

## 一个完整的例子

- ▶ 例0 计算一组圆（直径为n以内的正整数）的周长之和（计量单位为米）。

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

```
#define PI 3.14
```

```
int main( )
```

```
{ int n, d = 1;  
  double sum = 0;  
  char ch = 'm';  
  printf("Input n: ");  
  scanf("%d", &n);
```

```
.....
```

```
return 0;
```

```
}
```

```
while(d <= n)
```

```
{
```

```
    sum = sum + PI * d;
```

```
    d = d + 1;
```

```
}
```

```
printf("The sum is: %f ", sum);
```

```
putchar(ch); //显示计量单位
```

## 一个完整的例子（用C++的输入、输出）

- ▶ 例0 计算一组圆（直径为n以内的正整数）的周长之和（计量单位为米）。

```
#include <iostream>
using namespace std;
const double PI = 3.14;
```

```
int main( )
{ int n, d = 1;          //
  double sum = 0;        //
  char ch = 'm';
  cout << "Input n: " ;
  cin >> n;

  .....
  return 0;
}
```

```
while(d <= n)
{
    sum = sum + PI * d;
    d = d + 1;
}
```

```
cout << "The sum is: " << sum;
cout << ch;
```

## 编程实现猜数字游戏Guess Number

- 通常由两个人玩，一方出数字，一方猜。一个简单的版本：出数字的

需要两个变量表示“出的数字”和“猜的数字”。

人想好一个数字，猜的人就可以开始猜，每猜一个数字，出数者就要

需要判断大小，相等Great! 否则无论猜大了还是小了都要继续猜。

根据这个数字给出是猜大了还是小了，这样猜的人继续猜，直到猜对

“继续猜”是一种“迭代”，采用循环流程实现！

为止。

猜对了，功成名就！再“玩”一次或程序退出；再“玩”一次就是外层嵌套一个大循环。

过程抽象：把一次猜的过程用单独的函数进行实现。

# 内容回顾

// 根据调用的函数不同, 往往需要包含不同的头文件

#include <stdio.h> // 输入输出库

#include <stdlib.h> // 标准库

#include <time.h> // 包含时间函数

using namespace std; // 使用std名空间, 配合#include <iostream>

void menu(); // 函数声明, 函数的调用在定义点前, 需要先声明

void playgame(); // 函数声明, 函数的调用在定义点前, 需要先声明

int main() { // 主函数, 每个C/C++程序都有一个main函数

int input = 0; // 定义一个整型变量input, 初值为0

srand((unsigned int)time(0)); // 获取系统时间, 作为随机数的种子

do { // while, do while 循环语句

menu(); // 执行 menu函数

scanf\_s("%d", &input); // 键盘键入一个值, 赋给input

switch (input) { // switch语句根据input的值, 选择执行不同分支

case 1:

playgame(); // 当input等于1, 执行playgame函数

case 0:

exit(0); // 当input等于0, 退出程序, 不玩了!

break; // break 跳出循环

default:

printf("输入错误! \n"); // 输出窗口输出"输入错误! "并换行

}

} while (input);

return 0;

// 返回0给系统。

# 内容回顾

```
void menu() {                                     // menu函数的定义, void表示返回空类型
    printf("*****\n");                          // 输出窗口输出*****并换行
    printf("****1.plat****\n");                  // 输出窗口输出****1.plat****并换行
    printf("****0.exit****\n");                  // 输出窗口输出****0.exit****并换行
    printf("*****\n");                          // 输出窗口输出*****并换行
}

void playgame() {                                // playgame函数的定义, void表示返回空类型
    int magic = rand() % 100 + 1;                // 得到一个1-100的随机数
    int guess;                                  // 定义一个表示用户输入的值的整形guess
    int counter = 0;                            // 定义一个表示计数器的整形counter
    // while (1) {                               // 循环语句, 当 () 内表达式为true, 执行循环
        do {                                     // 循环语句, 当guess和magic不相等, 执行循环
            printf("请猜一个数字: ");            // 输出窗口输出请猜一个数字:
            scanf_s("%d", &guess);              // 用户键盘输入一个数字, 赋给guess
            counter++;                            // 计数器counter值加一
            if (guess > magic)                    // 如果guess大于magic
                printf("太大了! \n");           // 输出窗口输出: 太大了! 并换行
            else if (guess < magic)               // 如果guess小于magic
                printf("太小了! \n");           // 输出窗口输出: 太小了! 并换行
            else                                  // 否则的话
                printf("Great!\n");             // 输出窗口输出: Great!
        } while (guess != magic);                // 循环语句, 当guess和magic不相等, 执行循环
        printf("您一共猜了%d次, 你真是太棒了! \n", counter);
        // 输出窗口输出: 您一共猜了counter次, 你真是太棒了! 并换行
    }
}
```

## 本课程自定义标识符命名具体建议☆

---

前缀	类型	例子
n	int	nLength
c	char	cGrade
f	float	fScore
a	数组	aStu
p	指针	pFunc
C	类或者结构体	CDocument, CPrintInfo
g_	全局变量	g_Servers
m_	成员变量	m_pDoc, m_nCustomers



## 培养基本的编程思维

---

- ▶ **读题**：读懂题意、确定解法“算法”
  - ▶ **将算法思路转换为程序代码**
    - ▶ 变量定义：考虑清楚需要什么变量、类型是什么？
    - ▶ 确定程序流程结构：顺序、选择、循环...
    - ▶ 确定程序的主要功能函数，并分别定义（实现）
    - ▶ 主函数调用功能函数
  - ▶ **程序验证和调试Debug**
-

# 2 基本数据类型

郭延文

2022级苏州校区技术科学试验班



# 主要内容

---

- ▶ 数据类型的概念
- ▶ 常量与变量
- ▶ 基本数据类型
- ▶ 数据类型转换



# 主要内容

---

- ▶ 数据类型的概念
- ▶ 常量与变量
- ▶ 基本数据类型
- ▶ 数据类型转换



# 数据类型

---

- ▶ 数据是程序的一个重要组成部分，每个数据都属于某种数据类型。
- ▶ 一种数据类型可以看成由两个集合构成：
  - ▶ **值集**：规定了该数据类型能包含哪些值（包括这些值的结构）。
  - ▶ **操作集**：规定了对值集中的值能实施那些运算。
- ▶ 例如：**整型**就是一种数据类型，其值集就是由整数所构成的集合，操作集包括：加、减、乘、除等运算。



# 区分数据类型的好处

---

- ▶ 对数据进行分类，便于对数据进行描述、存储和处理；
- ▶ 提高程序的可靠性，便于编译程序自动进行类型一致性检查；
- ▶ 提高程序效率，便于产生高效的可执行代码。



# C/C++数据类型

---

## ▶ 基本数据类型

- ▶ C++语言预先定义好的数据类型，常常又称为**标准数据类型**或**内置数据类型**（built-in types），它们都是简单类型。

## ▶ 构造（复合）数据类型

- ▶ 用户利用语言提供的**类型构造机制**从其它类型构造出来的数据类型，大多为复合数据类型（枚举类型除外）。

## ▶ 抽象数据类型

- ▶ 用户利用**数据抽象机制**把数据与相应的操作作为一个整体来描述的数据类型(类)，一般为复合数据类型。
- 



# C/C++ 数据类型

## 基本数据类型

- 整数类型
- 实数类型
- 字符类型
- 逻辑类型
- 空值类型

## 构造数据类型

- 枚举类型
- 数组类型
- 结构与联合类型
- 指针类型
- 引用类型

## 抽象数据类型

- 类
- 派生类

# 主要内容

---

- ▶ 数据类型的概念
- ▶ 常量与变量
- ▶ 基本数据类型
- ▶ 数据类型转换



# 数据在程序中的表示

---

- ▶ 在程序中，数据以两种形式出现：
  - ▶ **常量**：用于表示在程序执行过程中不变（或不能被改变）的数据。
  - ▶ **变量**：用于表示在程序执行过程中可变的数数据。
  - ▶ 例如，在计算圆的周长表达式 $2*PI*r$ 中，
    - ▶ 2和圆周率PI是常量。
    - ▶ 半径r是变量，它的值可能在程序运行时从用户处得到，或由程序的其它部分计算得到。



# 常量

---

- ▶ 在C++程序中，常量可以用两种形式表示：
    - ▶ 字面常量：在程序中通过直接写出常量值来使用的常量，通常又称为直接量
    - ▶ 符号常量（命名常量）：通过常量定义给常量取一个名字并指定一个类型，在程序中通过常量名来使用这些常量
-

# 符号常量

- ▶ 符号常量是指先通过常量定义给常量取一个名字，并可指定一个类型；然后，在程序中通过常量名来使用这些常量。
- ▶ 符号常量的定义格式为：

`#define <常量名> <值> // C`

或

`const <类型名> <常量名>=<值>; // C++`

例如：

`#define PI 3.1415926`

或，

`const double PI=3.1415926;`

- ▶ 符号常量的使用：

`2*PI*r;` (此为语句；也可作为表达式参加运算)

# 使用符号常量的好处

---

- ▶ 保证程序对常量使用的一致性

- ▶ 圆周率:  $\text{PI} = 3.141\ 5926\ 5358$

- ▶ 增加程序的易读性

- ▶  $\text{PI} = 3.141\ 5926$ ; // 简明扼要

- ▶ `const int PASS_SCORE=60;`

- ▶ `const int MINUTES_PER_HOUR=60`

- ▶ 增强程序的易维护性

- ▶  $\text{PI} = 3.141\ 5926$ ; // 当需要修改其精度时...

