计算机程序设计

Computer Programming

C语言基础

主讲：李卫海

目录
CONTENTS

C语言的诞生

- 1970年，美国贝尔实验室的 Ken Thompson，以BCPL语言为基础，设计出B语言（取BCPL的首字母），并用B语言写了第一个UNIX操作系统
  - 为了玩游戏Space Travel（需要操作系统支持）

Dennis Ritchie (right)
Ken Thompson (left)

- 1972年，美国贝尔实验室的Dennis Ritchie改进B语言，最终设计出了一种新的语言，取了BCPL的第二个字母命名，即C语言
  - New B → C

C语言的标准化

- 1978年，Brian Kernighan和Dennis Ritchie著书“The C Programming Language”，该版本的C被成为K&R C
  - 不够严谨；新特性的出现
- 1989年，ANSI发布了第一个完整的C语言标准ANSI X3.159—1989，简称“C89”或“ANSI C”，1990年，C89被国际标准组织采纳，官方命名ISO/IEC 9899:1990，也通常被简称为“C90”或“ISO C”
- 1999年，发布新版ISO/IEC 9899:1999，简称“C99” ISO/IEC 9899:1999
- 2011年，发布新版，“C11” ISO/IEC 9899:2011
- 2018年，发布“C17” ISO/IEC 9899:2018

C语言的特点

- 是一种底层语言
  - 可以直接操作硬件
- 是一种小型语言
  - C本身支持的特性较少，依赖库的支持
- 是一种包容性语言
  - 提供更开放的自由度
  - 假定用户知道自己在做什么，执行较少的强制错误检查

C语言的优点

- 高效
  - 在有限内存空间里快速运行是它的“初心”
- 可移植
  - 支持可移植性
- 灵活，功能强大
  - 作为底层语言，拥有更多的自由度
  - 对程序员约束较少
- 标准库
  - 多年的发展形成了稳定的标准库
- 与UNIX系统的集成

C语言的缺点

- C程序更容易隐藏错误
  - 执行基本的词法、语法检查
  - 绝大多数语义层面问题需要程序员自己掌握
  - 不同编译器在一些模糊特性上采取不同的策略
- C程序可能难以理解
  - C语言是简明的、灵活的，允许特性结合使用的
- C程序可能难以修改
  - 不具有“类”“包”之类的封装特性
  - 大型程序的函数、变量之间关系复杂

编译与链接

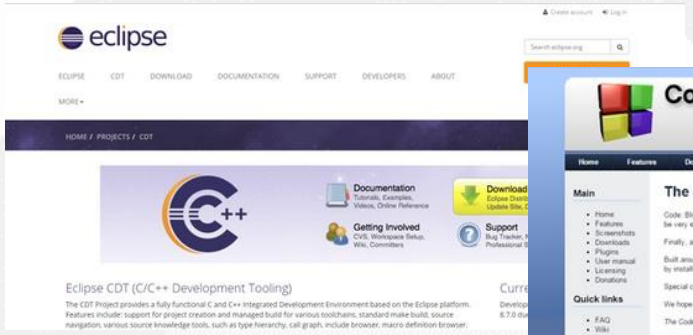
- 预处理过程对c源文件进行编辑，添加、修改代码
- 编译过程将c源文件翻译成机器的目标代码（汇编/机器代码）
- 链接过程将目标代码和其它需要补充的代码整合在一起，产生可执行程序

- 不同系统中，编译、链接等的指令不同
  - unix：% cc –o pun pun.c
  - linux：% gcc –o pun pun.c

◎ 集成开发环境

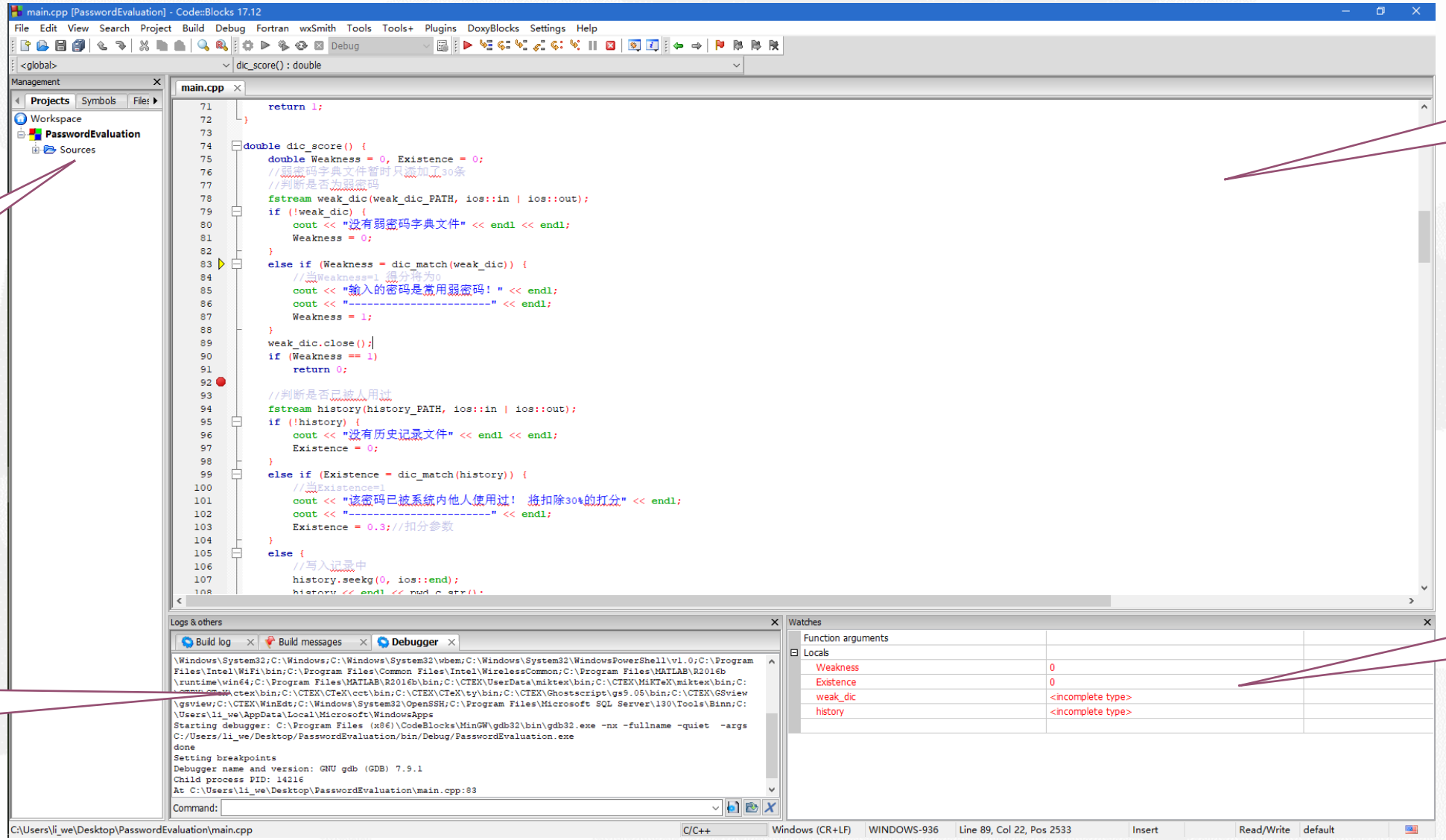
- IDE (Integrated Development Environment)
  - 将源程序编译、编译、链接、执行、调试等功能集成在一个软件包中

