

All things are numbers.

--- *Pythagoras and his students*



廈門大學  
XIAMEN UNIVERSITY



信息学院 黃 煒  
(特色化示范性软件学院) 博士·副教授  
School of Informatics Wei Huang

# C语言简介

理论课程



廈門大學  
XIAMEN UNIVERSITY



信息学院 黄 焯  
(特色化示范性软件学院) 博士, 副教授  
School of Informatics Wei Huang

# 内容要点

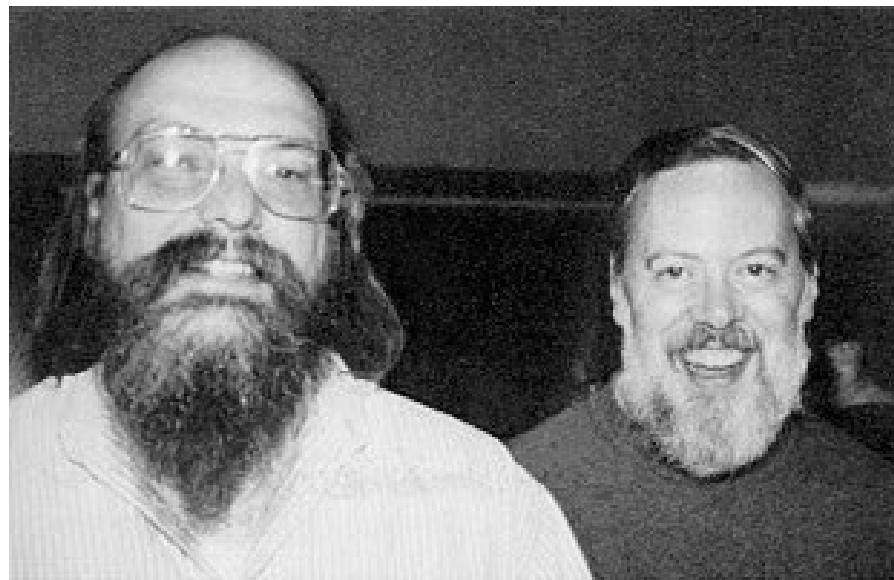
- C语言的历史
- C语言的基本语法
- C程序的可读性
- C程序设计实例

# 目录

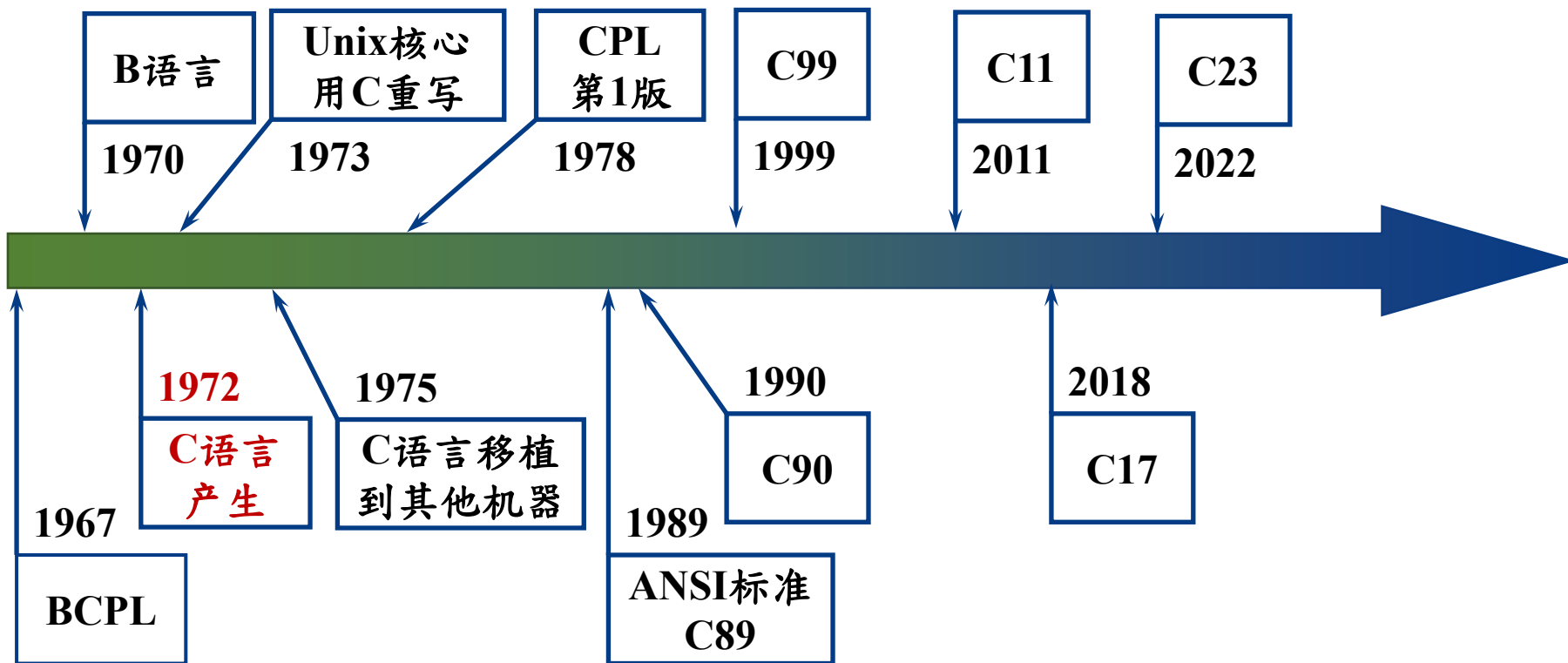
1	C语言的历史
2	C语言的基本语法
3	C程序的可读性
4	C程序设计实例
5	总结

# C语言之父

- **Dennis Ritchie ( 1941-2011 )**
  - C语言之父，UNIX之父，图灵奖
  - 1972年Ritchie ( 右 ) 与Ken Thompson ( 左 ) 设计 Unix
  - 来自于Thompson的B语言



# C的历史



# 目录

1	C语言的历史
2	C语言的基本语法
3	C程序的可读性
4	C程序设计实例
5	总结

# 程序设计的目的

- 程序设计的目的

- 编写计算机程序，控制计算机运行，利用计算机的高速运算和高容存储，提高人类的劳动能力和效率。

- 程序员书写程序的要求

- 让计算机能识别：符合程序设计语言的语法规则，否则程序将无法执行

- 让队友能看懂：符合程序员正常的读写习惯，否则无法融入团队



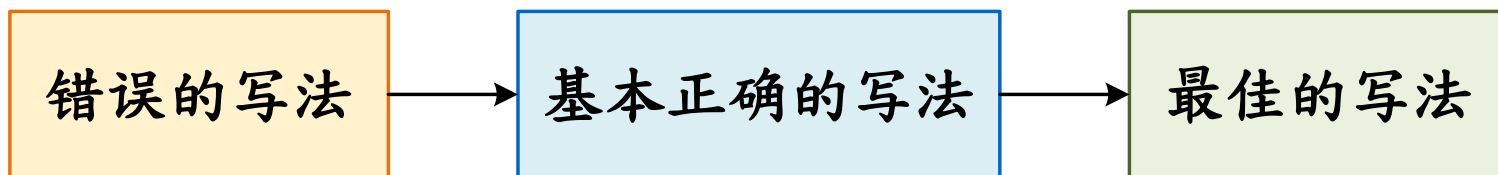
# 书写C程序应符合语法规则

- 语法：语言规则
- 书写程序应严格遵守语法规则。
  - 计算机无法理解存在语法错误的程序
  - 计算机无法理解弦外之音或表达错误

语法错误	示例	日常示例
顺序颠倒	<code>int main() → main int()</code>	“我饭吃”：我吃饭？饭吃我？
成分丢失	<code>return 0; → return;</code>	“我看见”：我看见什么？
未定义	<code>int a=3; b=a/5; → b=a/5;</code>	“我找他”：他是谁？

# 书写C程序应符合可读性良好

- 在不影响可读性的情形下，规则允许部分自由度。
  - 空白字符
  - 注释
  - 兼容的数据类型
  - 等价的写法



```
// first.c
#include <stdio.h>
int main(void)
{
    int num;
    num = 1;

    printf("I am a simple "); /* use the printf() function */
    printf("computer.\n");
    printf("My favorite number is %d because it is first.\n", num);

    return 0;
}
```

