第2章 基本词法语法规则与程序元素

华中科技大学计算机学院
甘早斌

目 录 6学时

2.1 字符及词法元素

2.1.1 字符集

- C程序是一个字符序列，字符序列先被分解为称之为记号（token）的词法元素，再根据语法规则检查这些记号组合是否合法。
- C语言字符集为：
  - 英文字母：a~z 和A~Z
  - 数字字符：0~9
  - 特殊字符：！ “ # % & ‘ ( ) * + ， - . / :
    ； < > = ? [ ] \ ^ _ { } | ~
  - 空白字符：空格、换行符、水平制表符（HT）、
    垂直制表符（VT）、换页符（FF）

三字符序列

- 有些国家的字符集不包括上述所有特殊字符。标准C语言定义了一组三字符序列，使C语言程序可以用ISO 646-1083不变代码集编写，这是7位ASCII代码集的子集，是许多非英语国家字符集公用的代码集。
- 以两个连续问号开头（教材表2-1）。所有的三字符序列都要用相应的单个字符替换，这种替换发生在其他任何处理之前。
- 例如，
  - int a??(4??)=??<0??>; 被替换成
  - int a[4]={0};

2.1.2 词法元素

- 词法元素称为记号，记号是程序中具有语义的最基本组成单元。记号共分5类：
  - 标识符
  - 关键字
  - 常量
  - 运算符
  - 标点符号
- 编译器从左至右收集字符，总是尽量建立最长的记号，即使结果并不构成有效的C语言程序。
- 相邻记号可以用空白符或注释语句分开。

词法分析举例

- 例2.1 sum=x+y
  - 分解成sum、=、x、+和y 共5个记号。
- 例2.2 int a，b=10，
  - 分解成int、a、，、b、=、10和； 共7个记号
- 例2.3 x+++++y
  - 分解成x、++、++、+、y 共5个记号,这是无效的C语法
  - 但可组合成x++ + ++y， 这是有效的C语法，遗憾！系统不会这样分解
- 这是一个非常重要的知识点

2.3 标识符、关键字及分隔符

- 2.3.1 标识符
- 2.3.2 关键字
- 2.3.3 分隔符

2.3.1 标识符

- 标识符是用来标识用户定义的常量、变量、数据类型和函数等名字的符号。其命名规则:
  - 以一个字母(a~z, A~Z)或下划线(_)开头，后跟字母、下划线或数字(0~9)
  - year， #123， .COM， Day ，Win3.2 ， $100， ATOK， 1996Y，x1，_CWS ，1_2_3 ，_change_to （?)
  - 不能把C语言关键字作为标识符，例如if, define, for等，也要避免使用C程序库中函数和常量的名称，例如 scanf。
- 这是一个重要的知识点

2.3.1 标识符

- 注意:
  - 标识符对大小写敏感，即区分大小写。如：Book≠book
  - 标识符长度是由机器上的编译系统决定的，一般的限制为8字符（注：8字符长度限制是C89标准，C99标准是32个字符，其实大部分工业标准都更长）。
  - 良好的编程风格是选择有助于记忆且有一定含义的标识符，这样可增强程序的可读性和程序的文档性。
  - 标识符命名的良好习惯——见名知意:
    - 见名知意是指通过变量名就知道变量值的含义。通常应选择能表示数据含义的英文单词（或缩写）作变量名，或汉语拼音字头作变量名。
    - 例如：name/xm (姓名)、sex/xb (性别)、age/nl (年龄)、salary/gz (工资)

2.3.2 关键字 1/4

- 关键字也称保留字 (Why?)
  - 关键字是被系统赋予特定含义并有专门用途的标识符
  - 关键字不能作为普通标识符给变量、函数或标号等命名，但可以作为宏名（Why? ）
  - 关键字都是采用小写
- ANSI C标准C语言共有32个关键字:
  - 1. auto： 声明自动变量
  - 2. short： 声明短整型变量或函数
  - 3. int： 声明整型变量或函数
  - 4. long： 声明长整型变量或函数

2.3.2 关键字 2/4

- 5. float：声明浮点型变量或函数
- 6. double： 声明双精度变量或函数
- 7. char： 声明字符型变量或函数
- 8. struct：声明结构体变量或函数
- 9. union：声明共用数据类型
- 10. enum： 声明枚举类型
- 11. typedef：用以给数据类型取别名
- 12. const： 声明只读变量
- 13. unsigned：声明无符号类型变量或函数
- 14. signed：声明有符号类型变量或函数
- 15. extern：声明变量是在其他文件中声明
- 16. register：声明寄存器变量
- 17. static： 声明静态变量
- 18. volatile：说明变量在程序执行中可被隐含地改变

2.3.2 关键字 3/4

- 19. void：声明函数无返回值或无参数，声明无类型指针
- 20. if:条件语句
- 21. else：条件语句否定分支（与 if 连用）
- 22. switch :用于开关语句
- 23. case：开关语句分支
- 24. for：一种循环语句
- 25. do：循环语句的循环体
- 26. while：循环语句的循环条件
- 27. goto：无条件跳转语句
- 28. continue：结束当前循环，开始下一轮循环
- 29. break：跳出当前循环
- 30. default：开关语句中的“其他”分支
- 31. sizeof：计算数据类型长度
- 32. return：子程序返回语句（可以带参数，也可不带参数）循环条件

2.3.2 关键字 4/4

- 1999年12月16日，ISO推出了C99标准，该标准新增了5个C语言关键字：
  - Inline, restrict, _Bool, _Complex, _Imaginary
- 2011年12月8日，ISO发布C语言的新标准C11，该标准新增了1个C语言关键字：
  - _Generic

2.3.3 分隔符

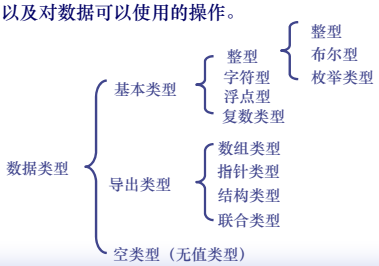
- 分隔符统称为空白字符(包括空格符、制表符、换行符、换页符及注释符)，在语法上仅起分隔单词的作用。
- 当程序中两个相邻的单词之间如果不用分隔符就不能区分开时则必须加分隔符（通常用空格符)。
- 例如，int x,y; 不能写成 intx,y;
  能写成 int x , y ;

2.4 基本数据类型

- 2.4.1 数据类型的分类
- 2.4.2 基本类型的名字
- 2.4.3 字符类型
- 2.4.4 整型类型
- 2.4.5 浮点类型

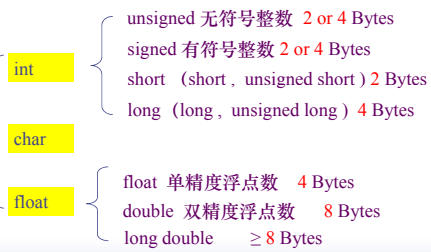
2.4.1 数据类型的分类

- 数据类型决定了数据的表示方式、占内存空间的大小、取值范围以及对数据可以使用的操作。



2.4.2 基本类型的名字

- 本小节介绍字符型、整型、浮点型 (表4.2)



