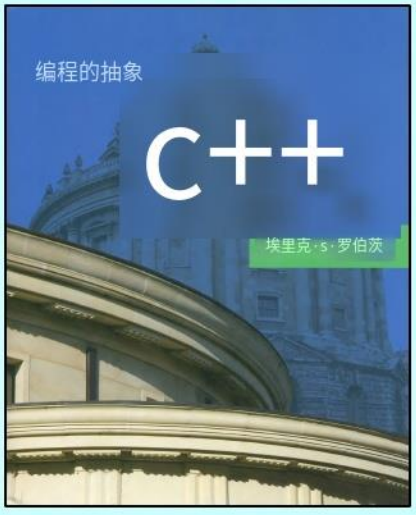
第二章

函数和库



你的库就是你的天堂。

——desiderius Erasmus，《费雪在鹿特丹的研究》。

程序设计范型

编程范式是一种“风格”或“方式”
编程。

编程语言的特征之一是 its
对特定范式的支持。一些语言做到了
在某些范式下很容易写，但在其他范式下就不容易了。

一种专门设计用于编程的语言
许多范式被称为多范式编程
语言，例如 c++，Python。你可以写程序或者
库，主要是过程的，面向对象的，或者
功能(即。，一些典型的范式)。

在一个大型程序中，甚至可以用不同的部分编写
不同的范式。

c++支持过程式和面向对象的范式
自然地，支持函数式范式通过
<函数式>接口，并支持许多其他范式
通过多种外部库

过程式编程范式

命令式编程范式:一个显式的
改变程序状态的语句，指定如何改变
实现结果。

-结构化:程序有干净、免 goto、嵌套
控制结构。

过程式:使用过程的命令式编程
对数据进行操作。

过程编程是一种编程范式，
从结构化编程衍生而来，基于概念
的过程调用。

过程，也称为例程、子程序或
函数，简单地包含了一系列的计算步骤
的运算。

在计算机程序中，函数是有名称的部分

函数的思想

• 函数是代数中一个熟悉的概念
  基于一个或多个函数指定一个数学计算
  未知值，称为参数。

• 以数学函数为例

$$f(x) = x2 + 1$$

  指定可以计算函数的值 *f* by
  将参数 x 平方，然后将结果加 1。因此，
  f(0)是 1，f(1)是 2，f(2)是 5，以此类推。

• 在 c++中，可以通过。实现一个数学函数
  在函数定义中编码计算，像这样:

```
Double  f(Double  x) {
    返 回   x  *  x  +  1;
}
```

编程中的函数

在编程语言中，函数是一段代码
被组织成一个独立的单元，并有一个名字。

定义函数后，使用函数名进行调用的行为
这段代码就是调用函数。

调用程序可以将信息传递给使用的函数
参数。

调用后，该函数将提供的数据作为参数，
它是否工作，然后返回到调用程序
在进行调用的代码点。当它返回时，
函数通常会将一个值传递回调用者。

本质上，你是在复制函数内的代码
定义到调用它的地方，用变量(即，
参数)替换为from 的实参
调用者。

参数 vs. Arguments

参数是一个参数的占位符

在函数调用中，在大多数方面就像一个局部变量

变量。

- 参数是函数声明中的变量

  定义，以及参数的占位符。

- 参数是调用时使用的表达式

  函数，以及形参的实现。

- 当函数被调用时，参数就是数据(实际值)你传递给函数的参数(局部变量)。

使用函数的优点

- 函数允许您缩短程序，允许您
  一次性地包含特定操作的代码
  你可以在各种不同的环境中经常重用它，
  通常用不同的参数。
- 函数使程序更容易阅读，通过允许您
  使用单一名称调用整个操作序列
  (抽象)对应于更高层次的理解
  功能的目的。
- 函数通过允许您简化程序维护
  将一个大程序分成更小、更容易管理的部分。
  这个过程叫做分解。你可以更新
  功能的实现而不影响程序
  调用函数。

自上而下的设计/逐步细化。

在 c++中定义函数

c++ 中函数定义的一般形式看起来很丰富和其他派生自 C 的语言一样，例如 Java，但比 Python 稍微复杂一些:

```
类型名称(参数列表){
    函数体中的语句
}
```

其中 type 表示函数返回的类型，name 是函数名，参数列表是的列表
变量声明用于保存每个变量的值
论点。

函数体中的语句实现了函数(例如:
算法)，至少包含一个 return 语句。

```
返回表达式;
```

问题

如果我想返回多个值怎么办?