亮蓝色文字是关键词

绿色文字是注释；枣红色文字是字符串

灰色底色的方框内是控制台输出

```
I am a simple computer
My favorite number is 1 because it is first.
```

# C源代码的基本语法元素

- 源程序由可见字符组成

- 大小写英文字母、数字和下划线 ( A~Z , a~z , 0~9 , \_ )
  - 大小写敏感 ( Case Sensitive ) : 大小写字符是不同的
  - 中日韩文等大字符集 ( **不推荐!** 编辑器不支持时会乱码 )
- 括号修饰符 ( { } [ ] ( ) )
- 操作符 ( + - \* / % , . < > = ! & | ' " : ? ^ ) , 语句结尾分号 ( ; )
- 空白字符 ( 空格、制表符、换行符 )
- 可用于字符串中的其它符号 ( 普通键盘可以打出的符号 )

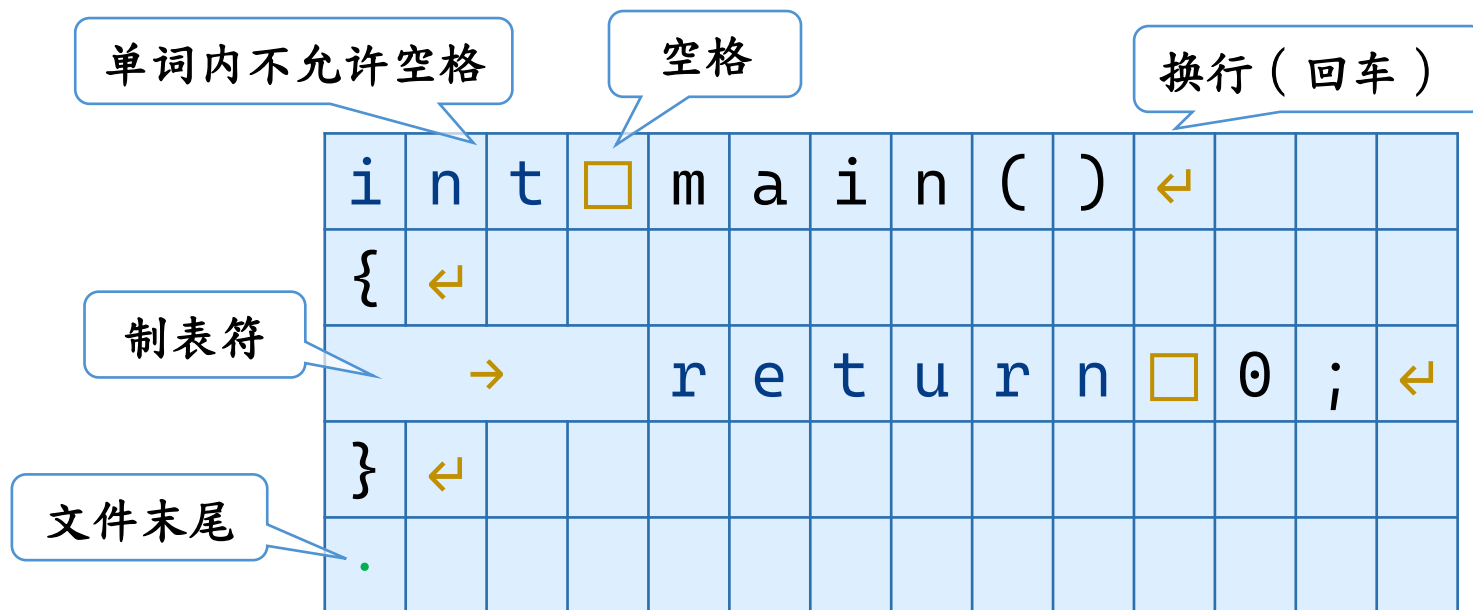
# C源代码的基本语法元素

- 空白字符：看不见的字符
  - 空格：键盘上最宽的键（空格键），显示宽度为1个字符。
  - 制表符：键盘上Tab键，显示宽度默认为4个字符。
  - 回车：键盘上Enter键，下一个字符将从下一行顶格显示。
- 空白字符其在源代码编译时是等价的。
  - 连续的空白字符，将简化为空白字符，最终简化为空格。

# C源代码的基本语法元素

- 文件的开头和结束

- 程序的第一个字符前一般不加空白字符
- 程序一般以回车符为结尾（不加回车符也可以）



# C源代码的基本语法元素

## • 注释

- 注释是写给队友看的帮助其正确理解代码的说明
- 计算机编译时，先删除注释再替换空白字符为空格

```
/*  
    A simple main function.  
*/
```

这是一个多行注释，以多行注释的开始符 ( /\* ) 开始，直到最近的一个结束符 ( \*/ ) 结束。中间可以换行，也可以不换行

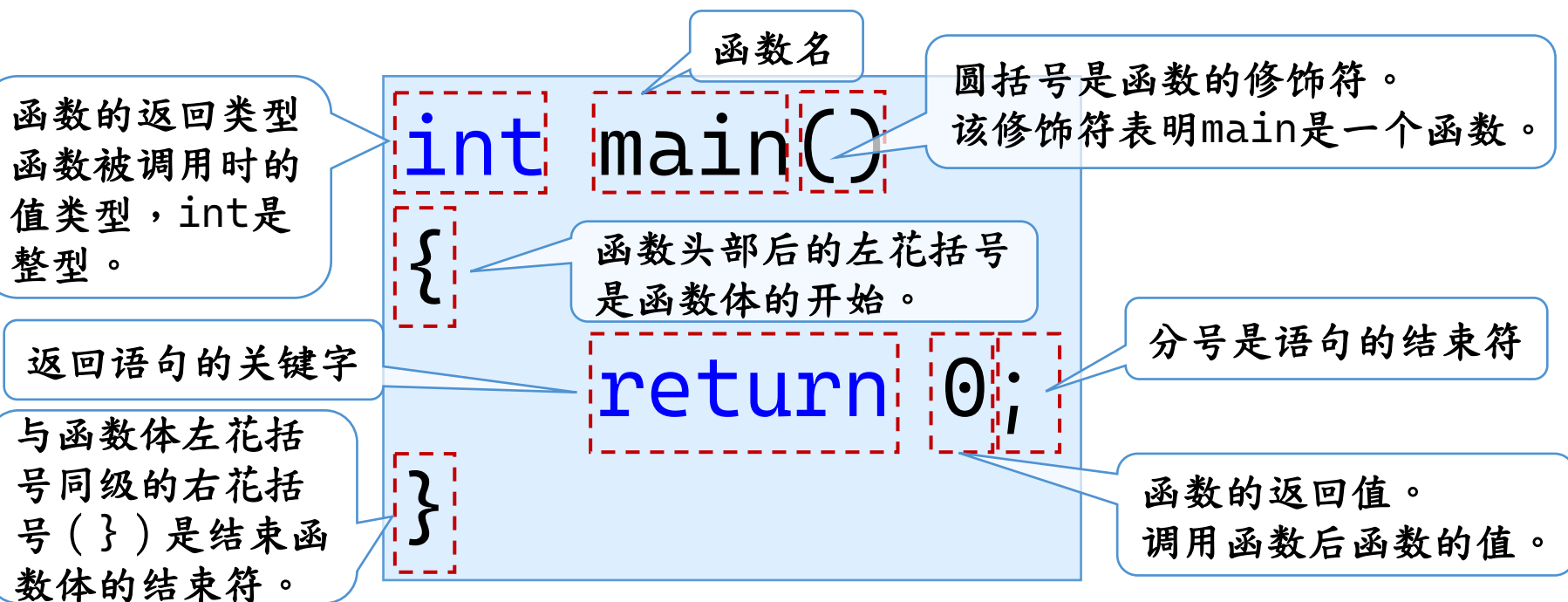
```
int main()  
{  
    return 0; // no error  
}
```

这是一个单行注释，以单行注释的开始符 ( // ) 开始，直到最近的一个结束符 ( 换行符 ) 结束。中间不可换行

# C源代码的基本语法元素

## • 函数

- 函数的特征是函数名后用圆括号修饰，如：main()
- 函数的定义由函数头部和函数体组成。





# C源代码的基本语法规则

- C源代码由一个或多个函数构成。
  - 可执行程序有且只有1个main()函数，否则有编译错误。
- 函数由语句或语句体组成。
  - 语句以分号结束

```
return 0;
```

