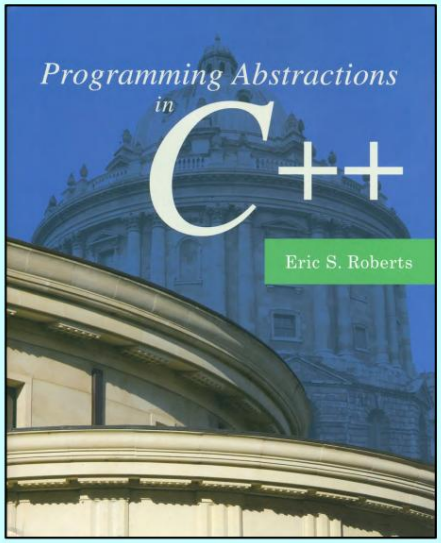
第2章

函数和库

你的图书馆就是你的天堂。

Desiderius Erasmus,费舍尔在鹿特丹的研究， 1524 年



编程范式

·编程范式是编程的“风格”或“方式”。 · 编程语言的特征之一是它支持特定的范例。有些语言可以很容易地用某些范式编写，但不能用其他范式编写。

· 一种旨在允许在多种范式中进行编程的语言称为多范式编程语言，例如C++、Python。您可以用这些语言编写程序或库，这些程序或库主要是过程性的、面向对象的或功能性的（即一些典型的范例）。

· 在大型程序中，甚至可以写入不同的部分
不同的范式。

· C++自然支持过程和面向对象的范式，通过<functional>接口支持函数范式，并通过各种外部库支持许多其他范式。

程序化编程范式

·命令式编程范例:改变程序状态的显式语句序列,指定如何实现结果。

–结构化:程序具有干净、无跳转、嵌套
控制结构。

–过程:使用过程的命令式编程
对数据进行操作。

· 过程式编程是一种编程范式,
源于结构化编程,基于过程调用的概念。

· 过程,也称为例程、子例程或
函数,简单地包含一系列要执行的计算步骤。

· 在计算机程序中,函数是执行一组特定语句的命名代码段。

函数的思想

·函数是代数中熟悉的概念,它们指定基于一个或多个未知值的数学计算,这些未知值称为参数。

· 例如,数学函数

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

指定您可以通过将参数x平方然后将结果加 1 来计算函数*f*的值。因此， f(0) 为 1， f(1) 为 2， f(2) 为 5,以此类推。

· 在 C++ 中,您可以通过以下方式实现数学函数
在函数定义中编码计算,如下所示：

```
双f（双x){
        返回 x * x + 1;
}
```

编程中的功能

- 在编程语言中,函数是被组织成独立单元并命名的代码块。

- 定义函数后,使用名称调用的行为
  该代码称为调用该函数。
- 调用程序可以将信息传递给函数使用
  论据。
- 一旦被调用,该函数接受作为参数提供的数据,完成它的工作,然后在代码中的调用点返回到调用程序。当它返回时,函数通常会将一个值传回给它的调用者。

- 本质上,您是将函数定义中的代码复制到调用它的位置,函数中的
  变量（即参数)由调用者的参数替换。

参数与参数

- 参数是函数调用中提供的参数之一的占位符,在大多数情况下就像局部变量一样。
  - 参数是函数声明和定义中的变量,是参数的占位符。
  - 自变量是调用函数时使用的表达式,是参数的实现。
  - 调用函数时,参数是您传递给函数参数（局部变量)的数据（实际值）。

使用函数的优势

- 函数允许您将特定操作的代码包含一次,然后在各种不同的上下文中（通常使用不同的参数)尽可能频繁地重用它,从而缩短程序。

- 函数允许您使用一个名称（抽象)调用整个操作序列,这对应于对函数用途的更高层次的理解,从而使程序更易于阅读。

- 函数允许您将大型程序分成更小、更易于管理的部分,从而简化程序维护。
  这个过程称为分解。您可以在不影响调用该函数的程序的情况下更新函数的实现。

- 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与从 C 派生的其他语言非常相似,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是函数的名称， parameter list 是用于保存每个参数的值的变量声明列表。

·函数体中的语句实现函数（例如算法）,至少包括一个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

- 返回布尔结果的函数称为谓词
  功能。

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

- 根本不返回值的函数通常称为过程，使用void作为结果类
  型。
- 所有函数都需要在被调用之前声明
  指定由标题行和分号组成的原型。

函数原型

·函数原型只是函数的标题行,后跟一个分号,您将其放在主函数前面,以便编译器在您实际定义函数之前知道该函数需要什么参数以及返回什么类型的值.

