信息与计算科学导论实验

流程控制、函数

仝伟

南京大学计算机科学与技术系

回顾：判断两个浮点数相等?

- 使用浮点型变量时，可能会产生精度误差。
  - 例如 `1.0 / 3 * 3 = 0.9999...` （与1不等）
- 可以使用

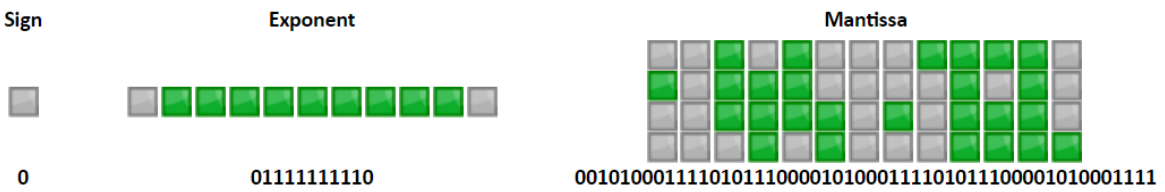
```
if (fabs(a - b) < epsilon) printf("1");
```

- 来判定a 和 b是否相等。其中epsilon为你需要自己定义的一个极小的常量，例如1e-6

- 对于a < b 或者 a > b，可以直接判断
- 思考：如何判断<=和>=?

浮点数的精度

0x3FE28F5C28F5C28F = 00111111 11100010 10001111 01011100
00101000 11110101 11000010 10001111



- 0.58　　5.79999999999999600319711134 94E-1

```
#include <iostream>
using namespace std;

int main() {

    float a = 0.58;
    float b = 5.799999999999996003197111349 4E-1;

    if (a > b)
        cout << "Yeah!" << endl;
    else
        cout << "Oops!" << endl;
}
```

循环语句

回顾：循环语句

- 循环语句一般由四个部分组成：
  - 循环初始化：为重复执行的语句提供初始数据
  - 循环条件：描述重复操作需要满足的条件
  - 循环体：描述要重复执行的操作
  - 下一次循环准备：为下一次循环更新数据（包括重复操作以及循环条件判断所需要的数据），它常常会隐式地包含在循环体中。

```
#define repeat(x) for(int rijk = (x); rijk > 0; --rijk)
```

回顾：while 语句

```
while (CONDITION) {
    STATEMENTS;
}
```

```
int i = 0;
while(i<n) {
    cout << "Hi!" << endl;
    i++;
}
```

- 执行while语句时，首先查看条件表达式是true还是false：
  - 若false，则循环终止，程序接着执行整个while语句后面的语句；
  - 若true，整个循环体被执行，然后程序返回到循环的开始再检查一次条件表达式。
- 通过一次循环体语句构成了循环的一个周期。

for 语句

```
for (INIT; CONDITION; STEP_EXPRESSION) {
    STATEMENTS;
}
```

- 在*INIT*中不可包含分号。
- for语句是C++中最重要的流程控制语句，它适合以特定的循环次数来重复执行某个操作
- 等价于下面的while语句

```
INIT;
while (CONDITION) {
    STATEMENTS;
    STEP_EXPRESSION;
}
```

for 语句

```
for (int i=0; i<n; i++)    cout << "Hi!" << endl;
```

- 等价于

```
int i = 0;
while(i<n) {
    cout << "Hi!" << endl;
    i++;
}
```

for 语句

- 求n的阶乘

```
int n; cin >> n;
int result = 1;
for (int num = 1; num <= n; num++) {
    result *= num;
}
cout "The factorial of " << n << " is " << result << endl;
```

- 求n的k次方

```
int n, k;
cin >> n >> k;
int result = 1;
for (int i = 0; i < k; i++) {
    result *= n;
}
cout << "The " << k << "th power of " << n << " is " << result << endl;
```

for语句的常见模式

```
for (int var = 0; var < n; var++) {
}

for (int var = start; var <= finish; var++) {
}
```

- var称为循环变量
- 常用的形式是使用++来使循环变量自增，但并非唯一形式：

```
for (int t = 10; t > 0; t--) {
    cout << t << endl;
}
```

```
int even_sum = 0;
for (int i = 2; i <= n; i += 2) {
    even_sum += i;
}
```

“死循环”