- 语句体是由花括号包含的一个或多个语句构成

```
{  
  
  
}  
  
}
```

```
int s = 0;  
return s;
```

这是一个语句体，以左花括号（{）开始，到同一级右花括号（}）结束。该级的代码与花括号对齐后缩进四个空格书写。

# C源代码的基本语法规则

- C源代码的任何位置可以包括预编译指令。
  - 预编译指令终将影响编译配置或展开为语句的一部分。

```
#include <stdio.h>
```

- 预编译指令不是语句，不能以分号结束
- 最常用的语句是表达式语句
  - 表达式语句以表达式和分号构成。

```
sum = a + b;
```

这是一个表达式，函数的调用也是表达式。

- 表达式有值，按优先级依次计算得到值。

# C源代码的基本语法规则

- 表达式之间可相互替换

```
int main()
{
    return 0;
}
```

```
int main()
{
    return 5+3;
}
```

```
#include <math.h>
int main()
{
    return 5*sin(0.5);
}
```

- 语句（体）之间可相互替换

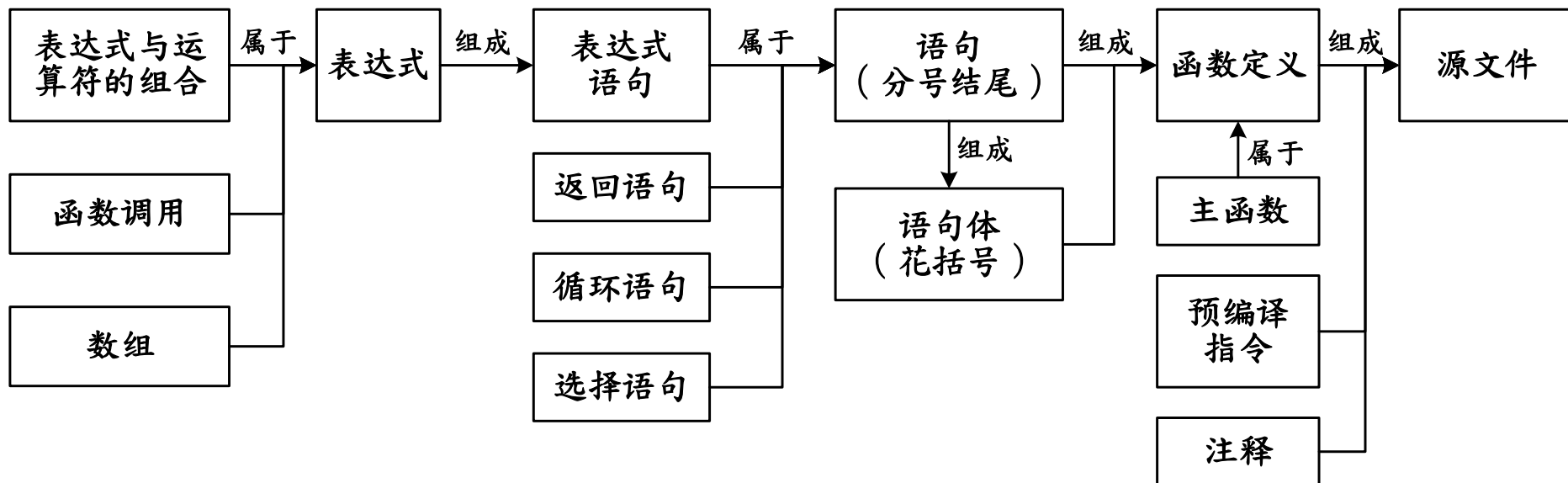
```
int main()
{
    return 0;
}
```

```
int main()
{
    int s = 4;
    return s;
}
```

```
#include <math.h>
int main()
{
    int s = 4;
    s = s * 5;
    return s;
}
```

# C源代码的基本语法规则

## • 语法规则

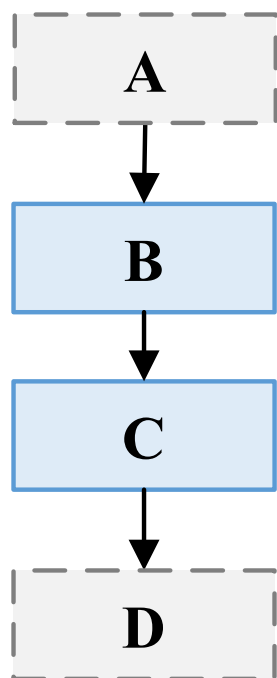


## • 编译器按语法规则解析代码

— 操作符（标点符号）和单词（含大小写）不可写错。

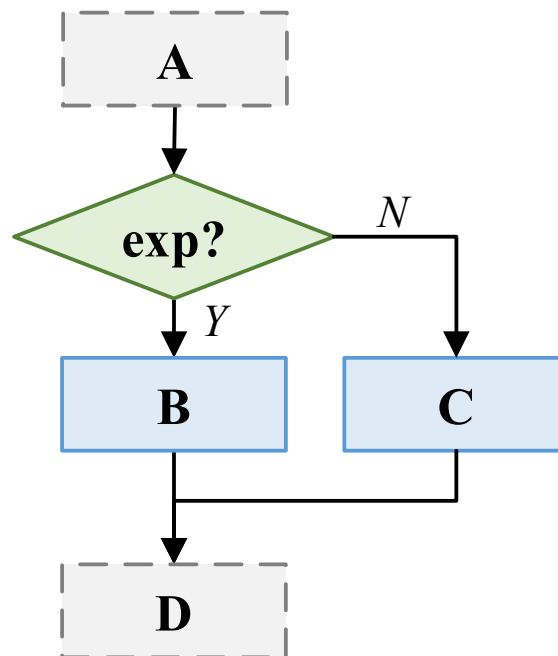
# C源代码的基本语法规则

- 结构化编程的三大结构



顺序结构

```
A;  
B;  
C;  
D;
```

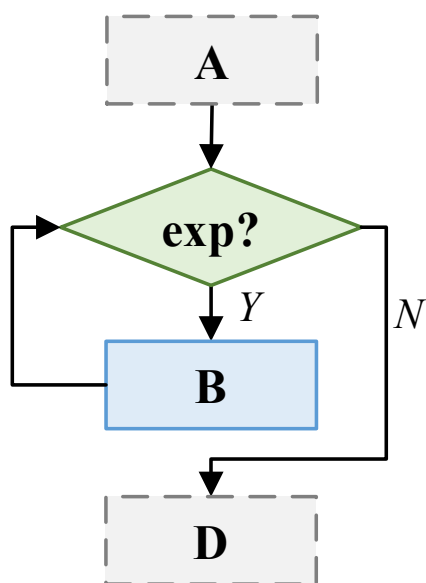


分支结构

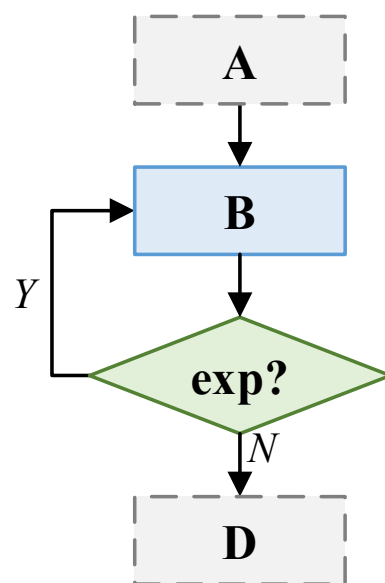
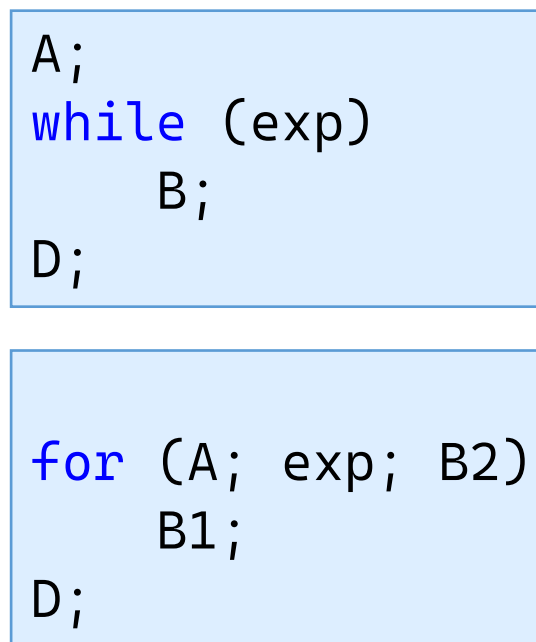
```
A;  
if (exp)  
    B;  
else  
    C;  
D;
```

