信息 (information)

- **现实世界中的对象及其属性在人们头脑中反映为各种不同的信息**
  - 数字
  - 文字
  - 声音
  - 图形
  - …

- **计算机中，这些信息一般是用一系列 0 和 1 来表示的（简单），它们对应着电器设备的两个稳定状态：开关的开/关、电压的高/低、电流的有/无。**

信息计量单位

- **对于基于 0 和 1 表示的信息，常用的计量单位**
  - 位（bit，比特，由一个 0 或 1 构成）
    - **计算机中最小的信息单位**
  - **字节（byte，B，由8个二进制位构成）**
    - **存储空间的基本计量单位**
    - 千字节（kilobyte，简称KB，由1024个字节构成）
    - 兆字节（megabyte，简称MB，由1024个千字节构成）
    - 吉字节（gigabyte，简称GB，由1024个兆字节构成）
    - 太字节（terabyte，简称TB，由1024个吉字节构成）
    - 更大的计量单位还有PB（petabyte），EB（Exabyte），ZB（zettabyte），以及YB（yottabyte）。
    - 上述计量单位可以用来衡量内存和外存等的容量，例如，内存的容量可以为512MB、1GB、2GB、8GB等，硬盘的容量可以为40GB、80GB、160GB、500GB等

step by step

进阶

起步：

认知与体验（硬件、软件、程序与C语言）

进阶：

判断与推理（流程控制方法、语句）

抽象与封装（模块设计方法、函数）

表达与转换（基本操作、数据类型）

提高：

构造与访问（数组、指针、结构）

归纳与推广（程序设计的本质）

程序中数据的描述

- 数据类型
- 程序中的基本数据类型（通过关键字或常量来描述简单的数据）
  - 字符型、整型、浮点型、逻辑型、枚举类型
  - 基本类型的选用
    - sizeof()操作符
  - 基本类型的转换
    - 伪随机数的生成
- 程序中的派生数据类型（通过关键字或符号构造新类型来描述复杂的数据）
  - 类型名的自定义

数据（data）

- **程序的处理对象和结果**
  - 表达式的值
  - 数据文件
  - 数据库

数据类型（data type）

- 程序设计语言往往将不同的数据分成不同的类型，并分别用专门的单词/符号来描述。

  求一个*非负整数*的*阶乘*　　把*大写字母*转成*小写字母*再显示

- 一种数据类型可以看成由两个集合构成：
  - **值集**：可以取哪些值(值域，包括这些值的结构)
  - **操作集**：值集中的值可参与哪些操作

程序中的数据可取的值除了受到数据类型的约束之外，
还会受计算机的存储空间和存储方式的限制，
例如，整型的值集只是数学里整数集合的一个子集。

C语言基本类型 (basic types/ fundamental types/ built-in types)

- 字符型 *char*
- 整型
  - 短整型 *short int*
  - 基本整型 *int*
  - 长整型 *long int*
  - 加长整型 *long long int*
- 浮点型
  - 单精度 *float*
  - 双精度 *double*
  - 长双精度 *long double*
- 逻辑型 *bool*

关键字

系统预先定义好，
对应机器指令能
直接操作的数据

值集？
操作集？
变量怎么定义？
常量的形式？
怎么输入？
怎么输出？

字符型

- 用于描述文字、符号类的数据
- 占用1个字节存储空间
  - 值集：256种字符(一般为ASCII码表中规定的字符)
    - [00000000, 11111111]
    - [00, FF]
    - [0, 255]
- 操作集：基本操作
- 变量与常量

实际应用中，数字字符更多是用来描述字符串的一分子，比如，“以3结尾的学号”，而不是用来参加数值运算。
数字字符 '3' - '0' 得到整数 3

signed / unsigned

更关注表示的字符，还是对应的存储的那个整数

```
int main( )
{
        char ch;
        scanf("%c", &ch);
        if('A' <= ch && ch <= 'Z')
                ch += 32;
        printf("%c", ch);
        return 0;
}
```

ch 0100 0001

printf("%c", 65);