2.4.3 字符类型char

- char的存储长度是一字节 。
- 多数系统中char与signed char同 (-128~127)。
- 字符数据以ASCII码存储在内存中 。
- 在不要求大整数的情况下，可用字符型代替整型 。

2.4.3 字符类型char

- char型变量的长度为1字节，最高位符号位，其最大值即是0111 1111，最小值为1000 0000（注意此数值是以补码形式表示的），换算为十进制即为127和-128。
- 机器多数是基于补码表示的: 11111111 = -1
  10000000 = -128

证明 10000000 表示的是 -128而不是-0

10000000 + 00000001 = 10000001 = -127

显然 -128 + 1 = -127

11111111 符号位为1表示负数

将数值位按位取反加一得到 0000000+1 = 0000001

所以 11111111是-1的补码，其值（原码） = -1

2.4.4 整型类型

- int型值存储在一个机器字中 .
- 假设字长为2B，int取值范围为-32768~32767 (-2^{15}～2^{15}-1)
  unsigned取值范围为0～65535 (0 ～ 2^{16}-1)
- 下面的代码是否正确

```
#define BIG 30000
int main(void)
{ short x,y,z;
  x=y=BIG;
  z=x+y;        /* 短整数溢出 */
   ……
}
```


- 程序员必须时刻保证整数表达式的值在合理范围内；
- 引入short和long的目的是为了提供各种满足实际要求的不同长度的整数；
- int通常反映特定机器的自然大小，short一般为2B，long一般为4B；
- 当关心存储时，用short；
- 当需要较大的整数值时，用long。

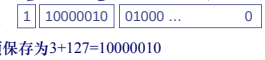
2.4.5 浮点类型

- 一个浮点数N可表示为：
  - S表示符号位，S=0,V为正；
    S=1，则V为负。
  - M表示有效数字，[1,2)
  - E表示指数位， float类型的偏移值为127
    double类型的偏移值为1023

$$V = (-1)^S \times M \times 2^E$$

数符 1b　尾数 23b　指数 8b

- -10.0=-$(0.101)_2$ ×2^4
- $(-1)^1 \times (1.010)_2 \times 2^{(011)_2}$
  - E=3时，float类型必须保存为3+127=10000010



- 写出float x=97.0在计算机内的浮点表示
  - 首先将10进制转化为二进制
  - $(97)_{10}$=$(0110\ 0001)_2$
  - 然后用浮点数的表示方法来表示该数
  - $(0110\ 0001)_2$ =$(-1)^0$× $(1.10\ 0001)_2$ ×2^{6} $^{(110)2}$
  - 指数E=110+01111111(127偏移位)=1000 0101
  - 尾数M为1.100001中去掉第一位变成100001，后面再补零
  - 则97.0在计算机内部表示为：

0 100 0010 1 100 0100 0000 0000 0000 0000

31 30　23 22　0

1.范围

- float和double的范围是由指数的位数来决定的。
- float的指数位有8位，而double的指数位有11位，分布如下：
- float： 1bit（符号位） 8bits（指数位） 23bits（尾数位）
- double： 1bit（符号位） 11bits（指数位） 52bits（尾数位）
- float的指数范围为-127~+128，而double的指数范围为-1023~+1024，并且指数位是按补码的形式来划分的。其中负指数决定了浮点数所能表达的绝对值最小的非零数；而正指数决定了浮点数所能表达的绝对值最大的数，也即决定了浮点数的取值范围。
- float的范围为-2^{128}～$+2^{128}$，也即-3.40E+38～+3.40E+38；
- double的范围为-2^{1024}～$+2^{1024}$，也即-1.79E+308～+1.79E+308。

2. 精度

- float和double的精度是由尾数的位数来决定的。
- 浮点数在内存中是按科学计数法来存储的，其整数部分始终是一个隐含着的“1”，由于它是不变的，故不能对精度造成影响。
- float：2^{23} = 8388608，一共7位，这意味着最多能有7位有效数字，但绝对能保证的为6位，也即float的精度为6~7位有效数字；
- double：2^{52} = 4503599627370496，一共16位，同理，double的精度为15~16位。
- 思考题：
  - 设计一个程序验证浮点数的精度、值域范围！

- 浮点数的表示可能只是近似的。其值与表示法之间的差称为“可表示误差”。
- 计算也可能造成可表示误差!
  - 不能使用= =和! = 运算符比较浮点数据!!!!!
- 可以用两个数值之差同一个预定的小正数epsilon比较的方法解决这个问题。
- 这是一个比较重要的知识点

浮点数溢出的处理

- 下溢
  - 如同整型，浮点数也是离散的，就是说零之后的那个正浮点数不是无穷小，而是一个确定的值，可以想象，中间必然存在无数个值，这些值是float表示不出来的，结果会被表示为0，称之为下溢。
  - 具体可以看下 计算机组成与体系结构 之类的书
- 上溢
  - 用称为“无穷大”的特殊位模式表示，即指数域全为1，尾数域0。
  - 有些系统中将输出+Infinity或 Infinity表示上溢的数据；指数域全为1，尾数域非0，表示这个数不是一个数（NaN）。
- 程序员必须能识别本地编译器和系统的上溢和下溢值，能识别并修正造成溢出的错误运算。

2.5 常量与变量

- 2.5.1 文字常量
- 2.5.2 符号常量
- 2.5.3 变量定义

2.5.1 文字常量 1/12

1. 整型常量
   - 有三种表示方法（通过前缀字符区分）：
     - 十进制：无前缀
     - 八进制：前缀为0
     - 十六进制：前缀为0x或0X时。
   - 例如，31可写成037，也可写成0x1f或0X1F

2.5.1 文字常量 2/12

- 整型常量可以带有后缀，用以指定其类型：
  - 字母u或U表示unsigned
  - 字母l或L表示long
  - 字母ul或UL表示unsigned long
  - 字母ll或LL表示long long (C99)
  - 字母ull或ULL表示unsigned long long (C99)
  - 无后缀时，表示int
- 当常量值超出指定类型的范围时，其实际类型取决于数值大小、前缀等，确定类型的规则很复杂，在标准化前的C语言、C89和C99中各不相同

2.5.1 文字常量 3/12

- 2. 浮点型常量
  - 有两种表示方式：
    - （1）带小数点的十进制数形式（可以小数点开头，也可以小数点结尾）
      如23.7，14.，.126
    - （2）指数形式（科学计数法）
    - 将指数部分跟在尾数部分后面。尾数部分的书写规则与第一种相同，但可以没有小数点，指数部分e(E)±n，代表$10^{\pm n}$。
    - 如 45e-3 = 45×10^{-3}， .15e5 = 0.15×10^{5}。

2.5.1 文字常量 4/12

- 可以使用后缀来指定其类型
  - 无后缀：double，
  - 后缀f或F：float，
  - 后缀l或L：long double。

2.5.1 文字常量 5/12

- 3. 字符常量
  - (1)用单引号包含的一个字符是字符常量
  - (2)只能包含一个字符

‘a’，’A’，‘1’

‘abc’ 、 “a” ✗

2.5.1 文字常量 6/12

- 转义序列
  - 以\开头的特殊字符称为转义序列,有两种形式：
  - 一种是“字符转义序列”，即反斜线后面跟一个图形符号，用于表示字符集中的非图形符号和一些特殊的图形字符。
  - \n 换行 \t 水平制表符
  - \\ 反斜杠 \‘ 单引号
  - \“ 双引号 \0 空字符
  - \? 问号
- 这是一个比较重要的知识点， !!!!! 见表2.5 P29
- \源程序\ex2_4.c