- 无限循环 – 循环永远结束不了
- 在循环体或for语句的***STEP_EXPRESSION***中一定要有能改变循环条件中操作数值的操作，并逐步使得循环条件有不满足的趋势，否则将会出现“死循环”！

```
int i = 0;
while(i < n) {
    cout << "Hi!" << endl;
    i++;
}
```

循环的种类

- 计数控制的循环
  - 循环前就知道循环的次数，循环时重复执行循环体直到指定的次数。
  - 用于计数的变量称为“**循环变量**”。
  - 循环的执行次数不依赖于循环体的执行结果。

- 事件控制的循环
  - 循环前不知道循环的次数，循环的终止是由循环体的某次执行导致循环的结束条件得到满足而引起的。
  - 循环的执行次数要依赖于循环体的执行结果。

循环语句的使用原则

- 三种循环语句在表达能力上是等价的，在解决某个具体问题时，用其中的一种可能会比其它两种更加自然。

- 一般来说，
  - 计数控制的循环一般用for语句；
  - 事件控制的循环一般用while。
  - 由于for语句能清晰地表示“循环初始化”、“循环条件”以及“下一次循环准备”，因此，一些非计数控制的循环也常用for语句实现。

例题：打印直角三角形

- 给定一个数N，打印一个由+, |, \ , -和空格构成的直角三角形。打印出的直角三角形应该包含N行，其中第 i 行应包含 2*i-1 个字符（包括空格）。以下是几个示例：

```
N = 1  2       3         4           5
    +  +       +         +           +
       +-+     | \       | \         | \
               +---+     |   \       |   \
                         +-----+     |     \
                                     +-------+
```

打印直角三角形 – 解决思路

- 打印首行 ‘+’
- 打印中间几行，对于第i行 ‘|’，2*i-3个 ‘ ’，‘\’
- 打印最后一行 ‘+’，2*N-3个 ‘-’，‘+’

打印直角三角形 – 代码

```
int main() {
    int  n; cin >> n;
    cout << '+' << '\n';          // output first line
    if (n == 1) return 0;
    for (int i=2; i<n; i++) {  // output the i-th line
        cout << '|';
        for (int j=0; j<2*i-3; ++j)   cout << ' ';
        cout << "\\\n";
    }
    cout << '+';                  // starting last line
    repeat(2*n-3) cout << '-';
    cout << "+\n";
    return 0;
}
```

流程控制

- 流程控制语句包括：
  - 顺序执行语句：按书写次序依次执行。
  - 选择执行语句：根据条件选择执行。
  - 循环执行语句：重复执行直到某个条件不满足。
  - 无条件转移语句：无条件转移到程序某个位置。

无条件转移控制

- 除了有条件的选择语句外，C++还提供了无条件的转移语句：
  - goto <语句标号>; - 程序程序转移到带有<语句标号>的语句（不要使用！）
  - break; - 在循环体中只要执行了break语句，就立即跳出（结束）当前这一层循环，循环体中跟在break语句后面的语句将不再执行，程序继续执行循环之后的语句。在循环体中，break语句一般作为某个if语句的子句，用于实现进一步的循环控制。
  - continue; - 立即结束当前这一轮循环，准备进入下一轮循环。
  - *return

关于goto语句

- goto语句会使得程序的静态结构和动态结构不一致，导致程序难以理解、可靠性下降和不容易维护。有时会导致程序效率的下降。
- 从结构化程序设计的角度讲，goto语句会破坏程序中的每一个结构所具有的单入口单出口的性质。
- 实际上，goto语句的使用可以分成两类：
  - 向前的转移（forward）（可用分支结构实现）
  - 往回的转移（backward）（可用循环结构实现）

不要使用goto语句！

while + if + break

- while循环适合于重复操作执行前需首先进行某些条件测试的情况。一种常见的使用情形是读取数据，直到遇到某个表示输入结束的特定数值（sentinel）

```
while (true) {
    // Prompt user and read in a value
    if (value == sentinel) break;
    // Process the data value
}
```

- while (true) 似乎引入了一个无限循环。该程序跳出循环的唯一方式就是通过执行循环体中的break语句

while + if + break

```
int main() {
    int n = 100000;
    while (n--) {
        double eps;
        ...
        if (eps <= 1e-6)
            break;
    }
}
```

while + if + continue

```
int main() {
    int n = 100000;
    while (n--) {
        if (n % 100 == 2)
            continue;
    }
}
```