在 c++中定义函数

- 返回布尔值结果的函数称为谓词
  功能。

```
bool isEven(int n) {
    返回 n % 2 == 0;
}
...
```

- 没有返回值的函数通常被调用
  过程，使用 void 作为结果类型。

所有函数都需要在被调用之前声明
指定一个包含标题行之后的原型
由分号。

函数原型

函数原型就是函数的标题行
后面是一个分号，你把它放在 main 前面
函数，以便编译器知道参数是什么
函数需要什么类型的值，在你之前
实际定义函数。

- 如果你总是在调用函数之前定义函数，
  原型就不需要了。
  - 例如，底层功能在开始时出现，
    然后是调用的中级函数
    它们，主程序在最下面。
- 虽然这种策略可以节省一些原型线，但确实如此
  在自上而下设计的意义上是违反直觉的，因为
  大多数通用功能出现在最后。
- 可能在某些情况下，你不能总是
  在调用函数之前定义函数(例如，相互递归)。

函数和算法

- 函数对编程至关重要，因为它们提供了
  用来表达算法的结构。
- 解决特定问题的算法可能有很大差异
  其效率(或复杂性)。思考是有意义的
  当你选择算法的时候要小心，因为
  做一个错误的选择可能会付出极大的代价。

接下来的几张幻灯片通过实现来说明这一原则
计算的最大公约数的两个算法
整数 x 和 y，定义为最大整数
这两个数都能被整除。

暴力方式

计算最大公约数的一种策略是
从较小的数开始倒数，直到找到一个
这样就能把两个数平分了。代码看起来像这样:

```
Int gcd(Int x, Int y) {
    Int guess = (x < y) ?X: y;
    While (x %猜 != 0 || y %猜 != 0){
        猜一猜,
    返回猜;
}
```

当 x 和 y 为正值时，该算法必须终止
因为 guess 的值最终会达到 1。在那
点，猜想一定是最大公约数，因为
while 循环应该已经测试过所有较大的除数了。

尝试所有可能性被称为蛮力策略。

欧几里得算法

- 如果你使用蛮力方法来计算最大的 1000005 和 1000000 的公约数，程序会走差不多 100 万步，就能告诉你答案是 5。

使用更好的方法，你可以更快地得到答案算法。亚历山大的希腊数学家欧几里得 23 世纪前描述了一种更高效的算法，它长这样:

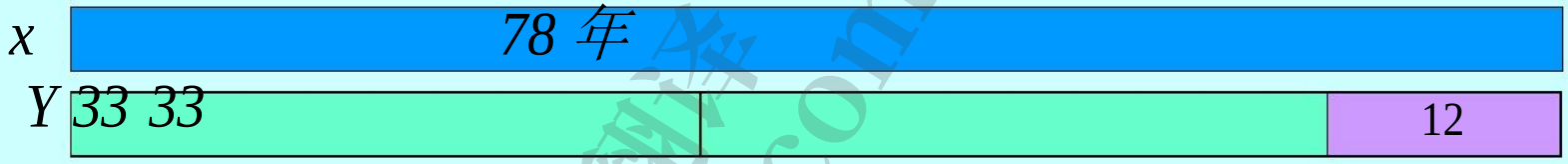
```
Int gcd(Int x, Int y) {
    Int r = x % y;
    While (r != 0) {
        x = y;
        y = r;
        r = x % y;
    }
    返回 y;
}
```

欧几里得算法是如何工作的

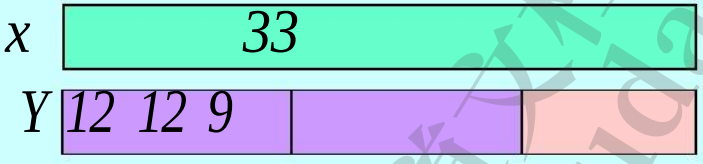
- 欧几里得的伟大见解是的最大公约数 *X 和 y 也一定是 y 和的最大公约数* x 除以 y 的余数。此外，他还能证明这个命题在他的《元素》第七卷里。
- 如果你在 1000005 和 1000000 上使用欧几里得算法，你只需 2 步就能得到正确答案，这样就好多了比蛮力需要的百万步强多了。
- 如果你思考一下，很容易看出欧几里得算法是如何工作的和欧几里得一样，从几何角度研究这个问题。下一个当 x 是 78 时，幻灯片将完成计算中的步骤 y 是 33。

欧几里得算法的一个例子

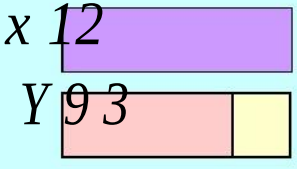
第 1 步:计算 78 除以 33 的余数:



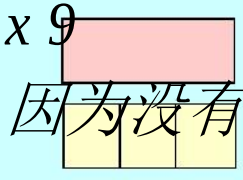
第二步:计算 33 除以 12 的余数:



第三步:计算 12 除以 9 的余数:



第四步:计算 9 除以 3 的余数:



因为没有余数，所以答案是 3。

对函数的 c++增强

- 函数可以重载，这意味着你可以
  定义几个与 long 同名的不同函数
  As 的正确版本可以通过查看来确定
  参数的数量和类型。
- 参数的模式(即:的数量和类型
  参数而不是形参名称)是必需的
  特定的函数称为它的签名。

使用重载的主要优点是这样做
  使你作为程序员更容易跟踪
  相同操作的不同函数名
  应用在稍微不同的上下文中。

- 原型与签名

练习:过载和签名

- 如果我们在每种情况下输入 1 或 1.0，输出是什么?

```
Int add1(Int x);
Double add1(Double

Int main() {
    双 x;
    Std::cin >> x;
    Std::cout << add1(x)/4;
    返回 0.
}
Double add1(int x){//模棱两可的签名
    Return x + 1;
}
Int add1(Int y) {
    返回 y + 1;
}
双 add1(双 x){
```

对函数的 c++增强

- 函数可以通过包含初始化器在函数原型中的变量名之后。例如，函数原型

```
setMargin(int margin = 72);
```

指示 setMargin 接受可选参数
默认值为 72。

定义一个 setInitialLocation 过程，它接受一个 x 和 y 坐标作为参数:

```
setInitialLocation(double x, double y);
```

- 给参数赋默认值使其可选:

```
setInitialLocation(双 x = 0，双 y = 0);
```

对函数的 c++增强

当你为可选参数使用默认值时，它会有所帮助
记住以下规则:

—默认值的规格(default . value)
参数)只出现在函数原型和
而不是在函数定义中。

-任何可选参数都必须出现在末尾
参数列表。后面可能没有必要的参数
可选参数。

锻炼

```
setInitialLocation(双 x = 0， 双 y = 0);
```

- 在上面的例子中，如果用户只提供一个参数，它将被分配给第一个参数。如果我们想要阻止用户调用 setInitialLocation 只有一个参数(即。，用户可以提供其中任何一个参数或完全不提供参数)?

使用重载来实现默认参数:

```
setInitialLocation(double x, double y);

无效 setInitialLocation(){
    setInitialLocation(0,0);
}
```

调用函数的机制

当你调用函数时，会发生以下动作:

1. 调用函数计算its中的参数表达式自己的上下文。

2.c++然后将每个参数值复制到相应的形参变量，它被分配在一个新赋值的称为栈帧的内存区域。这个任务遵循参数出现的顺序:第一个实参被复制到第一个形参变量中，以此类推。

3.c++然后计算函数体中的语句，使用新的栈帧来查找局部变量的值。

4.当 c++遇到 return 语句时，它会计算返回值，并用该值代替调用。

5.c++然后为被调用的函数丢弃栈帧返回到调用者，从中断的地方继续。

组合函数

- 为了说明函数调用，文中使用了函数 C(n,k) that 计算组合函数，即的个数从一组 n 个对象中选择 k 个元素的方法。
- 例如，假设你有一组五枚硬币:a便士，五分镍币，一角硬币，二角五分硬币和一美元:

选出两枚硬币的方法有多少种?

便士+镍币	Nickel + dime	一角硬币+四分之一	Quarter + dollar
便士+一角硬币	镍 + 25 美分		
便士+ 25 美分	镍+美元		
便士+美元			

一共 10 种方式。

组合和阶乘

幸运的是，数学提供了一种更简单的计算方法组合函数比所有方法都计算更有效。的组合函数的值由公式给出:

$$C(n, k) \quad \frac{n}{k! \times (n-k)!}$$

•将组合分解为阶乘。从顶层开始，
组合函数需要一个阶乘函数。鉴于
如果你已经有了一个事实函数，就很容易把它转过来
公式直接转化为 c++函数，如下:

```
Int combinations(Int n, Int k) {
    返回事实(n)/(事实(k)*事实(n - k));
}
```

计算阶乘

一个数n的阶乘(通常写成n!在在数学中)被定义为整数的乘积1到n。因此，5!等于120，也就是1 x 2 x 3 x 4 x 5。

- 下面的函数定义使用for循环进行计算阶乘函数:

```
Int 事实(Int n) {
    Int result = 1;
    For (int I = 1;I <= n;我 + +){
        结果*= i;
    返回结果;
}
```