2.5.1 文字常量 7/12

- 转义序列的另一种是“数字转义序列”，即\ooo（1~3个八进制数字）
  \xhh （1~2个十六进制数字）
- 例如，
  'A'、'\101' 和'\x41' 字符A;
  '\t'、'\11'、'\011'、'\x9'和'\x09' 水平制表符

2.5.1 文字常量 8/12

- 使用转义序列时注意以下两点：
  - 使用数字转义序列时，可能依赖于字符编码方式，不可移植。最好把转义符隐藏在宏定义中，便于修改。
    - #define NAK '\025' /*否认字符*?
  - 数字转义序列的语法是独特的。
    - 八进制转义序列在用完3个八进制位之后，或遇到第一个非八进制位时终止，如
      - "\0111" 包含 "\011"和"1"两个字符
      - "\090" 包含三个字符："\0","9"和"0"
    - 十六进制转义序列中的超过2位时，编译出错。为了终止十六进制转义，可以把字符分段，如
      - "\xabc" 出错，可改为"\xa","bc"

2.5.1 文字常量 9/12

- 4. 字符串常量
  - 写成用一对双引号括住0至多个字符的形式。
    - "string\n" /* 包含7个字符的字符串 */
    - "" /* 包含0个字符的空字符串*/
  - 字符串中的单引号可以用图形符号表示，但双引号和反斜线必须用转义序列表示。例如：
    - "3'40\""
      - /* 表示5个字符的字符串：3'40" */
    - "c:\tc"
      - /* 表示4个字符的字符串 */
    - "c:\\tc"
      - /* 表示5个字符的字符串 */

2.5.1 文字常量 10/12

- 如何将一个较长的字符串写成多行?
- 有两种方法：
  - （1）行连接：在前一行的末尾输入续行符（\） 再换行。

```
"Hello,\
 how are you"   /* 换行后应紧靠行首 */
```

  - （2）字符串连接：将字符串分段，分段后的每个字符串用双引号括起来。

```
"Hello, "
"how are you"   /* 换行后不必紧靠行首*/
```


2.5.1 文字常量 11/12

- 'a'与 "a"有何区别？
  - 'a'：字符常量，占1 B内存空间
  - "a"：字符串常量，占2B内存空间

a	\0

  - 存储时，系统自动在后面补上\0（空字符，ASC11值为0，作为字符串结束标志）
  - 字符串的存储长度比字符串的实际长度大1
  - 这是一个知识点 !!!!!
- \源程序\ex2_5.c

2.5.1 文字常量 12/12

```
1.  /*****************************************************************
2.  函数名称：mystrlen
3.  函数功能：求字符串的长度
4.  函数参数：s保存字符串的数组
5.  函数返回值：返回字符串的长度。
6.  *****************************************************************/
7.  #include<stdio.h>
8.  int mystrlen(char s[])
9.  {
10.         int i;
11.         i=0;
12.         while(s[i]!='\0') ++i;
13.         return i;
14. }
15. int main(void)
16. {
17.         printf("%d\n",mystrlen("world"));  /* 输出 5 */
18.         return 0;
19. }
```


2.5.2 符号常量 1/5

- 用一个标识符表示一个常量.
- C语言中有三种定义符号常量的方法：
  - (1) 用#define指令
  - (2) 用const声明语句
  - (3) 用枚举类型（在2.9节介绍）

2.5.2 符号常量 2/5

- 1．用#define定义符号常量
  - #define是一种编译预处理指令，格式为：
    #define 标识符 常量
  - 标识符就是符号常量的名字(一般用大写，以区分变量)，它代表后面的常量值
  - 编译程序在编译前，所有在程序中出现该标识符，都用对应的常量替换
  - #define指令通常放在文件顶端
  - 注意：#define指令行的末尾没有分号！

2.5.2 符号常量 3/5

- 例2.6 打印华氏和摄氏温度对照表，温度转换公式为：
  ℃=(5/9)(℉-32)
- \源程序\ex2_6.c

2.5.2 符号常量 4/5

- 2．用const定义符号常量
  - const是关键字，称为类型限定符。格式为：
    const 类型名 标识符=常量;
  - 例如：

```
const double PI=3.14159;
const int DOWN=0x5000; /* 下光标键的扫描码 */
const int YES=1, NO=0;
```

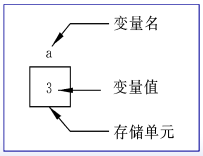

2.5.2 符号常量 5/5

- 用const和#define定义的符号常量的区别?
  - const声明的标识符是一个只读变量，编译时系统会根据定义的类型为该标识符分配存储单元，并把对应的常量值放入其中，该值不能再被更改，此后，程序中每次出现该标识符都是对所代表存储单元的访问。
  - #define定义的标识符没有对应的存储单元，只是在编译之前由预处理程序进行简单的文本替换。

2.5.3 变量定义 1/2

- 变量代表内存中具有特定属性的一个存储单元，它用来存放数据，这就是变量的值，在程序运行期间，这些值是可以改变的。
- 要求对所有用到的变量作定义，也就是“先定义，后使用”。

```
类型名 变量表;
  int   a, total, average;
```



2.5.3 变量定义 2/2

- 变量在声明时可以同时赋一个初值（称为变量的显示初始化），每个变量必须分别显示初始化。
  - char alert='\a', c;
  - int count=0, sum=0; （不能 int count=sum=0;）
  - 在编辑源程序时务必注意不能采用中文标点符号！
- 变量声明的目的？
  - 告诉编译器为变量分配适当长度的存储单元
  - 确定变量所允许的运算

2.6 运算符和表达式

2.6.1 C运算符简介 1/2

- 运算符是用运算的符号(+、-、*、/)表示执行对运算对象(称为操作数)的各种操作;
- 单个的操作数(包括常量、变量和函数调用)是表达式，由运算符和操作数组成的有意义的计算式子也是表达式。如:
  - sqrt(b*b-4*a*c)
  - x=x*PI/180
  - fabs(an)>=EPS
- 表达式的书写必须严格遵循语法和语义的规定，以便计算机能够识别和正确地解释，否则将导致语法出错或无法得出正确的计算结果。

2.6.1 C运算符简介 2/2

- 运算符是一种向编译程序说明特定的运算或操作的符号。每一种运算符都具有特定的数据操作规则
- 运算符的分类
  - （1）按运算符要求运算对象的个数分类：单目运算符、双目运算符、三目运算符
  - （2）按其数据操作功能分类：算术运算符、关系运算符、逻辑运算符、位运算符、C的特殊运算符 ，如数组下标运算符[]、函数调用运算符()、间接访问运算符*，等等
- 运算符都对操作数的类型有规定，比如%的操作数不能为浮点型
- 运算符具有优先级和结合性

2.6.2 运算符的优先级和结合性 1/9

- 运算符的优先级
  - 当一个表达式中出现多个运算符时，表达式的求值运算按运算符的优先级别从高到低的顺序执行。在C语言中，将44种运算符的优先级从高到低共分为15个等级。
- 所有运算符的优先级和结合性规则见表2.6 ??