更多流程控制语句

switch 语句

- 程序中有时需要从两个（组）以上的语句中选择一个（组）来执行。
- C++提供了一条多路选择语句：switch语句能根据某个表达式的值在多组语句中选择一组语句来执行。
- 表达式e为控制表达式，执行switch语句时，将其与c1、c2等值进行比较。此处c1、c2等值必须是**标量类型常数**。
- case子句最后的break语句，表明该case子句执行结束并跳出该case子句，接着执行**整个switch语句后面的语句**

整型常量
字符常量
*枚举类型

```
switch (e) {
    case c1:
        STATEMENTS
        break;
    case c2:
        STATEMENTS
        break;
    ...

    default:
        STATEMENTS
        break;
}
STATEMENTS
```

switch 语句 – 例

```
int main() {
    int day; cin >> day;
    if (day == 0) cout << "Sunday";
    else if (day == 1) cout << "Monday";
    else if (day == 2) cout << "Tuesday";
    else if (day == 3) cout << "Wednesday";
    else if (day == 4) cout << "Thursday";
    else if (day == 5) cout << "Friday";
    else if (day == 6) cout << "Saturday";
    else cout << "Input error";
    cout << endl;
    return 0;
}
```

```
int main() {
    int day; cin >> day;
    switch (day) {
        case 0: cout << "Sunday"; break;
        case 1: cout << "Monday"; break;
        case 2: cout << "Tuesday"; break;
        case 3: cout << "Wednesday"; break;
        case 4: cout << "Thursday"; break;
        case 5: cout << "Friday"; break;
        case 6: cout << "Saturday"; break;
        default: cout << "Input error";
    }
    cout << endl;
    return 0;
}
```

switch 语句 – break

- 在执行switch语句的某个分支时，需要用break语句结束该分支的执行。
- 在switch语句的一个分支的执行中，如果没有break语句（最后一个分支除外），则该分支执行完后，将继续执行紧接着的下一个分支中的语句序列。
- 很多时候是bug，有些情况是刻意为之

```
switch (e) {
    case c1:
        STATEMENTS
        break;
    case c2:
        STATEMENTS
        break;
    ...

    default:
        STATEMENTS
        break;
}
STATEMENTS
```

switch 语句 – 例

```
switch (month) {
    case 4:
    case 5:
    case 9:
    case 11:
        num_of_days = 30;
        break;
    case 2:
        if ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0) {
            num_of_days = 29;
        }
        else {
            num_of_days = 28;
        }
        break;
    default:
        num_of_days = 31;
        break;
}
```

Your library is your paradise.