转义符（escape sequence）

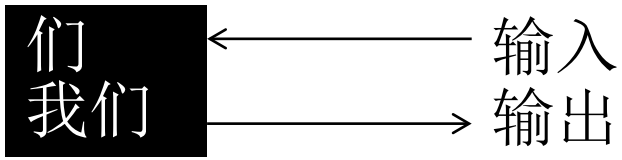
- 由两个单引号（'）括起来的一个特殊字符序列，其中的字符序列以\开头，后面是一个特殊字符或ASCII码
  - **特殊转义符**
    - '\n' (换行符)，'\a' (响铃符)，'\t' (制表符)，'\b' (退格符)，
      '\\' (反斜杠本身)，'\'' (单引号本身)，'\"' (双引号本身)，'\%' (百分号本身)，…
  - **ASCII码转义符**
    - 八进制：'\ddd'，如：'\101'（表示'A'）
    - 十六进制：'\xdd'（或'\Xdd'），如：'\x47'（表示'G'）

一般用于只有数字小键盘的场合

一般用于键盘只能输入数字和少数英文字母的场合

宽字符

```
…
#include <locale.h>
int main( )
{
  setlocale(LC_ALL,"");   //设置为本地区域字符库
  wchar_t wch1 = 25105;   //25105是"我"的机器数
  wchar_t wch2 = getwchar( );
  wprintf(L"%c \n",  wch1, wch2);
  return 0;
}
```



整型

- **用于描述整数**
- 包括：
  - 基本整型（int）
  - 短整型（short int，int可以省略）
  - 长整型（long int，int可以省略）
  - 加长整型（long long int，int可以省略）

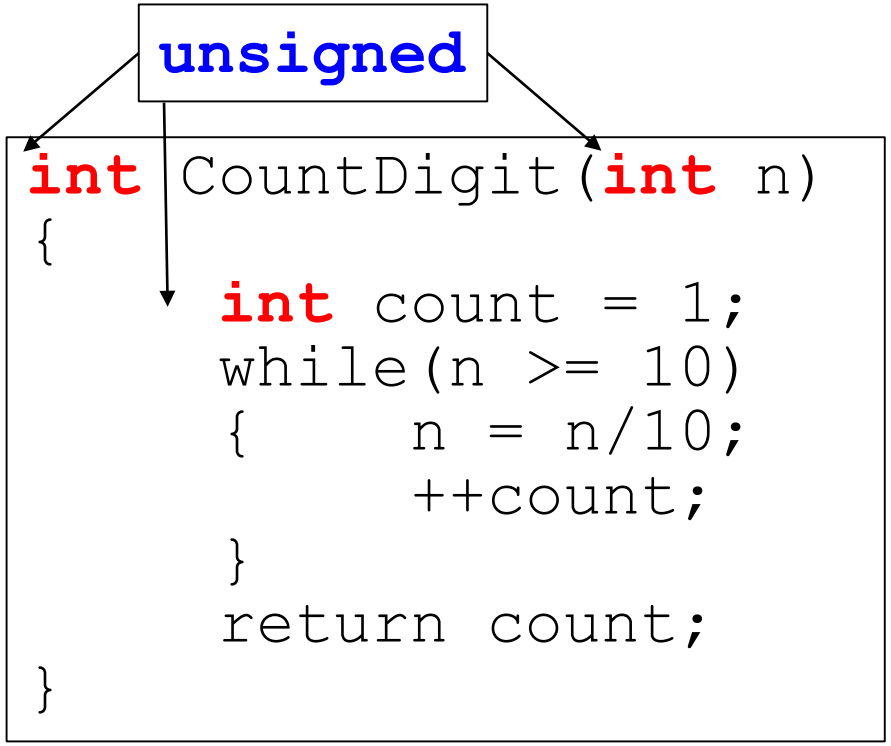
VS2019等还提供了_int64

- **int型数据一般占1个字空间**
  - signed char ⩽ short ⩽ int ⩽ long ⩽ long long
  - 值集：可查看文件 limits.h
    - [-2147483648, 2147483647]
    - [10000000 00000000 00000000 00000000, 01111111 11111111 11111111 11111111]
    - [80000000, 7FFFFFFF]

字（word）：计算机指令处理数据的单位，字的位数叫字长。16位机器的字长一般为16位；32位机的字长一般为32位，当时人们习惯字长为16位，所以又称32位为“双字”。本课程默认字长是32位（4个字节）

- 操作集：基本操作

整型变量与常量



count 00000000 00000000 00000000 00000001

八进制整数：073，0200，-0110；
十六进制整数：0x3B, 0X80, -0x48.

