2018-08-28

2.2.变量  
 (1)初始化不是复制， 初始化的含义是创建变量时赋予其一个初始值，而赋值的含义是把对象的当前值擦出，而以一个新值来代替可。

1. 初始化的方式，例如定义一个名为units\_sold的int变量并初始化为0：
   * 1. Int units\_sold = 0
     2. Int units\_sold = {0}
     3. Int units\_sold{0}
     4. Int units\_sold(0)

花括号初始化变量的方式称为列表初始化。列表初始化一个重要的特点就是：如果我们使用列表初始化且初始化值存在丢失信息的风险，则编译器将报错。

1. 变量声明和定义的区别：
   * 1. ”声明“使得名字为程序所知，一个文件如果想要使用别处定义的名字则必须包含对那个名字的声明，不开辟内存空间。如果向声明一个变量而非定义它，则在变量名前面添加关键字”extern”，但是如果包含显示的初始化声明则失效，声明变为定义。
     2. “定义”负责创建与名字关联的实体，开辟内存空间。
     3. 变量只能被定义一次，但是可以被声明很多次。
     4. 当你第一次使用变量的时候再定义它。
2. 复合类型，指的是基于其他类型定义的类型（指针和引用）：

2018-08-30

2.3.指针和引用

2.3.1.指针：

1. 指向常量的指针(常指针)：const int \*p
2. 指针本身是常量（常量指针）：int \* const p

2.3.2.引用：

1. 引用本身不是对象，声明时必须初始化应用某个实例对象。类中的引用成员变量必须用列表初始化
2. 引用本身不是对象所以没有指向引用的指针，但是有引用指针的引用声明：int \*&p，从右向左读p是引用，引用int指针。

2.4.const限定符

(1)const 与宏不一样在于，const是运行时确定值形如：const int i = get\_size()(运行时确定)合法，而宏是预编译器阶段确定其值。另一方面宏没有类型检查，宏是简单的替换。而const则有类型检查，const变量在运行时会将所有变量替换为实际的值。

（2）const特性只会在执行改变const变量的操作时发挥作用。将const常量赋值给非const变量是合法的。

（3）默认情况下const变量只在当前文件有效，在多个文件中定义同名const变量时，这些变量是互相独立的。特别的，当在const前添加extern修饰符时，该const变量是全局的，可在多个文件中共享，且变量初始值不能为常量表达式。

（4）const引用：对常量的引用，形如：const int &r = var，引用的变量不嫩被修改。

（5）引用类型必须与其所引用对象的类型一致。因为如果引用的类型不一致而发生类型转换（其实是生成一个额外的临时变量），引用一个临时变量不是我们想要的，所以编译器将这种行为定为非法行为。

（6）可以常量引用一个非常量的值。

2018.9.3

2.4.3顶层const和底层const（p58）

1. 底层const只与指针和引用等符合类型相关的，通常指的是指针（引用）的对象是常量。
2. 顶层const不影响拷贝操作，而底层则影响。

3.1命名空间的using声明

1. 头文件不应该包含using声明，因为头文件会被拷贝到所有引用它的文件中去，重复包含会引起名字冲突。

3.2.2string对象上的操作

（1）cin从输入流中读取字符串时会忽略空白（空格符、换行符、制表符等）

（2）当希望在最终读入的字符串中保留输入时的空格符，则使用getlin()函数代替>>，gitline()函数参数是一个输入流和一个string对象,函数从给定的输入流中读入内容，直到遇到换行符（注意换行符被读入，但是会被丢弃），函数返回一个cin对象。

（3）string中的size()函数返回一个无符号类型string::size\_type，千万不要将无符号类型和int混用！！！

（4）标准模板库允许把字符字面值和字符串字面值转换成string对象，所以在需要string对象的地方也可以使用字符字面值或者字符串字面值替代。

2018.09.06

3.5.1数组.char \*p和char p[]区别:

1. char \*p = “Hello World”.p指向的是一个保存在静态数据区,是个常量,所以不能通过指针p修改字符串的内容,当多个指针指向”Hello World”时,它们指向的是同一个内容.事实上,该语句应该写成:const char \*p = “Hello World”
2. Char p[] = “Hello World” p指向的内容是刚刚在栈中开辟的内存,并且可以随时通过指针p修改该内存的内容.

2018.09.07

3.51数组.