# 变量的定义

---

- ▶ C语言规定：程序中使用到的每个变量都要有定义（有的语言不需要）。变量定义格式为：

**<类型名> <变量名>;**

或者

**<类型名> <变量名>=<初值>;**

例如：

```
int a=0;
```

```
int b=a+1;
```

```
double x;
```

或：

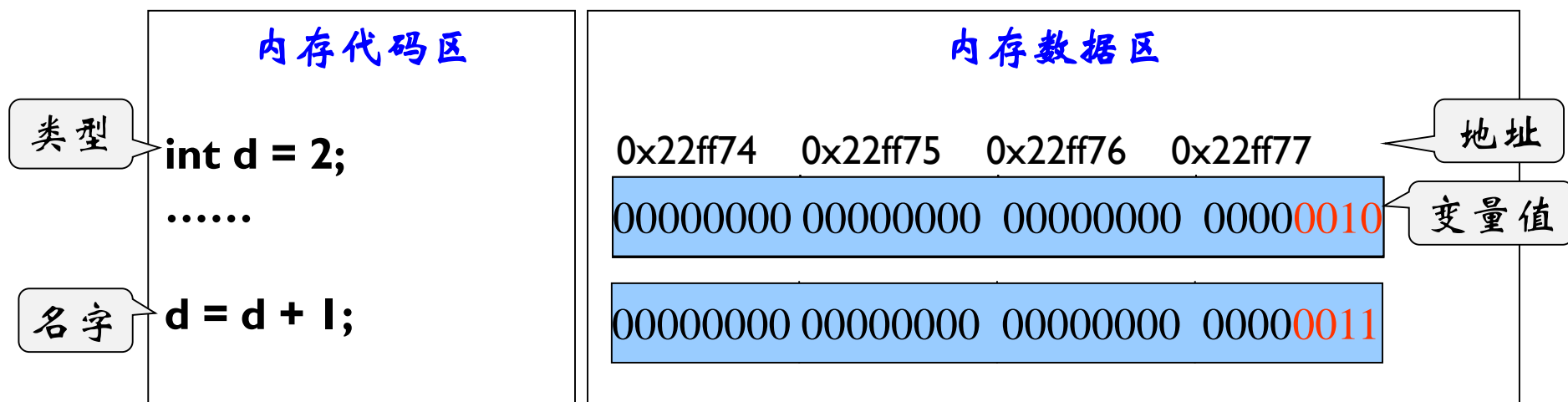
```
int a=0, b=a+1; //同类型变量可以写在一起，用‘,’分开
```

```
double x;
```

---

# 变量的属性

- ▶ 程序执行到变量定义处，系统会为变量分配一定大小的空间，用以存储变量的值。
- ▶ 存储空间里起初是一些0/1组成的无意义的值，可以通过赋值或输入值来获得有意义的值；存储空间由地址来标识，一般由系统自动管理。



用户指定变量类型和名字，  
系统决定地址，存储二进制值。

# 主要内容

---

- ▶ 数据类型的概念
- ▶ 常量与变量
- ▶ **基本数据类型**
- ▶ 数据类型转换



# C/C++基本数据类型

---

- ▶ 整数类型（整型）
- ▶ 实数类型（实型）
- ▶ 字符类型
- ▶ 逻辑类型
- ▶ 空值类型



# 整数类型

---

- ▶ 整数类型用于描述通常的整数。根据精度分成：
    - ▶ int
    - ▶ short int 或 short
    - ▶ long int 或 long
  - ▶ 一般情况下，  
“short int”的范围 ≤ “int”的范围 ≤ “long int”的范围
    - ▶ 具体大小由实现决定，例如
      - ▶ short int 占2个字节 [-32768 ~ 32767]
      - ▶ int 占4个字节 [-2147483648 ~ 2147483647]
      - ▶ long int 占4个字节 [-2147483648 ~ 2147483647]
      - ▶ 具体的值域可以查看文件limits.h
  - ▶ 在计算机内部，整数一般用补码表示。
-



# 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, 由一个0或1 构成)
    - ▶ 计算机中最小的信息单位
  - ▶ 字节 (byte, 由8个2进制位构成)
    - ▶ 存储空间的基本计量单位
    - ▶ 千字节 (kilobyte, 简称KB, 由1024byte构成)
    - ▶ 兆字节 (megabyte, 简称MB, 由1024KB构成)
    - ▶ 吉字节 (gigabyte, 简称GB, 由1024MB构成)
    - ▶ 太字节 (terabyte, 简称TB, 由1024GB构成)
    - ▶ 更大的计量单位还有PB (petabyte), EB (Exabyte), ZB (zettabyte), 以及YB (yottabyte)。
  - ▶ 字 (word)
    - ▶ 计算机进行数据处理和运算的单位
    - ▶ 由若干个字节构成, 字的位数叫**字长**, 不同档次的机器有不同的字长。例如32位机的字长为32位, 其1个字由4个字节构成。

# 整型的值域

(以32位机为例, int型数据的取值范围)

---

▶ 用二进制表示:

正数: 00000000000000000000000000000000 ~  
01111111111111111111111111111111

零: 10000000000000000000000000000000、

负数: 10000000000000000000000000000001 ~  
11111111111111111111111111111111

**$[-2147483648 \sim 2147483647]$**

# 实数类型(浮点型)

---

- ▶ 实数类型又称浮点型，它用于描述通常的实数。根据精度可分为：
  - ▶ float（单精度型）
  - ▶ double（双精度型）
  - ▶ long double（长双精度型）



# 值域与精度（以32位机为例）

---

- ▶ 单精度 (float): 4个字节

- ▶ 值域大约是:  $-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$
- ▶ 能够表示的最小正数大约是  $1.175 \times 10^{-38}$
- ▶ 分辨率大约是  $1.192 \times 10^{-7}$ , 即有6位数字有效

- ▶ 双精度 (double): 8个字节

- ▶ 值域大约是:  $-1.8 \times 10^{308} \sim 1.8 \times 10^{308}$
- ▶ 能够表示的最小正数大约是  $2.225 \times 10^{-308}$
- ▶ 分辨率大约是  $2.22 \times 10^{-16}$ , 即有15位数字有效

- ▶ 长双精度 (long double)

- ▶ 表示方法: IEEE 754标准

- ▶ 具体值域和精度可以查看文件float.h

---



# 如何选择float, double, long

---

- ▶ 使用double类型基本上“不会有错”（符合精度要求）
  - ▶ 8字节共64位，能保证 $10^{-15}$ 的所有精度。
- ▶ 在float类型中隐式的精度损失是不能忽视的，双精度计算的代价相对于单精度可以忽略
  - ▶ float型只能保证6位有效数字，而double型至少可以保证15位有效数字（小数点后的数位），long double型提供的精度通常没有必要，而且还要承担额外的运行代价



# 特别注意!

---

- ▶ 对浮点数进行关系操作时, 往往得不到正确的结果, 应避免对两个浮点数进行 “==” 和 “!=” 操作

- ▶  $x == y$       可写成:  $\text{fabs}(x-y) < 1e-7$

- ▶  $x != y$       可写成:  $\text{fabs}(x-y) > 1e-7$

- ▶  $z == 0.3$       可写成:  $\text{fabs}(z-0.3) < 1e-7$

或

```
#define ZERO 1e-7  
#include <cmath>  
 $\text{fabs}(x-y) < \text{ZERO}$ 
```

## 例：c语言实浮点型数据的输入/输出

---

```
#include<stdio.h>
int main( )
{
    float x, y = 12.3456779F;
    scanf("%f", &x);
    printf("%f \n", x);
    printf("%.11f \n", y);
    printf("%e \n", 3.14159265);
    printf("%e \n", 0x1f);
    return 0;
}
```

- ▶ **%e**是按科学计数法显示结果，默认情况下结果占13格，其中，小数点前的整数部分与小数点本身各占1格，小数部分占6格，然后是字母e与正（负）号各占1格，指数部分占3格。

# printf的输出格式符

[http://blog.163.com/chen\\_dawn/blog/static/11250632011101741153221/](http://blog.163.com/chen_dawn/blog/static/11250632011101741153221/)

---

`%a(%A)` 浮点数、十六进制数字和p-(P-)记数法(C99)

`%c` 字符

`%d` 有符号十进制整数

`%f` 浮点数(包括float和double)

`%e(%E)` 浮点数指数输出[e-(E-)记数法]

`%g(%G)` 浮点数不显无意义的零"0"

`%i` 有符号十进制整数(与%d相同)

`%u` 无符号十进制整数

`%o` 八进制整数 e.g. 0123

`%x(%X)` 十六进制整数0f(0F) e.g. 0x1234

`%p` 指针

`%s` 字符串

`%%` "%" 

---



# 字符类型

---

- ▶ 字符类型用于描述文字类型数据中的一个字符。
- ▶ 字符在计算机内存储的是它的编码(对应的“机器数”)