- 总有一款适合你
  - 深受C/C++程序员欢迎的11款IDE开发工具
  - C/C 开发者必不可少的15款编译器 IDE



Code::Blocks

- 免费、跨平台、线下



工程文件窗口

代码编辑窗口

编译信息窗口

调试监视窗口

集成开发环境

- 应当注意：
  - 不同的IDE平台，对同一个程序的编译执行结果可能不同
  - 不同的优化也可能会导致不同的运行结果
  - C对于某些运算的规定是模糊的，允许编译器的设计者自行决定

```
int i=1;
printf("%d, %d, %d, %d\n", i++, i++, i=2, i);
printf("%d\n", i);
```

- 代码中应尽量少使用模糊特性
  - 不要炫技

Hello, World

- C是一种“函数式”面向过程的语言
  - 将一个模块封装为一个函数，模块间的相互调用体现为函数的调用
  - main是一个C程序的主控函数，有且只能有一个
  - 使用{ }来标示函数的起始与结束

```
#include <stdio.h>
int main() {
        printf(“Hello, World\n”);
        return 0;
}
```

- 预处理指令
  - C提供标准库函数，也可以建立自定义的库
  - 本例使用了printf标准格式化输出函数，该函数定义在stdio标准库中
  - 标准库函数使用时需要用预处理指令告诉编译器使用了哪些标准库
- 语句
  - 函数中可以包含若干条语句
  - return语句用于结束函数并返回。main函数中的return将结束程序运行
  - 每条语句以“;”结束（复合语句除外）

另一个简单的程序

- 注释
  - 注释可以给出必要的文档说明，编译时被忽略
  - 标准工程文档对注释有严格的要求
    - 编写日期、作者、功能、使用方式等
    - 变量的说明、算法的说明、模块的说明
  - /* . . . */可用于书写若干行注释
  - // 将本行它后面的内容全部视为注释
- 善用缩进、空行、空格，便于阅读
  - 缩进可以展示出嵌套层次
  - 空行可以划分逻辑单元
  - 空格使各记号更加清晰
    - for (i=1, j=8; i<=10 && j>=3; i++, j++) 比 for(i=1,j=8;i<=10&&j>=3;i++,j++)看起来要舒服得多

```
/* 猴年马月狗日，张三
 * 程序功能：计算圆面积
 * 输入圆半径，输出面积
 */
#include <stdio.h>
#define PI 3.14159
main() {
        float r;                //圆的半径
        float area;             //圆的面积
        scanf("%f",&r);
        area=PI*r*r;
        printf("The area is %f\n", area);
}
```

另一个简单的程序

- 常量const
  - 程序运行期间，其值不允许改变的数据对象
  - 常量是一个固定的数值，如3.14159（字面值），可以为它起个名字（符号常量）如PI

    #define␣标识符␣字符序列
  - 此后在程序中出现PI时，代表的就是3.14159
  - 也可以为一个串（注意末尾没有“;”）命名，在预处理时，用此串替换所有该符号常量

```
#define abc 4 + 5
int main() {
    printf("%d", abc);
    return(0);
}
```

  - 命名常量的好处：便于阅读和统一修改

```
/* 猴年马月狗日，张三
 * 程序功能：计算圆面积
 * 输入圆半径，输出面积
 */
#include <stdio.h>
#define PI 3.14159
main() {
        float r;                //圆的半径
        float area;             //圆的面积
        scanf(“%f”,&r);
        area=PI*r*r;
        printf(“The area is %f\n”, area);
}
```

另一个简单的程序

- 变量variable
  - 程序一般是要计算的，计算就需要存储数据，变量就是用来一种数据的存储单元
  - 程序运行期间，其值可以改变的数据对象
  - C规定变量必须有类型

    类型␣变量名列表;
    - 类型规定了每种数据所占的内存空间大小，规定了内存中二进制数值的解释方式，约束了该种类数据的取值范围
    - 变量必须先声明，后使用
      - 标准C规定声明必须位于其它语句之前；C99之后无此约束
    - 变量在使用之前一般应当先赋初值
    - 可在定义变量的同时对变量赋值（初始化）
      - 例：int a, b=3;

```
/* 猴年马月狗日，张三
 * 程序功能：计算圆面积
 * 输入圆半径，输出面积
 */
#include <stdio.h>
#define PI 3.14159
main() {
        float r;                //圆的半径
        float area;             //圆的面积
        scanf("%f",&r);
        area=PI*r*r;
        printf("The area is %f\n", area);
}
```