- 默认情况下，整数一般按int型看待，不在int型范围内的整数按unsigned、long型或其他能够表示的类型看待。
- 给整型变量赋值时，最好用同类型的整型常量，比如，
  - `int i = 0;`
  - `long long sum = -2147483648LL;`
  - `unsigned size = 4294967295u;`

整数的越界问题

```
…
int main( )
{
  int a = 2147483647, b = 1;
  printf("%d \n", a + b);
  return 0;
}
```

2147483647 `01111111 11111111 11111111 11111111`

+1 `00000000 00000000 00000000 00000001`

`10000000 00000000 00000000 00000000`

-2147483648

```
printf("%u \n", a + b);
```

2147483648

```
printf("%d", 'A');
```

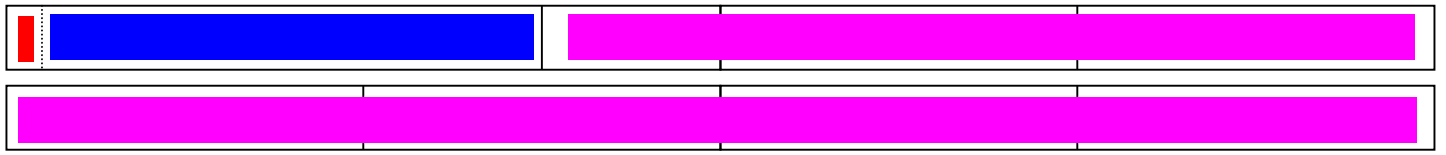
65

浮点型

- **用于描述浮点数**
- 包括：
  - 实浮点型（real floating types）
    - **单精度浮点型（*float*）**
    - **双精度浮点型（*double*）**
    - **长双精度浮点型（*long double*）**
  - 复型（complex types，含有实部和虚部两个元素）
    - 单精度复型（float _Complex）
    - 双精度复型（double _Complex）
    - 长双精度复型（long double _Complex）。

旧标准未规定复型

- **double型数据一般占2个*字*空间**



  - float ⩽ double ⩽ long double
  - 值集：可查看文件 float.h
    [-1.8×10^{308}，1.8×10^{308}]

分辨率：$1\times2^{-53}\approx2.22\times10^{-16}$

- 操作集：基本操作（%除外）

实浮点型变量与常量

```
#include <math.h>
double Dis(double x1, double y1, double x2, double y2)
{
      return fabs(x1 - x2) + fabs(y1 - y2);
}
```

- 默认情况下，实数一般按double型看待。加字母后缀可表示float型或long double型
  - `double d = 2.71828;`
  - `float ave = 82.04f;`
  - `long double disGreatCircle = 8256.012300045678000 09L;`
- C程序中实数的表示
  - **小数**：314.16
  - **科学表示法**：3.1416E2（即3.1416×10^2）
  - 十六进制科学表示法： 0x1.fP-3（即$0x\ 1.f\times16^{-3}$）

```
#include <stdio.h>
int main( )
{
   float x = 0.1F;
   float y = 0.2F;
   float z = x + y;
   if(z == 0.3)
       printf("They are equal.\n");
   else
       printf("They are not equal!);
   return 0;
}
```

```
#include <math.h>
#include <float.h>
if(fabs(z-0.3) <= FLT_EPSILON) //DBL_EPSILON
```

```
#include <math.h>
#define EPSILON 1e-6
```

```
if(z == 0.3F)
```

```
if(fabs(z-0.3) <= EPSILON)
```

- 求级数1 + x + $x^2/2!$ + $x^3/3!$ + … + $x^n/n!$ 的和.