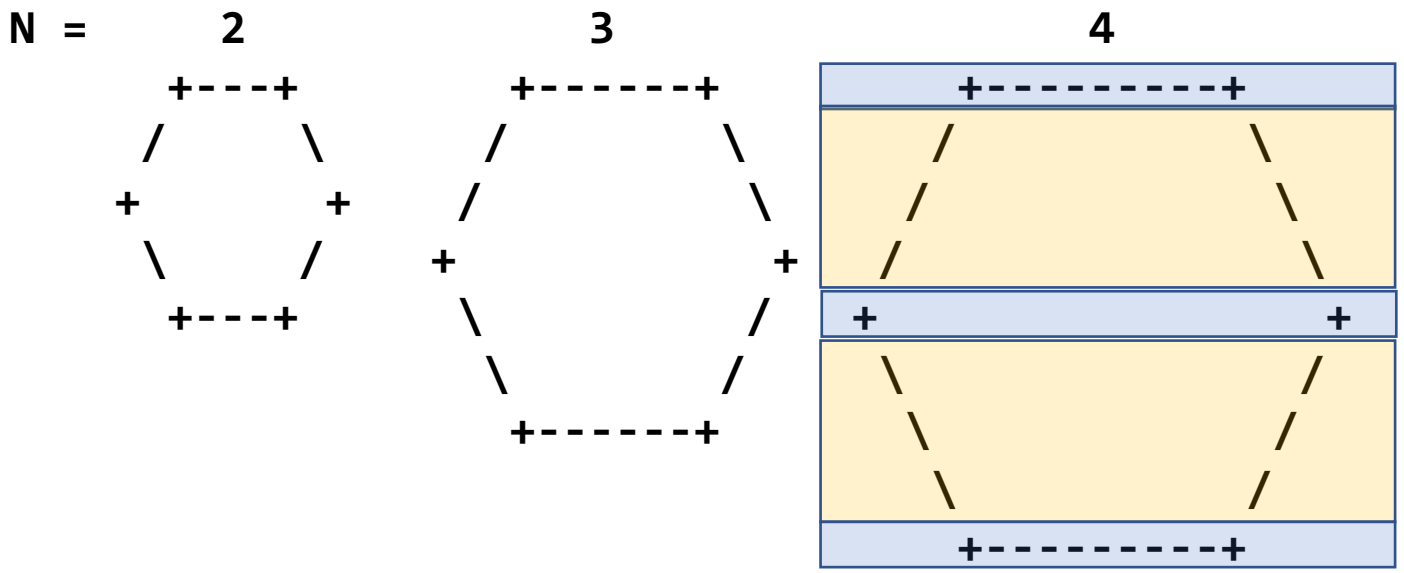
- Desiderius Erasmus

(Fisher’s Study at Rotterdam), 1524

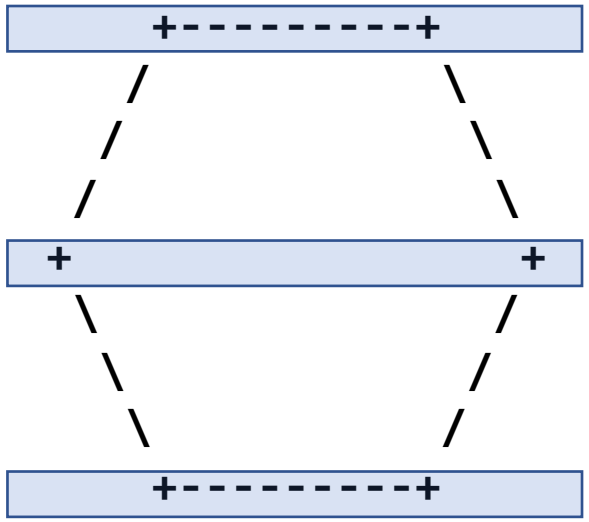
函数与库

例题：打印六边形

- 给定一个数N，打印一个大小为N的六边形。使用以下示例弄清楚所需实现的模式。打印出的形状所处位置应确保六边形最左侧顶点的左侧没有空格。



例题：打印六边形

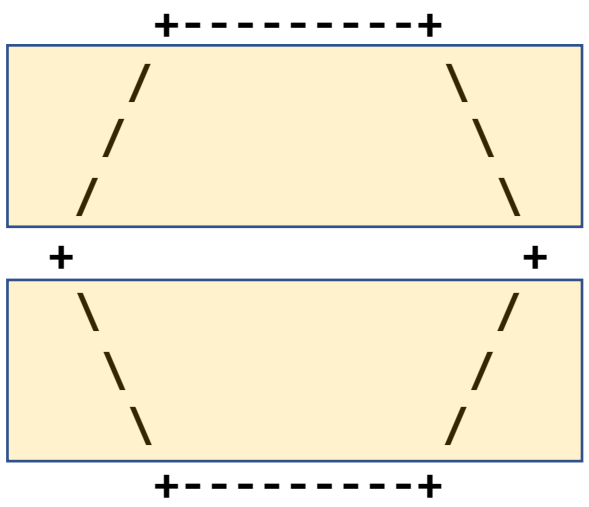


```
    for(int j = 0; j < n; j++) cout << ' ';
    cout << '+';
    for(int j = 0; j < (n-1)*3; j++) cout << '-';
    cout << '+' << endl;
```

```
    cout << '+';
    for(int j = 0; j < (n-1)*3 + n*2; j++) cout << ' ';
    cout << '+' << endl;
```

```
    for(int j = 0; j < n; j++) cout << ' ';
    cout << '+';
    for(int j = 0; j < (n-1)*3; j++) cout << '-';
    cout << '+' << endl;
```