标识符

- 为常量或变量起的名字，在C语言中称为“标识符”(identifier)
  - 变量名、常量名、函数名、类型名...
- 命名规范
  - 由字母、数字、下划线(Underscore, '_')组成
  - 第一个字符必须为字母或下划线（慎用）
  - C语言标识符是大小写敏感的（A和a是不同的名字）
- 用户定义的标识符不应与C语言关键字(Keywords)重复

ANSI C（32个）	auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while
C99增加5个	inline, restrict, _Bool, _Complex, _Imaginary
C11增加7个	_Alignas, _Alignof, _Atomic, _Static_assert, _Noretum, _Thread_local, _Generic

◎ 标识符

- 正确的标识符

  student、num1、_SUM、_1_2_3

  MyName、myname、my_name

- 错误的标识符

  room3-1　　包含了其他字符（减号）

  2men　　以数字开头

  long　　与关键字重复

  Mr.Wang　　包含了其他字符（小数点）

标识符命名规范

- 能表征含义 英文？拼音？
- 加前缀起到提示作用
  - 例如：cPI（c表示常数），nNum（n表示整数），pNode(p表示指针)
- 善用“_”，善用大小写

- 方便自己，方便他人，方便合作
- 大型企业都有自己的规范
- 不做硬性要求，但大家应养成习惯

◎ 整型

- 基本整型 int
- 短整型 short int（可简写为short）
- 长整型 long int（可简写为long）
- long long int（可简写为long long，C99支持）
- 修饰符
  - 有符号 signed（缺省）
  - 无符号 unsigned，例如 unsigned int i;
  - 有修饰符时，缺省为int。即signed=signed int
  - 虽然类型和修饰符的顺序可以任意，但还是按常规顺序比较有修养

◎ 整型

- 占据空间及取值范围

  sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)

  - 运算符sizeof(x)返回类型x（或变量x）占据内存的大小。注意 sizeof() 不是函数
  - 实际大小随CPU字长、操作系统字长、编译器不同而异
    - short不少于2字节，long不少于4字节，long long不少于8字节
  - 本课程约定：
    - short 2字节，int 4字节，long 4字节，long long 8字节
  - N字节无符号整数，大小范围$0\sim2^{8N}-1$
  - N字节有符号整数，大小范围$-2^{8N-1}\sim2^{8N-1}-1$

整型

整型常量的表示

- 十进制 (Decimal)
  - 不能以0开头
  - 例如，123、-456、0
- 八进制 (Octal)
  - 必须以0开头，不得含有8或9
  - 例如，0123、-0456
  - 错例，0138、-0912
- 十六进制 (Hexadecimal)
  - 以0x或0X开头，用a~f或A~F表示10~15
  - 例如，0x123、-0X45、0x3AB、-0xabc

整型

- 整型常量的类型后缀 (Suffix)
  - 没有后缀时，默认为 int
  - 当常数值超出范围时，根据大小按等级依次设置
    - 十进制设为 int, long, long long
    - 八进制和十六进制设为int, unsigned int, long, unsigned long, long long, unsigned long long
  - 通过后缀显式指定

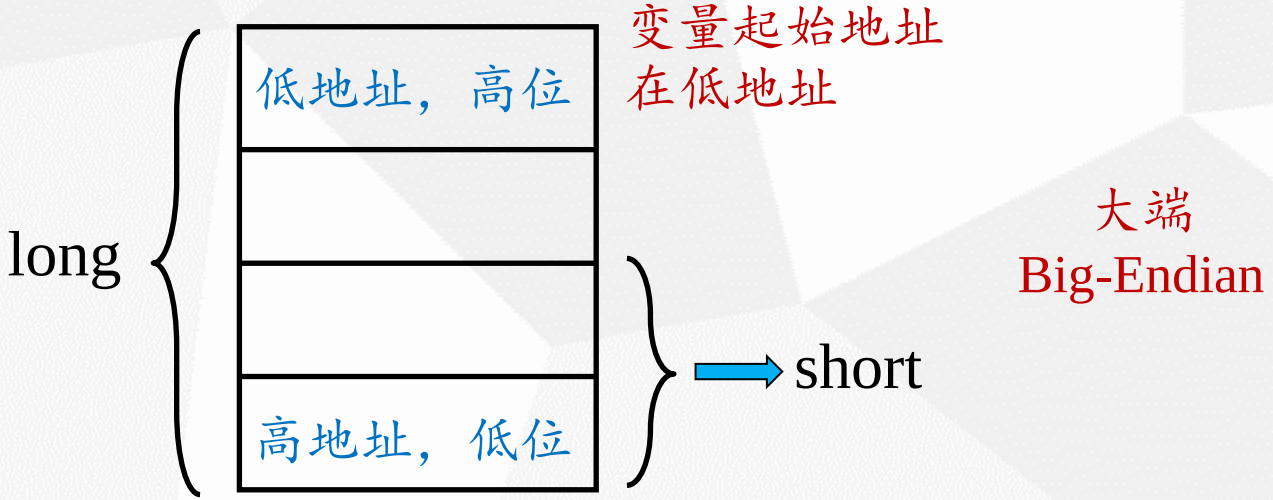
l、L	表示常量是long
u、U	表示常量是unsigned
ll、LL	表示常量是long long

两种后缀可以一起使用

举例	
123l	长整型常数123
456U	无符号整型常数456
789ul	无符号长整型常数789
101Lu	无符号长整型常数101

整型

- 整型赋值时的溢出与截断
  - 运算超出字节表示范围时，发生溢出
  - 无符号数与有符号数相互赋值时，符号位直接搬移
    - 整型赋值时，二进制数值直接搬移，不同类型进行不同的解读
  - 高精度向低精度赋值时，会发生截断
    - 常数赋值、计算时都可能发生



◎ 浮点型

- 单精度 float
- 双精度 double
- 长双精度 long double （C99新类型，大小随编译系统不同）

类　型	占据空间	有效十进制位数	绝对值范围参考
float	4字节	6 or more	0，10^{-38}~10^{38}
double	8字节	15 or more	0，10^{-308}~10^{308}
long double	8/10/12/16字节	19 or more	0，10^{-4932}~10^{4932}