A 

01000001
----------

a 

01100001
----------

- ▶ C标准规定普通**字符型数据**在计算机中占用**1个字节**空间，即8个2进制位空间。
- ▶ 根据字符型数据在计算机中占用空间的大小，可以推算出其取值范围。

▶ 值域：

- ▶ 2进制数为00000000~01111111、10000000、10000001~11111111，
- ▶ 对应的十六进制数为00~7F、80、81~FF，
- ▶ 对应的十进制数为0~127、128、129~255，
- ▶ 对应的**256**种字符一般为**ASCII**码表中规定的字符。

A 65 **01000001**

a 97 **01100001**

```
printf("%c\n", 65);    // printf("%c\n", 97);
```

# 常用的字符集及其编码

---

- ▶ ASCII码(美国标准信息交换码American Standard Code for Information Interchange):
    - ▶ 解决常用西文字的存储问题:
      - ▶ 0~9十个数字字符、
      - ▶ 26个大写英文字母以及26个小写英文字母的编码各自是连续的
      - ▶ 其它一些常用符号(如标点符号、数学运算符等)
    - ▶ 将字符转换成二进制数的标准代码;方便起见,常常用十进制数或十六进制数来描述二进制ASCII码
    - ▶ 在C++中用char类型描述
-

# ASCII码表八、十六、十进制对照表

<http://www.feiesoft.com/00007/>

41	21	33	!	141	61	97	a
42	22	34	"	142	62	98	b
43	23	35	#	143	63	99	c
44	24	36	\$	144	64	100	d
45	25	37	%	145	65	101	e
46	26	38	&	146	66	102	f
47	27	39	'	147	67	103	g
50	28	40	(	150	68	104	h
51	29	41	)	151	69	105	i
52	2a	42	*	152	6a	106	j
53	2b	43	+	153	6b	107	k
54	2c	44	,	154	6c	108	l
55	2d	45	-	155	6d	109	m
56	2e	46	.	156	6e	110	n
57	2f	47	/	157	6f	111	o
60	30	48	0	160	70	112	p
61	31	49	1	161	71	113	q
62	32	50	2	162	72	114	r
63	33	51	3	163	73	115	s
64	34	52	4	164	74	116	t
65	35	53	5	165	75	117	u
66	36	54	6	166	76	118	v
67	37	55	7	167	77	119	w
70	38	56	8	170	78	120	x
71	39	57	9	171	79	121	y
72	3a	58	:	172	7a	122	z
73	3b	59	;	173	7b	123	{
74	3c	60	<	174	7c	124	
75	3d	61	=	175	7d	125	}
76	3e	62	>	176	7e	126	~
77	3f	63	?	177	7f		

# 字符类型允许的操作集

---

- ▶ 算术操作
- ▶ 关系和逻辑操作
- ▶ 位操作
- ▶ 赋值操作
- ▶ 条件操作
- ▶ .....

实际上是其对应的**ASCII码**在参与操作



# 字符类型常量

---

- ▶ 在C++程序中，字符常量是由两个单引号（' '）括起来的一个字符构成，其中的字符写法可以是：
  - ▶ 字符本身，如：'A'，'3'
  - ▶ 注意： 3 V.S. '3'

# 字符型变量

---

- ▶ 定义字符型变量时用 **char**，可以加类型修饰符。
  - ▶ **char**: 一般被看作 **signed char** (VS 2013)
  - ▶ **signed char** (-128-127)
  - ▶ **unsigned char** (0-255)
  - ▶ **wchar\_t** (宽字符)
    - ▶ 可以描述: **Unicode**(国际通用字符集)

具体的值域可以查看文件 `limits.h`

---



# 转义字符（“转换意义的字符”）

---

- ▶ ‘\n’（换行符）、‘\r’（回车符）、‘\t’（横向制表符）、‘\b’（退格符）、‘\a’（响铃）等

- ▶ 注意下列字符的表示：

- ▶ 反斜杠（\）应写成：`\"`
- ▶ 单引号（'）应写成：`'`
- ▶ 双引号（"）可写成：`\"`或

**要知道：当cout或printf打印相应符号时用！**



# 关于字符的操作

char ch;

(ch >= 'a' && ch <= 'z' ) || ( ch >= 'A' && ch <= 'Z')

(ch >= '0' && ch <= '9')

## 字符型数据

▶ 字符变量：ch

▶ 字符常量：

'a' 'z' 'A' 'Z' '0' '9' '\n'

ASCII字符集：列出所有可用的字符

每个字符：惟一的次序值（ASCII 码）

'0'-'9'

'A'-'Z'

'a'-'z'

ASCII 码表

符 号	10进制	符 号	10进制	符 号	10进制	符 号	10进制
@	64	P	80	,	96	p	112
A	65	Q	81	a	97	q	113
B	66	R	82	b	98	r	114
C	67	S	83	c	99	s	115
D	68	T	84	d	100	t	116
E	69	U	85	e	101	u	117
F	70	V	86	f	102	v	118
G	71	W	87	g	103	w	119
H	72	X	88	h	104	x	120
I	73	Y	89	i	105	y	121
J	74	Z	90	j	106	z	122
K	75	[	91	k	107	{	123
L	76	\	92	l	108		124
M	77	]	93	m	109	}	125
N	78	^	94	n	110	~	126
O	79	-	95	o	111		127

# 关于字符的操作

## 字符型数据的输入和输出

### ▶ 字符输入函数getchar

输入一个字符

```
char ch;
```

```
ch = getchar();
```

### ▶ 字符输出函数putchar

输出一个字符

```
putchar(输出参数);
```



字符常量或字符变量

调用scanf和printf输入输出字符

```
double value1, value2;
```

```
char operator;
```

```
printf("Type in an expression: ");
```

```
scanf("%lf%c%lf", &value1, &operator, &value2);
```

```
printf("%.2f %c %.2f", value1, operator, value2);
```

```
char ch;  
ch = getchar();  
putchar (ch);  
putchar ('?');
```

a  
a?

## 重点：字符结合ASCII码的操作

例：用格式符%c将各种类型的数据显示为字符

---

```
#include <stdio.h>
int main()
{
    printf("ASCII code 65 in decimal represents the character: %c \n", 'A');
    printf("ASCII code 65 in decimal represents the character: %c \n", 65);
    return 0;
}
```

```
ASCII code 65 in decimal represents the character: A
ASCII code 65 in decimal represents the character: A
```

# ASCII码表八、十六、十进制对照表

<http://www.feiesoft.com/00007/>

41	21	33	!	141	61	97	a
42	22	34	"	142	62	98	b
43	23	35	#	143	63	99	c
44	24	36	\$	144	64	100	d
45	25	37	%	145	65	101	e
46	26	38	&	146	66	102	f
47	27	39	'	147	67	103	g
50	28	40	(	150	68	104	h
51	29	41	)	151	69	105	i
52	2a	42	*	152	6a	106	j
53	2b	43	+	153	6b	107	k
54	2c	44	,	154	6c	108	l
55	2d	45	-	155	6d	109	m
56	2e	46	.	156	6e	110	n
57	2f	47	/	157	6f	111	o
60	30	48	0	160	70	112	p
61	31	49	1	161	71	113	q
62	32	50	2	162	72	114	r
63	33	51	3	163	73	115	s
64	34	52	4	164	74	116	t
65	35	53	5	165	75	117	u
66	36	54	6	166	76	118	v
67	37	55	7	167	77	119	w
70	38	56	8	170	78	120	x
71	39	57	9	171	79	121	y
72	3a	58	:	172	7a	122	z
73	3b	59	;	173	7b	123	{
74	3c	60	<	174	7c	124	
75	3d	61	=	175	7d	125	}
76	3e	62	>	176	7e	126	~
77	3f	63	?	177	7f		

## 例： 格式符%d将各种类型的数据显示为十进制整数

```
#include <stdio.h>
int main( )
{
    printf("ASCII code of the character is: %d \n", 'A');
    printf("ASCII code is: %d \n", 65);
    printf("ASCII code in decimal is: %d \n", 0x41);
    printf("ASCII code of the character is: %d \n", '7');
    printf("ASCII code of the character is: %d \n", '\a'); // 转义字符响铃
    return 0;
}
```

ASCII code of the character is: 65

ASCII code is: 65

ASCII code in decimal is 65

ASCII code of the character is: 55

ASCII code of the character is: 7

## 例：数字字符与整数的区别示例

```
#include<stdio.h>
int main( )
{
    int i = 3;
    char ch = '3';
    printf("10i = %d, 10ch = %d \n", 10 * i, 10 * ch);
    return 0;
}
```

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。

30, 510

3

'3'

00000011

00110011

# ASCII码表八、十六、十进制对照表

<http://www.feiesoft.com/00007/>

41	21	33	!	141	61	97	a
42	22	34	"	142	62	98	b
43	23	35	#	143	63	99	c
44	24	36	\$	144	64	100	d
45	25	37	%	145	65	101	e
46	26	38	&	146	66	102	f
47	27	39	'	147	67	103	g
50	28	40	(	150	68	104	h
51	29	41	)	151	69	105	i
52	2a	42	*	152	6a	106	j
53	2b	43	+	153	6b	107	k
54	2c	44	,	154	6c	108	l
55	2d	45	-	155	6d	109	m
56	2e	46	.	156	6e	110	n
57	2f	47	/	157	6f	111	o
60	30	48	0	160	70	112	p
61	31	49	1	161	71	113	q
62	32	50	2	162	72	114	r
63	33	51	3	163	73	115	s
64	34	52	4	164	74	116	t
65	35	53	5	165	75	117	u
66	36	54	6	166	76	118	v
67	37	55	7	167	77	119	w
70	38	56	8	170	78	120	x
71	39	57	9	171	79	121	y
72	3a	58	:	172	7a	122	z
73	3b	59	;	173	7b	123	{
74	3c	60	<	174	7c	124	
75	3d	61	=	175	7d	125	}
76	3e	62	>	176	7e	126	~
77	3f	63	?	177	7f		

