

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>
```

编译预处理命令

- C++程序中可以写一些供编译程序使用的命令：**编译预处理命令**。
 - 编译预处理命令不是C++程序所要完成的功能，而是用于对编译过程给出指导，其功能由编译预处理系统来完成。
 - 编译预处理命令主要有：
 - ▶ 文件包含命令（**#include**）
 - ▶ 宏定义（**#define**）命令
 - ▶ **条件编译命令？**
-



条件编译

- 编译程序根据不同的情况来选择需编译的程序代码。例如，编译程序将根据宏名ABC是否被定义，来选择需要编译的代码：

<代码1> //必须编译的代码

`#ifdef ABC` // `#ifndef` <宏名>

 <代码2> //如果宏名ABC有定义，编译之

`#else`

 <代码3> //如果宏名ABC无定义，编译之

`#endif`

<代码4> //必须编译的代码

- <宏名>可以在程序中用`#define`定义，也可以在编译器的选项中给出

■ 利用标准库中定义的宏：**assert**

```
#include <cassert> //或<assert.h>
```

```
.....
```

```
assert(x == 1); //断言
```

■ assert的定义大致如下：

```
.....
```

```
#ifdef NDEBUG
```

```
#define assert(exp) ((void)0)
```

```
#else
```

```
#define assert(exp) ((exp)?(void)0:<输出诊断信息并调用库函数abort>)
```

```
#endif
```

```
.....
```




回顾：一个完整的例子（C++）

- ▶ 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;
```

为什么

- ▶ $10/3$ 值为 3?
- ▶ $/$ 表示整数取倍数, $\%$ 表示取余数
- ▶ 判断浮点类型变量相等为何需要用
`if (fabs(a-b) < 1e-6) ?`
- ▶ 字符和字符串如何定义?



5 程序数据描述（I） — 基本数据类型

郭延文

2019级计算机科学与技术系

主要内容

- ▶ 数据类型的概念
- ▶ 常量与变量
- ▶ C++基本数据类型



写“简单”程序的基本流程

- ▶ 理解问题（需求分析）
- ▶ 算法（流程）设计与问题分解
- ▶ Top-down实现（功能分解）
 - ▶ 处理什么数据，数据的属性是什么？
 - ▶ 函数设计、函数之间的通讯（接口）
 - ▶ 每个函数的实现
 - ▶ 处理什么数据，数据的属性是什么？
 - ▶ 流程设计
- ▶ 综合（功能复合）…归口到main函数

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（运算）集**：规定了对值集中的值能实施哪些运算。
- ▶ 例如：整数类型就是一种数据类型，它的值集就是由整数所构成的集合，它的操作集包括：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；
- ▶ 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，它们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）。它们一般为复合数据类型。
-



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

数据在C++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。

数据在C++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
-

字面常量（直接量）

▶ C++的字面常量有：

- ▶ 整数类型常量
- ▶ 实数类型常量
- ▶ 字符类型常量
- ▶ 逻辑类型常量
- ▶ 字符串常量



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>`

或

`const <类型名> <常量名>=<值>;`

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

▶ 常量定义的好处

▶ 保证程序对常量使用的一致性

▶ 圆周率: $PI = 3.141\ 5926\ 5358$

▶ 增加程序的易读性

▶ $PI = 3.141\ 5926$; // 简明扼要

▶ `const int PASS_SCORE=60;`

▶ `const int MINUTES_PER_HOUR=60`

▶ 增强程序的易维护性

▶ $PI = 3.141\ 5926$; // 当需要修改其精度时...

变 量

- ▶ 程序中可变的数据用变量来表示。
- ▶ 例如：在计算圆周长的式子“ $2*PI*r$ ”中，半径 r 就是一个可变的数据，它可能是通过用户输入得到，也可能由程序的其它部分计算得到。