算法1:

```
double x, sum = 1, item, b = 1;
int n, a = 1, i;
scanf("%lf%d", &x, &n);
for (i=1; i <= n; i++)
{   b *= x;        // 计算x^i
    a *= i;        // 计算i!
    item = b/a;    // 计算x^i/i!
    sum += item; // x^i/i!加到sum中
}
printf("sum = %f \n", sum);
```

分别计算 $item_i$ 累积误差更小

算法2:

```
double x, sum = 1, item = 1;
int n, i;
scanf("%lf%d", &x, &n);
for (i=1; i<=n; i++)
{   item *= x/i; // 计算x^i/i!
    sum += item; // x^i/i!加到sum中
}
printf("sum = %f \n", sum);
```

```
%.10f
```

利用 $item_i = item_{i-1}*x/i$ 可行性更好

逻辑型（布尔型）

- 用于描述真假是非这样的逻辑概念

- 逻辑型变量与常量

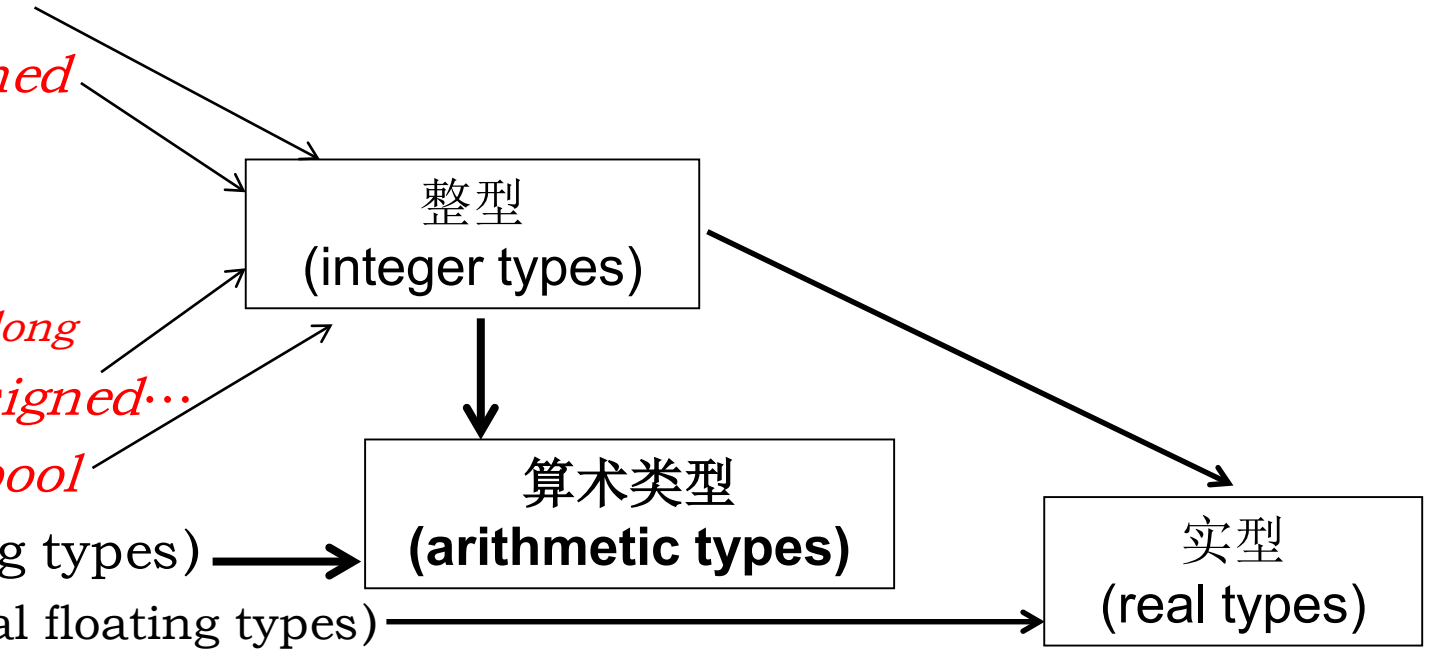
```
_Bool Prime(unsigned i)
{       if(i < 2) return 0;
        unsigned j = 2;
        while(j * j <= i)
        {       if(i%j == 0)
                        return 0;
                ++j;
        }
        return 1;
}
```

```
#include <stdbool.h>
bool Prime(unsigned i)
{       if(i < 2) return false;
        unsigned j = 2;
        while(j * j <= i)
        {       if(i%j == 0)
                        return false;
                ++j;
        }
        return true;
}
```