◎ 浮点型

浮点型常量的表示

- 十进制小数形式
  - 由符号、数字和小数点(必须)组成
  - 整数和小数部分都可省略，但不能同时省略
  - 举例，12.3　　-.123　　123.　　0.　　.0
- 指数形式
  - 由整数或小数、e或E、整数顺序组成
    e或E之前必须有数字，之后必须是整数
  - 举例，123.4e-5表示123.4×10^{-5}, 1E1表示1×10^{1}
  - 错例，e1　12e　　1.2e3.4　.e5　　e
- 规范化指数形式
  - 类似于“科学计数法”，e或E之前的小数中，小数点前面有且仅有一个非零数字
  - 例如，123.456的规范化形式为1.23456e2、1.23456E+002

◎ 浮点型

- 浮点型常量的类型后缀 (Suffix)
  - 无后缀默认是double
  - 通过后缀显式指定
    - f、F 表示常量是 float，如1.23f
    - l、L 表示常量是 long double，如3.14159265L

◎ 浮点型

- 赋值过程中，可能因精度不同，出现舍入误差或溢出的情况
- 例如：

float x=123456789; x实际存入的值变成123456792

float x=1.234E123; x溢出，会显示成1.#INF00

```
float a=12345678900, b, x=12345.6;
b = a + 20; /*b=12345678920?*/
printf(“ a=%f\n b=%f\n x=%18.10f”, a, b, x);
```

运行结果> a=12345678848.000000
b=12345678868.000000
x=12345.5996093750

◎ 特殊常数

- 无穷大 (Infinition)

  Inf　　举例，1.0/0.0

  -Inf　　举例，-1.0/0.0

- NaN (Not a Number)

  NaN　　举例，sqrt(-1) /*开平方*/

  -NaN　　举例，-sqrt(-1)

- 如果没有溢出，整型/浮点常量的值总是非负的。如果在常量数值的前面出现了负号-，这个负号不是常量的一部分，而是作用于这个常量的负号单目运算符。例如，-345中，”345”是常数，”-”是负号单目运算符。

◎ 字符型

- C将字符类型视为一种整型类型，只占用一个字节的存储空间，存放的是字符的ASCII码
- 定义字符类型的主要作用是便于操作字符

- 字符数据的分类
  - 字符型：char
  - 有符号字符型：signed char，取值范围-128~127
  - 无符号字符型：unsigned char，取值范围0~255

char缺省是有符号还是无符号，取决于编译器

◎ 字符型

字符型常量的表示

- 单引号括起来的一个字符
- 例如：
  - 正确：'a’　　'A'　　'1'　　'␣'　　'?'
  - 错误：'abc'　　''　　'我'

- 字符常量赋值

char ch1='a', ch2='6', ch3=66;

signed char ch1=97, ch2=54, ch3=66, b=-127;

unsigned char ch1=97, ch2=54, ch3=66, b=255;

◎ 字符型

- 当使用（无符号）字符型数据时，与使用这个字符的ASCII码(一个8位整数)是完全一样的
- 字符型数据的用法与整型数据完全一样，包括运算、赋值、输入、输出以及溢出问题等
- 是字符还是整数，只有在输出时才有区别和意义
  - 如，char a='b'; 把a的值当成整数输出时显示为98，当成字符输出时显示为b
- 有时也把字符型数据成为字节型的整型数据

◎ 字符型

- 转义字符
  - 用以表示特殊字符，以 \ 开头

转义字符	字符值	输出结果
\'	一个单撇号(')	输出单撇号字符'
\"	一个双撇号(")	输出双撇号字符"
\?	一个问号(?)	输出问号字符?
\\	一个反斜线(\)	输出反斜线字符\
\a	警告(alert)	产生声音或视觉信号
\b	退格(backspace)	将光标当前位置后退一个字符
\f	换页(form feed)	将光标当前位置移到下一页的开头
\n	换行	将光标当前位置移到下一行的开头
\r	回车(carriage return)	将光标当前位置移到本行的开头
\t	水平制表符	将光标当前位置移到下一个 Tab 位置
\v	垂直制表符	将光标当前位置移到下一个垂直制表对齐点
\o、\oo 或\ooo 其中 o 代表一个八进制数字	与该八进制码对应的 ASCII 字符	与该八进制码对应的字符
\xh[h…] 其中 h 代表一个十六进制数字	与该十六进制码对应的 ASCII 字符	与该十六进制码对应的字符

◎ 表达式

- 表达式
  - 用运算符把操作数连接起来，并符合语法规则的式子
  - 操作数包括常量、变量、函数调用、表达式
  - 广义上，程序里几乎所有语句都可看作表达式
- 举例

  a+1

  C=B-1+sqrt(a)

  printf(“a=%d”,a)
- 关注：
  - 运算符的种类
  - 运算符的优先级
  - 运算符的结合方向

◎ 运算符的种类

- 算术运算符　+ - * / % ++ --
- 求字节数运算符　sizeof
- 其它运算符　()
- 赋值运算符　= += -= *= /= %= >>= <<= &= ^= |=
- 逗号运算符　,
- 关系运算符　> < == >= <= !=
- 条件运算符　?:
- 逻辑运算符　! && ||
- 位运算符　<< >> ~ | ^ &
- 指针运算符　* &
- 强制类型转换运算符　(type)
- 分量运算符　. ->
- 下标运算符　[]

◎ 运算符的优先级

- 运算符的运算次序按优先级由高到低执行
- 算术→关系→逻辑→条件→赋值
- 善用圆括号( )改变/指定运算符的执行次序

优先级	运算符	含义	要求运算对象的个数	结合方向
12	||	逻辑或运算符	2（双目运算符）	自左至右
13	? :	条件运算符	3（三目运算符）	自右至左
14	= += -= *= /= %= >>= <<= &= ^= |=	赋值运算符	2（双目运算符）	自右至左
15	,	逗号运算符（顺序求值运算符）		自左至右