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;
```

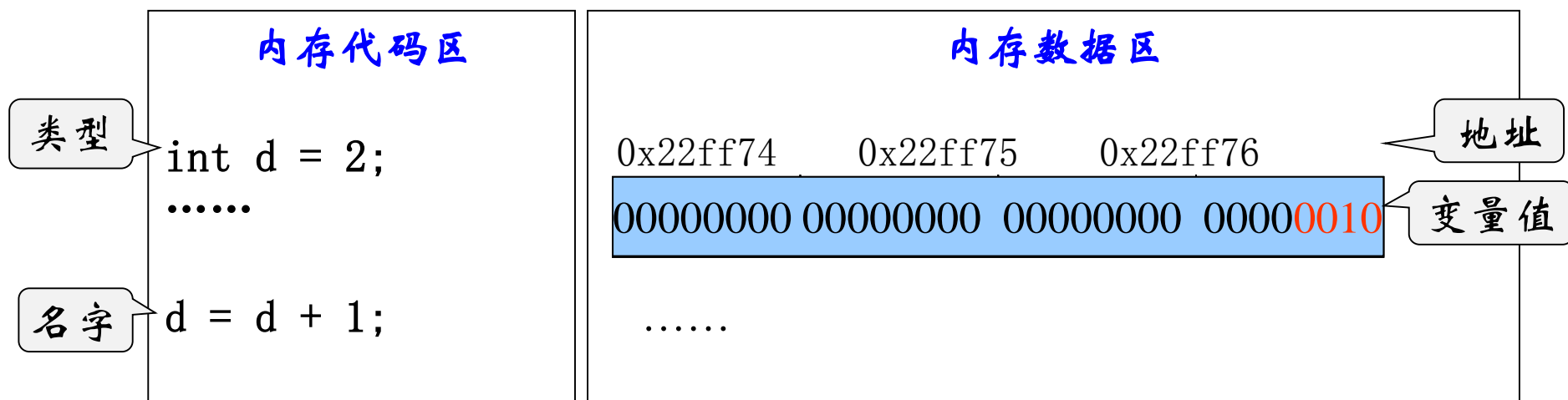
或：

```
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。



用户定义变量类型和名字

系统决定地址，存储二进制值

C/C++基本数据类型

- ▶ 基本数据类型对应着能由计算机直接表示和处理（机器指令能对它们直接进行操作）的数据类型，包括：
 - ▶ 整数类型
 - ▶ 实数类型
 - ▶ 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 \leq “int”的范围 \leq “long int”的范围
 - ▶ 具体大小由实现决定，例如
 - ▶ int 占4个字节，一般由计算机的字长决定
 - ▶ short int 占2个字节 (-32768~32767)
 - ▶ long int 占4个字节 (-2147483648~2147483647)
 - ▶ 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。

无符号整数类型

- ▶ 为了能对非负整数进行单独描述，C++提供了无符号整数类型：
 - ▶ `unsigned int` 或 `unsigned`
 - ▶ `unsigned short int` 或 `unsigned short`
 - ▶ `unsigned long int` 或 `unsigned long`
- ▶ 它们所占的内存大小与相应的有符号整数类型相同，但所表示的最大正整数比相应的有符号整数类型所表示的最大正整数要大（大约一倍）。



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

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

- ▶ 具体的值域可以查看文件limits.h

例：整型数据范围溢出示例

```
#include <stdio.h>
int main()
{
    int a = 2147483647, b = 1;
    printf("%d \n", a + b);
    return 0;
}
```

-2147483648

- ▶ 如果将输出格式符%d改成%u（即按unsigned int型数据输出），则结果为2147483648。

2147483647的补码

01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11

+1

10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

- 2147483648的补码

整数类型常量

- ▶ 在C/C++程序中，整数类型常量可以用下面形式来书写：
 - ▶ 十进制：由0~9数字组成，第一个数字不能是0（整数0除外），如：59，128，-72
 - ▶ 八进制：由数字0打头，0~7数字组成，如：073，0200，-0110
 - ▶ 十六进制：由0x或0X打头，0~9数字和A~F（或a~f）字母组成，如：0x3B，0x80，-0x48
- ▶ 整数类型字面常量的默认类型为int，可在整型常量的后面：
 - ▶ 加上l或L，表示long int类型的常量，如：32765L
 - ▶ 加上u或U，表示unsigned int类型的常量，如：4352U
 - ▶ 同时加上u（U）和l（L）表示unsigned long类型的常量，如：41152UL，或，41152LU