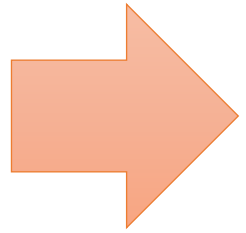
例题：打印六边形



```
    for (int i = 1; i < n; i++) {
        for(int j = 0; j < n-i; j++) cout << ' ';
        cout << '/';
        for(int j = 0; j < (n-1)*3 + i*2; j++) cout << ' ';
        cout << '\\' << endl;
    }
```

```
    for (int i = n-1; i > 0; i--) {
        for(int j = 0; j < n-i; j++) cout << ' ';
        cout << '\\';
        for(int j = 0; j < (n-1)*3 + i*2; j++) cout << ' ';
        cout << '/' << endl;
    }
```

```
    for(int j = 0; j < n; j++) cout << ' ';
    cout << '+';
    for(int j = 0; j < (n-1)*3; j++) cout << '-';
    cout << '+' << endl;
    for (int i = 1; i < n; i++) {
        for(int j = 0; j < n-i; j++) cout << ' ';
        cout << '/';
        for(int j = 0; j < (n-1)*3 + i*2; j++) cout << ' ';
        cout << '\\' << endl;
    }
    cout << '+';
    for(int j = 0; j < (n-1)*3 + n*2; j++) cout << ' ';
    cout << '+' << endl;
    for (int i = n-1; i > 0; i--) {
        for(int j = 0; j < n-i; j++) cout << ' ';
        cout << '\\';
        for(int j = 0; j < (n-1)*3 + i*2; j++) cout << ' ';
        cout << '/' << endl;
    }
    for(int j = 0; j < n; j++) cout << ' ';
    cout << '+';
    for(int j = 0; j < (n-1)*3; j++) cout << '-';
    cout << '+' << endl;
```



- 前驱空格
- 起始字符
- 中间字符
- 结束字符

例题：打印六边形

- 代码有些繁琐?
- 实现的代码中会存在很多看似重复却不完全相同的部分
- 此时适合用函数!

函数

基于过程抽象的程序设计

- 人们在设计一个复杂的程序时，经常会用到功能分解和复合两种手段：
  - 功能分解：把程序的功能分解成若干子功能，每个子功能又可以分解成若干子功能，等等，直到最终分解出的子功能相对简单、容易实现为止，从而形成了一种自顶向下（top-down）、逐步精化（step-wise）的设计过程。
  - 功能复合：先设计子功能，然后把已有子功能逐步组合成更大的子功能，最后得到完整的系统功能，从而形成一种自底向上（bottom-up）的设计过程。
- 功能分解和复合的程序设计基于了一种抽象机制 -- 功能抽象（过程抽象）：一个功能的使用者只需要知道相应功能是什么（what to do），而不必知道它是如何做的（how to do）。

子程序与函数

- 子程序是取了名的一段程序代码，在程序中通过名字来使用（调用）它们。
  - 减少重复代码，节省劳动力
  - 实现过程抽象
  - 封装和信息隐藏的作用
  - 语言功能的扩充

