1，函数基础

函数的返回类型不能是数组类型或函数类型，但可以是指向数组或函数的指针。

只存在于块执行期间的对象称为自动对象。形参是一种自动对象。

对于局部变量对应的自动对象来说，如果变量本身没有初始值，就执行默认初始化。内置类型的未初始化局部变量将产生未定义的值。

定义成static的局部变量的生命周期贯穿函数调用及之后的时间。

函数只能定义一次，但可以声明多次。函数的声明可以省略参数名字。

函数应该在头文件中声明，在源文件中定义。

定义函数的源文件应该把含有函数声明的头文件包含进来，编译器负责验证函数的定义和声明是否匹配。

2，参数传递

形参初始化的机理与变量初始化一样。

如果形参是引用类型，它将绑定到对应的实参上，否则，将实参的值拷贝后赋给形参。

熟悉C语言的程序员常常使用指针类型的形参访问函数外部的对象。在C++语言中，建议使用引用类型的形参替代指针。

拷贝大的类类型对象或者容器对象比较低效，甚至有的类类型（包括IO类型）根本不支持拷贝操作。只能通过引用形参访问该类型的对象。

如果函数无需改变应用形参的值，最好将其声明为常量引用。

一个函数只能返回一个值，有时函数需要同时返回多个值，引用形参为我们返回多个结果提供了有效的途径。

当用实参初始化形参时会忽略掉顶层const，形参的顶层const被忽略掉了。当形参有顶层const时，传给它常量对象或者非常量对象都是可以的。

尽量使用常量引用。

find\_char(string str);find\_char("Hello")会报错。

数组的两个特殊性质对我们定义和使用作用在数组上的函数有影响，这两个性质分别是：不允许拷贝数组，数组使用时会被转换成指针。

当我们为函数传递一个数组时，实际上传递的是指向数组首元素的指针。

void print(const int\*)和void print(const int[])以及void print(const int[10])等价。

main可以接受命令行选项。int main(int argc,char \*argv[])

为了编写能处理不同数量实参的函数的三种方法：1，如果实参类型相同，使用initializer\_list；2，实参类型不同，使用可变参数模版；

3，使用省略符(这种功能一般只用于与C函数交互的接口程序)

编程：打印变长的错误消息

3 返回类型和return语句

不要返回局部对象的引用或指针。

如果函数返回的是列表，那么返回值类型是vector<T>。

main函数可以没有return语句。

因为数组不能拷贝，所以函数不能返回数组。不过可以返回数组的指针或引用。

要想定义一个返回数组的指针或引用的函数比较烦琐，但是有一些办法可以简化这一任务，其中最直接的方法是使用类型别名。

比如：typedef int arrT[10],using arrT = int[10]; arrT\* func(int i);

如果我们想定义一个返回数组指针的函数，则数组的维度必须跟在函数名字之后。

不使用类型别名来定义一个返回数组的函数：int (\*func(int i ))[10];

尾置返回类型 auto func(int) -> int(\*)[10];

使用decltype：int odd[]={1,3,5,7,9}; decltype(odd) \*arrPtr(int c){}

4 函数重载

一个拥有顶层const的形参无法和另一个没有顶层const的形参区分开来。

比如lookup(Phone)和lookup(const Phone) lookup(Phone\*)和lookup(Phone\* const)

底层const是可以重载的：

lookup(Account&)和lookup(const Account&) lookup(Account\*)和lookup(const Account\*)

const\_cast可以将const string&转换成string &

5 特殊用途语言特性

本节将介绍三种函数相关的语言特性：默认实参，内联函数和constexpr函数。

默认实参

默认实参，一旦某个形参被赋予了默认值，它后面的所有形参都必须有默认值。

内联函数

调用函数一般比求等价表达式的值要慢一些，内联函数可避免函数调用的开销，它会在调用点上“内联地”展开。

在函数的返回类型前面加上关键字inline，这样就可以将它声明为内联函数。

内联说明只是向编译器发出一个请求，编译器可以选择忽略这个请求。

一般来说，内联机制用于优化规模较小，流程直接，频繁调用的函数。很多编译器都不支持内联递归函数。

constexpr函数

constexpr是指能用于常量表达式的函数。该函数的返回类型及所有形参的类型都得是字面值类型，而且函数体中必须有且只有一条return语句。

执行初始化任务时，编译器把对constexpr函数的调用替换成其结果值，为了能在编译过程中随时展开，constexpr被隐式的指定为内联函数。

constexpr size\_t scale(size\_t cnt){return new\_sz() \* cnt;} scale(2)是常量表达式；int i = 2;scale(i)不是常量表达式。

constexpr函数不一定返回常量表达式。

内联函数和constexpr函数通常定义在头文件中。

调试帮助

程序可以包含一些用于调试的代码，但是这些代码只在开发程序时使用，当应用程序编写完成准备发布时，要先屏蔽这些代码。

这种方法用到两项预处理功能：assert和NDEBUG

assert是一种预处理宏，所谓预处理宏其实是一个预处理变量，它的行为类似于内联函数。

assert(expr)首先对expr求值，如果表达式为假，assert输出信息并终止程序的执行。如果表达式为真（即非0），assert什么也不做。

assert宏定义在cassert头文件中，预处理名字由预处理器而非编译器管理，因此我们可以直接使用预处理名字而无须提供using声明，使用assert而不是std::assert。

assert的行为依赖于一个名为NDEBUG的预处理变量的状态。如果定义了NDEBUG，则assert什么都不做。

#define NDEBUG加到每个源文件中,但这条语句必须放在#include <assert.h>之前.

使用\_\_func\_\_输出当前调试的函数的名字。它是一个静态数组。

预处理器还定义了另外4个对于程序测试很有用的名字：

\_\_FILE\_\_存放文件名的字符串字面值；\_\_LINE\_\_存放当前行的整型字面值；\_\_TIME\_\_存放文件编译时间的字符串字面值；\_\_DATA\_\_存放编译日期。

6 函数匹配

7 函数指针

函数的类型由它的返回类型和形参类型共同决定，与函数名无关。

要想声明一个可以指向该函数的指针，只需要用指针替换函数名即可。

函数类型bool(const string &, const string &)的指针bool (\*pf)(const string &, const string &);

当我们把函数名作为一个值使用时，该函数自动转换成指针。

pf = lengthCompare和pf = &lengthCompare等价。

我们还能直接使用指向函数的指针调用该函数，无须提前解引用指针。

pf("Hello","Hi")等价于(\*pf)("Hello","Hi")

可以直接把函数作为实参使用，此时它会自动转换成指针。

直接使用函数指针类型显得冗长而烦琐，类型别名和decltype能让我们简化使用函数指针的代码。

typedef bool Func(const string&, const string&)

typedef decltype(lengthCompare) Func2

typedef bool(\*FuncP)(const string&, const string&)

typedef decltype(lengthCompare) \*FuncP2

以上都是等价的

要想声明一个返回函数指针的函数，最简单的办法是使用类型别名：using PF = int(\*)(int\*, int);

将decltype作用于某个函数时，它返回函数类型而非指针类型。