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）

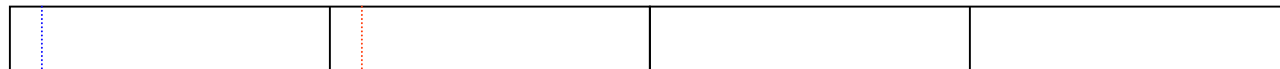


实数类型表示：IEEE754标准（了解即可）

▶ 以32位机为例，一般地，

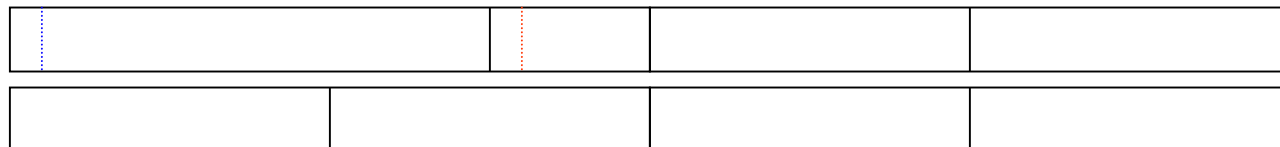
- ▶ 单精度数据占32位空间，其中尾数部分占24位，含1位符号位，指数部分占8位，含1位指数的符号位

float



- ▶ 双精度数据占64位空间，其中尾数部分占53位，含1位符号位，指数部分占11位，含1位指数的符号位

double



- ▶ 长双精度数据占96位空间（VS 2013, gcc）

值域与精度（以32位机为例）

- ▶ 单精度 (float)

- ▶ 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 1.175×10^{-38}
- ▶ 分辨率大约是 1.192×10^{-7} ，即有6位数字有效

- ▶ 双精度 (double)

- ▶ 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 2.225×10^{-308}
- ▶ 分辨率大约是 2.22×10^{-16} ，即有15位数字有效

- ▶ 长双精度

- ▶ 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



实数类型常量

- ▶ 在C++程序中，实数类型常量采用十进制形式书写。

实数类型常量有两种表示法：

- ▶ 1. 小数表示法：由整数部分、小数点“.”和小数部分构成，如：456.78，-0.0057，5.，.5
- ▶ 实数类型字面常量默认为double型，可以在实数类型常量后面：
 - 加上F(f)以表示float型，如：5.6F。
 - 加上L(l)表示long double型，如5.6L。

2. 科学表示法

- ▶ 实数可以用十进制或十六进制形式来书写：
 - ▶ 十进制实数：没有前后缀
 - ▶ 在小数表示法或整数后加上一个指数部分，指数部分由E(或e)和一个整数类型数构成，表示基数为10的指数，如：4.5678E2, -5.7e-3等。
 - ▶ 十六进制实数：以0x或0X为前缀，没有后缀
 - ▶ 由规格化十六进制小数、字母P(或p)、符号和十六进制整数四部分组成，比如0x1.fP-3(即 $0x\ 1.f \times 16^{-3}$)。

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/

%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
%%	"%"



cout输出浮点数的精度

使用`setprecision(n)`可控制输出流显示浮点数的数字个数。C++默认的流输出数值有效位是6。

如果`setprecision(n)`与`setiosflags(ios::fixed)`合用，可以控制小数点右边的数字个数。`setiosflags(ios::fixed)`是用定点方式表示实数。

如果与`setiosnags(ios::scientific)`合用，可以控制指数表示法的小数位。数。`setiosflags(ios::scientific)`是用指数方式表示实数。



举例

```
#include <iostream.h>
#include <iomanip.h> //要用到格式控制符

void main()
{
    double amount = 22.0/7;
    cout << amount << endl;
    cout << setprecision(0) << amount << endl
    << setprecision(1) << amount << endl
    << setprecision(2) << amount << endl
    << setprecision(3) << amount << endl
    << setprecision(4) << amount << endl;

    cout << setiosflags(ios::fixed);
    cout << setprecision(8) << amount << endl;

    cout << setiosflags(ios::scientific)

    << amount << endl;

    cout << setprecision(6); //重新设置成原默认设置
}
```

