6.1 函数基础

1.
函数定义包括返回类型，函数名，由0个或多个形式参数组成的参数列表和函数体

2. 函数调用通过调用运算符“()”来完成，该括号作用于一个表达式，表达式是函数或者指向函数的指针，括号内是用逗号隔开的实参列表。函数调用做2件事儿：

a) 用实参初始化对应的形参

b) 将控制权转移给被调函数，主调函数执行暂时被中断，被调函数开始执行

3. return语句：

a) 返回return语句中值（如果有）

b) 将控制权从被调函数转移回主调函数

4. 函数的形参列表可以为空，形参列表中的参数用逗号隔开，每个形参都必须有一个类型声明，且任意两个形参不能同名。

5. 形参的名字在函数声明时可写可不写，但是在定义时如果不写形参名，就无法使用形参；函数调用时，即使被调函数内某些形参不使用，主调函数也要为它提供对应的实参

6. 函数返回类型不能是数组或者函数，但可以是指向这二者的指针

6.1.1 局部对象

1. 形参和函数体内定义的变量统称为局部变量

2. 局部对象在程序执行第一次经过对象定义语句时初始化，直到程序结束被销毁，对象所在的函数结束执行并不会对它产生影响。在变量类型前添加关键字static可以定义局部静态对象，则该局部对象的生命周期会持续至整个程序终止

6.1.2 函数声明

1. 函数只能定义一次，但可以声明多次。函数声明也叫做函数原型，函数可以只声明无定义

2. 函数一般在头文件中声明，在源文件中定义

6.1.3 分离式编译

分离式编译允许把程序分割到几个文件中去，每个文件独立编译。每个文件产生对应的编译后的.o文件，之后编译器把对象文件链接（link）在一起形成库文件或者可执行文件

6.2 参数传递

6.2.1 值传递

1. 如果形参不是引用类型，则函数对形参做的所有操作都不会影响实参，使用指针类型的形参可以访问或修改函数外部的对象

```
void func(int *p){    *p = 10; //?p?????????    p = nullptr;
 //????p???????}
```

注意：上述尝试改变p的指向的操作实际上改变的是p的拷贝的指向，p本身指向并不会被改变；如果想要改变p的指向，需要传指针的指针

6.2.2 引用传递

1. 如果希望改变实参的值，可以使用引用形参

```
void func(int &i){    i = 10;}
```

2. 使用引用形参可以避免拷贝，拷贝类类型对象或容器对象比较低效，而且有些对象不支持拷贝操作，如果函数不需要修改引用形参的值，最好将形参声明为常量引用

6.2.3 const形参和实参

1. 实参初始化形参时，会忽略顶层const，即形参类型为const

T或者T是等价的，
因为对于值传递来说传递的是实参的副本，在函数体内不会改变实参的值

2. 当形参有顶层const时，传递给它常量对象或非常量对象都是可以的

3. 可以用非常量对象初始化一个底层const形参，但是不能用常量对象或字面值初始化普通引用或指针

4. 好习惯：如果函数内部不会改变对象，则尽量使用常量引用做形参

6.2.4 数组形参

1. 数组不能拷贝，无法以值传递的方式使用数组参数，且编译器处理数组时通常将其转换为指针，但可以把形参写成多种数组的变种：

```
void print(const int*);void print(const int[]);void print(const int[10]);//?????????10??????????????????????const int*??
```

2. 如果函数不需要对数组元素执行写操作，应该把数组形参定义成指向常量的指针

3. 形参可以是数组的引用，但此时维度是形参类型的一部分

```
void func(int &arr[10]); //error????????????????void func(int (&arr)[10]);//okint *arr[10];//????int (*arr)[10];//????
```

6.2.5 main：处理命令行选项

1. 形式：

```
int main(int argc, char *argv[]);//??int main(int argc, char **argv);//??????????????????????C???????
```

2. argv的第一个元素指向程序的名字或者一个空字符串，接下来的元素依次传递命令行提供的实参。最后一个指针之后的元素值为空

6.2.6 含有可变形参的函数

1. 两种形式：

a) 如实参类型相同，则传递一个该实参的initalizer_list类型

```
void error_msg(std::initializer_list<T> list){    for (auto
iter = list.begin(); iter != list.end(); ++iter)        std:
:cout << *iter << " " ;    std::cout << std::endl;}
```

b) 使用省略符形参传递可变数量的实参，一版只用在与C函数交换的接口程序中

2. initializer_list是一种标准库类型，定义在同名头文件中，其与vector类似，但是它的元素都是常量值。Initializer_list对象只能用花括号初始化

3. initializer_list提供的方法：

方法	含义
initializer_list<T> list;	默认初始化，空列表
initializer_list<T> list{a, b, c};	常量初始化
list2(list)	直接拷贝初始化， 注意这里并不拷贝元素，而是list2和list二者共享同一列表
list2=list	赋值拷贝，同上，元素也是二者共享
list.begin()/list.end()	返回首元素/尾后元素指针