(1)不能使用数组初始化另一个数组,同时也不能将数字直接赋值给另一个数组

3.5.3指针和数组

1. 在大多数表达式中,使用数组类型的对象其实是使用一个指向该数组首元素的指针.
2. c标准模板库中的cstring头文件(在c中为string.h).包含strlen(), strcmp(), strcat(), strcpy()等函数,只能用于操作c风格的字符串(以空格符”\0”结尾的字符串).
3. 常用的转义字符“\0”表示空格符, “\n”表示换行符, “\t”表示制表符.
4. String对象可以用”<”, “>”, “=”, “+”等符号比较字符串, 但是c风格字符串不可用, 如果哟使用比较的是子字符串对应的指针的大小.
5. String对象中的c\_str()函数返回一个c风格的字符串.
6. Int \*p[4]表示的是p是一个数组,纬度为4的且保存int\*的数组, int (\*p)[4]表示p是一个指针, 指向纬度为4的int数组.
7. 标准模板库中的begin(), end()函数用于获取数组首尾指针

3.6多维数组

1. 使用范围for循环时, 最外层循环控制变量必须声明为引用类型, 避免数组被自动转为指针.(当程序使用多维数组的名字时, 也会自动转换为指向数组首元的指针) 不然无法通过编译.如果需要修改数组中变量的值时,内层控制变量也要声明为引用类型.

2018.09.08

4.2算术运算符

1. 取余运算(取模运算)只能用于整数类型.
2. (-m)/n和m/(-n都表示-(m/n), m%(-n)等于m%n, (-m)%n等于-(m%n)
3. Io运算符满足左结合律, 所以”cout << c < 4”错误,(cout << 4)优先执行并返回一个cout对象, 此时cout < 4语句错误.
4. 当数组被用作decltype关键字的参数, 或者作为取址, sizeof()以及typid等运算符的运算对象时,数组名不会被转换为指针类型.

2018.09.09

6.2.4数组形参

1. 数组不能被拷贝,并且传入的实参数组名会被转为指针,所以传数组实际传入的是指针.
2. 管理数组实参的有效:
   * 1. 办法是传入数组的头指针和尾指针, 通常是使用标准模板库中的begin()和end()方法. 形如void print(begin(arr), end(arr)).
     2. 或者在传入首元指针的同时传入数组的大小.
     3. 数组以引用的形式传入, 形如void print (int (&arr)[10]), 该方式会限制传入的数组大小一定为10.
     4. 数组以引用传入的方式, 并且不限制大小的方法后续介绍.
3. Mian()函数可以接受外部传进来的参数, 此时main函数应该声明为main(int argc, char \*argv[])或者main(int args, char \*\*argv). args保存参数的数量, argv是个c风格字符串的数组, argv[0]保存这程序名,从argv[1]开始保存传入的参数, 并且数组最后一个位置保存着0表示结束.
4. 可变形参initializer\_list<T>, 可以像访问vector那样使用迭代器访问initalizer\_list中对象, 也可以使用范围for循环, 但是initializer\_list中的每个对象都是const对象, 即无法被修改.
5. Initalizer\_list使用时传入的时参应该用{}阔起来.
6. 函数的返回类型决定了函数调用是否是左值.调用一个返回引用的函数得到的是左值, 其他类型是右值. 也就是说可以像使用其他左值那样来使用返回引用的函数的调用.例如:

Char &get\_val(string &str, string::size\_type ix)

{

Return str[ix];

}

Int main()

{

String s(“adf”);

get\_val(s) = ‘A’’; //完全合法, 左值赋值

Return 0;

}

1. C++11规定,函数可以返回花括号包围的值的列表,即可以在return语句后以{}形式返回一个vector<T>列表.
2. Main函数返回值为0时表示执行成功,其他值表示执行失败.其他非0值含义与机器有关. cstdlib库中提供了两个变量表示执行成功或者失败.

Int main()

{

If (some\_failure)

Return EXIT\_FAILURE;

Else

;Return EXIT\_SUCCESS;

}

1. 返回数组的函数声明形式:int (\*func(int i)[10]);

6.4函数的重载

1. 一个拥有顶层const的形参和一个没有顶层形参的函数声明是无法区分的,不能以此来重载函数.例如:Record lookup(Phone), Recond lookup(const Phone).
2. 另一方面,如果形参是某种类型的指针或者引用,则通过区分其指向的是常量对象还是非常量对象可以实现函数重载.例如:Recond lookup(Account \*)和Recond lookup(const Account \*)可以区分.
3. 在许多函数声明中使用到常量引用的方式确保传入的参数不需要拷贝和修改,但是当需要返回一个可以被修改的对象时,必须用到const\_cast<>进行类型转换. 通常的做法是保留原本的常量引用函数, 新建一个传入非常量引用的形参的函数调用之前的函数, 传入的形参可以通过const\_cast<const String&> 将非常量实参转为常量实参,函数执行完后返回的常量引用值可以通过const\_cast<String &>去掉const属性.