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”
```

思考

- $(0.1)_{10}$ 转成2进制是多少？用2进制能否精确地表示？

不能。 $0.1_{10} = 0.000110011\cdots_2$ 0.099609375_{10}
有些十进制小数不能精确表示成2进制小数。

$$\begin{aligned}
(0.1)_{10} &= (0.0001100110011\dots)_2 \\
&= (0.0110011001100110011)_2 \times 2^{-2} \\
(0.2)_{10} &= (0.0011001100110\dots)_2 \\
&= (0.1100110011001100110)_2 \times 2^{-2} \\
(0.1+0.2)_{10} &= (1.00110011001100110011001)_2 \times 2^{-2} \\
(0.3)_{10} &= (0.01100110011\dots)_2 \\
&= (1.10011001100110011001101)_2 \times 2^{-2}
\end{aligned}$$

为什么？

1: 二进制

2: 浮点数的存储格式

```
...  
float a, b, c;  
a  =  1.345f;  
b  =  1.123f;  
c  =  a  +  b;  
  
if(c == 2.468)  
    printf("They are equal.\n");  
else  
    printf("They are not equal! The value of c is %.10f, or %f",  
c, c);  
...
```

They are not equal! The value of c is 2.4679999352 or

可以使用常数FLT_EPSILON(单精度浮点型, $1.192092896e-7F$) 或 DBL_EPSILON(双精度浮点型, $2.2204460492503131e-16$), 需要包含这些常数定义的float.h, 这些常数被当成最小的正数。

```
#define EPSILON 0.0000001 //Define your own tolerance
```

```
...
float a, b, c;
a  = 1.345f;
b  = 1.123f;
c  = a + b;

if(((2.468-EPSILON)<c) && (c<(2.468+EPSILON)))
    printf("They are equal.\n");
else
    printf("They are not equal! The value of c is %.10f, or %f",
c, c);
...
```

They are equal.

特别注意！（重点）

- ▶ 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: $\text{fabs}(z-0.3) < 1e-7$

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xi/i!
      sum += item;           // xi/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
-

字符类型

- ▶ 字符类型用于描述文字类型数据中的一个字符。

- ▶ 回顾函数重载：对于下述的重载函数：

```
void print(int);
```

```
void print(double);
```

根据提升匹配，下面的函数调用：

```
print( 'a' ); 绑定到函数：void print(int);
```

```
print(1.0f); 绑定到函数：void print(double);
```

- ▶ 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、
 - ▶ 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		

教材附录A

ASCII 字符集及其编码

A: 0x41 (16进制表示) 65 (十进制)

记住 ‘0’、‘a’ 和 ‘A’ 的ASCII码即可



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））



常用的字符集及其编码（续）

- ▶ Unicode（国际通用字符集）
 - ▶ 2~4个字节
 - ▶ 可用于大部分语言中的字符
 - ▶ C++用wchar_t描述
- ▶ GB2312（简体中文）
 - ▶ 2个字节
 - ▶ C++用2个char描述
- ▶ Big5（繁体中文）
 - ▶ 2个字节
 - ▶ C++用2个char描述
- ▶ Shift-JIS（日文）
 - ▶ 2个字节
 - ▶ C++用2个char描述

了解即可！



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！

转义字符ASCII码值（了解）

转义字符	含义	ASCII码（16/10进制）
\0	空字符 (NULL)	0X00/0
\n	换行符 (LF)	0X0A/10
\r	回车符 (CR)	0X0D/13
\t	水平制表符 (HT)	0X09/9
\v	垂直制表 (VT)	0X0B/11
\a	响铃 (BEL)	0X07/7
\b	退格符 (BS)	0X08/8
\f	换页符 (FF)	0X0C/12
\'	单引号	0X27/39
\"	双引号	0X22/34
\\	反斜杠	0X5C/92
\?	问号字符	0X3F/63
\ddd	任意字符	三位八进制
\xhh	任意字符	二位十六进制

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）

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）。
-

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
```

