1. const

1).const 分为底层coast,和顶层const,,

其中顶层const 表示,自身是一个常量.

常见写法: const type T=(初始化), type \*const T=&(),对于赋值行为,常常会忽略顶层 const,即可以将一个非常量,赋值给一个常量,但反过来不可以.

底层const 表示其指向的是一个常量

常见写法:const type \*T=&(),const type &T=(),同样需要在定义的同时进行初始化

2).auto,和decltype 对于含有顶层const 变量的推断是不同的,auto会忽略掉顶层cosnt,

Decltype不会

const int i=6;

auto(i) j // j为int类型可以不进行初始化

decltype(i) j //error :j为const 类型,必须在定义的时候进行初始化

3) const 一般不支持跨文件的操作,一般情况下,const 修饰的变量只作用与自己所在的文件,在不同的文件中定义相同的const 变量,通常代表不同的含义,如果想要在不同的文件中共享需要在,声明,和定义中都添加 extern

//t.h

#ifndef \_t\_h\_

#define \_t\_h\_

extern const int i=5;//定义

#endif

//a.cpp

#include”t.h”

extern const int i; //声明

int main ()

{

std::cout<<i<<std::endl;

return 0;

}

1. 定义与声明的区别

一个变量可以有多个声明,但只有一个定义,声明的意思是告诉,编译器,我在某个地方定义了一个变量,你自己去找,所有声明可以有许多个,定义一般伴随着内存的申请.

1. 类内初始值

c++11,允许直接在类内对变量进行初始化

1. 关于c++容器中内定义的size(),函数的返回值,是size\_t,size\_type等,他是一个无符号的整形,我们应该避免在一个表达式中同时出现,带符号数,和无符号数,避免出现一些意想不到的事情,例如:”fdggs”.size()<n,假如n是一个负数,该表达式,基本上是一个true的表达式,因为,一个负数,会被转换成一个较大的无符号数.
2. 一些容器使用的注意事项
3. 在一个范围for中,不要尝试改变容器的大小,这对于for循环来说,是危险的
4. 在有迭代器使用的情况下,也不要改变容器的大小,这会使得迭代器失效.
5. 理解复杂数组的声明

int arr[10];

int (\*p)[10]=&arr;//指向含有10个整数数组的指针

int (&p)[10]=arr;//指向一个含有10个整数的数组的引用

注意:要想理解一个数组的声明,最好,从数组的名字开始从内往外阅读

1. 多维数组

严格来说,c++并不存在多维数组,通常所属的多维数组,其实是数组的数组

当一个数组的元素还是数组的时候通常,可以采用两个,维度来表示他,一个维度是它自身的大小,另一个维度表示其所指的元素的大小.

int ia[2][3]

int (&raa)[4]=ia[1] //绑定ia第二个元素,也就是第二个四个元素的数组

访问多维数组:

for(auto &row:ia)

for(auto &l:row)

{

}

使用auto &是因为,c++将数组转变成指针

for(auto p=ia;p!=ia+2;++p)

for(auto q=\*p;q!=\*p+3;++q)

{

}

for(auto p=begin(ia);ia!=end(ia);++p)

for(auto q=begin(\*p);q!=end(\*p);++q)

{  
}

传递多维数组

void print (int (\*m)[10],int r);

void print (int m[][10],int r);

//不要忘记后面的维度

1. 求值顺序

c++虽然规定运算符的优先级和结合律,但是并没有规定.运算对象按照什么样的顺序进行求值

如:int i=0;

std::cout<<i<<’ ‘<<++i<<std::endl;

也许你期望的是 0 1;

但在devc++ 5.5.3中测试为 1 1;

这种没有规定的行为,完全取决与编译器做出的选择,应该,避免使用.

1. 除与余

c++规定取整,为直接切除小数点后面,

对于求余,m%n,符号有m 决定

m%(-n)=m%n , (-m)%n=-(m%n) , (-m%(-n)=-(m%n);

1. sizeof

两种形式

sizeof(type);

sizeof expr;

1. c++提供的显示的类型转换

1).T=static\_cats<type> (expr)

任何具有明确定义,且不含底层const,的都可以使用static\_cast<>

2).const\_cast<>(),可以改变对象的底层const 一般用在重载函数中

eg:

const string& shortstring (cosnt string &s1,const string &s2)

{

return s1.size()<=s2.size()?s1:s2;

}

对于这个函数,不管,你传变量,是不是常量,它都会返回,常量,我们希望,当我们传入的是,非常量时,它可以返回一个非常量

string &shortstring (string &s1,string &s2)

{

auto &r=shortstring (const\_cast<const string &>(s1),const\_cast<const string &> (s2));

return const\_cast<string &>(r);

}

对于一个非常量,我们使用这样的转换,显然是合理,合法的.

1. dynamic\_cast<>()一般在类之间的继承时,用来转换
2. reinterpret\_cast<>()

通常时运算符对象的位模式的低层次重新解释,不推荐使用.

1. switch内部变量的定义

在switch分支中定义变量有时,会报错,这是因为,因为有的分支可能并不会进行访问,c++不允许,跨过变量的初始化直接跳转到,该变量作用域的另一个位置.详解看p163;