2.6.2 运算符的优先级和结合性 2/9

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	[]	数组下标	数组名[常量表达式]	左到右	
	()	圆括号	(表达式)/函数名(形参表)		
	.	成员选择（对象）	对象.成员名		
	->	成员选择（指针）	对象指针->成员名		
2	-	负号运算符	-表达式	右到左	单目运算符
	(类型)	强制类型转换	(数据类型)表达式		
	++	自增运算符	++变量名/变量名++		单目运算符
	--	自减运算符	--变量名/变量名--		单目运算符
	*	取值运算符	*指针变量		单目运算符
	&	取地址运算符	&变量名		单目运算符
	!	逻辑非运算符	!表达式		单目运算符
	~	按位取反运算符	~表达式		单目运算符
	sizeof	长度运算符	sizeof(表达式)		

2.6.2 运算符的优先级和结合性 3/9

优先级	运算符	名称或含义	使用形式	结合方向	说明
3	/	除	表达式/表达式	左到右	双目运算符
	*	乘	表达式*表达式		双目运算符
	%	余数（取模）	整型表达式/整型表达式		双目运算符
4	+	加	表达式+表达式	左到右	双目运算符
	-	减	表达式-表达式		双目运算符
5	<<	左移	变量<<表达式	左到右	双目运算符
	>>	右移	变量>>表达式		双目运算符
6	>	大于	表达式>表达式	左到右	双目运算符
	>=	大于等于	表达式>=表达式		双目运算符
	<	小于	表达式<表达式		双目运算符
	<=	小于等于	表达式<=表达式		双目运算符

2.6.2 运算符的优先级和结合性 4/9

优先级	运算符	名称或含义	使用形式	结合方向	说明
7	==	等于	表达式==表达式	左到右	双目运算符
	!=	不等于	表达式!= 表达式		双目运算符
8	&	按位与	表达式&表达式	左到右	双目运算符
9	^	按位异或	表达式^表达式	左到右	双目运算符
10	|	按位或	表达式|表达式	左到右	双目运算符
11	&&	逻辑与	表达式&&表达式	左到右	双目运算符
12	||	逻辑或	表达式||表达式	左到右	双目运算符
13	?:	条件运算符	表达式1? 表达式2: 表达式3	右到左	三目运算符

2.6.2 运算符的优先级和结合性 5/9

优先级	运算符	名称或含义	使用形式	结合方向	说明
14	=	赋值运算符	变量=表达式	右到左	
	/=	除后赋值	变量/=表达式		
	*=	乘后赋值	变量*=表达式		
	%=	取模后赋值	变量%=表达式		
	+=	加后赋值	变量+=表达式		
	-=	减后赋值	变量-=表达式		
	<<=	左移后赋值	变量<<=表达式		
	>>=	右移后赋值	变量>>=表达式		
	&=	按位与后赋值	变量&=表达式		
	^=	按位异或后赋值	变量^=表达式		
	|=	按位或后赋值	变量|=表达式		
15	,	逗号运算符	表达式,表达式,…	左到右	从左向右顺序运算

2.6.2 运算符的优先级和结合性 6/9

- 运算符的结合性
  - （1）运算符的左结合性：当一个操作数两侧的运算符具有相同的优先级时，操作数先与左边的运算符结合，即自左至右的结合方向。
  - （2）运算符的右结合性：即自右至左的结合方向。
    - 除单目运算符、赋值运算符和条件运算符是右结合性外，其它运算符都是左结合性。

2.6.2 运算符的优先级和结合性 7/9

- 当表达式中包括多个运算符时，C语言会先按优先级规则解释表达式的意义。如：
  - 1+2*3 等价于 1+（2*3）
- 当一个操作数两侧的运算符优先级别相同时，则按“结合性”规则。
  - 1+2-3 等价于（1+2）-3
  - ---从左至右的结合性（左结合）
  - -a++ 等价于-（a++）
  - ----从右至左的结合性（右结合）

2.6.2 运算符的优先级和结合性 8/9

- 理解规则的技巧
  - （1）先（括号）内层，后（括号）外层
    - 有括号时，最内层的括号中的东西先算，再一层层向外
  - （2）先函数，后运算
    - 式中有函数，例如sqrt(), fabs(), sin()之类，先算函数
  - （3）先算术，后关系，再逻辑
    - 逻辑表达式中，混有算术运算，要先算算术(加减乘除，函数)运算，再算关系(大小，等不等)运算。最后算逻辑(或与非)，得到真假

2.6.2 运算符的优先级和结合性 9/9

- 理解规则的技巧
  - （4）先乘除，后加减
    - 同算术一样，先做乘除，后做加减。逻辑运算符“与”又叫逻辑乘，逻辑运算符“或”，又叫逻辑加，按先乘除，后加减，“与”比“或”优先
  - （5）先左，后右
    - 同级运算，先做左边的，后做右边的
  - （6）搞不清，加括号
    - 自己写算术表达式和逻辑表达式时，搞不清运算符优先级，可以加括号，括号里的总是先

2.6.3 算术运算 1/4

- 运算符:
  - + 加法 正值 3+6, +3
  - – 减法 负值 6– 4, –5
  - * 乘法 3*8
  - / 除法 8 / 5
  - % 求余 7 % 4 值为3

2.6.3 算术运算 2/4

- 注:
  - 两个整型数据相除 （结果为整, 舍去小数部分）
    –5/ 3 ⇒ – 1 1/ 2 ⇒ 0 1. / 2 ⇒ 0. 5
  - 使用时千万注意 int / int 出现数据丢失。

* %操作数必需为整数

2.6.3 算术运算 3/4

- 【例2.7】 求出所有的水仙花数
  - “水仙花数”是一个三位数，其各位数字立方和等于该数本身。例如，153是一个“水仙花数”，因为153=1^3+5^3+3^3。
  - 找出“水仙花数”的关键是怎样从一个三位数中分离出百位数、十位数和个位数，分解方法见下面的代码。
  - \源程序\ex2_7.c

2.6.3 算术运算 4/4

```
1.  #include<stdio.h>
2.  int main(void)
3.  {
4.     int x,i,j,k; /* x是三位数，i,j,k分别是组成x的三个数字 */
5.     for(x=100;x<=999;x=x+1)  /* 从100开始循环到999 */
6.     {
7.        i=x/100;        /* 分解百位数 */
8.        j=(x-i*100)/10;   /* 分解十位数 */
9.        k=x%10;         /* 分解个位数 */
10.       if(x==i*i*i+j*j*j+k*k*k) /* 是水仙花数 */
11.          printf("%5d",x);  /* 输出 */
12.    }
13.    return 0;
14. }
```


2.6.4 关系运算 1/4

- 有6个关系运算符 :
  - > 大于
  - >= 大于等于
  - < 小于
  - <= 小于等于
  - == 等于
  - != 不等于
- 关系表达式的类型: int
- 关系表达式的值:
  - 关系成立，值为1（代表“真”）
  - 关系不成立，值为0（代表“假”）