- 如果您总是在调用函数之前定义它们,
  不需要原型。
  - 例如,低级函数位于开头,然后是调用它们的中级函数,主程序位于最底部。

- 虽然这种策略可以节省一些原型线,但它是
  在自上而下的设计意义上违反直觉,因为最通用的功能出现在最后。
- 在某些情况下,您可能无法始终
  在调用函数之前定义函数（例如,相互递归）。

函数和算法

- 函数对编程很重要,因为它们提供了一种表达算法的结构。

- 解决特定问题的算法在效率（或复杂性)方面可能有很大差异。在选择算法时仔细考虑是有意义的,因为做出错误的选择可能会付出极高的代价。

- 接下来的几张幻灯片通过实施说明这一原则
  用于计算整数x和y的最大公约数的两种算法,它被定义为均分两者的最大整数。

蛮力方法

- 计算最大公约数的一种策略是从较小的值倒数,直到找到一个能将两者均分的值。代码如下所示：

```
int gcd(int x, int y) {
        整数猜测 = (x < y) ？  x : 是的；
        而 (x % 猜测 != 0 || y % 猜测 != 0) {
                猜测 - ;
        }
        返回猜测；
}
```

- 该算法必须在x和y为正值时终止
  因为guess的值最终会达到 1。此时， guess必须是最大公约数,因为while循环已经测试了所有较大的公约数。

- 尝试所有可能性被称为蛮力策略。

欧几里得算法

- 如果您使用蛮力方法计算 1000005 和 1000000 的最大公约数,程序将花费近一百万步来告诉您答案是 5。

- 如果您使用更好的算法,您可以更快地得到答案。亚历山大的希腊数学家欧几里德在 23 世纪前描述了一种更高效的算法,如下所示:

```
int gcd(int x, int y) {
        诠释 r = x % y;
        而（r != 0){
                x = y;
                y = r;
                r = x % y;
        }
        返回 y;
}
```

欧几里得算法的工作原理

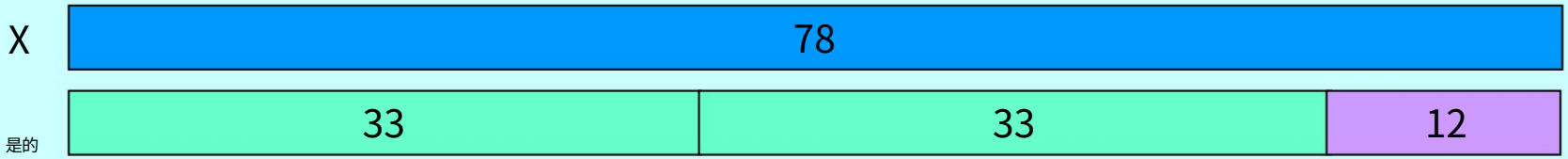
- Euclid 的伟大见解是x和y的最大公约数也必须是y的最大公约数和x的余数除以y。此外,他能够在他的《元素》第七卷中证明这个命题。

- 如果对1000005 和1000000 使用欧几里得算法,只需2步即可得到正确答案,这比蛮力所需的百万步要好得多。

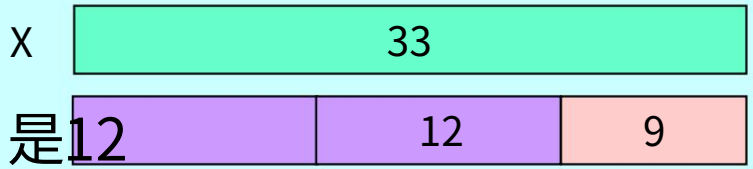
- 如果像 Euclid 那样从几何角度考虑问题,就很容易看出 Euclid 的算法是如何工作的。当x为 78 且y为 33 时,下一张幻灯片将完成计算中的步骤。