4. 如果像initializer_list形参中传递一个值序列，必须把序列放到花括号里

```
void func(std::initializer_list<int> list);func({1, 2, 3, 4}
);
```

5. 因为initializer_list包含begin和end成员，所以可以使用范围for循环处理其中的元素

6. 省略符形参是为了便于C++程序访问某些特殊的C代码而设置的，只能用于C和C++通用的类型，大多数类类型的对象传递给省略符形参都无法正确赋值，省略符形参只能出现于形参列表的最后

6.3 返回类型和return语句

1. return语句的作用：

a) 返回表达式的值

b) 跳出当前函数，将控制权返回给调用方

6.3.1 无返回值函数

1. 没有返回值的return语句只能用在返回类型是void的函数中

2. 一个返回类型是void的函数也能使用返回表达式，不过此时return语句的expression必须是另一个返回void的函数

6.3.2 有返回值函数

1. 这种情况下return语句必须返回一个值，并且返回值的类型必须与函数的返回类型相同，或者能隐式地转换成函数的返回类型（main函数例外）

2.
在含有return语句的循环后面应该也有一条return语句，否则程序是有问题的，但是编译器却发现不了

3. 返回一个值的方式和初始化一个变量或形参的方式完全一样：返回的值是用于初始化调用点的一个临时量，该临时量就是函数调用的结果

4. 函数不应该返回局部对象的指针或引用，因为一旦函数完成，局部对象将被释放

```
const std::string &func(){    return “Hello World”; //error}
```

5. 如果函数返回指针、引用或类的对象，则可以使用函数调用的结果访问结果对象的成员

6. 调用一个返回引用的函数会得到左值，其他返回类型得到右值，返回非常量引用时，可以为该左值执行赋值操作

```
func(a, b) = 1; //func???????
```

7. 从C++11开始，函数可以返回{ }列表，该列表也是用于初始化函数调用结果的临时量。如果列表为空，临时量执行值初始化；否则返回的值由函数的返回类型决定

```
std::vector<string> func(){      // a?b??std::string??
if (a.empty())          return {}; //?????      else if (a =
= b)          return {"ssss", "aaaa"}; //??????????      els
e          return {"aaaa", a, b};}
```

8. main函数结尾如果没有return语句，编译器会隐式插入一条返回0的return语句

9. 为了使main函数的返回值与机器无关，头文件cstdlib定义了EXIT_SUCCESS和EXIT_FAILURE这两个预处理变量，分别表示执行成功和失败

10. 递归函数使用注意事项：

a)
一定有某条路径是不包含递归调用的，否则函数会一直递归下去，直到程序栈空间耗尽为止

b)
相对于循环迭代，递归效率较低；循环适合处理先行问题，而递归适合处理非线性问题

c) main函数不能调用自身

6.3.3 返回数组指针

1. 因为数组不能被拷贝，所以函数不能返回数组，但可以返回数组的指针或引用

2. 如何生命一个返回数组指针的函数，有以下几种形式：

```
T (*func(params))[N]; //??????????????????????????auto func()
-> T (*)[N]; //??????using arrt = T[N]; arrt *func(); //????
??decltype(arr) *func(); //??decltype
```

3.
注意：尾置返回类型多用于
返回类型比较复杂的场景；
decltype作用于数值时，并不会把他转换为指针，所以为了让func返回函数指针，必须在func()前面再加个*符号

6.4 函数重载

1. 同一作用域内的几个名字相同但形参列表不同的函数叫做重载函数

2. 注意：

a) 两个函数除返回类型不同，其他都相同，不构成重载

b) main不能重载

c) 重载无法区分顶层const形参，但区分底层const形参

```
int func(int i);int func(const int i); //??const??????int fu
nc(int *p);int func(int* const p); //??const??????int func(i
nt *p);int func(const int *p); //??const?????int func(int &p
);int func(const int &p); //??const?????
```

3. const_cast可以用于函数的重载。当要重载常量引用与非常量引用的版本时，在非常量引用的版本中可以通过const_cast将哈数和返回值从常量引用转换为非常量引用，以实现对非常量引用版本的调用

```
const std::string &shorterString(const std::string &s1, cons
t std::string &s2){    return s1.size() <= s2.size() ? s1 :
s2;}std::string &shorterString(std::string &s1, std::string
&s2){    auto &r = shorterString(const_cast<const std::strin
g&>(s1),    const_cast<const std::string&>(s2));    return c
onst_cast<std::string&>(r);}
```

4. 调用重载函数时有三种可能的结果:

a) 最佳匹配

b) 无匹配

c) 有多个函数与实参匹配，但每个都不是明显的最佳选择，编译器会报二义性调用

6.4.1 重载与作用域

在不同的作用域中无法重载函数名。一旦在当前作用域内找到了所需的名字，编译器就会忽略掉外层作用域中的同名实体；C++的名字查找发生在类型检查之前

6.5 特殊用途语言特性

6.5.1 默认实参

1. 设置默认值的形参必须都放在没有默认值的形参后面

2. 调用含有默认实参的函数时，可以包含该实参，也可以省略，如果想使用默认实参，只要在调用时省略该实参即可，所以调用含有默认实参的函数，传入的实参数量小于等于函数实际使用的实参数量