2.6.4 关系运算 2/4

- 举例: 根据变量说明，给出表达式的值。

```
int  x=4, y=3,z=2;
char c='a';
```

(1) c = = 'A'+32 (1)
(2) c+1 ! = 'b' (0)
(3) x-y<=10 (1)
(4) z=x>y (1)
(5) x>y>z (0)

2.6.4 关系运算 3/4

- 注意
  - 数学上判断x是否在区间[a,b]中时，习惯写成:
    a<x<b
  - C中“a<x<b”的含义与数学中的含义不同，应写成:
    a<x && x<b

2.6.4 关系运算 4/4

- 常见的C语言编程错误
  - 将运算符= = 写成运算符=（赋值）。可能会因为运行时的逻辑错误而导致不正确的结果。例如，
  - if (grade = = 'A') printf("Very Good! ");
    /*当成绩的等级为A等时，输出Very Good!*/
  - 而:
  - if (grade ='A') printf("Very Good!");
    /*不论成绩的等级是几等，总输出Very Good!*/

2.6.5 逻辑运算 1/7

- 逻辑运算表示操作对象的逻辑关系
- 有3个逻辑运算符
  - ! 逻辑非(单目)
  - && 逻辑与(双目)
  - || 逻辑或(双目)
- 逻辑表达式:
  - 用逻辑运算符将关系表达式或逻辑量连接起来的式子

2.6.5 逻辑运算 2/7

- 逻辑运算的操作数可以是0和任何非0的数值，
- 系统最终以0判断属于“假”（以0代表“假”），
  以非0判断属于“真”（以1代表“真”）。
- 逻辑表达式的结果是int类型，其值为 1 (“真”) 或0 (“假”)
- 逻辑运算符的操作数可以是整型、浮点型、复数类型和指针类型，两个操作数各自独立进行整数提升，不进行一般的算术转换
- C语言中，认为是“假”的表达式有:
  - 具有0值的整数型表达式、具有0.0值的浮点型表达式、空字符'\0'和空指针
- 具有非0值的表达式都认为是真

2.6.5 逻辑运算 3/6

- && 和 || 的真值表

e1	e2	&&	||
0	0	0	0
0	非0	0	1
非0	0	0	1
非0	非0	1	1

2.6.5 逻辑运算 4/7

- ! 的真值表

e	!
0	1
非0	0

2.6.5 逻辑运算 5/7

- 注意
  - 编译程序在处理含有&& 、‖ 表达式时， 往往采用优化算法(提高速度)。
    - e1&&e2 一旦发现e1=0，不再计算 e2
    - e1‖e2 一旦发现e1=1，不再计算 e2
  - 例如，

```
x>=0.0 && sqrt(x)<=7.7
/* 如果x值为负，不求x的平方根 */
```


2.6.5 逻辑运算 6/7

- 熟练掌握C语言的关系运算符和逻辑运算符后，可以巧妙地表示一个复杂的条件。
  - (1) 整数a是偶数: !(a%2) 或 a%2= =0
  - (2) 字符c的值是英文字母。

```
c>='a' && c<='z' || c>='A' && c<='Z'
```

  - (3) 某一年year是闰年。如果某一年的年份能被4整除但不能被100整除，那么这一年就是闰年，此外，能被400整除的年份也是闰年。

```
!(year%4) && year%100 || !(year%400)
```

- 这是一个非常重要的知识点 ！！！！！

2.6.5 逻辑运算 7/7

例2.10 编程判断某一年是否是闰年

```
1.  #include<stdio.h>
2.  int main(void)
3.  {
4.      int year;
5.      printf("Input year: ");              /* 提示语 */
6.      scanf("%d",&year);                   /* 输入年份，存入变量year */
7.      if(!(year%4) && year%100 || !(year%400))      /* 是闰年 */
8.          printf("%d is a leap year\n",year);
9.      else                                 /* 不是闰年 */
10.         printf("%d is not a leap year\n",year);
11.     return 0;
12. }
```


2.6.6 自增和自减运算 1/8

- ++:自增,使内存中存储的变量值加1
- -- :自减,使内存中存储的变量值减1
- ++和--的奇特之处: 前缀式和后缀式都行
  - ++x
  - x++ 相当于 x=x+1;
  - --x
  - x-- 相当于 x=x-1;

2.6.6 自增和自减运算 2/8

- 注意前缀与后缀的区别
  - x = 10 ;
  - y = ++x ; /* y=11, x=11 */

    ---- 先执行对操作数的加运算，再使用该操作数的值
  - y = x++ /* y=10, x=11 */

    ---- 先使用该操作数的值，再对它作加运算，

2.6.6 自增和自减运算 3/8

- 【例2.11】 统计输入正文的字符数和行数
  - 正文是一行行字符组成的字符序列，每一行是以换行符为结束标志的一串字符，输入正文以Ctrl+Z（DOS系统）或Ctrl+D（UNIX系统）为结束标志（称为文件尾）。
  - 输入一个字符可以使用标准库函数getchar。
  - getchar遇文件结束标志时返回EOF，EOF是在头文件<stdio.h>定义的符号常量，其值为-1。
  - \源程序\ex2_11.c

2.6.6 自增和自减运算 4/8

```
1.  #include<stdio.h>
2.  int main(void)      /*统计输入正文的字符数和行数。*/
3.  {
4.          int c,nc,nl;
5.          printf("Input a text end of ctrl+z:\n");    /* 提示语 */
6.          nc=nl=0;
7.          while((c=getchar())!=EOF)
8.          {
9.                  ++nc;                   /* 字符数自增1，可以用nc++代替 */
10.                 if(c=='\n')  ++nl;          /* 行数自增1，可以用nl++代替 */
11.         }
12.         printf("nc=%d,nl=%d\n",nc,nl);
13.         return 0;
14. }
```


2.6.6 自增和自减运算 5/8

【例2.12】计算1+2+3+…+n，n从键盘输入

```
1.  #include<stdio.h>
2.  #define N  10
3.  int main(void)
4.  {
5.     int i,sum;   /* i是循环变量，sum是累加器变量，存放和值 */
6.     i=1;        /* 循环变量初始化为1  */
7.     sum=0;      /* 累加器初始化为0  */
8.     while(i<=N)   /* 循环求和 1+2+...+N */
9.           sum+=i++;        /* 等价于 sum=sum+i,i++; */
10.    printf("sum=%d\n",sum);  /* 输出和值 */
11.    return 0;
12. }
```


2.6.6 自增和自减运算 6/8

- 序列点
  - 后缀++(或--)计算延迟的终止点称为序列点。
  - 在序列点之前，用原值，序列点之后，该操作数是更改后的新值。
  - 下列条件出现序列点：
    - (1) &&、||、?: 或 ，运算符，即这些运算符的第一个操作数之后
    - (2)完整表达式结束时，即表达式语句、return语句中的表达式、if、switch或循环语句中的条件表达式（包括for语句中的每个表达式）之后

2.6.6 自增和自减运算 7/8

【例2.13】后缀式++(或--)表达式举例

int a=1,b=0;

(1) b++ + b++

表达式值为0，b为2

(2) a--&&a

表达式值为0，a为0

(3) b++ ? b : -b

表达式值为-1，b为1

2.6.6 自增和自减运算 8/8

几点注意:

1. 只能用于变量（即左值表达式）

如 5++, (a+b)--均不合法

2. 结合性为从右至左.