6.5特殊用途语言特性:

1. 在声明默认实参时,一旦某个形参被赋予了默认值,则它后面的所有形参都必须有默认值.
2. 内联函数(inline)可以避免函数调用的开销, 代码在编译过程中会自动在函数调用点中将代码展开, 常用于规模比较小,流程直接,调用频繁的函数.
3. Constexpr函数, 执行初始化任务时,编译器自动把constexpr函数转为函数的返回值,所以constexpr函数被隐性指定为内联函数, 并且函数的所有形参的类型都得是字面值类型, 而且函数体中必须只有一条return语句.
4. 利用assert预处理宏调试,

2018/09/10

7.3类的其他特性

1. 非const实例对象即可以调用非const函数，也可以调用const函数。而const实例对象只能调用const函数，不能调用非const函数。

其中的原理：类的成员函数调用时会隐式传入一个this指针，const成员函数中this指针被声明为”const Type \*”，即当const实例对象调用时传入const指针，调用成功，当非const实例对象调用时传入非const指针，此时非const指针可自动转换为const指针，调用成功。而当const实例对象调用非const成员函数时，传入的const指针无法转换为非const指针，所以调用失败。

1. 仅声明类而暂时不去定义它，这种声明称为前向声明。此时可以定义指向该类型的指针、引用或声明（不能定义）。一旦一个类名字出现过，它就被认为是被声明过了，所以，类允许包含指向它自身的指针或者引用。
2. 如果一个类指定了友元类，则友元类的成员函数可以访问此类包含非公有成员在内的所有成员。友元关系不具有传递性。

2018.9.11

7.5构造函数再探

1. 如果没有在构造函数的初始化列表中显式地初始化成员（在构造函数体中对成员赋值不是初始化），则该成员将在构造函数体之前执行默认初始化。
2. 如果类的成员是引用或者const时，必须在构造函数中列表初始化（初始化唯一机会）。
3. 构造函数初始值列表只说明用于初始化成员的值，而不限定初始化的具体顺序。成员初始化顺序只与成员在类中定义中出现的顺序有关。
4. 如果一个构造函数为所有参数都提供了默认参数，则它实际上也就定义了默认的构造函数。
5. 如果构造函数只接受一个实参，则它实际上定义了转换为此类类型的隐式转换机制，有时我们把这种构造函数称之为转换构造函数。假如class A中定义了接受string的构造函数和接受iostream的构造函数，则在需要使用class A的地方，我们可以使用string或者iostream作为代替。发生隐式转换的另一种情况是当我们执行拷贝形式的初始化时（使用=）。如果需要抑制这种隐式的转换机制，则在只含单个实参的构造函数前添加关键字”explicit”。此时使用exclicit声明的构造函数只能以直接初始化的形式使用，不能用于拷贝形式的初始化过程（=）。若要强行进行类型转换，可通过显示调用构造（即：sum(A(string))，显式将string类型转为A类类型）。
6. 因为静态数据成员函数不属于任何一个对象，所以它们并不是在创建类的对象时被定义，所以必须在类的外部定义和初始化静态成员，每个静态成员只能被定义一次。
7. 静态数据成员可以作为默认实参。

7.6补充（“>>”、”<<”运算符mmn重载为什么声明为友元函数）：

（1）为什么类中重载”<<”、”>>”等运算符时需要声明为友元函数？

类中重载二元运算符有两种方式，一种是在类中声明，另一种是在类外声明。在类外声明重载二元运算符时，需要传入两个参数，左边参数是类本身，右边是参与运算的另一个参数，比如重载+operator,+(obj1， obj2)，这种方法不太符合使用习惯；第二种是在类中声明重载运算符，只需要传入参数与运算的对象参数，比如重载+operator，obj1 + obj2，比较符合习惯。但是重载”>>”(或者“<<”)时，cin（或者cout）作为作则运算对象且不是类本身，要声明为第二种方式，则需要访问类中包含私有成员在内的数据成员，则需要声明为友元函数。