# C源代码的基本语法规则

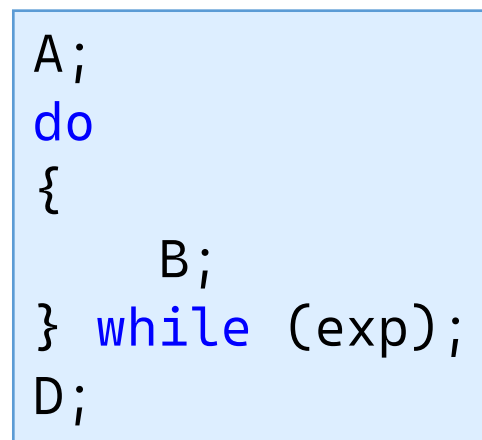
- 结构化编程的三大结构



循环结构 ( 1 )



循环结构 ( 2 )



# 最简单的程序

## • 最简单的规范程序（计算机视角）

C源代码由一个或多个函数构成。  
可执行程序从主函数开始执行

```
int main()
```

第1行说明这是主函数，类型是整型，圆括号说明这是一个函数。

```
{
```

第2行说明这个函数从这里开始

```
    return 0;
```

第3行是一个返回语句，程序运行到主函数的返回语句停止。

```
}
```

第4行说明主函数到这里结束

这个程序什么也不做，只是向调用它的程序返回0。

# 最简单的程序

## • 最简单的规范程序（程序员视角）

第1行主函数的值是整型（int），为了与其它函数保持一致，这里不推荐省略。

```
int main()
```

int为类型，之后以空格隔开，main为函数名，函数名后以圆括号修饰。

```
{  
    函数头结束后应有换行，换行后是一对花括号，  
    括号内为函数体，其中的语句缩进四格书写。
```

```
    return 0;  
}
```

返回语句在关键词return后应有空格，接着是返回值，最后是分号。一般一条语句后应换行。

说明函数结束，缩进也在这里停止。

约定俗成，在主函数返回值0，表明程序正常运行。



# 最简单的程序

- 简化版的简单程序

- 程序员应当能阅读不规范的程序，但不要书写它。

```
main()
```

这里省略了函数返回值类型int，也可以写为void main()，只有主函数可以省略。

```
{  
  
}
```

主函数没有返回语句时，默认返回值为0。

# 括号、体和分块

- 花括号 ( Braces ) {} 分隔了 main() 函数
  - 函数内的代码称为函数体 ( body )
  - 花括号也可用于函数中，将代码集中到单位或块 ( block )

```
#include <stdio.h>
```

```
int main(void)
```

此处括号中的void是可以不写的

```
/* a simple program
```

```
*/
```

```
{
```

```
    int num;
```

```
    num = 1;
```

```
{
```

```
        printf("My favorite number is %d because it is first.\n", num);
```

```
}
```

```
    return 0;
```

```
}
```

左花括号是否单独一行历来存再争议，  
我提倡换行（除非打印稿为节约纸张）

```
called num */
```

```
num */
```

# 程序的缩进

- 程序的缩进一般在以下情况进行
  - 语句体：语句体内的语句较花括号缩进一级
    - 花括号与上一语句应为同级
  - 分支：if(-else), switch分支内语句较关键字缩进一级
  - 循环：for, (do-) while循环内的语句较关键字缩进一级
    - do-while语句的语句体比do和while缩进一级
  - 中间换行的语句或表达式新行较其第一行语句缩进一级

# 程序的缩进

- 语句较其上一级语句往内增加缩进4个字符
- 低一级语句整体结束后应将缩进恢复原状
- 函数体花括号与较函数头同级，内层语句缩进一级
- 除此之外，语句应与上一条语句缩进相同
- 掌握缩进的相关知识
  - 可以使用DevC++或VS的自动缩进功能，观察其缩进模式

# 简单的缩进示例

- 打印4次“Hi!”的程序

```
#include <stdio.h>
```

```
int main()  
{
```

函数体的花括号与函数头同缩进

```
    int i;
```

语句体花括号内的语句降一级，增加缩进四个空格

```
    for (i = 0; i < 4; i++)  
        printf("Hi!\n");
```

语句默认与上一条语句同级

循环语句内的语句降一级

```
    return 0;  
}
```

循环语句结束后恢复原有级别

# 简单的输出到屏幕程序

- 打印“Hi!”的程序

- 程序应有零个以上的输入，有一个以上的输出。

```
#include <stdio.h>
```

第1行预编译指令说明该行在编译之前运行。

```
int main()
```

在主函数和预编译指令之间，空一行，提高可读性。

```
{
```

```
printf("Hi!\n");
```

主函数调用（使用）了其它函数，将字符串打印到屏幕上。

```
return 0;
```

在完成独立的且完整的功能后，空一行，提高可读性。

```
}
```

# 简单的输出到屏幕程序

- 打印“Hi!”的程序

- 函数应该先声明，有定义（在自带的库中定义），再使用。

```
#include <stdio.h>
```

尖括号表明在默认路径下包含，包含stdio.h是因为它里面提供了printf函数的声明。

```
int main()
```

```
{
```

这里是函数的调用，圆括号内为主函数向被调用函数传入的参数。

```
printf("Hi!\n");
```

由于篇幅关系，课件或课本可能会压缩这些空行。

```
return 0;
```

灰色底色的方框内是控制台输出

```
Hi!
```

# 简单的输出到屏幕程序

## • 打印“Hi!”的程序

```
// A sample  
#include <stdio.h>
```

这是注释，双斜杠到行末的部分为注释，它在编译之前会被编译器删除，但一个好的注释可以帮助队友更好地理解你程序的意图。

```
int main()  
{
```

输出一整行之后一般要加上换行符，这是排版上的考虑。整体输出最后换行，下次输出时提示符可以出现在行首，便于阅读。

```
    printf("Hi!\n");
```

这是表达式语句，因为函数的调用是表达式。语句以分号结尾，而指令不能以分号结尾。

```
    return 0;
```

字符串以一对双引号为起止，\n表示之后的输出将换行。

此处句末有换行，但不会特别指出。

Hi!



# 说明：#include 指令与头文件

- #include是C预编译器指令的一个例子
  - 编译前对源代码进行一些预备工作，称为预编译
  - 作用：将文件完整内容复制到#include出现的位置
  - 预编译器指令以#开头，不是语句，不加分号
- 包含文件提供了一个共享信息的简便方法
  - 函数应先声明后使用，类似列方程前需要设未知数
  - 函数的声明往往由一组语句构成，很难默写；而记住文件名，则容易得多。

# 说明：#include 指令与头文件

```
#include <stdio.h> /* Standard I/O header */
```

```
int main(char *argv, int argc)  
{
```

```
    double v0 = 0; //initial velocity  
    double a = 0; //acceleration  
    double t = 0; //time  
    double s = 0; //shift  
    printf("Enter the Initial Velocity (m/s): ");  
    scanf("%lf", &v0);  
    printf("Enter the Acceleration (m/(s*s)): ");  
    scanf("%lf", &a);
```

头文件包含  
在这里展开

```
00000140: 00 01 11 00 00 00 00 44 3A 5C 44 65 73 6B 74 6F  
00000150: 70 5C 73 68 69 66 74 2E 6F 62 6A 00 3A 00 3C 11  
00000160: 00 22 00 00 07 00 12 00 00 00 0D 52 01 00 12 00  
00000170: 00 00 0D 52 01 00 4D 69 63 72 6F 73 6F 66 74 20  
00000180: 28 52 29 20 4F 70 74 69 6D 69 7A 69 6E 67 20 43  
00000190: 6F 6D 70 69 6C 65 72 00 00 00 00 45 6E 74 65 72  
000001A0: 20 74 68 65 20 49 6E 69 74 69 61 6C 20 56 65 6C  
000001B0: 6F 63 69 74 79 20 28 6D 2F 73 29 3A 20 00 00 25  
000001C0: 6C 66 00 45 6E 74 65 72 20 74 68 65 20 41 63 63  
000001D0: 65 6C 65 72 61 74 69 6F 6E 20 28 6D 2F 28 73 2A  
000001E0: 73 29 29 3A 20 00 00 25 6C 66 00 45 6E 74 65 72
```

```
.....D:\Desкто  
p\shift.obj...<  
..R..... R..  
.. R.Microsoft  
(R) Optimizing C  
ompiler...Enter  
the Initial Vel  
ocity (m/s): ...%  
lf.Enter the Acc  
eleration (m/(s*  
s)): ...%lf.Enter
```

```
00000000: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  
00000010: B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 D8 00 00  
00000040: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  
00000050: 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  
00000060: 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  
00000070: 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  
00000080: 4A EC B7 36 0E 8D D9 65 0E 8D D9 65 0E 8D D9 65  
00000090: D3 72 12 65 0D 8D D9 65 0E 8D D8 65 43 8D D9 65
```

```
MZ?.....??..  
?.....@.....  
.....  
.....?...  
?..?!.L?!Th  
is program canno  
t be run in DOS  
mode....$.  
J??6??e??e??e  
?re.??e??eC??e
```