- 逻辑型数据*占1个字节空间*，实际存放的是0和1
  - 值集：false, true
    - 0, 1
- 操作集：基本操作

基本类型的选用

- **基本类型**
  - 字符型 *char*
  - 有符号整型 *signed*
    - 短整型 *short*
    - 基本整型 *int*
    - 长整型 *long*
    - 加长整型 *long long*
  - 无符号整型 *unsigned…*
  - 逻辑型 *_Bool*/ *bool*
  - 浮点型（floating types）
    - 实浮点型（real floating types）
      - ✓ 单精度 *float*
      - ✓ 双精度 *double*
      - ✓ 长双精度 *long double*
    - 复型



- **基本类型选用原则：**

表达是否自然；
可参与的操作与实际操作是否相符；
值域与实际需求是否协调（是否浪费空间或溢出）。

sizeof()操作符

- 用来计算基本类型或操作数实际占用内存空间的字节数
  - 单目操作符
  - 操作数可以是基本类型的关键字
    - `printf("%d \n", sizeof(long));`
  - 操作数也可以是各种表达式

编译时就能确定其值

    - `printf("%d \n", sizeof(3));`
    - `printf("%d \n", sizeof(x+3));  //double x;`

类型的转换

- 程序执行过程中，要求*双目*操作符连接的两个操作数类型相同
- 当不同类型的操作数参与这类操作时，会进行类型转换
  - 隐式类型转换：由系统自动按一定规则进行的转换
  - 显式类型转换：由程序员在程序代码中标明，强制系统进行的转换

```
int r = 10;
float c = 2 * 3.14 * r;                          //隐式类型转换
double s = 3.14 * (double)r * (double)r;         //显式类型转换
double v = 4.0 / 3 * 3.14 * r * r * r;           //隐式类型转换
printf("%f %f %f"), c, s, v);
```

- 不管哪一种方式，类型转换都是*“临时”*的
- 表达式中的类型转换过程是逐步进行的
  - **4/3***3.14*r*r*r → **1***3.14*r*r*r → ……
  - 3.14*r*r*r*4**/3**