8.1IO类

1. Io对象无拷贝或赋值，不能将形参或者返回类型设置为流类型。IO对象通常以引用的方式传递，因为读写一个IO会改变其状态，所以传递和返回的引用不能是const。
2. 如果想要每次输出操作后都刷新缓冲区，则可以使用unitbuf操作符，例如：cout<<unitbuf，使用cout<<nounitbuf回到正常的缓冲方式。
3. 输入流对象cin中的strm::iostate值反映当前输入流的情况，通过cin.rdstate()函数可以返回strm::iostate值，通过cin.setstate()设置当前cin对象状态，cin.clear()将当前输入流状态错误位全部清零。通过cin.good()查看当前流是否正常。

8.1文件输入输出

1. 当一个fstream对象离开它的作用域后，与之相关的文件会自动关闭，自动调用close()函数。
2. 与ifstream关联的文件默认以ifstream::in模式打开，与ofstream关联文件默认以ofstream::out模式打开，与iofstream关联文件默认以iofstream::in、iofstream::out模式打开。以out模式打开的文件默认以截断（trunc）方式存储。

8.3 string流

(1)sstream中的str()函数返回string流对象保存的string的拷贝，str(s)则将string s拷贝到string流对象中。

9.2顺序容器

1. 为了创建一个容器为另一个容器的拷贝，两个容器的类型及其元素类型必须匹配。不过，当传递迭代器参数拷贝一个范围时，就不要求容器类型是相同的了。
2. 如果元素类型是内置类型或者是具有默认构造函数的类类型，可以只为构造函数提供一个容器大小参数。如果元素类型没有默认构造函数，除了大小参数外，还必须提供一个显式的元素初始值。

2018.9.25

1. **vector表示的二维数组(m \* n)初始化方式：**
   1. 方法一

Vector<vector<int>> arr(m, vector<int>(n));

* 1. 方法二

vector<vector<int>> arr(m);

for (int i = 0; i < m; ++i)

{

arr[i].resize(n);

}

1. 产生随机数

需要的头文件：

#include<stdlib.h>

#include<time.h>

//根据time()产生不相同的随机数种子，提供给rand()

srand((unsigned)time(NULL));

//产生随机数

Rand();

//产生一定范围内的随机数

产生[a, b)之间的随机数：(rand() % (b - a)) + a;

产生[a, b]之间的随机数：(rand() % (b - a) + a);

产生(a, b]之间的随机数：(rand() % (b - a)) + a + 1;

通用公式：a + rand() % n; 其中a是起始值，n是整数的范围;

取得a到b之间的随机整数的另一种方式：a + (int)b \* rand() / (RAND\_MAX + 1);

获取0~1的浮点数：rand() / double(RAND\_MAX);

2018.10.10 中英文混合字符串处理

1. 读取和输出中文字符串：

locale china(“chs”);

wcin.imbue(china);

wcout.imbue(china);

wstring s;

wchar\_t wc = L”a”;

getline(wcin, s, wc);

wcout << s << endl;

1. 多字节和宽字符的转换：

#include <stdlib.h>

wchar\_t wstr[100]=L"wstr";

char str[100];

wcstombs(str,wstr,100);//宽字符转多字节，size\_t wcstombs(char \*dest, const wchar\_t \*src, size\_t n);

mbstowcs(wstr,str,100);//多字节转宽字符

1. C++中本质上有两种getline函数，一种在头文件<istream>中，是istream类的成员函数。一种在头文件<string>中，是普通函数。

一、在<istream>中的getline函数有两种重载形式：

istream& getline (char\* s, streamsize n );

istream& getline (char\* s, streamsize n, char delim );

作用是从istream中读取至多n个字符保存在s对应的数组中。即使还没读够n个字符，如果遇到换行符'\n'（第一种形式）或delim（第二种形式），则读取终止，'\n'或delim都不会被保存进s对应的数组中。

二、在<string>中的getline函数有四种重载形式：

istream& getline (istream& is, string& str, char delim);istream& getline (istream&& is, string& str, char delim);istream& getline (istream& is, string& str);

istream& getline (istream&& is, string& str);

