Notes for `C++ Primer`

Jingze Xing

2021 年 4 月 14 日

目录

1 变量和基本类型

(1). **声明 (declaration)**：声称存在一个变量、函数或别处定义的类型

(2). **声明符 (declarator)**：包括被定义的名字和类型修饰符（可以没有）

(3). **定义 (definition)**：为某一特定类型的变量申请存储空间

(4). 一条声明语句由一个**基本数据类型 (base type)** 和紧随其后的一个声明符列表组成

(5). **静态类型 (statically typed) 语言**：在编译阶段检查类型 (type checking) 的语言

(6). **作用域 (scope)**

使得名字具有特定含义（指向特定实体）的范围，始于名字的声明语句，以声明语句所在域的末端为结束，大多数作用域以花括号分隔

作用域声明某个名字后，仍允许在其**内层作用域 (inner scope)** 重新定义这个名字

(7). **复合类型 (compound type)**：基于其他类型定义的类型，如引用与指针

(8). **引用 (reference)**

为对象取得另外一个名字，通过将声明符写成 `&d` 定义引用类型

定义引用时程序将引用与其初始值**绑定 (bind)** 在一起

引用并非对象，只是一个别名

(9). **指针 (pointer)**

指针存放某个对象的地址，通过**取地址符 (&)** 获取对象地址，通过**解引用符 (*)** 来访问该对象

符号 `*` 与 `&` 既能做运算符，又能作为声明的一部分出现

可以构造指向指针的指针，如 `**p`

(10). **const 限定符**

const 一般修饰它前面的变量，与之构成基本数据类型，如 `int const *b = &a`（等价于 `const int *b = &a`），则 `const int` 是一个基本类型，`b` 是指向常量的指针

若 const 变量的初始值不是一个常数表达式,如 `const int a = func();` 则 `func()` 会在主程序之前被调用，且在此阶段 a 为默认初值

const“对象”若是引用，则相当于“常量引用”，如 `const int &b = 2 * a;`

(a). **顶层 const(top-level const)**

表示指针本身是个常量，如 `int* const ptr = &a` 中，`ptr` 是常量，但可以通过 `*ptr` 来修改所指对象

在**赋值/初始化**中，右值的顶层 `const` 可以被忽略

(b). **底层 const(low-level const)**

表示指针所指的对象是个常量，如 `const int *pa = &a`，表示 `pa` 是指向常量 `a` 的指针，`a` 必须是常量

一个常量的地址是带有底层 `const` 的指针，如 `const int a = 42`，`&a` 是 `const int*` 类型

(11). **constexpr 变量**

将变量声明为 constexpr 类型以便由编译器来验证变量的值是否是一个常量表达式

定义指针时，constexpr 仅能限制指针，不能限制指针所指的对象，即指针本身是常量

(12). **别名声明 (alias declaration)**

如 `using AB = Angel_Beats;`

`using ptr = int*; ptr a, b;` 则 `a,b` 均为指针类型

(13). **auto 类型说明符**：让编译器分析表达式所属的类型，必须有初始值

const 一般会忽略顶层 const，不忽略底层 const

(14). **decltype 类型指示符**

返回对象的数据类型，如 `decltype(f()) sum = f();`

decltype 返回的类型包含顶层 const 和引用，如果一个表达式的值可以作为**左值 (lvalue)**，则该式将会对 decltype 返回一个引用类型

(15). **范围 for(range for)**

for (*declaration*: *expression*) *statement*; 每次迭代中 *declaration* 部分的变量会**初始化**为 *expression* 序列中的下一个元素值；故若 *declaration* 定义的是一个引用，则可以修改 *expression* 中的值

2 字符串、向量和数组

2.1 数组

(1). 数组的维数 `[d]` 属于数组类型的一部分
如 `int (*Parray)[10]` 是指向大小为 10 的 int 数组的指针
而 `int *array[10]` 则是含有 10 个 int 指针的数组

(2). 在很多用到数组名字的地方，编译器会将其替换成一个指向首元素的指针

(3). `begin(arr),end(arr)` 分别返回指向首元素的指针与指向尾元素的下一位置的指针

2.2 迭代器

(1). `iterator` 可读可写，`const_iterator` 只可读不可写

(2). `cbegin(),cend()` 返回 `const_iterator` 类型

(3). 仅有 `vector,string` 的迭代器可以相减（返回值可正可负）、相互比较大小，或与整数进行加减运算

2.3 std::string

(1). `istream& getline(istream& is, string& str, char delim)`
读入直至遇到 delim 为止，丢弃 delim，此后读入从 delim 后继续，若没有给定 delim，则默认为换行符。字符串的生成方式类似于 `push_back`

(2). `string operator+ (...,...)`
字符串字面值不是 string 类型，表达式中必须有一个是 string 类型，string 类型可以与 char 相加

(3). `const char* c_str() const;`
返回一个 C 风格的字符串，原字符串被修改或删除时先前的指针会失效

2.4 std::vector

由类模板 (`vector`) 创建类 (`vector<T>`) 的过程称为**实例化 (instantiation)**

(1). `vector <T> v{e1,e2,e3};`
向量的列表初始化必须用花括号

(2). `vector <T> v(int,T);`
规定向量长度与填充的值，若不提供值，则填充默认值

(3). `vector <T> v(a + 1, a + n + 1);`
用两指针间的元素 `a[1]`,$\cdots$,`a[n]` 填充向量

3 表达式