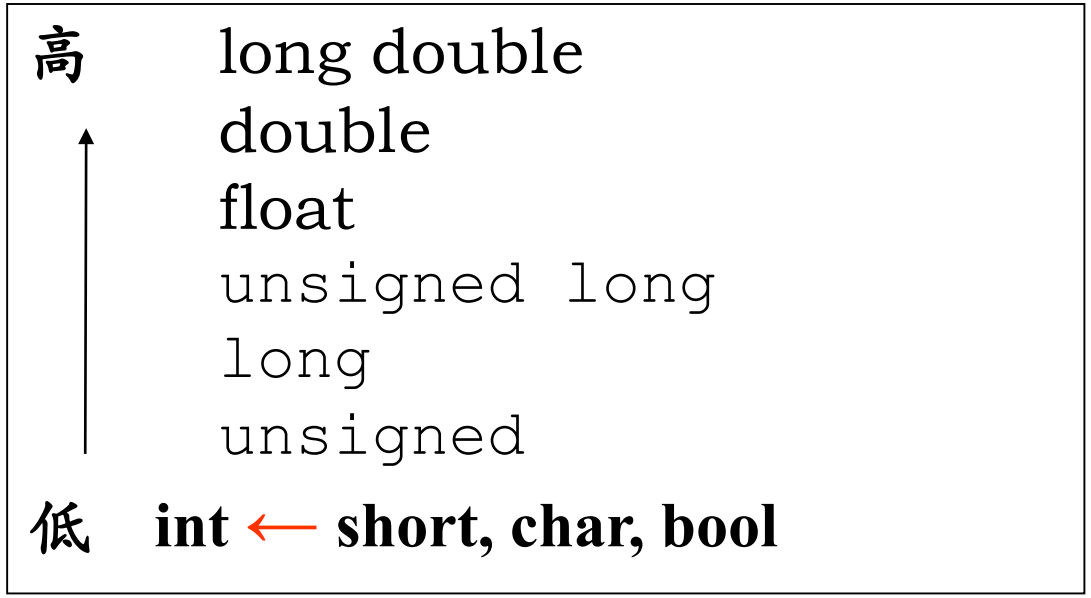
隐式类型转换规则

- 对于赋值操作，右操作数的类型转换为左边变量定义的类型；

- 对于逻辑操作和条件操作第一个表达式中的操作数，以及关系表达式的结果，如果不是逻辑型的数据，非0转换为true，0转换为false；

```
int i, sum=0;
scanf("%d", &i);
if(i)
      sum += i;
printf("%d \n", sum);
```

```
while(i)
```



- 对于其他双目操作（逗号操作除外），按“算术类型转换规则（usual arithmetic conversions）”和“整型提升转换规则（integral promotions）”进行转换。

整型提升转换规则（integral promotions）

1）bool、char、signed char、unsigned char、short int、unsigned short int型的操作数，***如果int型能够表示它们的值，则其类型转换成int***，否则，转换成unsigned int。其中，bool型的操作数，false通常转换为0，true通常转换为1。

2）wchar_t和枚举类型转换成下列类型中第一个能表示其所有值的类型：int、unsigned int、long int、unsigned long int。

算术类型转换规则(usual arithmetic conversions)

1）如果其中一个操作数类型为*long double*，则另一个的类型转换成long double。

2）否则，如果其中一个操作数类型为*double*，则另一个的类型转换成double。

3）否则，如果其中一个操作数类型为*float*，则另一个的类型转换成float。

4）否则，先对操作数进行*整型提升转换*，如果转换后操作数的类型不一样，则按下列规则再进行转换。

5）如果其中一个操作数类型为*unsigned long int*，则另一个的类型转换成unsigned long int。

6）否则，如果一个操作数类型为long int，另一个操作数类型为unsigned int，那么，如果long int能表示unsigned int的所有值，则unsigned int转换成long int，否则，两个操作数的类型都转化成unsigned long int。

7）否则，如果一个操作数类型为*long int*，则另一个的类型转换成long int。

8）否则，如果一个操作数类型为*unsigned int*，则另一个的类型转换成unsigned int。

- 隐式类型转换还会发生在函数调用及其值的返回过程中。

```
bool Prime(unsigned i)
{       if(i < 2) return 0;
        unsigned j = 2;
        while(j * j <= i)
        {       if(i%j == 0)
                        return 0;
                ++j;
        }
        return 1;
}
```

```
int main()
{
        int n;
        scanf("%d", &n);
        if(Prime(n))
                ……;
        return 0;
}
```

实型　去掉小数部分 → 整型

整型　加小数点 → 实型

逻辑型　1或0 → 整型

整型　true或false → 逻辑型

double

```
double a=3.3, b=1.1;
int i = a/b;
printf("%d \n", i); //2
```

```
printf("%.17f\n", 3.3/1.1);
```

2.99999999999999960

设计程序要防止因浮点数的不精确带来的问题

显式类型转换的作用

- 避免计算错误

```
int i = -10;
unsigned int j = 3;
if(i + (int)j < 0)
     printf("二者之和为负数. \n");
else
     printf("二者之和为非负数. \n");

if(i < (int)j)
     printf("i < j \n");
else
     printf("i > j \n");
```

```
double y = 3;
int x = 10 % (int)y;
```

*伪*随机数的生成-强制类型转换的应用

- 实际应用与程序设计中常常需要生成随机数。**随机数的特性是产生前其值不可预测，产生后的多个数之间毫无关系。**
- 真正的随机数是通过物理现象产生的，例如利用激光脉冲、噪声的强度和掷骰子的结果等等，它们的产生**对技术要求往往比较高。**
- **一般情况下，通过一个固定的、可以重复的计算方法产生的伪随机数就可以满足需求，它们具有与随机数类似的统计特征。**
  - 线性同余法是产生伪随机数的常用方法