优先级	运算符	含义	要求运算对象的个数	结合方向
1	( )	圆括号		自左至右
	[ ]	下标运算符		
	->	指向结构体成员运算符		
	.	结构体成员运算符		
2	!	逻辑非运算符	1（单目运算符）	自右至左
	~	按位取反运算符		
	++	自增运算符		
	--	自减运算符		
	-	负号运算符		
	(类型)	类型转换运算符		
	*	指针运算符		
	&	取地址运算符		
	sizeof	长度运算符		
3	*	乘法运算符	2（双目运算符）	自左至右
	/	除法运算符		
	%	求余运算符		
4	+	加法运算符	2（双目运算符）	自左至右
	-	减法运算符		
5	<<	左移运算符	2（双目运算符）	自左至右
	>>	右移运算符		
6	< <= > >=	关系运算符	2（双目运算符）	自左至右
7	==	等于运算符	2（双目运算符）	自左至右
	!=	不等于运算符		
8	&	按位与运算符	2（双目运算符）	自左至右
9	^	按位异或运算符	2（双目运算符）	自左至右
10	|	按位或运算符	2（双目运算符）	自左至右
11	&&	逻辑与运算符	2（双目运算符）	自左至右

运算符的操作数

按操作数的个数分为单目、双目、三目运算符

- 单目运算符：
  - 对一个数进行操作
  - 如 -125（负号）等
- 双目运算符：
  - 对两个数进行操作
  - 如 3-2（减号），a=‘0’（赋值）
- 三目运算符：
  - 对三个数进行操作
  - 唯一的三目运算符：条件运算符 ?:

运算符的结合方向

- 表达式中有多个相同优先级的运算符时的运算顺序

  +(正号) -(负号)：从右向左（右结合）

  +(加) -(减) * / %：从左向右（左结合）

  如3+5*2/4-6，*/优先级高，先算，*在左，先算

- 单目运算符除++(后缀)、--(后缀)外都是右结合（操作数在右边，也不存在连续多个存在的情况）
- 双目赋值运算符都是右结合
- 三目运算符?:是右结合
- 其它都是左结合

- 如何理解a++b？ 是a+(+b)？ 错

◎ 算术运算符

运算符	名称	优先级	举例	说明
+	正值运算符(正号)	2	+b	值不变
-	负值运算符(负号)	2	-d	符号取反
+	加法运算符	4	a+3	加法
-	减法运算符	4	c-4	减法
*	乘法运算符	3	a*3	乘法
/	除法运算符	3	c/d	除法
%	模运算符	3	e%4	求余数、取模

算术运算符

类型转换

- 以两个操作数中范围更大的类型作为结果的类型
  - 隐式类型转换
  - 两个整数相除，结果也为整数
    - 直接舍去商的小数部分（向下取整），不“四舍五入”
    - 如果需要得到精确的运算结果，可以把其中一个操作数改为浮点数

      5/3(=1)、5.0/3(=1.66666…)
    - 负数整数除法，结果的舍入方向没有规定

      -5/3(=-1?-2?)
- 求余运算符的两个操作数都必须是整数
  - 运算结果的符号与左操作数相同
  - 举例， -5%3=-2 5%-3=2

◎ 算术运算符

自增、自减运算符：

- 形式

  ++i, --i　　先计算，后取值

  i++, i--　　先取值，后计算

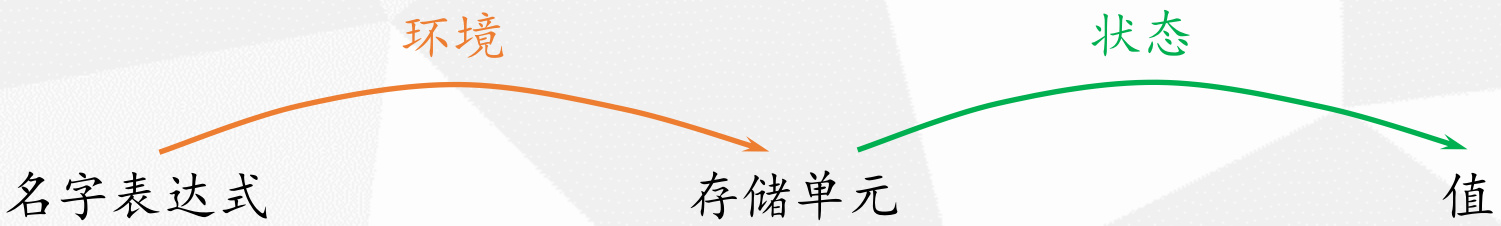
- 说明
  - 由于操作的结果将改变操作数的值，因此其操作数只能是左值表达式
  - 左值：其内容可以被改变，可看做变量
  - 标准C中，自增、自减的结果都不是左值(与C++不同）

    ++(++i)、(i++)++错误

◎ 算术运算符

左值的概念（left value，lvalue)

- 一种表达式，它可以作为赋值运算符的左操作数
- 左值表达式具有存放数据的空间，并且存放总是被允许的
- 左值可以是变量等，有明确地址的表达式
- 常量、无明确空间地址的表达式不能作为左值



- 运行时函数的调用改变环境，赋值改变状态

算术运算符

自增、自减运算符例子

```
int i=3, j, a, b=1, c=2;

j = ++i;                    //正确 i=i+1, j=i, i=4, j=4
j = i++;                    //正确 j=i, i=i+1, i=5, j=4
j = -i++;                   //正确 -(i++), i=6, j=-5
j = i++*2;                  //正确 (i++)*2, i=7, j=12
a = (b+c)++;                //错误，不是左值!
a = 34++;                   //错误，不是左值!
j = ++i++;                  //错误，不合语法!
j = a+++b+++c++;            //正确j=a++ + b++ + c++
j = a++++b;                 //错误 j=a ++ ++ b
j = a+++ +b;                //正确 j=a ++ + +b
```