如 –i++ 相当于 –(i++)

若 i=3，则该表达式结果为 –3，i为4

2.6.7 赋值运算 1/3

1. 简单的赋值运算
   - 赋值运算符：=
   - 赋值表达式一般形式为: <变量>=<表达式>
   - 右侧的“表达式”的值赋给左侧的变量
   - 左值 (lvalue) : 赋值运算符左侧的标识符
   - 变量可以作为左值，而表达式就不能作为左值(如a+b)
   - 常量也不能作为左值

2.6.7 赋值运算 2/3

- 赋值表达式的值和类型
  - 与左操作数的值和类型相同。
  - C把赋值处理为运算符，其好处是使赋值表达式可以像其它任何表达式一样当作一个数据来处理。 如：

```
a=2;
b=3;
x=a+b;  可以被简化为
x= (a=2) + (b=3)
```

  - 右结合性，如：

```
a=b=c=3;
等价于
a= (b= (c=3) ) ;
```

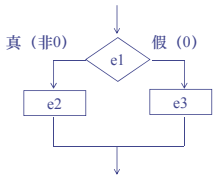

2.6.7 赋值运算 3/3

- 2. 复合的赋值运算
  - 在“=”号之前加一个双目运算符：
    +=，-=，*=，/=，%=
  - 如

```
i+=2        等价于 i=i+2
y/=x+10     等价于 y=y / (x+10)
x*=k=m+5    等价于 x=x * ( k=m+5)
s[i++]+=1  和s[i++] = s[i++] +1不等价
```


2.6.8 条件运算

- 三目运算符： e1 ? e2 : e3



- 举例
  - (1) x=10;
    y=x>9?100:200
    y= ?
  - (2) x = a > 0 ? 1 : a < 0 ? -1 : 0

等价于(因为是右结合性)

x = a > 0 ? 1 : (a < 0 ? -1 : 0)

2.6.9 逗号运算

- 在C语言的所有运算符中，逗号运算符的优先级最低，它是用一个表达式作为操作数的双目运算符，表达式为：
  e1，e2
- 计算规则
  - 先计算e1，再计算e2，逗号表达式的值和类型与e2的值和类型相同
- 举例
  - (1) x=(i=4，i%3)
    表达式的值为：1， x为1
  - (2) x=i=4，i%3
    表达式的值为：1， x为4

- **扩展形式**
  - 逗号表达式的一般形式可以扩展为
    e1，e2，e3，……，en
  - 它的值为最后一个表达式n的值。
- 逗号运算符特别适用于程序中需要执行多个表达式，而愈发上只允许为一个表达式的情况，如for语句中的表达式：

```
1. for(sum=0,i=1;i<10;i++)
2.    sum+=i;
```

  - 可改写为：

```
1. for(sum=0, i=1; i<10;  sum+=i , i++)
2.      ;
```

- 注意
  - 并不是任何地方出现的逗号都是作为逗号运算符。例如函数参数也是用逗号来间隔的。

```
printf("%d,%d", 2, 3);   /* 输出：2, 3 */
printf("%d,%d" ,(2, 3), 3 ); /* 输出：3, 3 */
```

“2，3”并不是一个逗号表达式，它是printf函数的2个参数

“（2，3）”是一个逗号表达式，它的值等于3。

- 【例2.14】 输入一串数字字符，将其转换为一个十进制整数赋值给x（模拟scanf("%d",&x）的功能）。
  - 当用scanf("%d",&x）输入一个整数时，实际上输入的是组成该整数的各位数字的一个字符串，scanf函数读到的每个数字都是该字符的字符码。
  - 为了将字符串转换为整数，必须先将每个数字字符转换成数字对应的一位整数，然后按相应的数位拼成一个十进制整数，最后赋给变量x。
  - \源程序\ex2_14.c


```
1.  #include<stdio.h>
2.  int main(void)
3.  {
4.     char c;   /* 用于存放当前输入的字符 */
5.     int x;    /* 用于存放转换的整数 */
6.     for(x=0,c=getchar(); c>='0'&&c<='9'; c=getchar())
                                  /* 遇非数字字符结束循环 */
1.        x=10*x+c-'0';        /* c-'0'：将数字字符转换成对应的一位整数 */
2.     printf("x=%d\n",x);  /* 输出 */
3.    return 0;
4.  }
```


改进版

```
1.  #include<stdio.h>
2.  int main(void)
3.  {
4.     char c;   /* 用于存放当前输入的字符 */
5.     int x;    /* 用于存放转换的整数 */
6.     for(x=0,c=getchar(), printf("c=%c\n",c) ; c>='0'&&c<='9'; \
7.              c=getchar(),   printf("c=%c\t",c) )
8.                                  /* 遇非数字字符结束循环 */
9.        x=10*x+c-'0';        /* c-'0'：将数字字符转换成对应的一位整数 */
10.    printf("x=%d\n",x);  /* 输出 */
11.   return 0;
12. }
```


2.6.10 sizeof运算

- 一个单目运算符，有两种形式：
  - (1) sizeof（类型名）
    给出指定数据类型占用的存储字节数
  - (2) sizeof 表达式
    给出表达式结果的类型占用的存储字节数

- 举例: 假设int类型占用2字节。
  - sizeof(long)　　/* 值为4 */
  - double x; sizeof x　　/* 值为8 */
  - int a[10]; sizeof (a)　　/* 值为20 */
  - int a=1,b=1; sizeof (a+b)
    /* 值为2, 而sizeof a+b，先求sizeof a，再和b加, 值为3 */

- sizeof是一个常量表达式
  - 其运算是在编译时执行的。因此，当sizeof的操作数是表达式时，则在编译时分析表达式以确定类型，运行时不对这个表达式求值。
  - 例如：

```
short x=1;
sizeof(++x);   /* x不递增，仍为1 */
```


2.7 位运算符和位表达式*

- 2.7.1 按位求反（～）
- 2.7.2 按位与、或、加运算 （&, | , ^ ）
- 2.7.3 左移和右移运算（<<，>>）
- 2.7.4 位运算符应用举例
- 2.7.5 打印整数各位

2.7 位运算符和位表达式

- 6个位运算符：
  - ～ (求反)
  - & (按位与)
  - | (按位或)
  - ^ (按位异或，或按位加)
  - << (左移)
  - >> (右移)

2.7.1 按位求反（～）

- 对操作数的每个二进制位取相反值
- 例如：
  - short a=5; unsigned short b=5;
  - a和b的二进制为：00000000 00000101
  - ~a的二进制为： 11111111 11111010
  - ~a的值为short型 –6。
  - ~b的值为unsigned short型 65530。

2.7.2 按位与、或、加运算（&,|,^ ）

表达式	二进制表示	十六进制值	说明
x	01101000 11010001	0x68d1	待处理数据
mask	11111111 00000000	0xff00	逻辑尺
x & mask	01101000 00000000	0x6800	将x的低字节置为0，保留高字节
x | mask	11111111 11010001	0xffd1	将x的低字节保留，高字节置为1
x ^ mask	10010111 11010001	0x97d1	将x的低字节保留，高字节翻转