- **许多编译器的stdlib头文件中定义了生成伪随机数的函数rand/srand和预先定义为32768的宏RAND_MAX，用户可以直接调用。**

```
unsigned long int next = 1;    //全局变量next用来传递随机数的种子
unsigned int rand(void)
{
      next = next * 1103515245 + 12345;   //两个常数是素数的乘积
      return (unsigned int)(next/65536) % RAND_MAX;
} //返回值范围: [0, RAND_MAX-1]
```

```
void srand(unsigned int seed) //seed获取的值赋给next
{
      next = seed;
}
```

- 函数中运用了显示类型转换
- 随机数种子next是在另一个库函数srand中通过参数seed设置的

- **实际上，可以利用系统的时间函数time(0)的返回值来设置seed。**

```
#include <time.h>
#include <stdlib.h> //其中定义了宏RAND_MAX（值为32768）

srand(time(0));
            //time(0)返回从1970年1月1日到程序运行时刻的秒数，会传给形参seed
rand();     //可避免短期内重新运行该程序产生的第一个数与上一组的第一个数相同

for(int i = 0; i < 10; ++i) //产生10个1~10之间的随机数
{
      int j = 1 + (int)(10.0 * rand() / RAND_MAX);
      printf("%d \n", j);
}
```

C语言派生类型（derived types， compound types ）

- 派生类型由程序员构造
  - 枚举 enum
  - 数组 [ ]
  - 指针 *
  - 结构 struct
  - 联合 union

C语言提供了内置的基本类型（关键字）来描述简单的数据。

程序语言一般不提供直接描述复杂数据的关键字。
C语言没有完整的用来定义 派生类型变量 的类型关键字，但提供了***构造派生类型的机制（用已有关键字、符号先构造类型，然后定义变量）***。
这些构造出来的派生类型（C++称为构造类型），可以用来描述复杂数据。

派生类型变量的存储方式和可取的值往往比较复杂，一般不能直接参与基本操作，需要程序员综合运用基本操作符、流程控制方法和模块设计方法设计特别的算法来处理。
课程后续内容将分专题介绍常见的复杂数据的处理

更为复杂的数据则需要用专门的方法（图模型、…）来描述和处理

类型名的自定义

- 用关键字 *typedef* 将已有的类型名定义成另一个类型标识符。例如，
  - `typedef unsigned int Uint;` //Uint 是unsigned int的别名
  - `typedef float Real;` //Real是float的别名
  - `typedef double Speedt, Sumt;` //Speedt和Sumt都是double的别名
- 类型的别名可以用来定义变量：
  - `Uint x;` //等价于unsigned int x;
  - `Real y;` //等价于float y;
  - `Speedt speed1, speed2;` //等价于double speed1, speed2;
  - `Sumt sum1, sum2, sum3;` //等价于double sum1, sum2, sum3;
- 实际上，typedef只是给已有数据类型取别名，并没有定义新类型。其作用是使程序简明、清晰，便于程序的阅读、编写和修改，增强程序的可移植性。特别是对于一些形式比较复杂，易于混淆、出错的类型（派生类型），可以用 typedef 定义成一个容易理解的别名，避免使程序晦涩难懂。

数据类型机制

- 按对数据类型的处理方式，程序设计语言可分为：
  - 静态类型语言与动态类型语言
    - 静态类型语言：要在运行前的程序中指定每个数据的类型
    - 动态类型语言：程序运行中才确定数据的类型
  - 强类型语言与弱类型语言
    - 强类型语言：自动严格检查类型
    - 弱类型语言：不作或很少作类型检查
- **C是静态的弱类型语言，C++是静态的强类型语言**

小结

- 将数据分类描述：
  - 有助于合理分配内存空间 char
  - 便于计算 %
  - 保护数据 *
  - 基本类型的数据通常可以由基本操作符直接操作，除逻辑类型之外数据可以由库函数直接输入、输出。
  - 基本数据类型在参加基本操作时，有可能按一定规则进行隐式或显式的类型转换。

- 要求：
  - 了解基本类型的值集与操作集
  - 掌握基本类型的变量与函数定义方法，恰当选用基本类型实现简单的任务
    - 一个程序代码量≈30行
  - 继续保持良好的编程习惯

Thanks!