# 嵌套命名空间

在一个已存在的命名空间中再次添加命名空间，这是被允许的。这样构建的命名空间我们称之为嵌套命名空间。

调用时需要从最外层逐层向内进行关联调用

# 内联命名空间

借助关键字inline（放在namespace之前）可以构建内联命名空间。主要目的，可以在调用空间内容时，省略空间名称，直接进行关联调用。

**内联命名空间的最主要作用是剥离了空间命名的复杂层次，解决了空间污染问题，并且还隐藏了过深的嵌套关系。**

格式

namespace ZQ

{

inline namespace ZZ //内联命名空间

{

int H = 10;

}

}

**注意：**

1. 任何明空空间均可以使用inline关键字进行描述。但是我们不建议将inline关键字放到全局命名空间上。这将导致命名空间意义丢失，造成空间命名污染。
2. 内联空间中的内容，禁止和外层空间内容相同。这将导致调用的二义性。
3. 内联命名空间中，可以继续存放内联命名空间，调用时省略空间名称。
4. 内联命名空间中可以存放普通命名空间，调用时只需要省略内联空间名称，即可完成调用。
5. 内联命名空间只是降低了命名空间中的层级调用关系，并没有解决大问题。
6. 内联命名空间命名需要与上层空间命名不重叠

# 未命名命名空间

直接通过关键字namespace进行构建，不提供空间名称。未命名命名空间仅在特定的文件内部有效，其他作用范围不会跨越多个文件。如果为未命名空间在全局作用域中，则容易产生二义性，如果在嵌套在其他命名空间中，则可以通过外层空间名称访问。

**未命名命名空间可以放在任何命名空间中。**

# Static关键字

静态关键字

Static关键字可以修饰函数，也可以用来修饰变量

### 静态全局变量

将全局变量之前加入static关键字，则构建出来了静态全局变量。静态全局变量构建在静态内存区域中。静态全局变量只作用域当前CPP文件中。如果静态全局在头文件中声明（**定义也需要在头文件中进行**），则引入该头文件的CPP文件均具有一个该静态变量（**引入头文件的CPP，每个都有一个该变量，并且互不干扰**）

静态全局变量特点：

1. 该变量在静态数据区域分配内存
2. 没有被初始化的静态全局变量会被程序自动初始化为0（指针为空）
3. 静态全局变量在声明的CPP文件中可见，在文件外不可见
4. 不同CPP中允许有相同名称的静态全局变量

### 静态全局常量

全局常量之前加入关键字static即为静态全局常量

所有常量都需要声明后初始化，确认常量值，静态全局常量也必须遵循此规则

**不同的CPP中允许有相同的静态全局常量**

### 静态局部变量

在局部变量之前加入static关键字，即声明了静态局部变量

特点：

1. 该变量在静态存储区域分配内存
2. 当程序执行到该处时被初始化，即以后的函数调用不再进行初始化
3. 静态局部变量一般在声明处初始化，如果没有显式初始化，则被程序自动初始化为0
4. 它始终驻留在内存区域中，直到程序执行结束被销毁
5. 静态局部变量的作用域为局部作用域
6. 在不同的CPP中可以命名相同的静态局部变量

### 静态全局函数

在全局函数之前加入static关键字，构建成为静态全局函数。

注意：

1. 如果声明在头文件中，定义必须写在头文件中（定义在CPP中无法进行关联）。使用需要引入声明头文件
2. 声明在CPP中，允许在不同的CPP中存在命名相同的静态全局函数（文件域不同，所以不会冲突）
3. 静态全局函数只能操作全局类型数据，如全局变量，静态全局函数，全局函数，静态全局变量，不能操作级别低于自己的内容。

### 静态成员变量

在类内部定义的静态变量称之为静态成员变量，受访问修饰符约束。禁止在变量声明的地方初始化，禁止在初始化参数列表中初始化，需要在CPP文件中进行初始化，不初始化无法使用。

声明格式：

static int Color;

定义：

int Hero::Color = 0;

使用：

类名：：变量名

Hero::Color = 100;

特点：

1. 在类中只有一份，不会随着对象的创建而增加
2. 不能在初始化参数列表中初始化
3. 使用前必须进行初始化
4. 禁止在类声明的地方初始化
5. 存储在静态内容空间中
6. 受访问修饰符约束

### 静态成员常量

在类内部的static修饰的成员常量，被称为静态成员常量。需要在声明时给予初始化，也可以在外部进行初始化。

外部初始化需要注意格式问题，不要忘了const关键字：

声明：

static const int Color;

外部定义：

const int Hero::Color = 10;

特点：

1. 在类中只有一份，不会随着对象创建而增加
2. 不能在初始化参数列表中初始化
3. 声明完成后需要进行初始化赋值，也可以到外部进行定义赋值
4. 受访问修饰符影响

### 静态成员函数

普通的成语函数之前加上static关键字，即成为了静态成员函数。定义方式不变。

访问无需使用类实例对象进行访问，直接通过类名加双冒号进行访问即可

静态成员函数不属于类的任何一个对象，属于类本身。

访问

类名：：函数名（参数）

特点：

1. 可以在头文件中直接定义，也可以在cpp文件中定义。
2. 只存在一份函数体，并且不属于任何类的实例对象，没有this指针（不属于某一个对象）。
3. 只能操作类的静态成员变量，静态成员函数，静态成员常量。
4. 访问受访问修饰符约束

# 动态数组

创建动态数组：



主要借助了关键字new将数组内容构建在堆上，并且是一块连续的内存空间

**释放：**

delete[] Array;

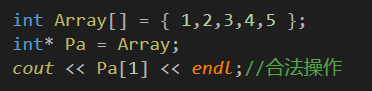
一次释放所有数据

**使用：**

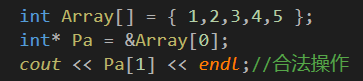
和普通数组一样进行使用

Array[0] = 100;

特殊操作



或是



第一数组名称储存的是数组第一个元素的地址

特殊操作[]

对于数组使用[]操作等同于：\*（array+n）

Data[N]相当于\*(Data + N)

注意点：

1. 数组名称储存的是数组中第一个元素的地址
2. 使用运算符[]进行数组名称操作时，相当于在进行数组地址偏移，偏移长度相当于数组的数据类型大小，然后进行解引用操作。
3. 动态数组无法通过sizeof获取数组长度
4. 动态数组无法在确定大小后，再次调整大小
5. 动态数组切记使用完成后使用delete[]进行释放

##### 柔性数组

存放在结构体中最后一个成员数组，并且数组本身没有进行大小标记，我们称之为柔性数组

struct Box

{

int Size;

int data[];

};

**新标准C++中已经取消了使用，但是目前还可以使用，编译过程会提示扩展内容警告，所以不建议使用**

借助动态内存申请关键字malloc进行创建结构体的大小，然后再使用结构体内的数组进行数据装填

Box\* b = (Box\*)*malloc*(sizeof(Box) + sizeof(int) \* 5);

释放

*free*(box);

问题：

1. 柔性数组在新标准中已经不再使用，不建议使用
2. 柔性数组具备数组基本特性，不做越界检查（在结构体内部，数据是连续的，数组对于下标使用负值，将向前进行数据获取）