# Lab3

**任务3**

**Typescript的模块与C++的内外部连接**

Typescript中某一模块，如果这个模块被另一个模块连接，那么本模块就要在类前面写export表明要被其他模块使用。而import则将其他模块连接进来。模块是在其自身的作用域里执行，并不是在全局作用域，这意味着定义在模块里面的变量、函数和类等在模块外部是不可见的，除非明确地使用 export 导出它们。类似地，我们必须通过 import 导入其他模块导出的变量、函数、类等。如果将Typescript模块间的关联看作连接的话（如果lab3的题意我没理解错的话），与C++进行内部连接使用static静态变量或者函数，而外部连接通过extern关键字来连接。

import id = require("./Identify")

export class Id\_number implements id.Identify{

public print(){

var message:string = "Id\_number is printed."

console.log(message);

}

}

**static int** variable = 0;  
**extern int** variable2;

**Typescript与C++的命名空间**

**ts的命名空间内可以有打印语句和相关操作，这和C++是不同的，因而在命名空间内的操作更加灵活；可以使用export进行嵌套定义； C++中啰嗦地使用using namespace而ts不使用。ts的命名空间的使用可以用.来进行，而C++需要使用::。**

**两者都能起到防止命名冲突的作用。**

namespace Information{

var index:number = 2

console.log("Id\_number information "+index)

}

namespace other{

var index:number = 20;

console.log("Id\_number other "+index)

}

**namespace** Fuc\_k{  
 **int** a = 1;  
 **int** b = 2;  
}

综上：

3

**（1）TypeScript中的模块，命名空间和c++ 中的命名空间作用很相似，都是为了解决重名问题。命名空间定义了标识符的可见范围，一个标识符可在多个名字空间中定义，它在不同名字空间中的含义是互不相干的。这样，在一个新的名字空间中可定义任何标识符，它们不会与任何已有的标识符发生冲突，因为已有的定义都处于其他名字空间中。**

**（2）较为不同的是TypeScript中的命名空间使用起来更灵活，命名空间只是全局命名空间中的JavaScript对象。这使命名空间成为一个非常简单的构造。它们可以跨多个文件，并且可以使用--outFile连接。有趣的是，正因为模块和命名空间的灵活性，TypeScript中没有引入内部连接，外部连接之类的概念。但这会造成全局命名空间污染，很难识别组件依赖性，尤其是在大型应用程序中。**

**（3）相比于C++，ts命名空间、模块可以包含代码和声明。主要区别在于模块声明了它们的依赖关系，显然这一点在大型应用程序中，具有长期模块化和可维护性优势。**