欧几里得算法的说明

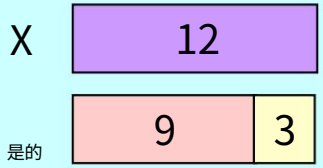
第 1 步:计算 78 除以 33 的余数:



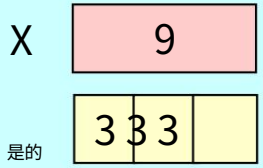
第 2 步:计算 33 除以 12 的余数:



第 3 步:计算 12 除以 9 的余数:



第 4 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,这意味着您可以
  只要可以通过查看参数的数量和类型来确定正确的版本,就可以定义几个具有相同名称的不同函数。

- 特定函数所需的参数模式（即参数的数量和类型,但不是参数名称)称为其签名。

- 使用重载的主要优点是,当应用在稍微不同的上下文中时,作为程序员,您可以更轻松地跟踪同一操作的不同函数名称。

- 原型与签名

练习:重载和签名

- 如果我们在每种情况下输入 1 或 1.0,输出是什么?

```
#include <iostream>

int add1(int x);双加1（双
x）；

int main() { 双 x;标
        准::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;


} double add1(double x) { return x + 1;

}
```

对函数的 C++ 增强

- 函数可以通过在函数原型中的变量名后面包含一个初始化器来指定可选参数。
  例如函数原型

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

  指示setMargin采用默认为 72 的可选参数。

- 定义一个以x和y坐标作为参数的过程setInitialLocation：

```
无效 setInitialLocation（双 x,双 y）；
```

·为参数提供默认值使它们成为可选的：

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

对函数的 C++ 增强

- 当您为可选参数使用默认值时,它会有所帮助
  牢记以下规则:
  - 默认值的规范（默认
    arguments)只出现在函数原型中,而不出现在函数定义中。
  - 任何可选参数都必须出现在
    参数列表。可选参数后面不能出现必需参数。

锻炼

```
无效 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。如果我们想阻止用户调用 setInitialLocation我们应该怎么做

只有一个参数（即用户可以提供所有参数或根本不提供参数）？

- 使用重载实现默认参数：

```
无效 setInitialLocation（双 x,双 y）；

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

调用函数的机制

调用函数时,会发生以下操作:

1. 调用函数在它自己的上下文中计算参数表达式。

2. 然后,C++ 将每个参数值复制到相应的参数变量中,该参数变量分配在新分配的称为堆栈帧的内存区域中。这个赋值遵循参数出现的顺序:第一个参数被复制到第一个参数变量中,依此类推。

3. C++ 然后评估函数体中的语句,使用新的堆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++ 然后丢弃被调用函数的堆栈帧并返回给调用者,从中断处继续。

组合功能

- 为了说明函数调用,本文使用了计算组合函数的函数C(n, k),即从n 个对象的集合中选择k个元素的方法数。

- 例如,假设您有一组五枚硬币:一分钱、五分钱、一角硬币、四分之一硬币和一美元:

有多少种方法可以选择两个硬币?

一分钱+镍
一分钱+一角钱
一分钱+四分之一
一分钱+一美元

镍 + 一角钱
镍 + 四分之一
镍+美元

一角钱+四分之一
一角钱+美元

季度 + 美元

共有10种方式。

组合和阶乘

- 幸运的是,数学提供了一种比计算所有方式更简单的方法来计算组合函数。组合函数的值由以下公式给出:

$$C(n,k) = \frac{\text{嗯!}}{! \quad x\ (n-k)!}$$

- 将组合分解为阶乘。从顶层看,组合函数需要一个阶乘函数。假设你已经有一个fact函数,很容易把这个公式直接变成一个 C++ 函数,如下:

```
整数组合（整数 n,整数 k){
        返回事实（n）/（事实（k)*事实（n - k））；
}
```

计算阶乘

- 数n （在数学中通常写为n！）的阶乘定义为从1 到n 的整数的乘积。因此,5!等于 120,即 1 x 2 x 3 x 4 x 5。

- 以下函数定义使用for循环来计算 阶乘函数：

```
诠释事实（诠释 n){
        整数结果 = 1;
        for (int i = 1; i <= n; i++) {
                结果 *= 我;
        }
        返回结果；
}
```