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
}
```

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

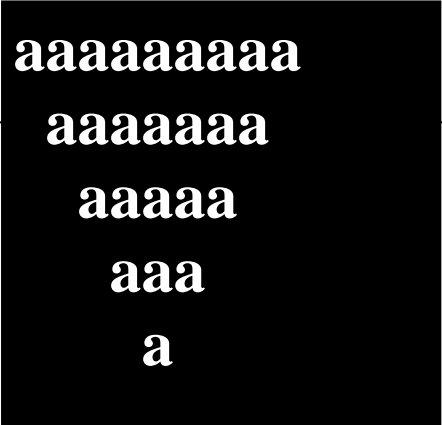
'3'

00000011

00110011

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

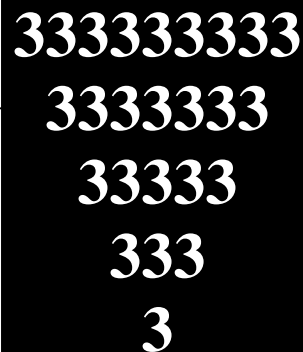
```
for(int i = 0; i < 5; i++)  
{  
    for(int j = 0; j < i; j++)  
        printf(" ");  
    for(int j = 0; j < 10-2*i-1; j++)  
        printf("%c", 97);  
    printf("\n");  
}
```



aaaaaaaaa
aaaaaaa
aaaaa
aaa
a

例：打印字符'3'构成的“倒三角”形状

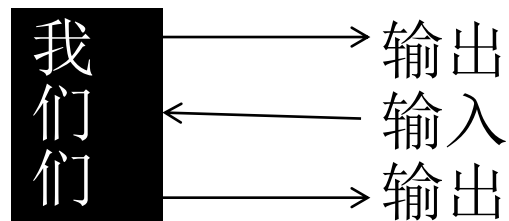
```
char ch = '3';  
for(int i = 0; i < 5; i++)  
{  
    for(int j = 0; j < i; j++)  
        printf(" ");  
    for(int j = 0; j < 10-2*i-1; j++)  
        printf("%c", ch);  
    printf("\n");  
}
```



```
333333333  
 3333333  
   33333  
    333  
     3
```

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;
}
```



回 顾

```
if (...)    // 条件成立时
{
    ...
}
```

```
float a = (b > c)? b : c;
```



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) {…}`



- ▶ $3 > 4$ 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

逻辑型数据不可以直接输入输出

```
#include <iostream>
using namespace std;
Int main( )
{
    _Bool b = true;
    if(b)
        cout<<"true."<<endl;    //或用puts函数
    else
        cout<<"false. "<<endl; //或用puts函数
    return 0;
} //间接输出
```

回顾

```
... ..
```

```
int main()
```

```
{
```

```
    cout<<"Hello
```

```
world! "<<endl;
```

```
    return
```

```
}
```

```
... ..
```

```
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 *)



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

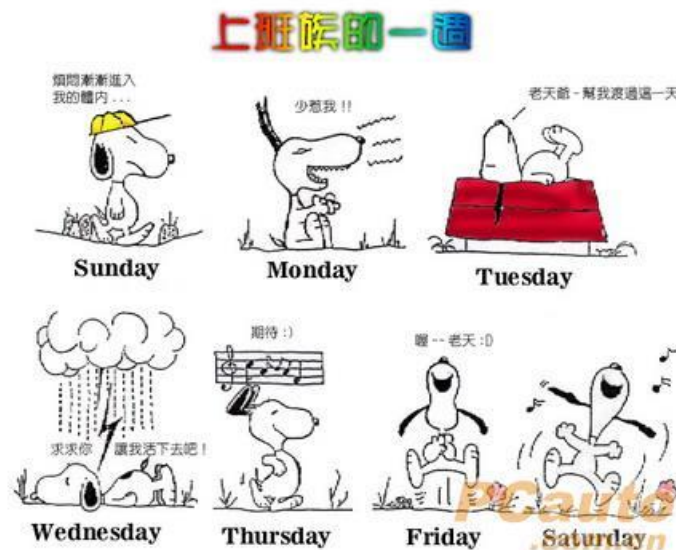
- 枚举类型
- 数组类型
- 结构与联合类型
- 指针类型
- 引用类型

抽象数据类型

- 类
- 派生类

可否有这样的数据类型

- 表示day, 其取值为
一星期的七天



- 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。

-
- ▶ 程序执行到枚举类型的构造时，内存数据区不开辟存储各个枚举值的空间
 - ▶ 执行到变量的定义，内存数据区才会开辟空间存储变量的值
 - ▶ 同一作用域里，不能有相同的枚举值

...