2.7.3 左移和右移运算（ <<和>> ）

- e<<n 将e的值向左移n位，低n位填入0。
  - 左移1位相当于该数值乘以2
- e>>n 将e的值向右移n位，而高n位可能填入？。
  - --e是无符号类型，填入0；
  - --e是有符号类型，一些机器填入0（即“逻辑移位”），而另一些机器则填入符号位（即“算术移位”）。
- 在使用右移运算符时应经常用无符号类型。
  - 右移一位相当于该数值除以2

移位运算符的例子

表达式	二进制表示	值	行为
a	00000000 00001011	11	
b	00000000 00001111	15	
-a	11111111 11110101	-11	对a进行负号运算
a<<2	00000000 00101100	44	a左移2位
b>>3	00000000 00000001	1	b右移3位
-a>>2	11111111 11111101	-3	-a右移2位(填入符号位1，而其他机器可能是0)

【例2.15】写一个表达式，取一个整数x从第m位开始向右的n位，并使其向右端靠齐。一个整数的各个二进制位从右至左依次编号为第0位、第1位、第2位、……。

- （1）将要取出的那n位移到最右端

  x>>(m-n+1)
- （2）设计一个逻辑尺：低n位全为1，其余位全为0

  ~(~0<<n)
- （3）将上面二者进行&的运算

  x>>(m-n+1) & ~(~0<<n)

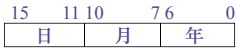
在一个整数x中，从第m位开始向右的取n位，使其成为一个新的整数。

2.7.4 位运算符应用举例

【例2.16】 压缩和解压示例。可以把表示21世纪日期的日、月和年3个整数压缩成1个16位的整数。写一个表达式，实现压缩存储日、月和年。

分析：

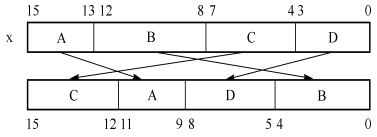
- 可以把表示21世纪日期的日、月和年3个整数压缩成1个16位的整数，因为日有31个值，月有12个值，年有100个值，所以可以在一个整数中用5b表示日，用4b表示月，用7b表示年.



day<<11 | month<<7 | year

【例2.17】 简单加密示例。

- 将一个短整型数x分成4个长度不等的部分：
- A（3b）、B（5b）、C（4b）和D（4b），然后将它们按照CADB的顺序重新拼凑在一起，实现对其加密的功能。写一个表达式，要求其值为x的密文。



- （1）设计4个逻辑尺，分别用于取出A、B、C和D 4个部分。
  - A的逻辑尺应设计成：高3位全为1，其余全为0，即0xE000；
  - B的逻辑尺应设计成：第8～12位全为1，其余全为0，即0x1F00；
  - 同理C和D的逻辑尺分别为0x00F0和0x000F。
- （2）取出A、B、C和D，再移到新位置。将x分别和逻辑尺进行&运算，可取出每一部分。用每部分第1位的初始位置减去其新位置得到移位数，正数右移，负数左移。
- （3）将各部分重新组装在一起。
- 得到表达式为：

(x&0xE000)>>4 | (x&0x1F00)>>8 | (x&0xF0)<<8 | (x&0xF)<<5

2.7.5 打印整数各位的软件工具

例2.19 函数PrintBit的功能是用屏蔽字显示int型数据的每一个二进制位，通过这个函数可以了解到，在内存中如何表示表达式的值。

```
1.  /* 按位显示 int 的内存中的表示*/
2.    #include<limits.h>
3.    void bit_print(int x)
4.    {
5.      int  i;
6.      int  n=sizeof (int) * CHAR_BIT;     /* CHAR_BIT 在limits.h中定义，*乘号*/
7.      int  mask = 1 << (n-1);             /* 逻辑尺mask=100……0 */
8.      for ( i=1; i<=n; ++i ) {
9.          putchar ( ! ( x & mask ) ? '0': '1');
10.         x<<=1;
11.         if ( ! ( i % CHAR_BIT ) && i<n )   putchar(' ');
12.     }
13.   }
```


2.8 类型转换

- C语言允许双精度、单精度、整型及字符数据之间混合运算
  10 + '1' + 6.5
- 也就是说在C语言中，表达式中各操作数的类型可以不一样！
- C语言表达式计算规则:
  - 先进行数据类型转换，转换成同一类型，然后计算其结果！

2.8.1 整数提升

- 任何表达式中的char、unsigned char、short和unsigned short都要先转换成int或unsigned;
- 如果原始类型的所有值可以用int表示，则转换成int，否则转换成unsigned，把这称为“整数提升”。

char / short → int → unsigned → long → unsigned long → float → double → long double

2.8.3 赋值转换

- 右操作数的值被转换为左操作数的类型
- 例如:

```
short s=5;  double d=2.9;
则
s = d   /*把d转换为short，再赋给s。值为2*/
d = s   /*把s转换为double，再赋给d。值为5.0*/
```


2.8.4 强制类型转换

- （类型名）操作数
- 例如，

```
(double) i  /* 将i转换成double，i的类型保持不变*/
(long)('a'-32) /*'a' 被自动转换成int，
                 相减的结果被强制转换为long*/
(float)x+y      /*等价于( (float)x)+y*/
(double)x=10  /*错误*/
```


2.9 枚举类型

- 枚举类型是用户自定义类型，它是用标识符命名的整型常量的集合，其中的标识符称为枚举常量。
- 从效果上看，枚举常量是自动设置值的符号常量。

2.9.1 枚举类型的定义

- 在未指定值的缺省情况下，第一个枚举常量的值为0，以后的值依次递增1
- 可以指定一个或多个枚举常量的值，未指定值的枚举常量的值比前面的值大1.
  - enum sizes { SMALL, MEDIUM=10, BIG , TOO_BIG=20 } ;
  - enum 后面也可以不出现枚举名 .
  - enum { WIN, LOSE, TIE, ERROR};

```
enum week { SUN, MON, TUE, WED, THU, FRI, SAT };
```

关键字　枚举名　枚举常量

2.9.2 用枚举类型定义符号常量

- #define WIN 0
- #define LOSE 1
- #define TIE 2
- #define ERROR -1

- 可用下面的枚举类型定义来代替。
- enum { WIN, LOSE, TIE, ERROR=-1};

2.9.3. 枚举变量的声明

- (1) 在定义枚举类型的同时说明枚举变量

```
enum color { RED,GREEN,BLUE} c1,c2;
```

- (2) 利用枚举名来说明枚举变量

```
enum color { RED,GREEN,BLUE} c1;
enum color c2;
```

- 或者

```
enum color { RED,GREEN,BLUE};
enum color c1,c2;
```


- 一个枚举变量的值是int型整数，但值域仅限于列举出来的范围。枚举变量值的输入和输出都只能是整数。

```
c1=BLUE;
/* 等价于 c1=2; 使用有意义的标识符有助于读者理解程序 */
printf("%d",c1); /* 输出2，而不是BLUE */
scanf("%d",c2); /* 输入0，不能输入RED */
```

- 下面的语句是错误的：

```
c1=3; /* 变量c1的值域为：0、1和2 */
printf("%s", GREEN );
      /* 输出错误的结果，而不是GREEN */
if(c1= =RED)  printf("Red" );
```