//用法和上一种类似，不过要读取的istream是作为参数is传进函数的。读取的字符串保存在string类型的str中。

2018.10.13

（1）在函数体内声明[1] char \*str=”abc”;和[2] charstr[]={‘a’,’b’,’c’}有什么区别？

 [1][2]中str变量都分配在栈上

 [1]中str指向常量存储区的字符串”abc”，其中字符串末尾会补0

 [2]中str数组的内容存储于栈空间，数组大小为3，字符串不会补0、

2018.10.14

1、类的基本思想是**数据抽象**和**封装**。

* 1. 数据抽象是一种依赖接口和实现分离的编程技术。类的结构包含用户所能执行的所有操作；类的实现则包括类的数据成员、负责接口实现的函数题以及定义各种私有函数。
  2. 封装：实现类的接口和实现的分离。封装后的类隐藏了它的实现的细节，也就是说，类的用户只能使用接口，而不能访问实现部分。

1. **内置类型变量**是否自动初始化取决于变量定义的位置。函数体外定义的变量初始成0；函数体内定义的变量不进行自动初始化。除了用作赋值操作的左操作数，其他任何使用未初始化变量的行为都是未定义的，不要依赖未定义行为。
2. 总结：未被初始化的枚举值的值默认将比其前面的枚举值大1。
3. 类中的const函数返回值通常也应声明未const。
4. 类中如果存在重载的同名const和no-const函数。可以先实现const函数,然后在no-const函数体中通过强制类型转换等操作调用const版本同名函数。具体方法如下：

使用const\_cast<Type>(static\_cast<const T>(\*this)(Type))，其中**static\_cast<>**将当前对象\*this强制转为const类型，以此调用const版本函数，再通过**const\_cast<>**将函数返回的const类型强制转为no-const类型。

1. 类中成员变量可以声明未mutable，这样即使在const成员函数中也可以对该变量修改。
2. 对象的成员变量的初始化动作发生在进入构造函数本体之前。
3. 定义了调用操作符“（）”的类的对象称为“**函数对象”。**

2018.10.15

1. 某年编译单元内的某个non-loca static对象的初始化动作使用了另一编译单元的某个non-loca static对象，图所使用的这个对象可能尚未被初始化，因为c++对“定义在不同编译单元内的non-local static对象”的初始化次序并没有明确的定义。

解决办法：

将每一个non-local static对象搬运到自己的专属函数内（该对象在函数内被声明为static）。这些函数返回一个reference指向它所含的对象。然后用户调用这些函数，而不是直接指涉这些对象。**换句话说就是，non-local static对象被local static对象替换了。**因为c++保证，函数内的local static对象会在“该函数被调用期间”，首次遇上该对象的定义式时被初始化。

2018.10.26（c++ primer 第九章）

1. forward\_list是单向链表，性能几乎等同于手写的最好的单项链表。但是forward\_list迭代器不支持递减运算符（--）和反向迭代器，同时forward\_list中也没有size()成员函数。
2. 与内置类型一样，标准库中的**array**的大小也是类型的一部分，指定一个array时，要指定元素的类型，还要指定容器大小，例如：array<int, 12>。Array不支持普通容器的构造函数，因为这些构造函数或是显示、或是隐式指定容器的大小，与array容量固定前提相悖。Array支持拷贝和赋值操作，即array<int, 10> ia{1, 2, 3}; array<int, 10> ib = ia; //正确。因为array中保存了容量大小。Array不支持拷贝一个迭代器对指定的元素范围。Array也不支持assign、花括号包围的值列表进行赋值，以及push\_back操作。
3. 内置类型数组不支持拷贝和赋值，即int a[] = {1, 2, 3}; int b[] = a; //错误 b = a; //错误。因为内置类型的数组设计之初只保留了指向数组首元的指针，并没有保存大小。即c在处理数组时通常是退化为指针处理。数组的大小未知，所以无法进行拷贝和赋值操作。
4. 标准库中的swap函数，（除array外）在交换两个容器内容时只交换了两个容器内部的数据结构，并未对容器拷贝删除和插入操作，所以很快。Vector<string> svec1(10); vector<string> svec2(10); swap(svec1, svec2);。对除string使用swap函数后，原先指向容器的指针，引用和迭代器都不会失效，但是string使用swap后会失效。对array来说执行完swap操作后虽然原有的指针、引用和迭代器都不变，但他们指向的值是交换过后的。
5. 除了无序的关联容器（仅支持==和!=）外，每个容器都支持关系运算符（>、>=、<、 <=）。比较过程是进行元素的逐对比较，即按字典顺序。

2018.10.27

1、迭代器使得算法不依赖于容器，但依赖于元素的类型。