下一张幻灯片模拟了组合和的操作

事实在一个简单的主要功能的上下文中。

组合程序

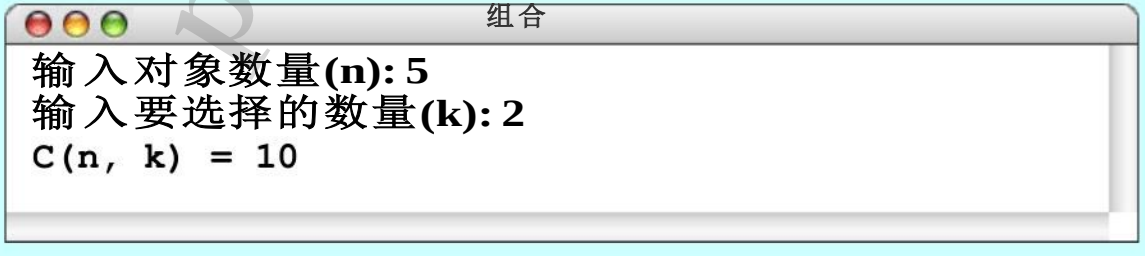
```
Int main() {
 Intintcombinations (int n, k;N, int k) {
  cout << "输入对象数量(n): ";返回的事实
  (n)
  Cinfact (int intn){/(事实(k) *事实(n - k));
  > > n;

 }
    coutint<<result"Enter120= the 1;number2 to be chosen (k): ";
    Cinfor >>(k;int I = 1;I <= n;i++) {
```

22
543216

在 this 程序点，callallsmakesnowprogramcallsfactanothertheyetcallsfactagaincallthetofunction,
combinationswithfactn，-withk askits applyingasfunction，它是相同的进程。应对。论点。如下:

1.TheThisEvaluatefinalfactcall tocallfunctionthefacttoarguments factreturnsreturnsthentheandvaluethevaluekvalueto2.120get6。此例
itsintegerscaller。5和 2。



问题

- 这个动画让你想起了什么?

锻炼

编写一个函数交换两个整型变量的值。

给定下面的函数/过程:

```
Void swap(int x, int y) {
    Int TMP = x;
    X = y;
    Y = tmp;
}
```

下面调用函数的结果是什么:

```
Int n1 = 1, n2 = 2;
交换(n1、n2);
```

,为什么?

- 这个函数不能正常工作的原因在于之前讨论过的调用函数的机制。

在 c++中引用变量

引用变量是别名，即，的另一个名称
已存在变量。

```
Int n1 = 1, n2 = 2;
Int & x = n1，  & y = n2;// x = ?，  y = ?
N1 = 2, n2 = 1;// x = ?，  y = ?
X = 1. v = 2;// n1 = ?，  n2 = ?
```

这些声明中的&字符表示引用
变量。

引用必须在声明时初始化。一次
引用被变量初始化后，就不能重新赋值了
引用一个不同的变量。

现在变量名和引用名都可以是
用来指向同一个变量。

引用调用示例

下面的函数互换两个整数的值:

```
Void swap(int & x, int & y) {
    Int TMP = x;
    X = y;
    Y = tmp;
```

要交换的参数必须是可赋值对象，这对于moment表示变量，但不能是常量。

- 如果在参数声明中省略了&字符，调用这个函数将不会有任何影响参数，因为函数会交换本地副本。

问题

如果我想返回多个值怎么办?

使用参数。

引用参数

c++ 允许调用者和函数使用一种称为引用调用的技术。

c++ 通过添加 &(&) 来表示引用调用在参数名之前。单个函数通常同时具有这两个参数*值形参和引用形参，如图所示*

二次解函数:

```
Int main() {
    双 a, b, c, r1, r2;
    getCoefficients(a, b, c);
    二次解(a, b, c, r1, r2);
    printRoots (r1, r2);
    返回 0;
}

二次解(double a, double b, double c,
                        双 & r1， 双 & r2) {
    if (a == 0)误差(“系数 a 必须非零。”);
    双盘 = b * b - 4 * a * c;
    if (disc < 0)误差(“此方程无实根。”);
    双 sqrtDisc= sqrt(disc);
    r1 = (-b +sqrtDisc)/ (2 * a);
```

引用参数

引用调用有两个主要目的:

- 它创建了一种共享关系，使传递成为可能
  通过参数列表双向传递信息。
- 它通过消除 an 的复制来提高效率
  论点。当。这个考虑变得更加重要
  论证是一个很大的对象。

c++中的引用

- 在 c++ 中，引用是一种简单的引用数据类型
  功能强大，但比继承自 C 的指针类型更安全。
- 它可以被认为是现有对象的新名称，但是
  而不是它所指对象的副本。
- 从 c++程序员的角度来看，引用不是
  占用任何内存空间;来自编译器实现者的
  从这个角度来看，引用可以像
  指针(即。，一个地址)，因此它可能占用内存
  地址的空间。

我们将推迟讨论参考文献，等到有时间再讨论
更深入地理解指针。现在，只需要记住 a
Reference 是一个现有对象的新名称，请使用 call by
引用在两个方向上传递信息
参数列表。

结束