- C++中函数是用于实现子程序的语言成分

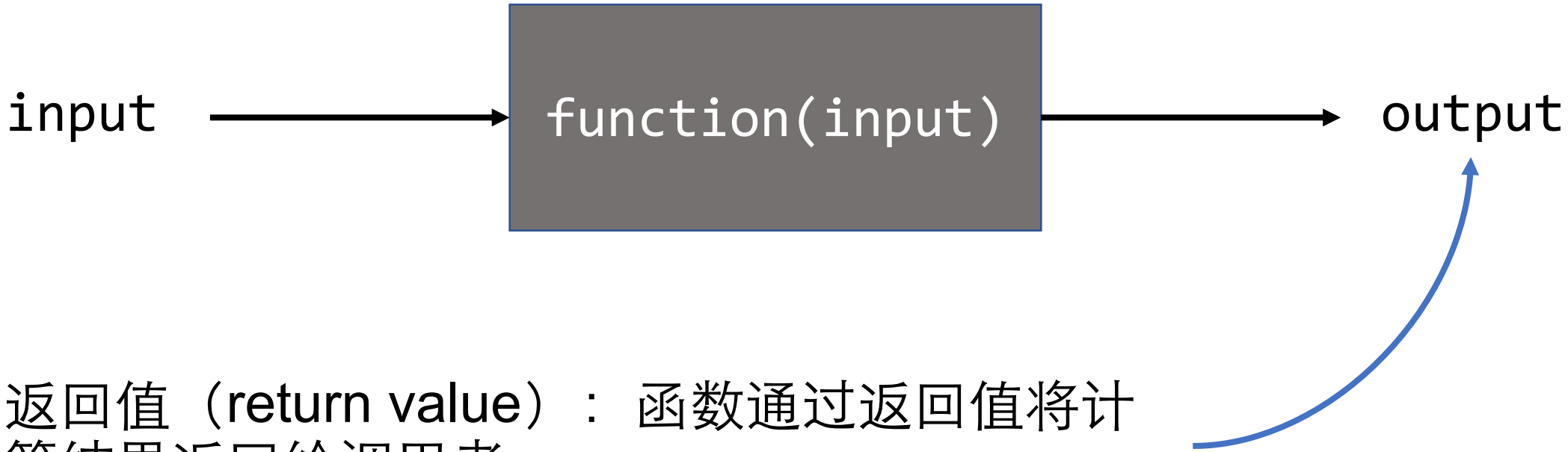
子程序的数据传递

- 一个子程序所需要的数据往往要从调用者（也是一个子程序）那里获得，计算结果也需要返回给调用者。
- 子程序之间的数据传递
  - 全局变量：所有子程序都能访问到的变量。
  - 参数：
    - 形式参数（形参）：用于被调用者接受数据
    - 实在参数（实参）：用于调用者提供数据
    - 值传递：把实参的值复制一份给形参。
    - 地址或引用传递：把实参的地址传给形参。
- 返回值机制：返回计算结果。

函数

- 形式参数（parameters, 形参）： 函数期望作为输入的变量
- 实在参数（arguments, 实参）： 传入函数的值，并且将其赋值给形参变量

函数



- 返回值（return value）： 函数通过返回值将计算结果返回给调用者

函数：声明与定义

函数的定义

```
type name(varType parameter1, varType parameter2, ...) {
    ... body ...
}
```

返回值类型：可以为任意的C++数据类型

- 如果函数不返回任何值？可以使用void作为返回值类型
- void只可以作为返回值类型，不用作为形参或变量类型

void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 *）

函数的定义

```
type name(varType parameter1, varType parameter2, ...) {
    ... body ...
}
```

函数名：用于标识函数的名字，用标识符表示

返回值类型：可以为任意的C++数据类型

- 如果函数不返回任何值？可以使用void作为返回值类型
- void只可以作为返回值类型，不用作为形参或变量类型

形式参数表（函数期望的输入）：由零个、一个或多个形参说明（用逗号隔开）构成

- 和变量声明的方式很相似
- 可以将形参看作一系列作用域为该函数的局部变量

函数体

```
type name(varType parameter1, varType parameter2, ...) {
    ... body ...
}
```

函数体：函数体*body*用于实现相应函数的功能

- 函数体内可以包含return语句，格式为：

  return *expression*;

- 当函数体执行到return语句时，函数立即返回到调用者。如果有返回值，则把返回值带回给调用者。

函数体

```
type name(varType parameter1, varType parameter2, ...) {
    // ... body ...
    varType variable = /* some fancy code. */

    return variable;
}
```

返回值

- 如果return中的*expression* (*variable*)的类型*varType*与函数返回值类型*type*不一致，则进行隐式类型转换，基本原则为：把*expression*转成*type*。

return;

- 对于返回值类型为void的函数，函数体中也可以没有return语句，执行完最后一个语句返回。

```
return; //返回值类型为void
```

```
void print(double eps) {
    if (eps > 0.01) {
        cout << "The parameter eps is too large. -- " << eps << endl;
        return;
    }
    cout << "Computing the noise level ... " << endl;
    // do some computation ...
}
```

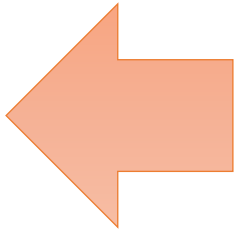
```
int add(int a, int b) {
    int c = a + b;
}
```

```
main.cpp:25:1: warning: no return statement in function returning non-void [-Wreturn-type]
   25 | }
      | ^
```

打印六边形

- 写个函数?