Source Code (.c)

源代码

Compiler  
编译器

Object Code (.obj)

目标代码

Linker  
链接器

Executable Code (.exe)

可执行代码

# 带格式的输出生程序

- 为了在输出中控制格式，使用格式标识符

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("My favorite number is %d.\n", 1);
```

```
    return 0;
```

```
}
```

第4行在输出语句第一个参数字符串增加了格式标识符：%d ( decimal )

函数printf()的参数个数在声明时是不确定的，因此可以接受1个或多个参数，但一般函数的参数应是固定的。

第4行向printf()函数传入了2个参数，多个参数之间应以逗号间隔。

My favorite number is 1.

# 带格式的输出生程序

- 为了在输出中控制格式，使用格式标识符

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int num;
```

第4行增加整型变量num的声明语句。

声明语句的格式为，数据类型，空格，变量名，最后以分号结束。

```
    num = 1;
```

第6行增加变量num的赋值。赋值语句格式为变量名，赋值号(=)，值，最后以分号结束。

```
    printf("My favorite number is %d.\n", num);
```

```
    return 0;
```

第7行增加变量num的使用，将num的值(1)传给printf函数。

```
}
```

My favorite number is 1.

# 带格式的输 出程序

- 为了在输出中控制格式，使用格式标识符

```
#include <stdio.h>
```

```
int main()
```

```
{
```

变量名的命名应有物理意义

```
int num;
```

赋值表达式是表达式的一种，赋值表达式的格式为：  
变量名，单个等号，值。语句以分号结束。

```
num = 1;
```

```
printf("My favorite number is %d.\n", num);
```

```
return 0;
```

printf第一个参数是格式字符串，第二个参数起，每个参数值按顺序替换格式标识符。

```
}
```

My favorite number is 1.

# 声明 (Declarations)

- 变量应先声明再使用
  - 将所有变量声明放置一处，有利于他人阅读
  - 编译器更容易对内存进行管理
  - 声明变量有利于防范难以发现的输入错误

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int covnt = 3;
```

```
    printf("%d", count);
```

```
    return 0;
```

```
}
```

a.c:5:18: error: 'count' undeclared (first use in this function); did you mean 'covnt'?

绿色底色的方框中是  
编译器的输出信息

# 声明 (Declarations)

- 传统C语言要求所有变量在语句体的最开头做出声明
  - C99和C11借鉴C++允许声明语句放置于语句体的任何部分
    - 你的程序应尽量兼容旧版

```
int main() // traditional rules
{
    int doors;
    int dogs;
    doors = 5;
    dogs = 3;
    {
        int num;
        num = 42;
    }
    // other statements
    return 0;
}
```

```
int main() // traditional rules
{
    // some statements
    int doors;
    doors = 5; // first use of doors
    // more statements
    int dogs;
    dogs = 3; // first use of dogs
    {
        int num;
        num = 42;
    }
    // other statements
    return 0;
}
```

# 打印函数 printf()

- printf()带来新的用法：

```
printf("My favorite number is %d because it is first.\n", num);
```

- 打印时，变量 num 的值1代替了字符串中的符号组 “%d”
  - “%” 提醒程序打印一个值；“d” 说明应以十进制(decimal) 打印
- 调用（ call / invoke ）函数的实例
  - 当程序执行到这一行，控制权将传递给新的函数
  - 当函数中的代码执行完毕（遇到return语句或执行至最后一行），控制权交回调用它的函数（如：本例的main()）



```
// fathm_ft.c -- converts 2 fathoms to feet
```

程序以注释开始：文档(Documentation)

```
#include <stdio.h>
```

```
int main(void)
```

此处void表示空，强调main()  
函数不接受参数，可以不写。

```
{
```

```
    int feet, fathoms;
```

程序允许在一行声明多个  
变量，用逗号隔开。

```
    fathoms = 2;
```

```
    feet = 6 * fathoms;
```

乘法：查找变量fathoms的值，将  
其与6相乘并将结果赋值给feet

```
    printf("There are %d feet in %d fathoms!\n", feet, fathoms);
```

```
    printf("Yes, I said %d feet!\n", 6 * fathoms);
```

```
    return 0;
```

第一个%d 替换为格式字符串后的第一个参数 (feet)  
第二个%d 替换为格式字符串后的第二个参数 (fathoms)

```
}
```

```
There are 12 feet in 2 fathoms!  
Yes, I said 12 feet!
```

# 带有算式的输出程序

- 带有算式的输出程序

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
double num;
```

这里使用了双精度浮点型  
( double ) 来表示实数。

```
num = 2 * 2.5;
```

这里使用了算式

```
printf("My favorite number is %lf.\n", num);
```

```
return 0;
```

这里使用%lf显示浮点数num，%lf  
的显示格式为六位小数

```
}
```

My favorite number is 5.000000.

# 复杂些的程序

- 带有算式的输出程序

```
#include <stdio.h> /* Standard I/O header */
```

```
int main()  
{
```

```
    double v0 = 10; // initial velocity
```

```
    double a = 3;    // acceleration
```

```
    double t = 2;    // time
```

```
    double s = 0;    // shift
```

这里如果用1/2，结果为0

```
    s = v0*t + 1.0 / 2 * a*t*t; // main equation
```

```
    printf("The total shift is %.3lf m.\n", s);
```

```
    return 0; // return code 0: no error
```

%.3lf显示3位有效小数。  
%lf与double对应。

```
}
```

```
The total shift is 26.000 m.
```

# 接受键盘输入的程序

- 接受键盘输入的程序

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    double num;
```

```
    scanf("%lf", &num);
```

```
    printf("My favorite number is %lf.\n", num*2.5);
```

```
    return 0;
```

```
}
```

这里使用了scanf输入函数，输入函数也没有固定的参数个数，第一个%lf参数表明接收键盘输入并以双精度浮点型解析该输入，第二个参数的取地址操作符&表明，将其解析得到的值写入num所在的内存空间中。

# 带有循环的程序

## • 带有循环的程序

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int num;
```

```
    for (num = 0; num < 5; num++)
```

```
        printf("No. %d\n", num);
```

```
    return 0
```

```
}
```

for循环是循环的一种，它将初始状态、循环条件、状态改变写在同一行。以括号中的两个分号为界，for(A;B;C) D;的运行顺序是A;B;D;C;B;D;C;B.....在B环节为判断B是否成立，直到B不成立时结束。

循环语句，在for之后缩进四格；  
如果循环内容为多条语句，应写在一个语句体内，由花括号包含。

No. 0

No. 1

No. 2

No. 3

No. 4

```

#include <stdio.h> /* Standard I/O header */
int main(void)
{
    double v0 = 0;          // initial velocity
    double a = 0;           // acceleration
    double t = 0;           // time
    double s = 0;           // shift
    double t_start = 0;     // start time
    printf("Enter the Initial Velocity (m/s): "); scanf("%lf", &v0);
    printf("Enter the Acceleration (m/(s*s)): "); scanf("%lf", &a);
    printf("Enter the Time (s): "); scanf("%lf", &t);

    while (t >= t_start)
    {
        s = v0*t + 1.0 / 2 * a*t*t;    // main equation
        printf("The shift at %.1f sec is %.3lf m.\n", t, s);
        scanf("%lf", &t);
    }
    return 0;                  // return code 0: no error
}