# 字符型变量值的输入及其参与关系和算术操作 (重点)

## 例：对输入的大写字母A-Z，转化为小写字母

---

```
#include <stdio.h>
int main( )
{  char ch;
   do
   {
       printf("Input Y or N (y or n) :");
       scanf("%c", &ch); // ch = getchar();
       if(ch >= 'A' && ch <= 'Z')
           ch += 32;
       // 可改写为: ch = (ch >= 'A' && ch <= 'Z') ? ch + 'a' - 'A' : ch
       printf("%c", ch);
   } while(ch != 'y' && ch != 'n');
   if(ch == 'y')
       .....
   else
}
```

---



# 字符串常量

---

- ▶ 在C++程序中，字符串常量是由两个双引号（“ ”）括起来的字符序列构成，其中的字符的写法与字符类型常量基本相同，包含字符本身和转义序列。如：
    - ▶ "This is a string."
    - ▶ "I'm a student."
    - ▶ "Please enter \"Y\" or \"N\":"
    - ▶ "This is two-line \n message! "
  - ▶ 存储字符串时，往往要在最后一个字符的后面存储一个字符 `'\0'`，表示字符串结束。
  - ▶ 字符串常量的类型为 **一维的常量字符数组**（构造数据类型）。
-

# C语言数组 (array)

---

- ▶ 数组用以表示**确定个数**的**同类型**数据按**一定次序**构成的数据群体
- ▶ 数据群体中的每个个体即数组中的每个**元素**
- ▶ 对于数组类型的数据，编译器将在内存中分配**连续的空间**来存储数组元素
- ▶ 通常情况下，对数组**不能进行整体操作**

# 一维数组变量的定义

---

两种方式：

1. 先构造类型，再定义变量，便于定义多个同类型变量

```
typedef int A[6];
```

```
A a, b;    //定义了一个一维数组a和一个一维数组b
```

2. 构造类型的同时定义变量

```
int a[6];    // int、[]和6构造了一个一维数组类型
```

```
    //并用该类型定义了一个一维数组a
```



# 字符数组

---

## ▶ 字符数组的定义

```
char str[10];
```

- ▶ 用char、[]和10构造了一个一维字符数组类型，并用该类型定义了一个一维字符数组str
- ▶ 系统会为该字符数组分配10个内存单元  
(10\*sizeof(char))，以存储10个字符型元素

# 字符数组的初始化

```
char str[10] = {'H', 'e', 'l', 'l', 'o', ' ', 'N', 'J', 'U', '\0'};
```

- ▶ 该初始化方式最好在最后加一个字符串结束标志'\0'（转义符，对应的ASCII码是0）。

```
char str[10] = "Hello NJU";
```

- ▶ 该初始化方式不必加'\0'，因为C语言中的字符串常量最后会自动加'\0'，并赋给定义的数组。
- ▶ 如果 **const** char str[10] = "Hello NJU";  
//这样数组元素的值不可更改

- ▶ 定义的字符数组长度应该保证足以存储该结束标志，这是一个约定俗成的做法，可以方便字符串的相关操作。

- 
- ▶ 如果是如下的不完全初始化形式:

```
char str[10] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char str[10] = "Hello";
```

- ▶ 系统会为str分配10个字节的内存空间，其前6个元素被初始化为'H'、'e'、'l'、'l'、'o'、'\0'，没有被初始化的元素均默认为'\0'。

- ▶ 如果是如下的初始化形式:

```
char str[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char str[] = "Hello";
```

- ▶ 字符数组的长度省略后，系统为str只分配6个字节的内存空间

- 
- ▶ 如果输出没有结束符的字符串，则在字符串的后面会显示若干乱码（未使用过的内存初始信号往往是汉字“烫”的机内码）。

- ▶ 比如，

```
char str[] = {'H', 'e', 'l', 'l', 'o'};
```

```
printf("The string is %s. \n", str); //通常会显示Hello烫烫...
```

- ▶ 用格式符%s输出str时，会将str数组里的字符以及其后的若干乱码全部输出，直至遇到0为止（内存中总有一些单元里的信号是0（'\0'的ASCII码））。应防范出现乱码问题！

# 字符数组的输入/输出

- ▶ 用 `cin>>`, `cout<<` (C++)

```
char* str;
```

```
cin>>str;
```

当键入的字符串为: Object\_Oriented Programming!

结果是: str指向的字符串为: Object\_Oriented

- ▶ 通过 `scanf/printf` 函数 (C), 借助格式符 `%s` 输入/输出 (一个 `%s` 对应一个字符数组名):

```
char str[10];
```

```
scanf( "%s" , str);      //输入一个字符串至str
```

```
printf( "The string is %s. \n" , str); //将str里的字符串输出
```

用 `cin/scanf` 的输入, 遇到空白字符为止!



# 关于字符串的操作（用库函数）

---

## ➤ **gets()** 函数

用来从标准输入设备(键盘)读取字符串直到回车结束,但回车符不属于这个字符串,其调用格式为:gets(s);

与scanf("%s",&s)相似但不完全相同.使用scanf("%s",&s) 函数输入字符串时如果输入了空格会认为输入字符串结束,空格后的字符将作为下一个输入项处理.但gets() 函数将接收输入的整个字符串直到回车为止。

## ➤ **puts()** 函数

用来向标准输出设备(屏幕)写字符串并换行,其调用格式为:puts(s); 其中s为字符串变量(字符串数组名或字符串指针)。puts()函数的作用与语printf("%s\n",s)相同。

---

# 关于字符串的操作（用库函数）

---

- ▶ 用库函数gets/puts输入/输出一个字符串：

gets(str); / puts(str);

以回车符作为输入的结束标志，也就是说回车符不会转存到str中，不过gets可以读取空格符！（解决cin和scanf的空格后不能输入的问题）

getchar(str[i]); / putchar(str[i]);

单字符的输入和输出

# 逻辑类型

---

- ▶ 逻辑类型用于描述“真”和“假”这样的逻辑值，分别表示条件的满足和不满足
  - ▶ 在C++中，逻辑类型用**bool**表示，它的值只有两个：**true**和**false**，分别对应“真”和“假”
  - ▶ 在大多数的C++实现中，**bool**类型的值一般占用一个字节的空間，**true**存储的是**1**，**false**存储的是**0**
    - ▶ 反过来：非零即是真！例如 `if (-3) {...}`
-

# 回顾

---

```
... ..  
int main()  
{  
    cout<<"Hello world!"<<endl;  
    return 0;  
}
```

```
... ..  
main()  
{  
    cout<<"Hello world!"<<endl;  
}
```

```
... ..  
void main()  
{  
    cout<<"Hello world!"<<endl;  
}
```



# 空值类型

---

- ▶ 在C++中提供了一种值集为空的类型：空值型

(void)，用以表示：

- ▶ 没有返回值的函数的返回类型
- ▶ 通用指针类型 (void \*)



# 数据类型

## 基本数据类型

- 整数类型
- 实数类型
- 字符类型
- 逻辑类型
- 空值类型

## 构造数据类型

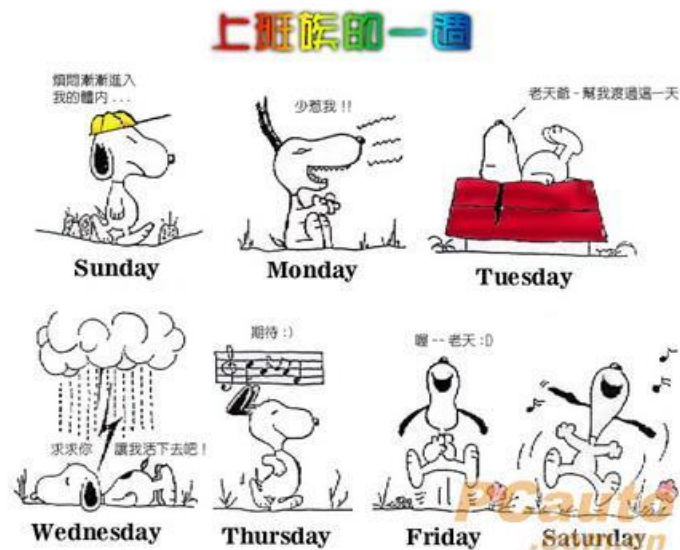
- 枚举类型
- 数组类型
- 结构与联合类型
- 指针类型
- 引用类型

## 抽象数据类型

- 类
- 派生类

# 可否有这样的数据类型

- ▶ 表示day, 其取值为一周的七天



- ▶ 表示color, 其取值为“赤、橙、黄...”



- ▶ ...