前驱空格
起始字符
中间字符
结束字符



```
    for(int j = 0; j < n; j++) cout << ' ';
    cout << '+';
    for(int j = 0; j < (n-1)*3; j++) cout << '-';
    cout << '+' << endl;
    for (int i = 1; i < n; i++) {
        for(int j = 0; j < n-i; j++) cout << ' ';
        cout << '/';
        for(int j = 0; j < (n-1)*3 + i*2; j++) cout << ' ';
        cout << '\\' << endl;
    }
    cout << '+';
    for(int j = 0; j < (n-1)*3 + n*2; j++) cout << ' ';
    cout << '+' << endl;
    for (int i = n-1; i > 0; i--) {
        for(int j = 0; j < n-i; j++) cout << ' ';
        cout << '\\';
        for(int j = 0; j < (n-1)*3 + i*2; j++) cout << ' ';
        cout << '/' << endl;
    }
    for(int j = 0; j < n; j++) cout << ' ';
    cout << '+';
    for(int j = 0; j < (n-1)*3; j++) cout << '-';
    cout << '+' << endl;
```

一个函数：打印一行

```
void printLine(int n1, char first, int n2, char middle, char last) {
    for (int i = 0; i < n1; i++)
        cout << ' ';
    cout << first;
    for (int i = 0; i < n2; i++)
        cout << middle;
    cout << last << '\n';
}
```

一个函数：打印一行

返回值类型

函数名

形式参数（形参）

```
void printLine(int n1, char first, int n2, char middle, char last) {
    …

}
```

函数体

函数调用

实际参数（实参）

```
int main() {

    print_line(n, '+', (n-1)*3, '-', '+');        // Top row
}
```

打印六边形：代码

```
int main() {
    int  n;    cin >> n;
    printLine(n, '+', (n-1)*3, '-', '+');       // Top row
    for (int i=1; i<n; i++)
        printLine(n-i, '/', (n-1)*3 + i*2, ' ', '\\');
    printLine(0, '+', (n-1)*3 + n*2, ' ', '+'); // Middle row
    for (int i=n-1; i>0; i--)
        printLine(n-i, '\\', (n-1)*3 + i*2, ' ', '/');
    printLine(n, '+', (n-1)*3, '-', '+');       // Bottom row
    return 0;
}
```

函数的声明 – 函数原型

```
type name(varType parameter1, varType parameter2, ...);
```

- 程序中调用的所有函数都要有定义。
- 如果在调用前没见到函数的定义（如在C++的标准库、在本源文件中调用点之后或在其它源文件中定义），则在调用前需要对被调用的函数进行声明。
- 在函数声明中，形式参数表中可以只列出形参的类型而不写形参名

```
type name(varType, varType, ...);
```

函数定义的例子

```
double power(double x, int n) {
    if (x == 0) return 0;
    double product = 1.0;
    if (n >= 0) {
        while (n > 0) {
            product *= x;
            n--;
        }
    }
    else {
        while (n < 0) {
            product /= x;
            n++;
        }
    }
    return product;
}
```

main

- 每个C++程序都要定义一个名字为main的函数，C++程序的执行是从main开始的。函数main返回值类型为int，参数常省略。例如：

```
int main() {
    ...
    return -1;
    ...
    return 0;
}
```

- 一般情况下，返回0表示程序正常结束；返回负数（如-1）表示程序非正常结束。

函数：调用

函数的调用

- 对于定义的一个函数，必须要调用它，它的函数体才会执行。
  - 除了函数main外，函数的调用都是从main开始的。
  - main一般是由操作系统来调用。
- 函数调用的格式如下：

  name(*arguments*)

  - *name* 为某个已定义函数的名字；
  - *arguments* 是实在参数表：由零个、一个或多个表达式（用逗号隔开）构成
  - 实参的个数和类型应与相应函数的形参相同。类型如果不同，编译器会试图进行隐式转换，转换规则是把实参类型转换成形参类型 。

函数调用的方式

- 以函数在程序中的出现位置和形式来看，函数的调用方式可分为以下3种
  - 函数调用作为独立语句。调用函数完成某项功能，没有任何的返回值:

```
printLine(n, '+', (n-1)*3, '-', '+’);
```

  - 函数作为表达式的一部分:

```
x = sqrt(a*a + b*b) / 2.0;
```

  - 以实参形式出现在其它函数的调用中:

```
num = min(sum(1,100), candidate);
```

函数调用的例子

```
double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

被调用者（callee）

