函数包括函数原型；函数定义和函数调用三部分。

用于接收传递值的变量叫形参(parameter)，传递给函数的值叫实参(argument)。多个参数用(,)分隔。

将数组传递给函数：

声明，int sum\_arr(int arr[], int n); // 后面的n表示数组个数

不能将数组大小的n放到方括号里面。实际上传递的arr是一个地址，因此替换成int\* arr也是等价的。在函数原型中，int arr[]和int\* arr是等价的。P218示例，可re。为了防止函数在无意中修改了数组的内容，可在声明形参时使用关键字const，如void show\_array(const double ar[], int n);

传递数组时，可以传递数组的头地址(或指针)和数组长度，如上所示；也可以将数组头和尾的指针传递给函数，如P220的程序所示。

把const关键字用到指针pt上可以防止修改pt所指向的值，有两种方法，1. 让指针指向一个常量；2. 将指针声明为常量，防止改变指针指向的位置。第二种方法，只能将常量指针指向常量，而不能指向变量。参考P222.

const和指针 re

将二维数组作为参数给函数时，原型为：

int data[3][4]; // 二维数组格式

int sum(int (\*ar2)[4], int size); // 函数原型

其中括号必不可少，因为：

int (\*ar2)[4] // 表示一个指向由4个int组成的数组的指针

int \*ar2 [4] // 表示一个由4个指向int的指针组成的数组

另一种形式：

int sum(int ar2[][4], int size);

ar2[r][c]和\*(\*(ar2+r)+c)等价

传递字符串到函数给出的是char指针，包括：char数组；由引号括起来的字符串常量和设置为字符串的地址的char指针。不必传递字符串长度。不能返回字符串，但可以返回字符串地址。

递归

函数指针re

一般函数调用需要保存参数并跳转，但内联函数将代码与其他程序代码内联起来，而无需跳转。内联函数比一般函数稍快，但会占用更多内存。一般适用于代码量小但调用次数多的函数。内联函数需要：1. 在函数声明前加上关键字inline；2. 在函数定义前加上关键字inline。内联函数不能递归。

引用变量

引用变量是已定义变量的别名，主要用途是作为函数的形参，函数将使用原始数据而非副本。这里用&符号来定义引用变量，例子：

int rats;

int & rodents = rats; //rodents是rats的引用变量

这里的&不是地址运算符，而是类型标识符，就像int \*表示int型的指针一样，int &表示指向int的引用。rats和rodents指向相同的值和内存单元，操作一个变量改变时，另一个变量也改变。必须在说明引用变量时初始化，一旦关联，引用变量将一直效忠于它。

引用变量作为函数参数时，被调用函数可以访问调用函数的中的变量，称为按引用传递。与之功能对应的是C中的指针传递。C中常用的按值传递不能改变调用函数中的变量，参考P259的例子。

void swapr(int & a, int & b); //按引用传递

void swapp(int \* a, int \* b); //按指针传递

void swapp(int a, int b); //按值传递

引用传递会改变调用函数中变量的值，若不希望改变，则应该在函数原型和函数头前使用const：double refcube(const double &ra);。引用变量限制更加严格，不能用表达式来赋值。

若引用参数是const，则编译器在下面两种情况下生成临时变量：1)实参的类型正确，但不是左值；2)实参的类型不正确，但可以转换为正确的类型，例子参考P262的代码。左值：是可被引用的数据对象，如变量、数组元素、结构成员、引用和解除引用的指针等。非左值包括字面常量(用引号括起来的字符串除外)和包含多项的表达式。应尽可能使用const。

引用变量适合结构、类等，因为可以节省时间和内存。传统的按值传递是将返回结果复制到一个临时位置，再复制给调用函数。但引用变量是之间返回值，效率更高，参考P264的例子和之后的说明。返回引用时应避免返回函数终止时不再存在的内存单元。

经类对象传递给函数时，C++通常的做法是使用引用。参考P269代码。

定义一个接受基类引用作为参数的函数，调用时可以将派生类对象作为参数。

打开文件示例：

const char \*fn = “ep-data.txt”;

ofstream fout;

fout.open(fc);

if (!fout.is\_open()){

cout<<”Can’t open”<<fn<< “. Bye. \n”;

exit(EXIT\_FAILURE);}

P272一些文件写入的示例，更多可参考17章。

P274 8.2.7关于引用参数的使用总结。

默认参数：当函数调用中省略了实参时自动使用的一个值。通过原型设置默认值：

char \*left(const char \*str, int n=1)

其中n=1为默认值，若传递了n值，则覆盖n=1。要设置参数的默认值，必须为它右边的所有参数都设置默认值。

函数重载(函数多态)指可以有多个同名的函数，可以用函数重载来设计一系列函数—他们完成相同的工作，但使用不同的参数列表。函数重载的关键是函数的参数列表—也称为函数的特征标(function signature)，C++允许定义名称相同的函数，条件是他们的特征标不同。使用被重载的函数时，需要在函数调用中使用正确的参数类型。

函数模板，使用泛型来定义函数。如用建立一个交换模板的示例：

template <typename T>

void swap(T &a, T &b){

T temp;

temp = a;

a = b;

b = temp;}

这里的T可以是int，double等各种类型，关键词template和typename是必需的，除非用关键词class代替typename，在这种上下文中，两者是等价的。

可以像重载常规函数定义那用重载模板定义。

显式具体化

1. 对于给定的函数名，可以有非模板函数，模板函数和显式具体化模板函数以及他们的重载版本
2. 显式具体化的原型和定义应以template<>打头，并通过名称来指定类型
3. 具体化优先于常规模板，而非模板函数优先于具体化和常规模板

设job为一个结构名，以下为三种函数原型的例子：

void Swap(job &, job &); // 非模板函数原型

template <typename T>

void Swap(T &, T &); // 模板原型

template <> void Swap<job>(job &, job &);// 具体化模板原型

上式的<job>是可选的。