- 下一张幻灯片在一个简单的主函数的上下文中模拟组合和事实的操作。

组合计划_

```
{int main(整数n,k;int)

  << combination(int
    输入对象数量(n):    ; cout <<    输入对象数量(n):    ;
    返回事实(n)/(事实(k)*事实(结果=1;整数结果 fact(int n){int
    输入要选择的数字(k):    ; cout <<    输入要选择的数字(k):    ; 120 2 6 for
  (i>k;返回0;返回;返回结果;返回结果;cout{for (int i, k)1;cout<<i++C(n,k)

                          << 组合(n, k) << endl; << 组合(n, k) << endl;

                                        nnn            nn 结果24 120     kkiii
                                   10                       2461
}}                                                                     321 2 2
   }}}                                    3 2                            543216 4321
```

此时再次调用fact函数,程序再次调用fact函数,参数为n - k 。

1. 计算参数n和k以获得整数5和2。返回给它的调用者。

2. 为组合功能创建一个新框架。

3. 将参数变量n和k初始化为参数值。

问题

- 这个动画让你想起了什么？

锻炼

- 编写一个函数来交换两个整数变量的值。
- 给定以下功能/程序：

```
无效交换（int x,int y){int tmp = x； x = y; y
        = tmp;


}
```

以下调用函数的结果是什么：

```
整数 n1 = 1, n2 = 2;交换（n1，
n2）；
```

为什么？

- 该函数不能按预期工作的原因在于调用前面讨论的函数的机制。

C++ 中的引用变量

- 引用变量是别名,即已存在变量的另一个名称。

```
整数 n1 = 1, n2 = 2;
整数& x = n1, & y = n2; // x = ?, y = ?
n1 = 2,n2 = 1;  // x = ?, y = ?
x = 1,y = 2;  // n1 = ?, n2 = ?
```

- 这些声明中的&字符表示引用
  变量。

- 引用必须在声明时进行初始化。一旦
  引用是用变量初始化的,它不能被重新分配以引用不同的变量。

- 现在变量名和引用名都可以是
  用来指代同一个变量。

引用示例调用

- 以下函数交换两个整数的值：

```
无效交换（int & x,int & y){
        诠释 tmp = x;
        x = y;
        y = tmp;
}
```

- swap的参数必须是可赋值的对象，目前这意味着变量，但不能是常量。

- 如果您在参数声明中省略了&字符，调用此函数将不会影响调用参数，因为该函数将交换本地副本。

问题

- 如果我想返回多个值怎么办？

  使用参数。

参考参数

- C++ 允许调用者和函数使用称为引用调用的技术共享信息。

- C++ 通过在参数名称前添加与号(&)来表示引用调用。单个函数通常同时具有值参数和引用参数,如solveQuadratic函数所示：

```
int main() {
        双a,b,c,r1,r2;
        获取系数（a,b,c）；
        求解二次（a,b,c,r1,r2）；
        打印根（r1,r2）；
        返回0;
}

无效的solveQuadratic（双a,双b,双c,
                                        双 & r1, 双 & r2) {
        if (a == 0) error(    系数 a 必须不为零。    );
        double disc = b * b - 4 * a * if (disc < 0) error(    这    c;
        个方程没有实根。    );
        双 sqrtDisc = sqrt(光盘);
        r1 = (-b + sqrtDisc) / (2 * a);
        r2 = (-b - sqrtDisc) / (2 * a);
}
```

参考参数

- 引用调用有两个主要目的：
  - 它创建了一种共享关系，可以通过参数列表在两个方向上传递信息。
  - 它通过消除复制的需要来提高效率
    争论。当参数是一个大对象时，这种考虑变得更加重要。

C++ 中的参考

- 在 C++ 中,引用是一种简单的引用数据类型,它不如从 C 继承的指针类型强大但更安全。
- 可以将其视为现有对象的新名称,但不是它所指对象的副本。
- 从 C++ 程序员的角度来看,引用不占用任何内存空间。从编译器实现者的角度来看,引用可以像指针（即地址)一样实现,因此它可能会占用地址的内存空间。
- 我们将推迟对引用的讨论,直到我们对指针有了更深入的了解。现在,只需记住引用是现有对象的新名称,并使用按引用调用通过参数列表双向传递信息。

The End