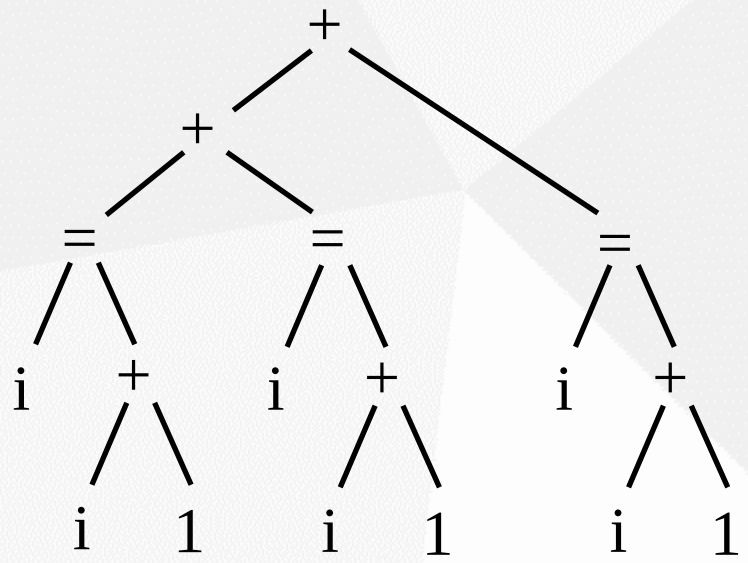
C编译程序自左向右尽可能多的将若干字符组合成一个运算符、标识符、关键字

算术运算符

- 建议不要在一个表达式内对同一个变量混合进行取值、自增、自减运算

```
i=3;
j=(++i)+(++i)+(++i);
```

j=?



如果先将i读入寄存器，计算的中间结果保存在寄存器中，结果应该是 j=15

如果机器有INC指令的话，极可能产生一条INC指令来完成++i。上层计算对它的引用落在另一个++i 的计算的后面。结果是 j=5+5+6=16

C规定了自增自减与取值的顺序，但未严格规定“后”到什么时候

◎ 长度运算符 sizeof()

- 获取操作数对象占据内存的大小

- 两种书写形式

  sizeof(类型名称)，如sizeof(int)

  /* 产生该类型的数据对象将占据内存空间的字节数*/

  sizeof(表达式)，如sizeof(a+2.3)

  /*分析表达式的类型后确定空间大小，但不求值*/

- sizeof后面的括号可以省略
  - sizeof(a) 和 sizeof a 一样
  - 但必须注意运算优先级
    - 设 int a，sizeof(a+2.3)得到4，sizeof a+2.3得到6.3

◎ 赋值(Assignment)运算符

- 基本赋值运算符：=
  - 左操作数只能是左值
- 复合算术赋值运算符：+= -= *= /= %=

  左值(左操作数) op = 表达式(右操作数)

  等价于 左操作数 = 左操作数 op 右操作数

  如 a+=1 等价于 a=a+1
- 简单形式也称为赋值语句：变量 赋值运算符 表达式;
- 所有赋值运算符优先级相同，都是右结合
- 赋值表达式不是左值

  例如，(a=3)=4错误

赋值(Assignment)运算符

- 例子

```
int a, b, c;
double x=2.6, y=2.0, s=3.0, t=1.0;

a=5;                    // 表达式的值为5
a=b=c=1;                // a=(b=(c=1))
a=(b=4)+(c=3);          // b=4, c=3, a=7
a+=a*=b+2;              // a=a+(a=a*(b+2)); 84
x+=5.0;                 // x=x+5.0; 7.6
y*=s+t;                 // y=y*(s+t); 8.0
```

如何理解？第一个加号如果按从左到右计算，则a值尚未修改

当该变量被多次引用时，不在子表达式中赋值，

不要考验编译器，不要给自己和他人制造麻烦

◎ 逗号运算符

exp1, exp2

- 逗号运算符的优先级最低，“自左向右”结合
- 逗号表达式的值是exp2的值
- 把多个表达式组合成一个表达式使用
- 举例

```
x=(a=3, 6*a)        /* a=3, x=18, 表达式的值为18 */
x=a=3, 6*a          /* a=3, x=3, 表达式的值为18 */
```