- 【例2.22】用枚举变量day控制循环，输出存储在数组weekName中的英文星期名。
  - 一个枚举变量的值是一个int整数，它可以出现在整数允许出现的任何地方。
  - 如果要输出与枚举值相对应的枚举常量标识符或代表其含义的完整字符串，可以定义一个字符串数组，以枚举值作为下标。
  - 源程序\ex2_22.c

2.10 新增数据类型**

- C99标准增加了支持64位的整数类型long long，还增加了布尔类型和复数类型。
- CodeBlocks编译器能够较好地支持C99标准，本节中的例子程序均由CodeBlocks运行通过。

2.10.1 long long类型

- long long类型的长度至少应为64位。和其他整型类型一样，有带符号和无符号两种。
- signed long long的数值范围为-263～ 263-1，unsigned long long的数值范围为0～ 264 -1。long long类型的常量用后缀“LL”或“ll”表示。
- 在所有整型类型中，类型long long的值域最宽，因此，C99标准的自动转换所遵循的转换方向为：
  - …→unsigned long→long long→unsigned long long→float→…

- 【例2.23】输出斐波那契（Fibonacci）数列的前60个数。
  - Fibonacci数列是由1和1 开始，之后的数是它前面两数的和，即 1，1，2，3，5，8，……
  - 分析：由于Fibonacci数列从大约第45个数起就超过了long类型的范围，为避免溢出，应将结果变量说明为long long类型，从输出结果可以看到，它的长度为8字节。
  - 源程序\ex2_23.c

2.10.2 布尔类型

- C语言中可以用任何整数类型表示布尔值，0表示假，所有非0表示真。布尔表达式为假时值为0，为真时值为1。例如，i=(a<b)，在a小于b时对整型变量i赋值1；在a不小于b时对变量i赋值0。
- C99标准引入了真正的布尔类型_Bool，_Bool类型长度为1，只能保存数值0和1（分别表示“false”与“true”）。虽然也可以使用其他整数类型表示布尔值，但如果C语言实现支持C99标准，那么使用_Bool类型更清晰。
- C99标准为了让C和C++兼容，增加了一个头文件stdbool.h。里面定义宏名bool为_Bool的同义词，定义false与true分别为0和1。因此，只要在源文件中包含stdbool.h这个头文件，就可以在C语言里像C++那样使用bool定义布尔类型变量，通过true和false对布尔变量进行赋值。
- 将任意非零值赋值给_Bool类型，都会先转换为1，表示真。将零值赋值给_Bool类型，结果为0，表示假。

- 【例2.24】 捕鱼和分鱼问题
  - A、B、C、D、E五人在某天夜里合伙捕鱼，到第二天凌晨时都疲惫不堪，于是各自找地方睡觉。
  - A第一个醒来，他将鱼分为五份，把多余的一条扔掉，拿走自己的一份。
  - B第二个醒来，也将鱼分为五份，把多余的一条扔掉，拿走自己的一份。
  - C、D、E依次醒来，也按同样的方法拿鱼。问他们合伙捕了多少条鱼?

- 分析：总共将所有的鱼进行了5次平均分配，每次分配的策略是相同的，即扔掉一条后剩下的分为五份，然后拿走自己的一份，余下其他的四份。
- 假定鱼的总数为X，则X可以按照题目的要求进行五次分配：X-1后可被5整除，余下的鱼为4*(X-1)/5。若X满足上述要求，则X就是题目的解。
- 源程序\ex2_24.c

2.10.3 复数类型

- C99标准新增了 _Complex 和_Imaginary 2个复数类型关键字，_Imaginary表示只有一个实数组成的纯虚数类型（即bi）。由于复数的实部和虚部均为实数，在说明复数类型时，需用浮点类型名进一步指定对应实数类型，因此复数或纯虚数各有以下3种类型：
  - float _Complex、double _Complex、long double _Complex
  - float _Imaginary、double _Imaginary和long double _Imaginary
- 复数类型_Complex表示为对应实数类型的二元数组，第1个元素表示复数的实数部分，第2个元素表示复数的虚数部分。而复数类型_Imaginary是实数部分总为0的纯虚数类型。

- C99标准还引入了头文件complex.h，其中定义complex宏为关键字_Complex的同义词，定义imaginary宏为关键字_Imaginary的同义词，定义_Complex_I(或I)为复数类型的虚数常量表达式，取值为虚数单位i，或$\sqrt{(-1)}$。
- 复数型常量用包含了虚数常量 _Complex_I(或I)的浮点型常量表达式来表示，如1.0+2.0*I，1.0+2.0*I，4.5*I等。
- 复数的运算必须要有相应的库函数支持。在C99标准中，复数的库函数很多，如三角函数、双曲函数、指数函数和其他函数等，这些函数的原型在头文件complex.h中说明。

- 复数类型转换规则
  - （1）一个复数类型转换成另一个复数类型时，实数和虚数浮点数成员按照浮点类型转换规则分别转换。
  - （2）从Complex类型转换为_Imaginary类型时，放弃实数部分；从_Imaginary类型转换为_Complex类型时，将实数部分设置为0。
  - （3）将一个实数（整数或浮点数）转换成复数类型时，该实数转换成复数值的实数部分，而虚数部分为0。
  - （4）将一个复数类型转换成实数（整数或浮点数）时，虚数部分放弃。

- 【例2.25】 求两个复数的乘积。
  - 源程序\ex2_25.c
  - 程序中定义了3个双精度的复数a、b和c，第一条printf函数调用语句输出复数a所需的内存大小，它为double的2倍。
  - 接着，给复数变量a和b分别赋一个复数常量，复数a和b的乘积赋值给复数变量c。
  - 函数creal和cimag分别为取得复数的实部和虚部。

本章小结

- 记号是程序中具有语义的最基本组成单元，共分5类：标识符、关键字、常量、运算符和标点符号。
  - 标识符是程序员用于对变量和函数等命名的符号。
  - 关键字有明确的含义，不能被用作普通标识符。
  - 常量包括字符常量、字符串常量，以及各种类型的整型常量、浮点型常量，必须熟练掌握各种类型常量的表示方法。
- C语言提供的基本数据类型有：字符型、整型、短整型、长整型，以及单精度、双精度和高精度浮点型。
- 在C99标准中增加了布尔类型、长长整型、复数类型等。
- 枚举类型是用标识符命名的整型常量的集合，其中的标识符实际上是自动设置值的符号常量。

- C语言提供了很多运算符，除通常的算术、关系、逻辑和赋值运算符外，还有一些C语言特有的运算符：自增、自减、条件、逗号、复合赋值、sizeof和位运算符。
- 自增自减运算符有前缀式和后缀式，但效果可能不同，需要特别注意。
- 位运算符使程序员可以访问字节或字中的实际二进制位，要重点掌握位运算符的应用。
- 每个表达式都有一个值和类型，运算符的优先级和结合性规则决定了表达式求值的顺序，在表达式的计算过程中，会涉及类型之间的转换问题，有自动类型转换和强制类型转换两种规则。

Assignments:

- 作业题
  - 2.3 2.4 2.6 2.7 2.8 2.10 2.11 2.16 2.18
- 我建议
  - 后面习题，每题都做，并且搞懂！
- 必做题
  - 在作业自动提交系统中必须按时提交指定的编程作业题，并计入期末考试的总成绩！
  - http://115.156.155.1/CTest_1228/