3. 在给定作用域内，一个形参只能被赋予一次默认实参，后续声明只能为之前没有默认值的形参添加默认实参，而且需要保证该形参右侧的所有形参必须有默认值

```
std::string screen(int, int, char = ‘ ’);std::string screen(int, int, char = ‘*’); //error????std::string screen(int = 20, int = 20, char); //ok
```

4. 通常在函数声明中指定默认实参，并将声明放在合适的头文件中

6.5.2 内联函数和constexpr函数

1. 定义内联函数时需要在函数的返回类型前添加关键字inline，内联函数会在每个调用点上展开，省去函数调用的时间开销

2. 内联函数适用于规模小/流程直接/频繁调用的函数，但是内联函数中不允许有循环和switch语句，否则即使声明了inline关键字，也会被编译为普通函数

3. constexpr函数是指能用于常量表达式的函数，他的返回类型及所有形参的类型都必须是字面值类型。C++11规定constexpr函数必须有且仅有一条return语句，但是该规定在C++14中被删除

```
constexpr int new_sz(){     return 42;}constexpr int foo = new_sz();//constexpr??????????????????????????????constexpr size_t scale(size_t cnt){    return new_sz() * cnt;}int arr[scale(2)]; // ok: scale(2) ????????int i = 2;int a2[scale(i)
```

```
]; // error: scale(i)???????
```

4. constexpr函数被隐式地指定为内联函数

5.
与其他函数不同，
inline函数和constexpr函数可以多次定
义
。因为在编译过程中，编译器需要函数的定义来随时展开函数。对于某个给定的inline函数或constexpr函数，它的多个定义必须完全一致，所以通常将他们定义在头文件中

6.5.3 调式帮助

1. assert预处理宏，包含在头文件<cassert>中，用法为assert(expr);如果expr为假，那么assert输出信息并终止程序，否则什么也不做

2. NDEBUG预处理变量：如果定义了NDEBUG，则assert什么也不做，默认NDEBUG没有定义

3. 对程序调试比较有用的5个名字：

```
__func__; //?????????__FILE__; //?????????__LINE__; //????__TIME__; //??????????__DATA__; //??????????
```

6.6 函数匹配

1. 有多个重载函数时，它们都称为候选函数；在候选函数集合中，实参数量与形参数量相同且类型相同或能转换为形参的类型的函数称为可行函数；编译期，编译器会从可行函数中找到最佳匹配，如果不存在最佳匹配，编译器会报二义性错误。

2. 调用重载函数尽量避免强制类型转换

6.6.1 实参类型转换

1. 实参到形参类型的转换分为以下几个等级，按优先级排序如下：

a) 从数组类型或函数类型转化为对应的指针类型，添加顶层const或删除顶层const

b) 通过const转换实现的匹配

c) 通过类型提升实现的匹配

d) 通过算术类型转换或指针转换实现的匹配

e) 通过类类型转换实现的匹配

2. 像上文介绍的，函数重载会忽略顶层const，但不忽视底层const，所以两个函数由于形参是否是底层const而发生重载时，编译器会根据实参是否具有底层const属性来精确匹配对应的函数版本

6.7 函数指针

1. 函数指针是指向函数类型的指针，想声明一个可以指向某种函数的指针，只需要用指针替换函数名称即可

```
bool compare(const std::string &, const std::string &);bool
(*pf)( const std::string &, const std::string &);pf = compar
e; //pf?????pf = &compare; //?????????????bool b1 = pf(“hel
lo”, “goodbye”);bool b2 = (*pf)(“hello”, “goodbye”);//??????
??????
```

2. 可以把函数的形参定义成指向函数的指针。调用时允许直接把函数名当作实参使用，它会自动转换成指针，函数指针也可以指向0或nullptr;

```
//?????????????????????void func(const std::string &s1, cons
t std::string &s2, bool pf(const std::string &, const std::s
tring &));//?????????????????????void func(const st::string &s
1, const std::string &s2, bool (*pf)(const std::string &, co
nst std::string &));
```

3. 定义函数别名：

```
typedef bool func(const std::string &, const std::string &);
typedef decltype(compare) func2;using func3 = bool(const std
::string &, const std::string &);//?????????????????????????
?????????????decltype?????????????????????*???typedef bool
 (*func)(const std::string &, const std::string &);typedef d
```

```
ecltype(compare) *func2;using func3 = bool (*)(const std::st
ring &, const std::string &);//??????????????????????????
```

4. 函数不能作为形参，但函数指针可以，函数名作为形参会被编译器自动转换为函数指针类型

5. 函数返回函数指针：

```
//????bool (*f(int))(double);//f?????????int?????????????????
??????double??????bool????//??????auto f1(int)->boo (*)(doub
le);//??decltypebool func(double);decltype(func) * f1(int);/
/????using pf = bool(*)(double);pf f1(int);
```