# 枚举类型

- ▶ 程序员用关键词enum构造出来的数据类型，程序员构造这种类型时，要逐个列举出该类型变量所有可能的取值。根据构造的枚举类型再定义具体的枚举变量。

- ▶ 比如，

```
enum Color {RED, YELLOW, BLUE};
```

```
Color c1, c2, c3;
```

- ▶ Color是构造的枚举类型名，花括号里列出了Color类型变量可以取的值，它们又叫枚举符或枚举常量（标识符的一种，习惯用大写字母的英文单词表示）
- ▶ c1、c2和c3是三个类型为Color的枚举变量，这三个变量的取值都只能是RED、YELLOW或BLUE。



# Notes

---

- ▶ 枚举变量所占空间大小与int型变量的相等
- ▶ 在计算机中实际存放的是枚举符对应的整数，默认情况下，花括号里第一个枚举符对应0，后面依次加1
- ▶ 可以指定（不是赋值，因为构造类型时不在内存开辟空间）所对应的整数
  - ▶ 比如，`enum Color {RED=1,YELLOW,BLUE};`  
则YELLOW对应2，BLUE对应3
- ▶ 若人为指定不当，可能会带来程序运行的错误
  - ▶ 比如，`enum Color {RED=2,YELLOW=1,BLUE};`  
则BLUE对应2，这样，RED和BLUE对应相同的整数，会给后面的程序带来意想不到的错误

# 常见枚举类型

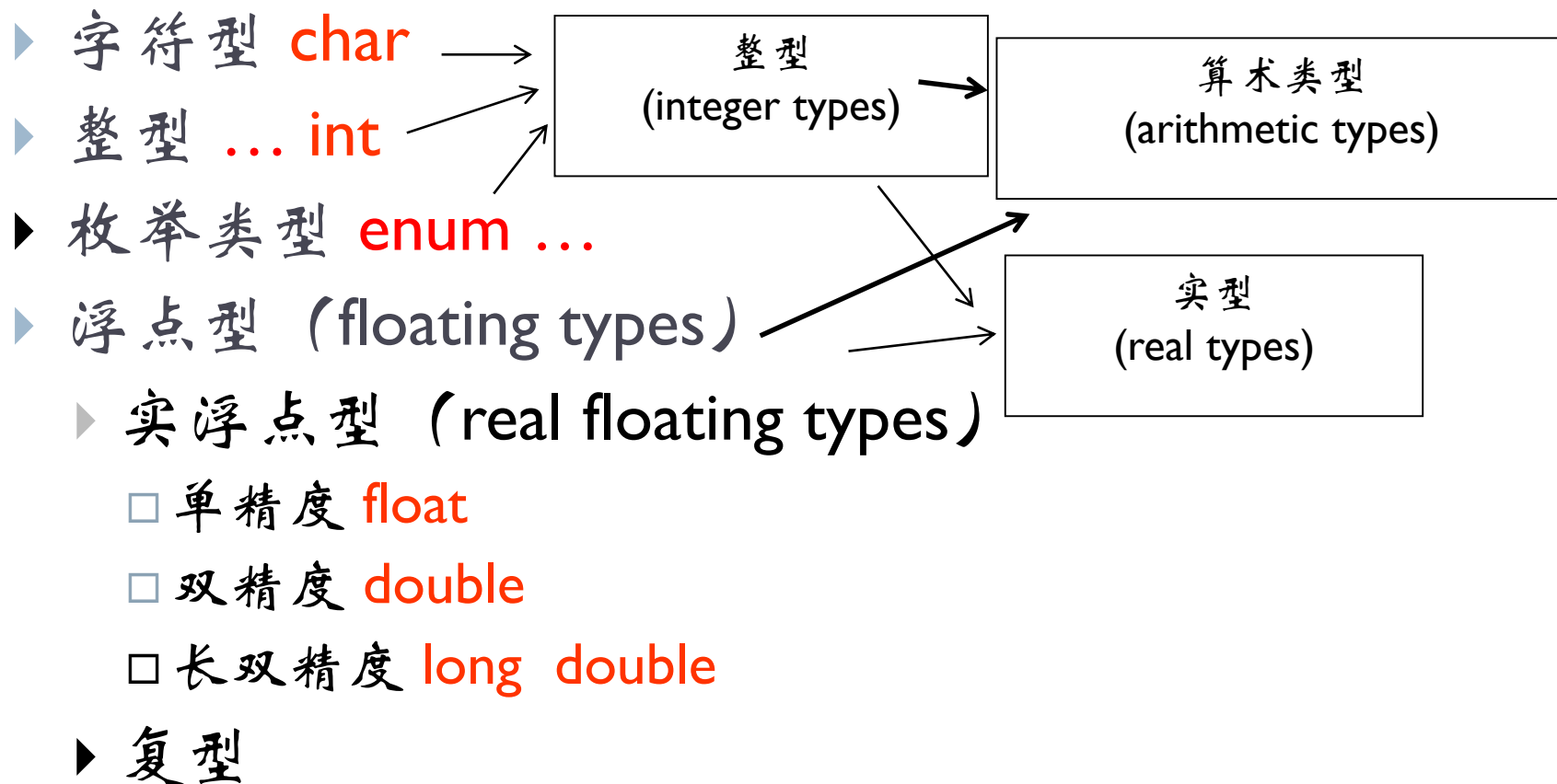
---

## ▶ 常见的枚举类型还有：

- ▶ `enum Weekday {SUN, MON, TUE, WED, THU, FRI, SAT};`
- ▶ `enum Month {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC};`
- ▶ `// 还可以表示Color ...`



# 数据类型总结



# sizeof：返回变量所占字节数

---

- ▶ 可以通过“sizeof(类型名)”或“sizeof(变量名)”来得到各种数据类型的数据所占的内存空间大小（字节数）
- ▶ 标准库的头文件climits（或limits.h）定义了所有整型的取值范围
- ▶ 标准库的头文件cfloat（或float.h）定义了所有实数类型的取值范围



## 例：写程序求部分基本数据类型占的位数

---

...

```
unsigned char uchar8 = 0;      signed char char8 = 0;
unsigned short uint16 = 0;     signed short int16 = 0;
unsigned int uint32 = 0;       signed int int32 = 0;
unsigned long ulong = 0;       float fp32 = 0;
double fp64 = 0;
```

```
printf("unsigned char is %d bit\n\r", sizeof(uchar8)*8);
printf("signed char is %d bit\n\r",  sizeof(char8)*8);
printf("unsigned short is %d bit\n\r", sizeof(uint16)*8);
printf("signed short is %d bit\n\r",  sizeof(int16)*8);
printf("unsigned int is %d bit\n\r",   sizeof(uint32)*8);
printf("signed int is %d bit\n\r",     sizeof(int32)*8);
printf("unsigned long is %d bit\n\r",  sizeof(ulong)*8);
printf("float fp32 is %d bit\n\r",     sizeof(fp32)*8);
printf("double fp64 is %d bit\n\r",   sizeof(fp64)*8);
```

...

---



## 例：写程序求部分基本数据类型占的位数

---

运行结果：

- ▶ unsigned char is 8 bit
- ▶ signed char is 8 bit
- ▶ unsigned short is 16 bit
- ▶ signed short is 16 bit
- ▶ unsigned int is 32 bit
- ▶ signed int is 32 bit
- ▶ unsigned long is 32 bit
- ▶ float fp32 is 32 bit
- ▶ double fp64 is 64 bit



# typedef: 类型名重定义

---

- ▶ C++允许在程序中给已有数据类型取一些别名，格式为：

**typedef <已有类型> <别名>;**

- ▶ 例如：

```
typedef unsigned int Uint;
```

则：

Uint x; 等价于：

unsigned int x;

- ▶ typedef并没有定义新类型。其作用是便于程序的阅读和编写，并使程序简明、清晰和易于维护。
-

# 主要内容

---

- ▶ 数据类型的概念
- ▶ 常量与变量
- ▶ 基本数据类型
- ▶ 数据类型转换





# 基本类型的转换

- ▶ 程序执行过程中，要求参加双目操作的两个操作数类型相同；当类型不同时，会进行类型转换，即一种操作数的类型会转换成另一种数据类型
- ▶ 常指基本类型，不是基本类型的两个不同类型操作数往往不能转换
- ▶ 类型转换方式有两种：
  - ▶ 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，进行强制转换
- ▶ 不管哪一种方式，类型转换都是“临时”的，即在类型转换过程中，操作数本身的类型并没有被转换，只是被临时“看作”另一种类型的数值而已。

## 例：基本类型的转换示例

---

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

```
int main( )
```

```
{
```

```
    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 \n", c, s, v);
```

```
    return 0;
```

```
}
```

# 注意数据类型转换对表达式值的影响

- ▶ 对于含有多个操作符的表达式，其类型转换过程是**逐步进行**的，而不是一次性将所有操作数转换成同种类型的数据再分别参加操作：

- ▶ 比如，“**double v = 4/3\*3.14\*r\*r\*r;**”，转换步骤为：

$1*3.14*r*r*r \rightarrow 1.0*3.14*r*r*r \rightarrow 3.14*r*r*r \rightarrow \dots$