```
int main( )
```

```
{ enum Color3{RED, YELLOW, BLUE};
```

```
enum Color7{RED, ORANGE, YELLOW, GREEN, CYAN, BLUE, PURPLE};
```

✗

```
...
```

枚举类型变量的定义（四种形式都可以）

- ▶ **<枚举类型名> <枚举类型变量名>;**

```
enum Color {RED, YELLOW, BLUE};  
Color c1, c2, c3;
```

- ▶ **enum <枚举类型名> <枚举类型变量名>;**

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

```
enum Color c1, c2, c3;    //加enum, 提醒Color是枚举类型
```

- ▶ **enum <枚举类型名> {<枚举值表>} <枚举类型变量名>;**

```
enum Color {RED, YELLOW, BLUE} c1, c2, c3;
```

- ▶ **enum {<枚举值表>} <枚举类型变量名>;**

```
enum {RED, YELLOW, BLUE} c1, c2, c3;
```


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

▶ 值域：

- ▶ 实际上是若干个有名字的整型常量的集合
- ▶ 枚举类型往往也被归入整型。

▶ 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
- ▶ ...
- ▶ 实际上是其对应的整数在参与操作

▶ 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 ...`

-
- ▶ 只能把**相同枚举类型**的数据赋给枚举变量

```
Weekday d1, d2;
```

```
d1 = SUN;
```

```
d2 = d1;
```

```
d1 = 1 ✗
```

```
d1 = RED ✗
```

例 枚举类型数据不可以直接输入输出

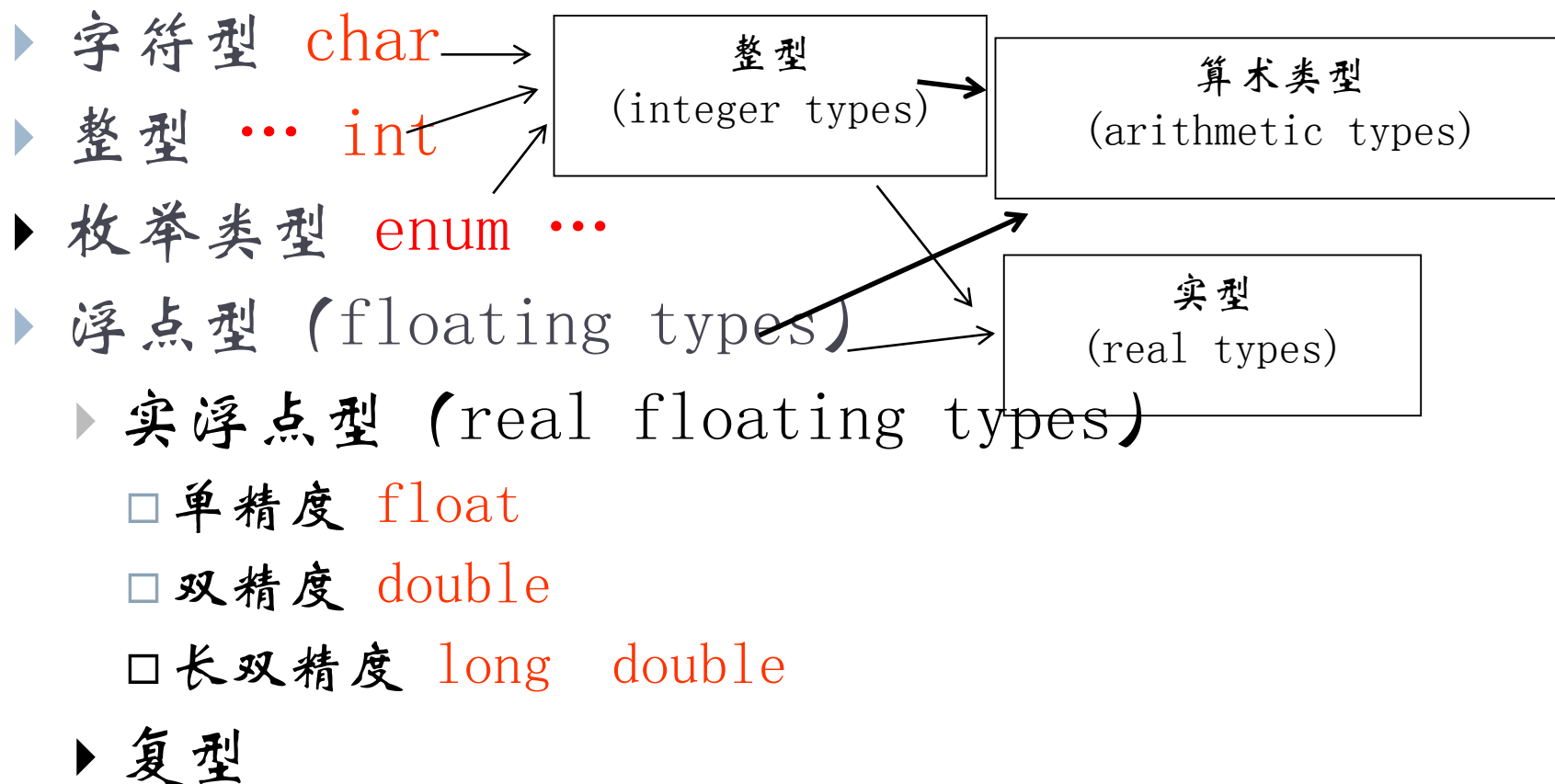
```
#include<stdio.h>
int main( )
{
    enum Weekday {SUN, MON, TUE, WED, THU, FRI, SAT};
    Weekday d1 = SUN, d2 = SAT;
    if(d1 < d2)
        printf("Sunday is the first day of a week. \n");
    else
        printf("\Which day is the first day of a week? \n");
    return 0;
}
```



枚举类型的好处和使用特点

- ▶ 可以约束操作数不在所列举的值之外取值。
 - ▶ 比如，对于星期这样的数据，如果用int型来描述，将会面临“1到底表示什么意思？”、“星期日用0还是7表示？”等问题，如果用0~6表示一个星期的每一天，则对于表示星期的int型变量d，不易防止“d = 10;”、“d = d*2;”等逻辑错误。
- ▶ 当一个操作数可取的值只是有限的几个（“语义明确的”）整数时，可以将其定义成某种枚举类型的变量，而不是定义成int型变量。
- ▶ 枚举类型通常与结构类型或联合类型（最后一章内容）结合起来使用。

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



打印各种类型的精度

► 参考以下网址，自己实现体会！

<http://blog.csdn.net/xuexiacm/article/details/8122267>