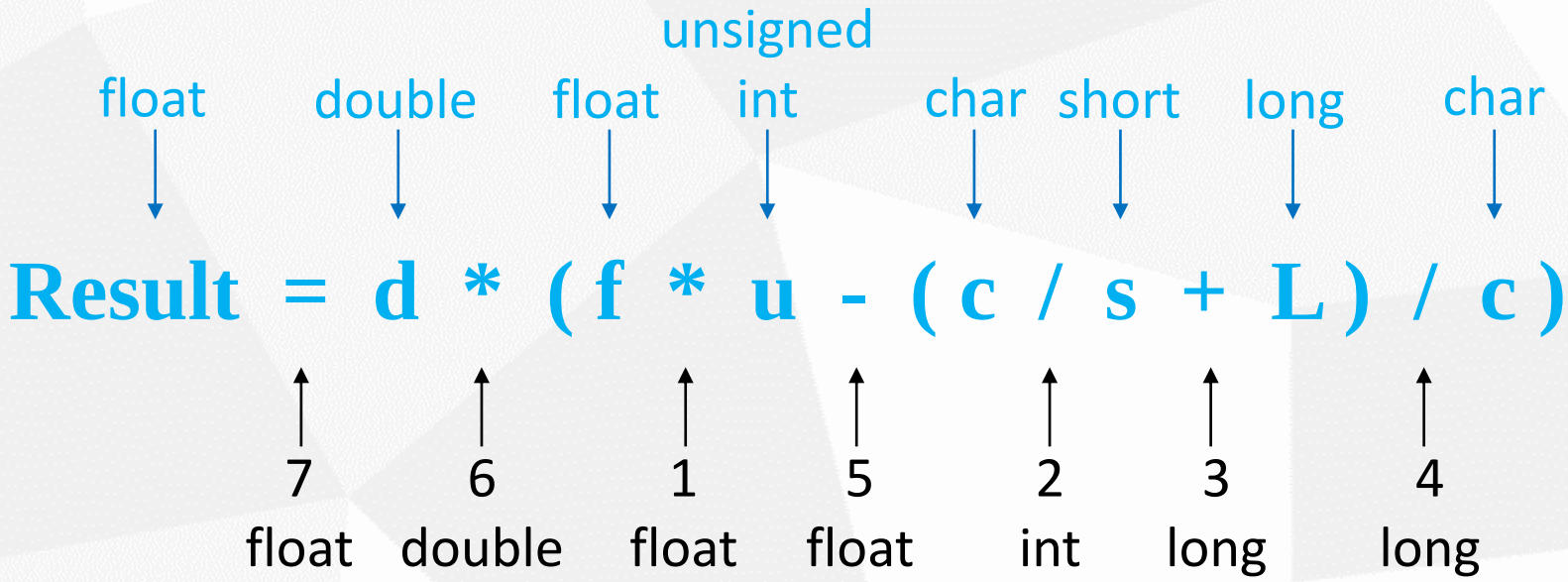
- 主要是方便特殊情况下简化代码

表达式的类型

- 表达式的类型就是表达式的值的类型，取决于操作数的类型以及所做的运算
- 隐式类型转换
  - 当操作数类型不同时，编译器将尝试隐式类型转换
  - 赋值操作时，右值被隐式转换成左值类型
  - 函数调用时，实参类型隐式转换成形参类型，返回值类型隐式转换成函数类型隐式
  - 算术运算时，随着逐个运算符的运算进行，参与运算的操作数转换成相同类型
    - 朝表达数据能力更强的方向进行转换
    - int → unsigned int → long → unsigned long → float → double → long double
    - short和char被转换成int
    - 如果一个是long，另一个是unsigned int，分两种情况：
      - 若long范围包含unsigned int，则转换为long
      - 若long不能包含unsigned int，则转换为unsigned long

◎ 表达式的类型

- 隐式类型转换例



◎ 表达式的类型

- 表达式的类型也可以强制转换（显式类型转换）
  - 形式：(类型名)操作数/表达式
    - 强制类型转换的优先级比较高（括号运算符）
    - 强制类型转换不会改变操作数自身的类型，只是得到一个所需类型的中间量
    - 多用于隐式类型转换不能实现的目的，或为了增强程序可读性

例子

```
float x=2.8, y=3.7, z;
int a, b, d=5, e=2;
a=(int)x+y;    // x=2.8, 2+3.7 => 5.7,a=5
b=(int)(x+y); // (int)(6.5), b=6
z=(float)d/e;  // 5.0/2, z=2.5
z=(int)y/e;    // 3/2, z=1.0
```

表达式的类型

- 类型转换时数值的变化
  - 无符号整数与有符号整数类型转换时，直接复制，不对补码和符号位做特殊处理
  - 短整数转换为长整数时，低位同上处理；若转换前后均为有符号整数，则高位采用符号位扩展，否则高位补零
  - 长整数转换为短整数时，直接截断
  - 涉及到浮点数转换时，进行数值的处理。受精度影响需要舍入时，策略各编译器不同

基本输入输出

- 输入输出库函数
  - 格式化输出函数　　printf()
  - 格式化输入函数　　scanf()
  - 字符输出函数　　putchar()
  - 字符输入函数　　getchar()
- 它们都定义在stdio标准库中，使用时，需要在源文件前面添加

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

◎ 格式化输出库函数printf()

- 函数调用格式

  printf(“输出格式串”, 输出项表列);

  - 按指定格式输出若干项数据（表达式的值）
  - 输出格式串的组成：格式控制字符、转义字符、普通字符

```
int a=3, b=4;
printf(“%d␣b=%6d\n", a, b);
```

- 常用格式控制字符

  %d 输出十进制整数　　%f 以小数形式输出实数

  %c 输出单个字符　　.....

- 数据输出宽度

  %6d输出6位整数 %18.10f输出总计占18个字符位置（含小数点）的浮点数，保留10位小数（四舍五入）

```
float x=12345.6;
printf(“x=%18.10f\n”, x);
```

输出结果：x=␣␣12345.5996093750

◎ 格式化输出库函数printf()

- 对齐标识符（部分）：
  - – 输出结果在指定宽度内左对齐，右边填空格
  - + 输出结果在指定宽度内右对齐，左边填空格

```
int a=3;
printf(“%-6d\n", a);
```

  - 若实际数据位小于宽度则补空格，大于宽度则按实际宽度输出，缺省宽度说明按实际宽度输出
- 转义字符
  - 用于输出控制代码和特殊字符，以 \ 打头
  - 如 \n 表示换行
  - 为输出特殊字符 \' \" ，应使用 \\ \' \"
  - 特例：为输出 % 应使用%%。若%与后面的内容不构成控制字符，也可以直接用%
  - 当 \ 与后面字符不构成转义字符时，\ 被忽略
  - \%d是什么意思？

◎ 常用输出格式字符

格式字符	说 明 与 示 例
d i	输出一个十进制整数 int x=1; printf(“%d”,x);
f	输出一个小数形式的单、双精度数（6位精度） float x=1.0; double y=3.14; printf(“%f %f”,x,y);
c	输出一个ASCII码字符 char x=‘a’; printf(“%c”,x);
s	输出字符串，直到遇到'\0'或达到指定宽度 char x[10]=“abcde”; printf(“%s”,x);

◎ 其它输出格式字符

格式字符	说　明
o	无符号八进制整数，不输出前导0
x, X	无符号十六进制整数，不输出前导0x或0X
u	无符号十进制整数
e, E	以规范化指数形式输出浮点数，缺省精度为6位小数
g, G	以f和e(E)两种格式中较短的一种输出
h	有符号或无符号短整型
l	有符号或无符号长整型
L	长双精度型

◎ 输出格式前缀

格式前缀	说　明
m	一个正整数，输出数据的最小宽度
n	一个正整数，输出的小数位数或字符串截取个数
.	分隔m和n
-	输出的结果向左靠齐
+	输出的结果向右对齐，且总带符号，即正号也输出
0	输出的数字不满规定宽度时，用前导0填充
␣	有符号数字如果没有输出符号，则前面加一个空格
#	八进制和十六进制数加前缀，浮点数保证小数点存在
*	取代m或n，其值用实参传入。例printf(“%*.*f”, 10, 5, a);