- ▶ 思考：如果写成“**double v = 3.14\*r\*r\*r\*4/3;**”呢？

# 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；
- ▶ 对于逻辑操作与条件操作中第一个表达式的操作数，不是bool型的数据，非0转换为true（非0暨真），0转换为false
- ▶ 比如，在a为0时，!(a)为true

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

- ▶ 对于其他双目操作，按整型提升转换规则和算术类型转换规则进行转换（一般是低精度类型转换为高精度类型）

# 整型提升转换规则

---

- 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  
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值的类型：int、unsigned int、long int、unsigned long int

# 考虑以下表达式的值

---

▶ !3 的结果为：

A) true      B) false

▶ -3 的结果为：

A) true      B) false

▶ true < false 的结果为：

A) true      B) false

▶ 30 > 20 > 10 的结果为：

A) true      B) false

# 类型优先级表

---

高	long double	
	double	
	float	
	unsigned long	
	long	
	unsigned	
低	int	← short, char, _Bool

# 显式（强制）类型转换

---

- ▶ 隐式类型转换有时不能满足要求，于是C语言提供了显式类型转换机制，由程序员用类型关键词明确地指出要转换的类型，**强制**系统进行类型转换：

`i + (int) j ;`



# 建议进行强制（显式）类型转换！

---

- ▶ 避免理解的“模糊”（提升程序可读性）和执行的“问题”
- ▶ 此外，对于一些对操作数类型有约束的操作，可以用显式类型转换保证操作的正确性。
- ▶ 比如，C语言中的求余数运算要求操作数必须是整型数据，  
“`int x = 10%3.4;`”应改为  
“`int x = 10%(int)3.4;`”，否则编译会出错。



# 类型转换后的数据精度问题

- ▶ 操作数类型转换后，有的精度不受损失，有的则会损失精度。损失精度的隐式类型转换会得到**编译器的警告(warning!)**。
- ▶ 隐式类型转换中，对于赋值运算，右操作数的类型转换为左边变量定义的类型，有可能会损失精度；对于其他运算，按“整型提升转换规则”和“算术类型转换规则”进行转换，一般精度不受损失。

```
int x = 4.3;
```

```
int a = 10;
```

```
float b = a + 3.4;
```

## 例：类型转换后的数据精度问题

```
//...  
int main()  
{  
    double a=3.3, b=1.1;  
    int i = a/b;  
    printf("%d \n", i);  
    return 0;  
}
```

2

```
//...  
int main()  
{  
    double a=3.3, b=1.1;  
    printf("%.0f \n", a/b);  
    return 0;  
}
```

3

思考：左边的例子如何得到3？

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

```
2.99999999999999960
```

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

```
3.000000
```

要点：设计程序时，防止因为浮点数的不精确带来的问题

默认6位小数，且四舍五入



## 例：实浮点型数据的精度问题

---

```
#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! The value of z is %.10f", z);
    return 0;
} //输出 “They are not equal! The value of z is 0.3000000119”
```

# C++标准库函数

- 为了方便程序设计，C++语言提供了标准库，其中定义了一些语言本身没有提供的功能：
  - ▶ 常用的数学函数
  - ▶ 字符串处理函数
  - ▶ 输入/输出
  - ▶ ...
- 在标准库中，根据功能对定义的程序实体进行了分类，把每一类程序实体的声明分别放在一个头文件中
- 在C++中，把从C语言保留下来的库函数重新定义在名空间std中；对相应的头文件进了重新命名：\*.h -> c\*

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

# 一些标准数学函数 (cmath或math.h)

- ▶ `int abs( int  $n$  );` //int型的绝对值
- ▶ `long labs( long  $n$  );` //long int型的绝对值
- ▶ `double fabs( double  $x$  );` //double型的绝对值
- ▶ `double sin( double  $x$  );` //正弦函数
- ▶ `double cos( double  $x$  );` //余弦函数
- ▶ `double tan( double  $x$  );` //正切函数
- ▶ `double asin( double  $x$  );` //反正弦函数
- ▶ `double acos( double  $x$  );` //反余弦函数
- ▶ `double atan( double  $x$  );` //反正切函数
- ▶ `double ceil( double  $x$  );` //不小于 $x$ 的最小整数 (返回值为以  
// double表示的整型数)
- ▶ `double floor( double  $x$  );` //不大于 $x$ 的最大整数 (返回值为以  
// double表示的整型数)
- ▶ `double log( double  $x$  );` //自然对数
- ▶ `double log10( double  $x$  );` //以10为底的对数
- ▶ `double sqrt( double  $x$  );` //平方根
- ▶ `double pow( double  $x$ , double  $y$  );` // $x$ 的 $y$ 次幂
- ▶ ... ..

## cmath和math.h的用法区别

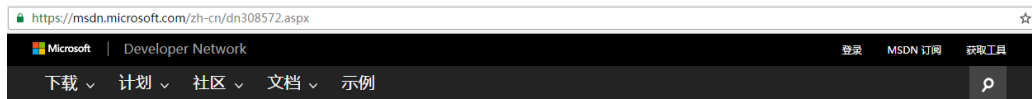
---

- ▶ C++把从C语言保留下来的库函数，重新定义在名空间std中；对相应的头文件进了重新命名：\*.h -> c\*
- ▶ cmath: 标准c++库文件  
#include <cmath>  
using namespace std;
- ▶ math.h: c语言头文件，兼容c风格的库文件  
#include "math.h"





# 如何查阅函数的使用

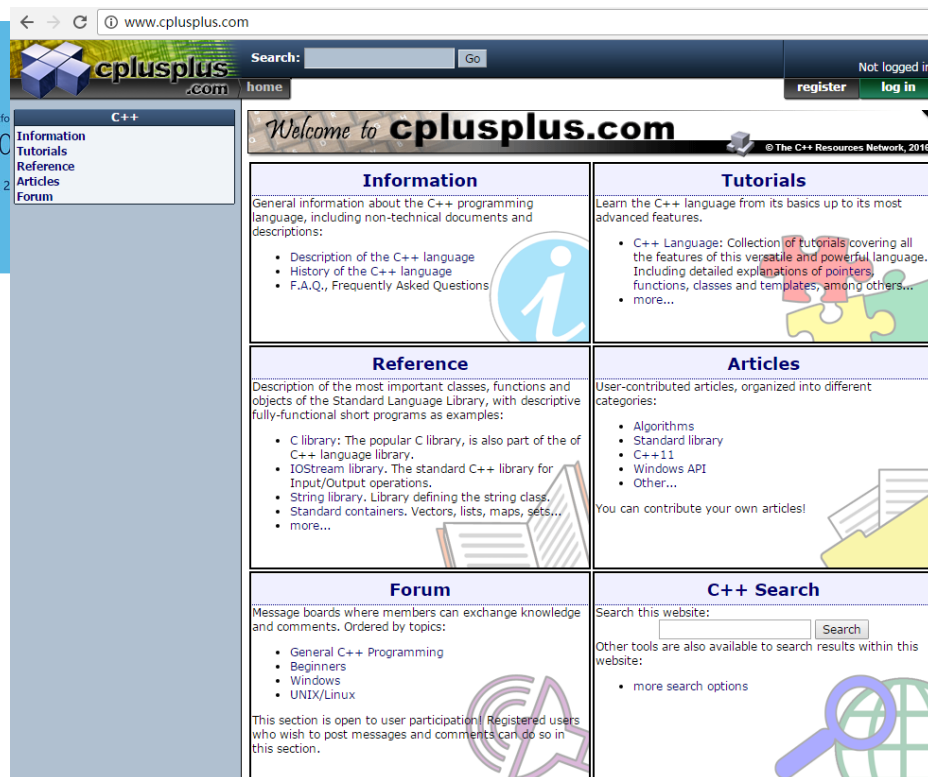


## DevDays Asia 2016

微软 DevDays Asia 2016 北京技术峰会 & Office 365 应用开发马拉松

MSDN

<https://msdn.microsoft.com/zh-cn/default.aspx>



<http://www.cplusplus.com/>

pow, powf, powl

printf, \_printf\_l,  
wprintf, \_wprintf\_l

\_printf\_p, \_printf\_p\_l,  
\_wprintf\_p,  
\_wprintf\_p\_l

printf\_s, \_printf\_s\_l,  
wprintf\_s, \_wprintf\_s\_l

\_purecall

putc, putwc

\_putc\_nolock,  
\_putwc\_nolock

putch

\_putch, \_putwch

\_putch\_nolock,  
\_putwch\_nolock

putchar, putwchar

\_putchar\_nolock,  
\_putwchar\_nolock

putenv

\_putenv, \_wputenv

\_putenv\_s,  
\_wputenv\_s

puts, \_putws

putw

\_putw

\_query\_new\_handler

\_query\_new\_mode

quick\_exit

qsort

# printf, \_printf\_l, wprintf, \_wprintf\_l

Visual Studio 2015 其他版本 ▾

将格式化输出打印至标准输出流。有关这些函数的更多安全版本，请参见 [printf\\_s](#)、[\\_printf\\_s\\_l](#)、[wprintf\\_s](#)、[\\_wprintf\\_s\\_l](#)。

## 语法

```
int printf(  
    const char *format [,  
    argument]...  
);  
int _printf_l(  
    const char *format,  
    locale_t locale [,  
    argument]...  
);  
int wprintf(  
    const wchar_t *format [,  
    argument]...  
);  
int _wprintf_l(  
    const wchar_t *format,  
    locale_t locale [,  
    argument]...  
);
```