```
cout << "int: \t\t" << "所占字节数: " << sizeof(int);  
cout << "\t最大值: " << (numeric_limits<int>::max)();  
cout << "\t最小值: " << (numeric_limits<int>::min)() << endl;
```

```
cout << "float: \t\t" << "所占字节数: " << sizeof(float);  
cout << "\t最大值: " << (numeric_limits<float>::max)();  
cout << "\t最小值: " << (numeric_limits<float>::min)() << endl;
```

```
cout << "double: \t" << "所占字节数: " << sizeof(double);  
cout << "\t最大值: " << (numeric_limits<double>::max)();  
cout << "\t最小值: " << (numeric_limits<double>::min)() << endl;
```



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。
-

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



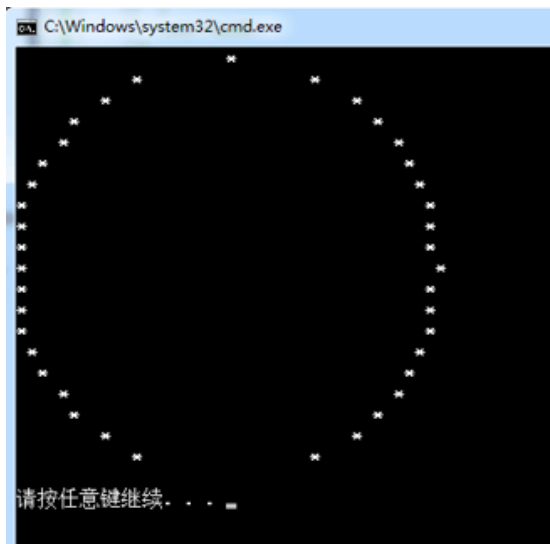
Q & A



测试第二题：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 $r=5\cdots$ 正常

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（运算）集：规定了对值集中的值能实施哪些运算。
 - ▶ 例如：整数类型就是一种数据类型，它的值集就是由整数所构成的集合，它的操作集包括：加、减、乘、除等运算。
-



内容回顾

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

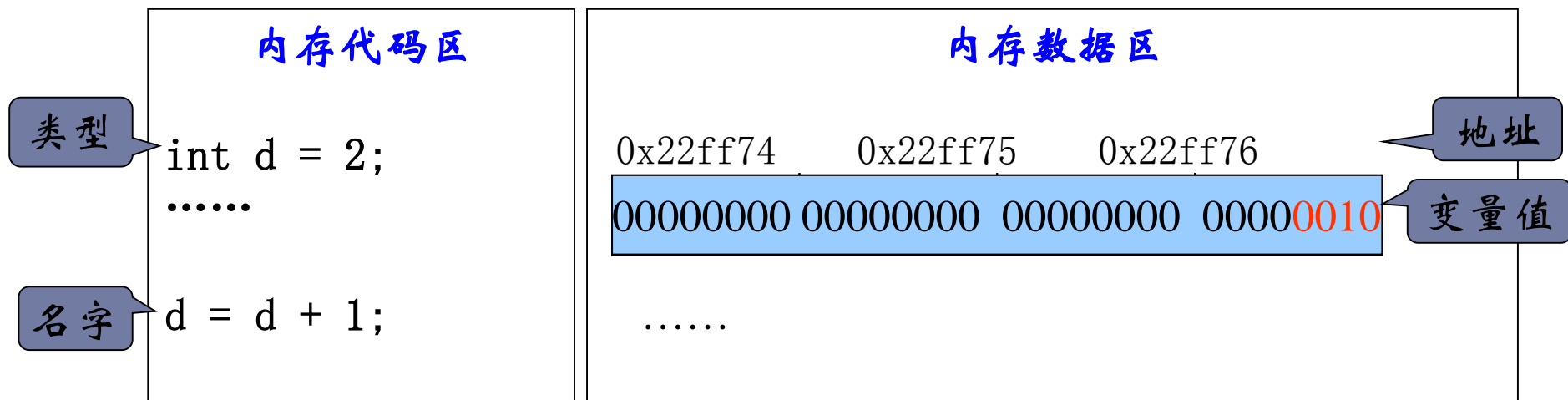
枚举类型
数组类型
结构与联合类型
指针类型
引用类型

抽象数据类型

类
派生类

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 定义变量类型和名字

系统 决定地址，存储二进制值

例：整型数据范围溢出示例

```
#include <stdio.h>
int main()
{
    int a = 2147483647, b = 1;
    printf("%d \n", a + b);
    return 0;
}
```

2147483647的补码

01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11

+1

10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

- 2147483648的补码

-2147483648

- 如果将输出格式符%d改成%u（即按unsigned int型数据输出），则结果为2147483648。

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）：有6位数字有效
 - ▶ 用于数值相关的计算时可能不符合精度要求
 - ▶ double（双精度型）：有15位数字有效
 - ▶ 一般情况下够了
 - ▶ long double（长双精度型）



可以使用常数FLT_EPSILON(单精度浮点型, 1.192092896e-7F) 或 DBL_EPSILON(双精度浮点型, 2.2204460492503131e-16), 需要包含这些常数定义的float.h, 这些常数被当成最小的正数。

内容回顾

```
#define EPSILON 0.0000001 //Define your own tolerance
```

```
...
float a, b, c;
a = 1.345f;
b = 1.123f;
c = a + b;

if(((2.468-EPSILON)<c) && (c<(2.468+EPSILON)))
    printf("They are equal.\n");
else
    printf("They are not equal! The value of c is %.10f, or %f",
c, c);
...
```

They are equal.

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: $\text{fabs}(x-y) < 1e-6$
 - ▶ $x != y$ 可写成: $\text{fabs}(x-y) > 1e-6$
 - ▶ $z == 0.3$ 可写成: $\text{fabs}(z-0.3) < 1e-6$

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

ASCII码表八、十六、十进制对照表

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

内容回顾

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		

字符结合ASCII码的操作

内容回顾

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
```



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
}
```