◎ 格式化输出库函数printf()

- 例子

```
int i=1234;
float f=-56.78;
printf("i=%+6d", i);          /* "i=␣+1234" */
printf(“%c”, i);              /* “π” */
printf("%#08x", i);           /* "0x0004d2" */
printf("%10.5E", f);          /* "-5.67800E+01" */
printf("%7.4f%%", f);         /* "-56.7800%" */
printf("%+6.3G", f);          /* "␣-56.8" */
printf("%-6.3s", "Hello");    /* "Hel␣␣␣" */
```

不确定

.3指有效数字

输出占6位，
取前3个字符

◎ 格式化输出库函数printf()

- 通常，我们要求输出项表列与格式控制字符一一对应
  - 类型不匹配时，会怎样?
  - 个数不匹配时，会怎样?

```
#include “stdio.h”
void main()
{ int i=1, j=2, a=3, b=4;
  printf(“%d,%d,%d,%d,%d\n”);
  printf(“%d,%d,%d,%d,%d\n”, i);
  printf(“%d,%d,%d,%d,%d\n”, i, j);
}
```

◎ 格式化输入库函数scanf()

- 函数调用格式

  scanf(“输入格式串”，输入项地址表列);

  例：int a; scanf(“%d”, &a);

  - 接收用户从键盘输入的数据，并按照格式控制符的要求进行类型转换后送入地址表列中对应的变量存储单元中
  - 输入格式串的组成基本与printf()相同
  - &是取地址操作符，用来获取变量所占存储空间的首个字节的内存地址
  - 输入格式串中，可以指定输入数据的宽度（但不能指定小数点的位数），系统自动按照此宽度截取读入的数据

◎ 格式化输入库函数scanf()

- 函数调用格式

  scanf(“输入格式串”，输入项地址表列);

  例：int a; scanf(“%d”, &a);

  - 若输入格式串中加入了格式符以外的其它字符，则输入时必须同样输入，否则结果不确定；若无其它字符，则可用空格、回车或制表符<Tab>作为间隔标记
    - *间隔标记个数不限，输入时也不限（包括0个）*
  - 使用“%c”时，输入的字符不能加间隔标记
  - 输入数据遇到空格、回车或制表符<Tab>以及其它各种非法输入时，认为该项数据输入结束。注意，这些输入并没有被跳过去
  - 当遇到格式符%*时，表示跳过该输入数据项不读入
  - 输入数据比表列更多时，会被自动用于下一次输入
    - 有一个缓冲区用于暂存输出内容

◎ 常用输入格式字符

格式字符	说 明 与 示 例
d i	输入一个十进制整数。i还可匹配8/16进制数
	int x; scanf(“%d”, &x);
f	输入一个小数形式的单精度浮点数
	float x; scanf(“%f”, &x);
lf	输入一个小数形式的双精度浮点数
	double y; scanf(“%lf”, &y);
c	输入一个字符，包括空格
	char x; scanf(“%c”, &x);

◎ 其它输入格式字符

格式字符	说　明
i	有符号整数，可以是八进制(带前导0)或十六进制(带前导0x或0X)
o	有符号八进制整数，可以带或不带前导0
x, X	有符号十六进制整数，可以带或不带前导0x或0X
u	无符号十进制整数
f, e, E, g, G	浮点数，可以用小数形式或指数形式

◎ 输入格式前缀

格式字符	说　明
h	有符号或无符号短整型
l	有符号或无符号长整型或双精度型
L	长双精度型
m	一个正整数，输入数据的最大宽度
*	需要输入数据，但是在赋值时被跳过。用于从格式文件中选择性读入某些字段

◎ 字符输出函数putchar()

- 函数调用形式

  putchar(字符或整数);

  - 输出一个字符，范围是ASCII码表
  - 整数若超出字符型数据范围，则只使用低字节
  - 示例

```
putchar('a');          // 输出字符'a'
putchar('\n');         // 输出换行符
putchar(101);          // 输出字符'e'
putchar('\101');       // 输出字符'A'
putchar(1000);         //低字节是232，输出字符'è'
```

◎ 字符输入函数getchar()

- 函数调用形式

  [整型/字符型变量 =] getchar();

  - 每执行一次接收从键盘输入的一个字符
  - 只有输入回车后，函数才会执行完
  - 示例

```
int c;
c = getchar();                  // 输入'abcd'后回车
putchar(c);                     // 输出'a'
putchar(getchar());             // 输出'b'
```

  - 输入数据多于一个字符时，会被自动用于下一次输入
  - 回车也会被认为是一个字符。当getchar()之前曾有过一次scanf()时，至少会留下一个回车符被getchar()获取

◎ 例1：输入球体半径，求球的体积 $V = \frac{4}{3}\pi r^3$

```
#include <stdio.h>
#define PI 3.1415926
void main() {
    float r, V;
    printf("r=");
    scanf("%f", &r);
    V = 4.0 / 3 * PI * r * r * r;
    printf("V=%f\n", V);
}
```

◎ 例2：输入大写字母，输出为小写字母

```
#include <stdio.h>
void main() {
    char cup, clow;
    cup = getchar();
    clow = cup - 'A' + 'a’;
    printf("%c\n", clow);
}
```

- 一个人静静坐在电脑面前写代码的感觉，那是什么感觉？那是武林高手闭关修炼的感觉。

- 真正的程序员的程序未必会在第一次就正确运行，但是他们愿意守着机器进行若干个30小时的调试改错。

作业

A1. 编写程序，验证 short 型数据和 unsigned short 型数据互相赋值时的规则。

- 提示，将合适的常数赋给一个变量，输出该变量的值

A2. 编写程序，验证高精度无符号整型数向低精度有符号整型变量赋值时的规则。

A3. 编写程序，验证高精度有符号整型数向低精度无符号整型变量赋值时的规则。

A4. 编写程序，验证你所用的编译器将char型视为有符号数还是无符号数。