## 参数

*format*  
格式控件。

*argument*  
可选参数。

*locale*  
要使用的区域设置。

## 返回值

返回打印的字符数，或在发生错误时返回负值。如果 *format* 是 **NULL**，则会调用无效参数处理程序，如 [参数验证](#) 中所述。如果允许执行继续，则该函数返回 -1 并将 **errno** 设置为 **EINVAL**。如果在 *argument* 中遇到 **EOF** (0xFFFF)，则函数返回 -1。

<queue>  
<set>  
<stack>  
<unordered\_map>  
<unordered\_set>  
-----<vector>  
**Input/Output:**  
<fstream>  
<iomanip>  
<ios>  
<iosfwd>  
<iostream>  
<istream>  
<ostream>  
<sstream>  
<streambuf>  
**Multi-threading:**  
<atomic>  
<condition\_variable>  
<future>  
<mutex>  
<thread>  
**Other:**  
<algorithm>  
<bitset>  
<chrono>  
<codecvt>  
<complex>  
<exception>  
<functional>  
<initializer\_list>  
<iterator>  
<limits>  
<locale>  
<memory>  
<new>  
<numeric>  
<random>  
<ratio>  
<regex>  
<stdexcept>  
<string>  
<system\_error>  
<tuple>  
-----<typeindex>  
<typeinfo>  
<type\_traits>  
<utility>  
-----<variant>

If the *base* is finite negative and the *exponent* is finite but not an integer value, it causes a *domain error*.  
If both *base* and *exponent* are zero, it may also cause a *domain error* on certain implementations.  
If *base* is zero and *exponent* is negative, it may cause a *domain error* or a *pole error* (or none, depending on the library implementation).  
The function may also cause a *range error* if the result is too great or too small to be represented by a value of the return type.

If a *domain error* occurs, the global variable `errno` is set to `EDOM`.

If a *pole* or *range error* occurs, the global variable `errno` is set `ERANGE`.

If a *domain error* occurs:

- And `math_errhandling` has `MATH_ERRNO` set: the global variable `errno` is set to `EDOM`.

- And `math_errhandling` has `MATH_ERREXCEPT` set: `FE_INVALID` is raised.

If a *pole error* occurs:

- And `math_errhandling` has `MATH_ERRNO` set: the global variable `errno` is set to `ERANGE`.

- And `math_errhandling` has `MATH_ERREXCEPT` set: `FE_DIVBYZERO` is raised.

If a *range error* occurs:

- And `math_errhandling` has `MATH_ERRNO` set: the global variable `errno` is set to `ERANGE`.

- And `math_errhandling` has `MATH_ERREXCEPT` set: either `FE_OVERFLOW` or `FE_UNDERFLOW` is raised.

### 💡 Example

```
1 /* pow example */
2 #include <stdio.h>      /* printf */
3 #include <math.h>       /* pow */
4
5 int main ()
6 {
7     printf ("7 ^ 3 = %f\n", pow (7.0, 3.0) );
8     printf ("4.73 ^ 12 = %f\n", pow (4.73, 12.0) );
9     printf ("32.01 ^ 1.54 = %f\n", pow (32.01, 1.54) );
10    return 0;
11 }
```

Output:

```
7 ^ 3 = 343.000000
4.73 ^ 12 = 125410439.217423
32.01 ^ 1.54 = 208.036691
```

### 🔧 See also

<b>log</b>	Compute natural logarithm (function )
<b>exp</b>	Compute exponential function (function )
<b>sqrt</b>	Compute square root (function )

# 网站推荐

---

- ▶ MS Visual Studio 联机帮助
- ▶ <https://msdn.microsoft.com>
- ▶ <http://www.cplusplus.com>
- ▶ <http://www.72up.com/c/function.htm>
- ▶ CSDN 网址: <http://www.csdn.net/> 简介: 于1999年3月成立, 是中国最大的软件开发人员网站, 社区热心高手众多, 很多开源代码



# Q & A

---



## 2、8、10、16进制

---

- ▶ 2进制

- ▶ 0 1

- ▶ 8进制

- ▶ 0 1 2 3 4 5 6 7

- ▶ 10进制

- ▶ 0 1 2 3 4 5 6 7 8 9

- ▶ 16进制

- ▶ 0 1 2 3 4 5 6 7 8 9 A(10) B(11) C(12) D(13) E(14) F(15)



# 整型的值域

## (以32位机为例, int型数据的取值范围)

- ▶ 用二进制表示:

正数: 00000000000000000000000000000000 ~  
01111111111111111111111111111111、零:  
10000000000000000000000000000000、  
负数: 10000000000000000000000000000001 ~  
11111111111111111111111111111111

- ▶ 对应的十六进制数为

00000000~7FFFFFFF、  
80000000、  
80000001~FFFFFFFF

- ▶ 对应的十进制数为

0~2147483647、	0~2147483647
-0、	-2147483648
-1~-2147483647、	-2147483647~-1

## 例：求级数 $1 + x + x^2/2! + x^3/3! + \dots + x^n/n!$ 和

分析：

0、初始化：

定义一个变量**sum**用于存储和，初始值为**1**；

用户输入两个值，分别放在变量**x**和**n**中；

1、累加(外层大结构：循环！)：

依次将每一项**item**的值加到**sum**中去；

其中的“依次...”隐含着循环（循环**n**次）。

2、构数(内层)：

计算某一项  $x^i/i!$  时隐含着循环（循环**i**次）；

将某一项的值保存在变量**item**中；

循环变量为**i**

**i**从1到**n**

**b/a**

循环变量为**j**

**j**从1到**i**



# 例：求级数 $1 + x + x^2/2! + x^3/3! + \dots + x^n/n!$ 和

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

算法1



利用 $x^i = x * x^{i-1}$ 和  $i! = i * (i-1)!$ 减少重复计算

```
int main()
```

```
{ double x, sum, item, b;
```

```
int n, a, i, j;
```

```
scanf("%lf%d", &x, &n);
```

```
sum = 1, a = 1, b = 1;
```

```
for (i=1; i <= n; i++)
```

```
{ a = 1, b = 1;
```

```
for (j=1; j <= i; j++)
```

```
{ b *= x; // 计算 $x^j$ 
```

```
a *= j; // 计算 $j!$ 
```

```
}
```

```
item = b/a; // 计算 $x^i/i!$ 
```

```
sum += item; //  $x^i/i!$ 加到sum中
```

```
}
```

```
printf("sum = %d \n", sum);
```

b \*= x; // 计算 $x^j$   
a \*= j; // 计算 $j!$

例：求级数 $1 + x + x^2/2! + x^3/3! + \dots + x^n/n!$ 和

例：求级数  $1 + x + x^2/2! + x^3/3! + \dots + x^n/n!$  和

```
int main()
{ double x, sum, item, b;
  int n, a, i, j;
  scanf("%lf%d", &x, &n);
  sum = 1, a = 1, b = 1;
  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 = %d \n", sum);
  ...
```

算法2

利用  $\text{item}_i = \text{item}_{i-1} * x/i$  进一步减少计算量

```
int main()
{ double x, sum, item, b;
  int n, a, i, j;
  scanf("%lf%d", &x, &n);
  sum = 1, a = 1, b = 1;
  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 = %d \n", sum);
  ...
```

$\text{item} = 1$

$\text{item} *= x/i;$

例：求级数  $1 + x + x^2/2! + x^3/3! + \dots + x^n/n!$  和

例：求级数  $1 + x + x^2/2! + x^3/3! + \dots + x^n/n!$  和

---

```
int main()
{ double x, sum, item;
  int n, i;
  scanf("%lf%d", &x, &n);
  sum = 1, item = 1;
  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 = %d \n", sum);
  ...
```

算法3



- 
- ▶ 算法3除较高效外,可靠性更好: 当 $x^i/i!$ 不太大, 而 $x^i$ 或 $i!$ 很大以至于超出计算机所能表示的数值范围时, 算法1和2就不能得出正确的结果; 算法3直接计算 $x^i/i!$ , 不存在超出表示范围的问题
  - ▶ 算法3会带来精度损失:  $x^i/i!$ 是基于 $x^{i-1}/(i-1)!$ 的计算结果的, 而 $x^{i-1}/(i-1)!$ 的计算结果有精度损失, 因此精度损失会叠加
-

# 八或十六进制表示字符

---

- ▶ 八进制字符和十六进制字符表示的是字符的ASCII码对应的数值
- ▶ 八进制字符的一般形式是'\ddd', d是0-9的数字。
  - ▶ 字符'3': 用'\063'表示, '3'的ASCII码对应63 (八进制) (33 (十六进制), 51 (十进制))
- ▶ 十六进制字符的一般形式是'\xhh', h是0-9或A-F内的一个。
  - ▶ 字符'A': 用'\x41'表示, 因为'A'的ASCII码是41 (十六进制) (65 (十进制), 101 (八进制))



# 转义字符（可用八或十六进制表示）

---

- ▶ 八进制转义字符和十六进制转义字符(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八进制ASCII码或十六进制ASCII码)





