下面的写法等价：

```
1 "one two three"
```

3.1.4 语句的写法

- 请注意，如果//注释后面不幸地有一个行连接符，那么下一行也依然是注释。
- 示例

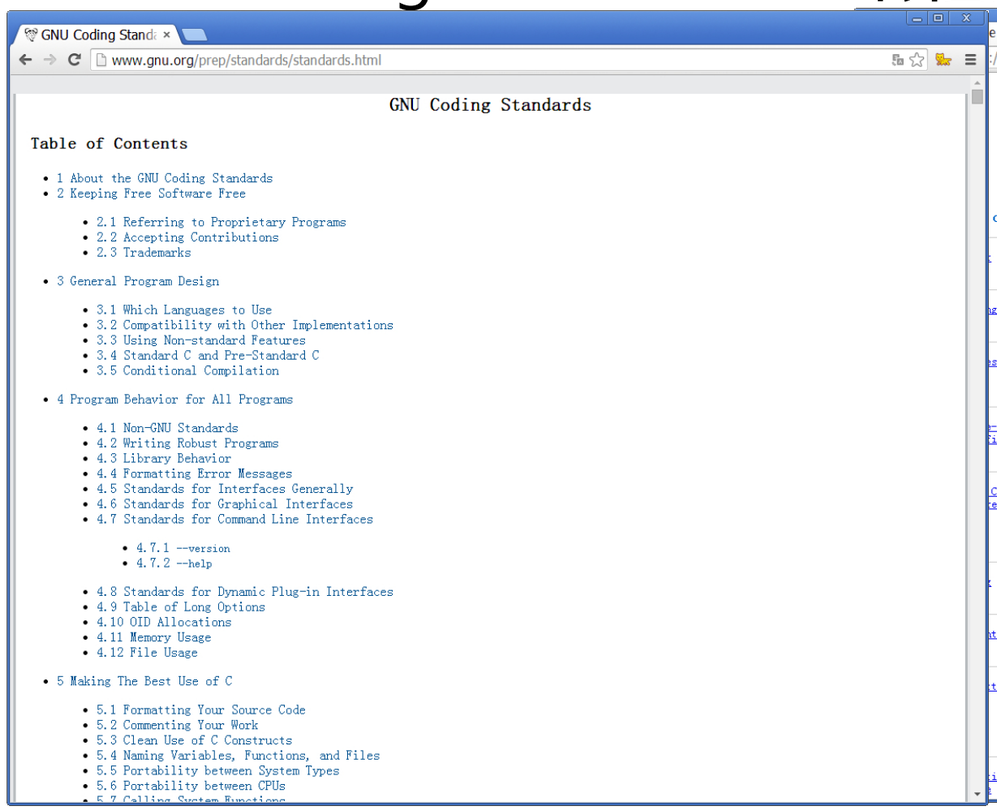
```
1 int t, a=10, b=7; // 本行的注释\
2 t=a, a=b, b=t;
```

- 与下面等价：

```
1 int t, a=10, b=7; // 本行的注释t=a, a=b, b=t;
```

3.1.4 语句的写法

- GNU、Google、Microsoft编程规范




程序设计