```

篇幅所限，刪去一些換行。

while是循环语句中突出循环条件的形式。while (A) B; 的运行顺序是ABABA.....直到A不成立。

# 带有分支的程序

- 带有分支的程序

```
#include <stdio.h>
int main()
{
    int num = -2;

    if (num <= 0)
        printf("num is non-positive\n");

    printf("num is positive\n");

    return 0;
}
```

if分支是分支的一种。if (A) B; 的运行顺序是如果A成立则运行B。但B以后的语句不论A是否成立，都会被运行。

```
num is non-positive
num is positive
```

# 帶有分支的程序

## • 帶有分支的程序

```
#include <stdio.h>
int main()
{
    int num = 2;

    if (num <= 0)
        printf("num is non-positive\n");
    else
        printf("num is positive\n");

    return 0;
}
```

if-else分支是分支的一種，else與其最近的if相匹配。if (A) B; else C;的運行順序是如果A成立則運行B否則運行C。

if可以單獨使用，不帶else，但意義不同。

if和其後的表达式之間以空格間隔，表达式後換行，else的前後也換行。if和else後的語句縮進四格空格。

num is positive



```
#include <stdio.h> /* Standard I/O header */
```

```
int main(void) {  
    double v0 = 0; // initial velocity  
    double a = 0;  // acceleration  
    double t = 0;  // time  
    double s = 0;  // shift  
    printf("Enter the Initial Velocity (m/s): ");  
    scanf("%lf", &v0);  
    printf("Enter the Acceleration (m/(s*s)): ");  
    scanf("%lf", &a);  
    printf("Enter the Time (s): ");  
    scanf("%lf", &t);  
    if (t < 0) {  
        printf("Time cannot be negative.");  
        return -1; // error code -1: time<0  
    }  
    s = v0*t + 1.0 / 2 * a*t*t; // main equation  
    printf("The total shift is %.3lf m.\n", s);  
    return 0; // return code 0: no error  
}
```

程序员不应该放任程序输出错误的结果而不告知用户。在输入错误时，应及早告知错误。

# 带有数组的程序

## • 带有数组的程序

```
#include <stdio.h>
int main()
{
```

声明语句中在数组名后加上方括号修饰符，  
将声明一个数组，方括号内为数组长度

```
    double score[10], average = 0,
    int i;
    printf("Enter 10 scores: ");
    for (i = 0; i < 10; i++)
    {
        scanf("%lf", &score[i]);
        average += score[i];
    }
    printf("Average score: %.3lf\n", average / 10);
    return 0;
}
```

多个变量的基本类型相同，适用范围相近，在声明时用逗号隔开，成为变量列表。在声明时可以赋初始值。

除声明语句外，数组名后的方括号中数组下标，a[i]表示数组a的第i个元素。数组的第一个下标为0，最后一个下标为长度减一。

# 带有宏的程序

## • 带有宏的程序

```
#include <stdio.h>
#define SCORE_LENGTH 10
int main()
{
```

宏是一种预编译指令，它将文件中独立的宏名替换为宏的内容。格式为#define，空格，宏名，空格，宏的内容。

```
    double score[SCORE_LENGTH], average = 0;
```

```
    int i;
```

```
    printf("Enter %d scores: ", SCORE_LENGTH);
```

```
    for (i = 0; i < SCORE_LENGTH; i++)
```

```
    {
```

```
        scanf("%lf", &score[i]);
```

```
        average += score[i];
```

```
    }
```

```
    printf("Average score: %.3lf\n", average / SCORE_LENGTH);
```

```
    return 0;
```

```
}
```

使用宏可以使10有一个具备物理意义的命名。这样做也便于在复杂的程序中修改一些参数而不影响其它代码。

# 多个函数

- 如何定义自己的函数（除 main() ）并加到程序中
  - 函数 butler() 在程序中出现了三次
    - 声明；调用；实现

```
...  
void butler(void);           /* ANSI/ISO C function prototyping */  
...  
    butler();  
...  
void butler(void)           /* start of function definition */  
{  
    ...  
}
```

声明：通知编译器即将使用的函数名

调用：在main()函数中以函数调用的形式出现

定义：最后给出函数的定义，即函数本身的源代码

```
/* two_func.c -- a program using two functions in one file */  
#include <stdio.h>  
void butler(void);      /* ANSI/ISO C function prototyping */
```

函数应先声明，有定义，再使用。

```
int main(void)  
{  
    printf("I will summon the butler function.\n");  
    butler();  
    printf("Yes. Bring me some tea and writeable DVDs.\n");  
  
    return 0;  
}
```

void类型的函数可以没有return语句。  
在最后一个语句运行后，跳出该函数。

```
void butler(void)      /* start of function definition */  
{  
    printf("You rang, sir?\n");  
}
```

I will summon the butler function.  
You rang, sir?  
Yes. Bring me some tea and writeable DVDs.

# 目录

	1	C语言的历史
	2	C语言的基本语法
	3	C程序的可读性
	4	C程序设计实例
	5	总结

# 命名选择：合规规则

- 合规 ( valid ) 的命名包括
  - 小写字母、大写字母、数字、下划线 ( \_ )
  - 不可与保留字一致

合规命名	非法命名
wiggles	\$Z]**
Cat2	2cat
Hot_Tub	Hot-Tub
taxRate	tax□rate
_kcab	don't

## C语言的关键字

```
auto extern short while break
float signed case for sizeof
char goto static const if struct
continue inline switch default
int typedef do long union double
register unsigned else restrict
void enum return volatile
_Alignas _Alignof _Bool _Complex
_Generic _Imaginary _Noreturn
_Static_assert _Thread_local
```

# 命名选择：合规规则

- C命名大小写敏感（case sensitive），但不可混用
  - 大小写字母含义不同（stars与Stars、STARS不同）
- 命名长度
  - 传统编译器只识别前63个字符，现在编译器会重命名处理。
- 选取合适的变量或函数、文件名称具有重要意义
  - 它使得阅读代码的程序员能迅速掌握代码的意图
  - 错误或随意的命名迷惑或误导读者，队友不愿意和你合作
  - 思考合理命名的过程，也是理清编程思路的过程

但不应同时出现在同一段代码里。  
读者会混淆：是不是同一个东西？



# 命名选择：注意事项

- 使用物理意义命名（功能、意义、单位）

- 避免用注释来代替命名

```
int d; // eclapsed time, in day  
if (x[0] == 4)
```

```
int ecalpsedTimeInDays, daysSinceCreation, fileAgeInDays;
```

- 避免易混淆的命名，避免单独的0、I、o、l、@和1。

```
double XYZControllerForEfficientHandlingOfThings;  
double XYZControllerForEfficientStorageOfThings;
```

- 使用易于诵读的名称，避免汉语拼音或自创英文缩写

```
int GenDateYYMMDDHHIISS();  
int DtaRcrd102();  
int mianyangzhishu;
```

# 命名选择：注意事项

- 删除无意义的冠词或冗余词
  - 避免使用Data、Information、Value和Variable等词。
  - 避免flag、temp等词。

```
struct Product;  
int GetMoney();  
int sheepAmount;
```

```
struct ProductData, ProductInfo;  
int theSheepAmount;  
int dayValue, sizeVariable;  
char * nameString;  
int GetMoneyAmmount();
```

# 命名选择：注意事项

- 使用可搜索的名称，不要大量用数字常量，单字母

```
int e = 4;  
s += t[j] / 34 + 4;
```

```
int MAX_STRING_LENGTH = 100;
```

- 避免使用编码（匈牙利命名、成员前缀、接口和实现）

```
int XMUZipCode;
```

```
struct Point {  
    double PointX;  
    double PointY;  
};
```

- 勿用口语，禁用脏话

```
int taskToKill;
```

```
int taskToGoToHell();
```

- 少用解决方案领域的名称（读者是计算机专业人员）

# 注释 ( Comments )

- 封装于 `/* */` 符号的部分是注释 ( comments )
  - 此种情况下可以换行，但起始标记不可没有结束标记 ( `*/` )

```
/* This is a C comment. */  
/* This comment, being somewhat wordy, is spread over  
two lines. */  
/*  
    You can do this, too.  
*/
```

```
/* But this is invalid because there is no end marker.
```

- 使用 `//` 可以增加一行注释

```
// Here is a comment confined to one line.  
int rigue; // Such comments can go here, too.
```

# 注释

- 切勿轻易删除多行注释终止符号：“\*/”

```
/*  
    I hope this works.  
*/  
x = 100;  
y = 200;  
/* Now for something else. */
```