## 例：打印字符'a'构成的“倒三角”形状

```
char ch = 'a';
for(int i = 0; i < 5; i++)
{
    for(int j = 0; j < i; j++)
        printf(" ");
    for(int j = 0; j < 10-2*i-1; j++)
        printf("%c", ch);
    printf("\n");
}
```



```
aaaaaaaaa
aaaaaaa
aaaaa
aaa
a
```

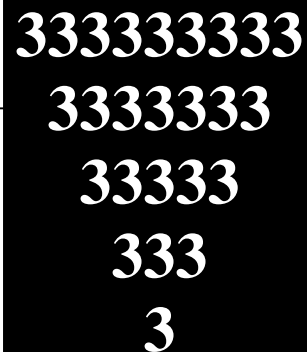
## 例：打印字符'a'构成的“倒三角”形状

```
for(int i = 0; i < 5; i++)  
{  
    for(int j = 0; j < i; j++)  
        printf(" ");  
    for(int j = 0; j < 10-2*i-1; j++)  
        printf("%c", 97);  
    printf("\n");  
}
```

```
aaaaaaaaa  
aaaaaaa  
aaaaa  
aaa  
a
```

## 例：打印字符'3'构成的“倒三角”形状

```
char ch = '3';  
for(int i = 0; i < 5; i++)  
{  
    for(int j = 0; j < i; j++)  
        printf(" ");  
    for(int j = 0; j < 10-2*i-1; j++)  
        printf("%c", ch);  
    printf("\n");  
}
```



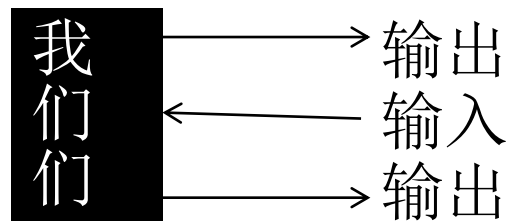
```
333333333  
3333333  
33333  
333  
3
```

- ▶  $3 > 4$  的结果为false
- ▶  $'a' < 'b'$  的结果为true
- ▶  $!(20 > 10)$  的结果为false
- ▶ 当m为11时,  $!(m < 10)$  的结果为true
  
- ▶ 当m为3, n为4时,  $(m > 1) \&\& (n < 20)$  的结果为true
- ▶ 在m为0, n为4, 或m为3, n为21, 以及m为0, n为21时,  $(m > 1) \&\& (n < 20)$  的结果均为false
  
- ▶ 当m为0, n为21时,  $(m > 1) \|\ (n < 20)$  的结果为false
- ▶ 在m为0, n为4, 或m为3, n为21, 或m为3, n为4时,  $(m > 1) \|\ (n < 20)$  的结果均为true

## 例：宽字符操作简单示例（了解）

---

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



## 例 枚举类型数据不可以直接输入输出

---

```
#include<stdio.h>
int main( )
{
    enum Weekday {SUN, MON, TUE, WED, THU, FRI, SAT};
    Weekday d1 = SUN, d2 = SAT;
    if(d1 < d2)
        printf("Sunday is the first day of a week. \n");
    else
        printf("\Which day is the first day of a week? \n");
    return 0;
}
```

# 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，比如掷骰子的结果、噪声的强度、福利彩票抽奖等，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

## 例：生成随机数（范围无限制）

---

- ▶ 利用rand()函数，rand()会返回一随机数值，范围在0至RAND\_MAX间。RAND\_MAX定义在stdlib.h, 其值为2147483647

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    for(int i=0;i<10;i+)
        printf("%d/n", rand());
}
```





## 例：生成随机数（在一定范围内）

---

例如：随机生成10个0~100的数：

```
#include<stdio.h>
#include<stdlib.h>
#define random(x) (rand()%x)

void main()
{
    for(int x=0;x<10;x++)
        printf("%d/n",random(100));
}
```



## 例：生成随机数（几次操作得到不一样的随机数）

---

```
▶ #include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define random(x) (rand() % x)

void main()
{
    srand((int)time(0)); // srand(time(NULL));
    for(int x=0;x<10;x++)
        printf("%d/n",random(100));
}
```



# 为什么用srand()函数

---

- srand和rand()配合使用产生伪随机数序列
- rand函数在产生随机数前，需要系统提供的生成伪随机数序列的种子，rand根据这个种子的值产生一系列随机数。如果系统提供的种子没有变化，每次调用rand函数生成的伪随机数序列都是一样
- srand(unsigned seed)通过参数seed改变系统提供的种子值，从而可以使得每次调用rand函数生成的伪随机数序列不同，从而实现真正意义上的“随机”
- 通常可以利用系统时间来改变系统的种子值，即srand(time(NULL))，可以为rand函数提供不同的种子值，进而产生不同的随机数序列

# 用rand()和srand()产生伪随机数的方法总结

---

课外阅读：

<http://blog.chinaunix.net/uid-26722078-id-3754502.html>



# 例：隐式类型转换存在的问题示例

```
#include<stdio.h>
int main()
{
    int i = -10;
    unsigned int j = 3;
    .....
    if(i + j < 0)
        printf("-7\n");
    else
        printf("error.\n");           //结果显示error

    if(i < j)
        printf("i<j\n");
    else
        printf("i>j\n");             //结果显示i>j
    return 0;
}
```

不同类型的数据（i和j）在一起操作（算术、比较操作），隐式类型转换的结果违背了常识。

# 例：隐式类型转换存在的问题示例

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

```
int main( )
```

```
{
```

```
    int i = -10;
```

```
    unsigned int j = 3;
```

```
    if(i + j < 0)
```

```
        printf("-7\n");
```

//应改成if(i+(int)j<0)     printf("-7\n");

```
    else
```

```
        printf("error.\n");
```

```
    if(i < j)
```

```
        printf("i<j\n");
```

//应改成if(i<(int)j)

printf("i<j\n");

```
    else
```

```
        printf("i>j\n");
```

```
    return 0;
```

```
}
```

利用显式类型转换，则结果可显示-7和i<j。

# 关于字符, 字符串的操作

字符串: 一个特殊的一维字符数组 '\0'

- ▶ 把字符串放入一维字符数组 (存储)

数组长度足够

- ▶ 字符数组初始化: `char s[20] = "Happy";`
- ▶ 赋值: `s[0] = 'a'; s[1] = '\0';`

`char str[10] = {'H', 'e', 'l', 'l', 'o', ' ', 'N', 'J', 'U', '\0'};`

`char str[10] = "Hello NJU";`

H	e	l	l	o		N	J	U	\0		
---	---	---	---	---	--	---	---	---	----	--	--

- ▶ 输入: 输入结束符 ==> 字符串结束符 '\0'

`i = 0;`

`while((s[i]=getchar()) != '\n')`

`i++;`

`s[i] = '\0';`

输出: `for(i = 0; s[i] != '\0'; i++)`

`putchar(s[i]);`

# 小 结

- ▶ 数据为什么要分成不同的类型
  - ▶ 表示具有不同取值“属性”的数据
  - ▶ 便于合理分配内存，产生高效代码
  - ▶ 便于运算，便于数据的处理
  - ▶ 便于自动进行类型一致性检查，保护数据，提高程序的可靠性

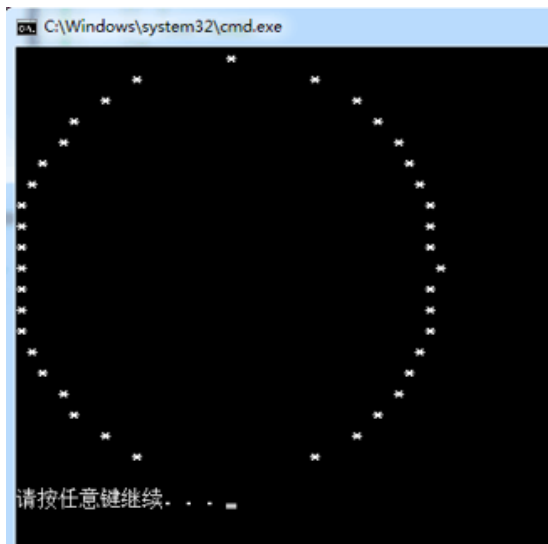


# 测试

# : 打印圆形

```
int main()
{
    cout<<"请输入半径r(r>0)"<<endl;
    int r;
    do
    {
        cin>>r;
        if(r<=0)
            cout<<"半径r输入错误请重新输入"<<endl;
        else
        {
            cout<<"半径输入正确开始画圆"<<endl;
            break;
        }
    }while(r<=0);

    double i=0, j=0;
    for(i=0; i<=2*r; i=i+0.2)
    {
        for(j=0; j<=r+0.1; j=j+0.2)
        {
            if(fabs(j-(r-sqrt(r*r-(r-i)*(r-i))))<=1e-6 || fabs(j-(r+sqrt(r*r-(r-i)*(r-i))))<=1e-6)
                cout<<"*";
            else
                cout<<" ";
        }
        for(j=r; j<=2*r+0.1; j=j+0.2)
        {
            if(fabs(j-(r-sqrt(r*r-(i-r)*(i-r))))<=1e-6 || fabs(j-(r+sqrt(r*r-(i-r)*(i-r))))<=1e-6)
                cout<<"*";
            else
                cout<<" ";
        }
        cout<<endl;
    }
    cout<<endl;
    return 0;
}
```



当r=2,3,4圆不封闭;

当r=5...正常