1. 什么时候编译器会合成默认的移动构造函数或者移动赋值运算符？

只有当我们未定义任何版本的拷贝控制成员时，并且所有的成员都是可移动构造或者移动赋值时，编译器为我们合成默认的移动构造函数或者移动赋值运算符。

1. 编译器什么时候会为我们合成默认的构造函数？

class A{

int I;

int \*p;

}

Int main(){

A a;

Cout <<a.i<<endl;

}

报错：使用了未初始化的变量a

因为类A只有内置的类型和复合类型，所以编译器并没有为我们合成默认构造函数。

编译器只有在需要的时候为我们合成默认的构造函数。“需要”指的是编译器需要而不是程序需要。以下两个认识错误：

* 1. 任何类如果没有定义构造函数，编译器都会为我们合成构造函数
  2. 合成的默认构造函数会为我们执行成员变量的初始化。（合成的构造函数不会为我们初始化内置和复合类型的数据成员）

在以下情况下被认为是“编译器需要”：

1. 调用类变量成员或者基类的默认构造函数（尽管如此，默认构造函数也不会对内置类型和复合类型初始化）
2. 含有虚函数表或者虚基类

# C++ Primer

## 运算符重载

1. 重载的运算符特点
   1. 不能改变其含义
   2. 不能创造新的运算符
   3. 必须含有一个类类型
   4. 优先级和结合性不变
2. 不能被重载的运算符

. \* sizeof ：： ?:

1. 不应该被重载的运算符

逗号（，）、取址符（&）、或运算符（||）、与运算符（&&）

1. 必须重载为成员函数的运算符

= 、 ->、（）、[]、new、delete

1. 应该被重载为成员函数的运算符

复合运算符+=，会改变对象状态的运算符

1. 应该被重载为友员函数的运算符

对称性运算符（+、-、\*等）

1. 输出运算符和输入运算符
   1. << ：两个参数（ostream &, const T& t）,非常量引用，常量引用；注意：不要添加格式输出符，比如换行
   2. >>：两个参数（istream&, T& t）,非常量引用，非常量引用；注意：必须进行流状态判断
   3. 必须是非成员变量
2. 相等运算符 ==和比较运算符<
   1. 定义了==的类还需要定义！=（使用==定义！=）；
   2. 当定义 < 号时，必须保证 == 和 < 的含义相同；即 当==表示类的各个成员都相等时，< 的含义应表示各个成员 < .
3. 赋值运算符 =
   1. 除了拷贝赋值和移动赋值，还可以处理等号右边是其他类型的情况，比如初始化列表，初始化列表不需要考虑自身赋值情况。
   2. 一般使用复合运算符+=定义=
4. 下标运算符 [ ]
   1. 两个版本：返回引用的非常量版本
   2. 返回常量的const版本

## const #define inline static

1. static和全局变量：

|  |  |
| --- | --- |
| 全局变量 | 全局变量在整个工程内可见，但不一定能用，想要使用需要加入extern |
| 静态局部变量 | 在整个程序存在，但是只在函数体可见  未赋初值，自动赋值为0 |
| 静态全局变量 | 作用域只是其源文件，不是整个工程 |

总结，局部变为静态，改变其生存期；把全局变为静态，改变其作用域。

|  |
| --- |
| 使用静态对象需要注意的问题：   1. 多线程 2. 静态对象的析构函数是在main函数退出，或者exit时调用析构函数，析构函数不能调用exit，否则无限循环。 3. 静态全局变量是内部链接的（const在C中是外部链接的，在c++中是内部链接的，要想外部链接，需要使用extern） |

1. #define const inline

|  |  |
| --- | --- |
| #define和const | #define是简单的文本替换，没有类型概念；const修饰变量，有类型。 |
| #define和inline | #define预处理阶段展开；inline编译器控制，类型检查 |

## 模板

1. 模板特例化

## 函数指针

## m\_map

## 多线程