如果你删除了这一行，后面各行就被纳入注释范围了。

- 注释的目的

— 代码能为别人所看懂，吸引更多人改进代码

养成书写注释的习惯，  
从作业代码做起。

# 注释：要求

- 注释应在言之有物前提下尽量丰富
  - 尊重读者：太简单的代码不应注释，不应误导读者
  - 注释做什么和为什么，而不是怎么做
  - 不应用注释代替变量命名规范
- 需要和它人分享的函数应详细说明
  - 用途、输入参数列表及其作用、返回值类型及其意义等
  - 作者的姓名和联系方式、日期、版本号、版权声明

# 注释：注意事项

- 尽量不要对代码写注释
  - 不写注释的前提是让代码自己说话
  - 注释是弥补我们在用代码表达意图时遭遇的失败
  - 注释可能会撒谎（注释本身也是一种耦合）
- 注释不能美化糟糕的代码
- 用代码来阐述

```
// check to see if the employee is eligible for full benefits  
if ((employee.flags & HOURLY_FLAG) && (employee.age > 65))
```

```
if (IsEligibleForFullBenefits(employee))
```

# 注释：好的注释

- 法律信息

```
// Copyright (C) 2003-2012 ABCD Corporation  
// Released under the terms of the GNU General Pubic License
```

- 对意图的注释

```
// This is our best attempt to get a race  
// condition by creating large number of threads  
for ( i=0 ; i<2000 ; i++)
```



# 注释：好的注释

- 阐释

```
if (strcmp(name[i], name[j])>0) // name[i]>name[j]
```

- 警示

```
// NOTE: This function lasts one hour for huge array  
void slowSort(int array[])
```

- 待完成列表

```
// TODO: out of bound has not been checked  
void bubbleSort(int array[])
```

- 放大

```
// This repetition is really important. It removes  
// starting space  
strtrim(name);
```

- 为公共API写注释

# 注释：坏的注释

- 越说越乱问题越多

```
if (name==NULL)
    // null pointer means defaults are loaded
```

- 勿做多余的注释

```
// check password given the password
int checkPassword(const char * password);
int dayOfMonth; // the day of month
if (pointer) // normal
```

- 勿用注释代替命名

```
// return an instance of name being tested
const char * nameInstance;
```

```
const char * nameInstanceBeingTested();
```

# 注释：坏的注释

- 勿注释掉代码，代码完成后这类注释要清理

```
// int i = 0;
```

- 位置标记（代码会淹没在背景噪音中）

```
////////// Check the pointer here //////////  
if (pointer)
```

- 括号后的注释

```
if (pointer) {
```

- 不明显的联系

```
    ...  
} // if
```

- 和代码无关的注释
- 注释本身还需要注释

# 格式

- 格式影响可维护性和可扩展性
- 垂直格式（参考报纸）
  - 文件行数：不要太长
  - 用空白行隔开：函数间、声明后
  - 概念相关的代码应当相近
    - 紧密联系的函数、依赖的函数、变量和调用
    - 调整顺序，删掉多余的注释
  - 变量应当靠近使用位置（即在最小作用域）内声明

# 格式

- 橫向格式

- 尽量不要超过80个字符
- 空格字符将较弱的事物分开
  - 赋值操作符周围加上空白字符
  - 不要做没必要的对齐
- 缩进：不要因为程序短就不回车违反缩进规则
- 空范围
  - 除非空语句另起一行并处理缩进，否则很难引起察觉

# 格式：让你的程序可读的技巧

- 编程应可读注意可读性。

```
int main( void ) { int four;  
four  
=  
4  
;  
printf(  
"%d\n",  
four); return 0;}
```

```
int main(void) /* converts 2 fathoms to feet */  
{  
    int feet, fathoms;  
  
    fathoms = 2;  
    feet = 6 * fathoms;  
    printf("There are %d feet in %d fathoms!\n", feet, fathoms);  
    return 0;  
}
```

选用有意义的名字

使用注释

利用空行

每个语句一行

# 目录

1	C语言的历史
2	C语言的基本语法
3	C程序的可读性
4	C程序设计实例
5	总结

# 需求

- 题目

- 一辆汽车以一定的初始速度做匀加速直线运动，对给定的初始速度、加速度和时间，计算位移。

- 输入：初始速度、加速度和时间

- 样例输入：1 1 1

- 输出：位移

- 样例输出：1.5

需求方提出  
写在草稿纸上



# 算法设计

- 一辆汽车以一定的初始速度做匀加速直线运动，对给定的初始速度、加速度和时间，计算位移。
  - 初始速度( $v_0$ )：量化为  $\text{m/s}$
  - 加速度( $a$ )：量化为  $\text{m/s}^2$
  - 时间( $t$ )：量化为  $\text{s}$
  - 位移( $s$ )：量化为  $\text{m}$
  - 问题抽象成  $s = v_0 t + \frac{1}{2} a t^2$

第一步  
写在草稿纸上

# 代码草稿

- 问题抽象成  $s = v_0 t + \frac{1}{2} a t^2$
- 主要流程
  - $v_0$  <----- 键盘
  - $t$  <----- 键盘
  - $a$  <----- 键盘
  - $s = v_0 * t + 1/2 * a * t * t$
  - $s$  -----> 屏幕

第二步  
写在草稿纸上

# 位移公式

- 先写出主要公式

```
s = v0*t + 1.0 / 2 * a*t*t; //main equation
```


```
s = v0*t + 1.0 / 2 * a*t*t; //main equation
```

第三步  
输入开发环境里

# 位移公式

- 接着完善声明语句

```
double v0 = 0; //initial velocity  
double a = 0; //acceleration  
double t = 0; //time  
double s = 0; //shift
```



```
double v0 = 0; //initial velocity  
double a = 0; //acceleration  
double t = 0; //time  
double s = 0; //shift
```

```
s = v0*t + 1.0 / 2 * a*t*t; //main equation
```

第三步  
输入开发环境里

# 位移公式

- 补充输入及其提示语

```
double v0;
double a;
double t;
double s;
printf("Enter the Initial Velocity (m/s): ");
scanf("%lf", &v0);
printf("Enter the Acceleration (m/(s*s)): ");
scanf("%lf", &a);
printf("Enter the Time (s): ");
scanf("%lf", &t);

printf("Enter the Acceleration (m/(s*s)): ");
scanf("%lf", &a);
printf("Enter the Time (s): ");
scanf("%lf", &t);
```

```
s = v0*t + 1.0 / 2 * a*t*t; //main equation
```

第三步  
输入开发环境里

# 位移公式

## • 补充输出及其提示语

```
double v0 = 0; //initial velocity
double a = 0; //acceleration
double t = 0; //time
double s = 0; //shift
printf("Enter the Initial Velocity (m/s): ");
scanf("%lf", &v0);
printf("Enter the Acceleration (m/(s*s)): ");
scanf("%lf", &a);
printf("The total shift is %.3lf m.\n", s);
printf("Enter the time (s): ");
scanf("%lf", &t);

s = v0*t + 1.0 / 2 * a*t*t; //main equation
printf("The total shift is %.3lf m.\n", s);
```

第三步  
输入开发环境里

# 位移公式

## • 完成框架

```
int main(void)
{
    double v0 = 0; //initial velocity
    double a = 0; //acceleration
    double t = 0; //time
    double s = 0; //displacement

    printf("Enter the initial velocity (v0): ");
    scanf("%lf", &v0);
    printf("Enter the acceleration (a): ");
    scanf("%lf", &a);
    printf("Enter the time (t): ");
    scanf("%lf", &t);

    s = v0*t + 1.0 / 2 * a*t*t; //main equation
    printf("The total shift is %.3lf m.\n", s);
    return 0; //return code 0: no error
}
```

Diagram illustrating the code structure and flow:

- An arrow points from the `int main(void)` line to a callout box containing the function signature `int main(void)`.
- Another arrow points from the `return 0;` line to the same callout box, which also contains `return 0; //return code 0: no error`.