```
int main() {
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}
```

调用者（caller）

```
double average(double a, double b);

int main() {
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}

double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

- 次序很重要：函数在被调用前需要先被定义

函数调用的例子

```
void f1() {

}

void f2(int, int);
int g(double, double);

int main() {
    ...
    f1();
    f2(3, 3);
    int x = g(1.2, 2.3);
    ...
    return 0;
}
```

```
void f2(int a, int b) {
    ...
}
int g(double u, double v) {
    ...
}
```

函数调用机制

- 调用者计算实参的值
- 把实参分别传递到被调用函数的形参变量中；（值或地址）
- 执行函数体的语句，直到遇到return或者没有多余可执行的语句
  - 如果函数有返回值，函数体内return语句表达式的值将被计算，并作为函数值返回给调用者
- 将函数返回值带入到函数调用点位置
  - 有返回值的函数调用作为操作数放在表达式中参加运算

```
x + power(x, y) * z
```

函数调用是如何进行的

```
// This is function definition
void printLine(int n1, char first, int n2, char middle, char last)
{ repeat(n1) cout<<' ';   … }

// This is a function call, using actual arguments
print_line(n, '+', (n-1)*3, '-', '+');
// First evaluate the expressions in actual arguments,
// assume n == 4, then it becomes
print_line(4, '+', 9, '-', '+');
// which becomes something like
{ n1=4;  first='+';  n2=9;  middle='-';  last='+';
     repeat(n1) cout<<' ';   … }
```

函数的参数传递

- C++提供了两种参数传递机制：
  - 值传递：把实参的值赋值给形参。
  - 地址或引用传递：把实参的地址赋值给形参。
- 参数传递方式在函数定义的参数类型中指出。
- C++默认的参数传递方式是值传递（pass by value）
  - 在调用时，采用类似变量初始化的形式把实参的值传给形参。
  - 函数执行过程中，通过形参获得实参的值
  - 函数体中对形参值的改变不会影响相应实参的值

值传递

```
#include <iostream>
using namespace std;

int doubleValue(int x) {
    x *= 2;
    return x;
}

int main() {
    int my_value = 5;
    int result = doubleValue(my_value);

    cout << "my_value: " << my_value << " ";
    cout << "result: " << result << endl;
    return 0;
}
```

问题：均方根误差

- 图图和豪豪各自有一套预测房价的方法，他们的预测方法基于五个信息：房屋面积、所处楼层、卧室个数、客厅（餐厅）个数、卫生间个数。
- 需要计算他们两人的预测结果的均方根误差。打印结果时，保留两位小数。
- 平方误差等于预测值与实际值之差的平方，均方误差等于一组平方误差值的平均值，均方根误差是均方误差的平方根。

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{n}(\widehat{y_t}-y_t)^2}{n}}$$

样例输入输出

Sample Input 0

```
3
1400000  95   8   3  1  1
2500000  160  12  4  2  2
1430000  105  15  2  1  2
```

Sample Output 0

```
150498.06 458341.67
Tu's RMSE is 32.84% of Hao's RMSE.
```

均方根误差

```
int (int s, int f, int b, int g, int r) {
  //图图预测5000 - 2000 * b+ 13000 * s + 30000 * r + 24000 * g + 10000 * f;
  return 5000 - 2000 * b+ 13000 * s + 30000 * r + 24000 * g + 10000 * f;
}

int hhPredict(int s, int f, int b, int g, int r) {
  //豪豪预测20000 * b + 7500 * f + 10000 * r + 5000 * g + 10000 * s;
  return 20000 * b + 7500 * f + 10000 * r + 5000 * g + 10000 * s;
}
```

均方根误差

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{n}(\widehat{y_t}-y_t)^2}{n}}$$

```
for (int i = 0; i < n; i++) {
    int price, s, f, b, g, r;
    cin >> price >> s >> f >> b >> g >> r;
    double err = ttPredict(s, f, b, g, r) - price;
    sum_t += err * err; // 累计图图的错误
    err = hhPredict(s, f, b, g, r) - price; // 预测
    sum_h += err * err; // 累计豪豪的错误
}
double rmse_t = sqrt(sum_t / n);
double rmse_h = sqrt(sum_h / n);
```