第三步  
输入开发环境里

# 位移公式

## • 包含标准头

```
#include <stdio.h> /* Standard I/O header */
```

```
int main(void)  
{
```

```
#include <stdio.h> /* Standard I/O header
```

```
double v0 = 0; //initial velocity
```

```
double a = 0; //acceleration
```

```
double t = 0; //time
```

```
double s = 0; //shift
```

```
printf("Enter the Initial Velocity (m/s): ");
```

```
scanf("%lf", &v0);
```

```
printf("Enter the Acceleration (m/(s*s)): ");
```

```
scanf("%lf", &a);
```

```
printf("Enter the Time (s): ");
```

```
scanf("%lf", &t);
```

```
s = v0*t + 1.0 / 2 * a*t*t; //main equation
```

```
printf("The total shift is %.3lf m.\n", s);
```

```
return 0; //return code 0: no error
```

```
}
```

第三步  
输入开发环境里



# 位移公式

- 调试、测试后，加入约束条件

```
#include <stdio.h> /* Standard I/O header */

int main(void)
{
    double v0 = 0; //initial velocity
    double a = 0; //acceleration
    double t = 0; //time
    double s = 0; //shift
    printf("Enter the Initial Velocity (m/s): ");
    scanf("%lf", &t);
    if (t < 0) //time cannot be negative
        return -1; //error code -1: time<0
    printf("Enter the Acceleration (m/s^2): ");
    scanf("%lf", &a);
    printf("Enter the Time (s): ");
    scanf("%lf", &t);
    if (t < 0) //time cannot be negative
        return -1; //error code -1: time<0
    s = v0*t + 1.0 / 2 * a*t*t; //main equation
    printf("The total shift is %.3lf m.\n", s);
    return 0; //return code 0: no error
}
```

第三步  
输入开发环境里

```

/*****
* Input the initial velocity, acceleration and time,
* calculates the shift, by the equation
*   s = v0*t + 1.0 / 2 * a*t*t;
* Author: Wei Huang <whuang@xmu.edu.cn>
* Version: 1.0 [Jun. 11, 2014]
*****/
#include <stdio.h> /* Standard I/O header */

```

```

/*****
*
* Input the initial velocity, acceleration and time,
* calculates the shift, by the equation
*   s = v0*t + 1.0 / 2 * a*t*t;
* Author: Wei Huang <whuang@xmu.edu.cn>
* Version: 1.0 [Jun. 11, 2014]
*****/

```

```

scanf("%lf", &t);
if (t < 0) //time cannot be negative
    return -1; //error code -1: time<0
s = v0*t + 1.0 / 2 * a*t*t; //main equation
printf("The total shift is %.3lf m.\n", s);
return 0; //return code 0: no error

```

```

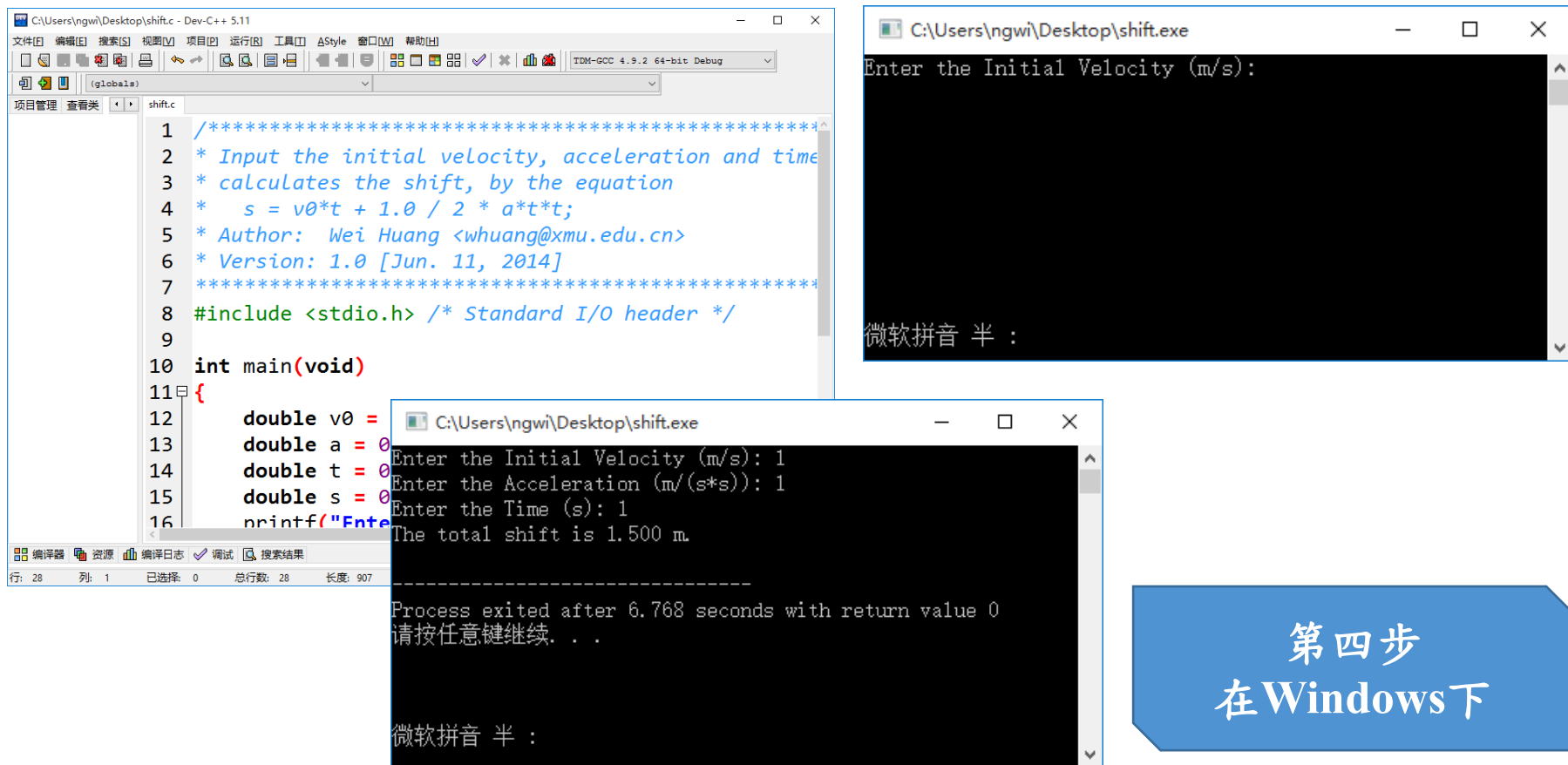
}

```

第三步  
输入开发环境里

# 编译并运行

- 输入到编译环境（IDE）中（例如：DevC++）



```
1  /*  
2  * Input the initial velocity, acceleration and time  
3  * calculates the shift, by the equation  
4  *  $s = v_0 * t + 1.0 / 2 * a * t * t;$   
5  * Author: Wei Huang <whuang@xmu.edu.cn>  
6  * Version: 1.0 [Jun. 11, 2014]  
7  */  
8  #include <stdio.h> /* Standard I/O header */  
9  
10 int main(void)  
11 {  
12     double v0 = 0;  
13     double a = 0;  
14     double t = 0;  
15     double s = 0;  
16     printf("Enter the Initial Velocity (m/s): ");  
17     scanf("%lf", &v0);  
18     printf("Enter the Acceleration (m/(s*s)): ");  
19     scanf("%lf", &a);  
20     printf("Enter the Time (s): ");  
21     scanf("%lf", &t);  
22     s = v0 * t + 1.0 / 2 * a * t * t;  
23     printf("The total shift is %.3f m.\n", s);  
24     return 0;  
25 }
```

Enter the Initial Velocity (m/s):  
微软拼音 半 :

Enter the Initial Velocity (m/s): 1  
Enter the Acceleration (m/(s\*s)): 1  
Enter the Time (s): 1  
The total shift is 1.500 m.  
-----  
Process exited after 6.768 seconds with return value 0  
请按任意键继续. . .  
微软拼音 半 :

第四步  
在Windows下

# 后续步骤

- 测试

- 实际输出与预期输出之间的审核或者比较过程。
- 为了发现软件在何种情况下无法得到预期输出。

- 调试

- 通过观察软件运行到特定位置的状态，找出程序错误、修补程序错误的过程。



# 目录

1	C语言的历史
2	C语言的基本语法
3	C程序的可读性
4	C程序设计实例
5	总结

# 总结

- C语言的历史
- C语言的基本语法
- C程序的可读性
- C程序设计实例

谢谢观看



廈門大學  
XIAMEN UNIVERSITY



信息学院 黄 焯  
(特色化示范性软件学院) 博士, 副教授  
School of Informatics Wei Huang