9. & 10. Inline Functions Name Control

Hu Sikang
skhu@163.com

School of Computer
Beijing Institute of Technology

9. Inline Functions

- How to improve the efficiency ?
- In C, one of the ways to preserve efficiency is through the use of ***preprocessor macros***. The preprocessor *replaces* all macro calls directly with the macro code.
- In C++, there are two problems with preprocessor macros:
  - A macro can bury difficult-to-find bugs.
  - The preprocessor macros cannot be used as class member functions.
- To retain the efficiency of the preprocessor macro, but to add the safety and class scoping of true functions, C++ has the ***inline function***.

9.1 Preprocessor pitfalls

```
#include <iostream>
using namespace std;

#define  f(x)  x*x
int main( ) {
    int x(2);
    cout << f(x) << endl;
    cout << f(x+1) << endl;
    return 0;
}
```

#define   f(x)   (x)*(x)

Replace with : 2*2

Replace with : 2+1*2+1

Output:

4

5

9.2 Inline Functions

- When a function has several lines code but may be called frequently, we can use ***inline*** to save time and improve efficiency.
- *An* ***inline*** function is a true function, which is expanded in place, like a preprocessor macro, so the overhead of the function call is eliminated.
- You should (almost) never use macros, only inline functions.

9.2.1 Inline Functions

Note:

- *Inline function definition must be appeared before its called.*

- The body of inline function don’t include exception handling.
- The body of inline function don’t be recursive.

9.2.2 Inlines inside classes

- The “**inline**” keyword is not necessary inside a class definition.
- Any function you define inside a class definition is *automatically* an inline function.

```
// Inlines inside classes
#include <iostream>
#include <string>
using namespace std;
class Point  {
    int i, j, k;
 public:
    Point( ) : i(0),  j(0),  k(0)  { }
    Point(int ii, int jj, int kk) : i(a),  j(b),  k(c)  {   }
    void print(const string& msg = "") const
    {
        if(msg.size() != 0)  cout << msg << endl;
        cout << i << ", " << j << ", " << k << endl;
    }
};
```

inline

Initilization List is used for initializing the member data

```
// Inlines outside classes
#include <iostream>
#include <string>
using namespace std;
class Point {
    int i, j, k;
 public:
    Point( );
    Point(int ii, int jj, int kk);
    void print(const string msg = "") const;
};

inline Point::Point( ): i(0), j(0), k(0) {  }
inline Point::Point(int ii, int jj, int kk): i(a), j(b), k(c) {   }
inline void  Point::print(const string& msg) const {
    if(msg.size() != 0) cout << msg << endl;
     cout << i << ", " << j << ", " << k << endl;
}
```

The definition of an **inline** is placed outside the class to keep the interface clean, using the **inline** keyword.

9.3 Hidden activities in constructors & destructors

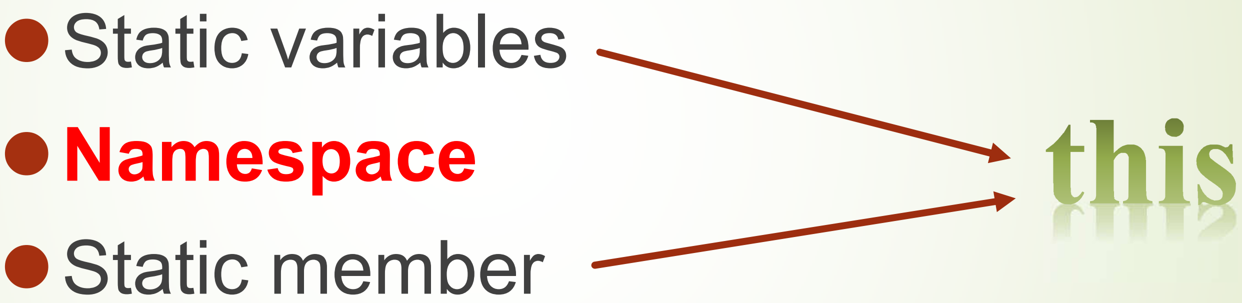
```
class Member {
    int i;
public:
    Member(int x = 0)  : i(x)   {  }
    ~Member() {   cout << "~Member, i = " << i << endl;   }
};
class WithMembers {
    Member q, r, s;            // Have constructors?
    int j;
public:
    WithMembers(int a, int b)  :  q(a), r(b)  { j = a; }
    ~WithMembers() { cout << "~WithMembers" << endl;  }
};
```

```
int main()
{
    WithMembers  wm(2, 5);
     return 0;
}
```

What’s the outputs?

WithMembers(int a, int b) : r(b), q(a);

Name Control



- Static variables
- **Namespace**
- Static member

this

10. Namespaces

- Although names can be nested inside classes, the names of global functions, global variables, and classes are still in a single global name space.
- In a large project, lack of control over the global name space can cause problems.
- You can subdivide the global name space into more manageable pieces using the *namespace* feature of C++.

10.1 Creating a namespace

```
//MyLib.cpp
namespace MyLib
{
   // members
}

int main()
{   return 0;   }
```

Differences from class:

- It can only appear at global scope, or nested within another namespace.
- **“;”** is not necessary after the closing brace.
- The name MyLib can be used in multiple header.
- The name can be *aliased* to another name:

  namespace **Lib = MyLib;**
- You cannot create an instance of a namespace.

10.2 Scope resolution

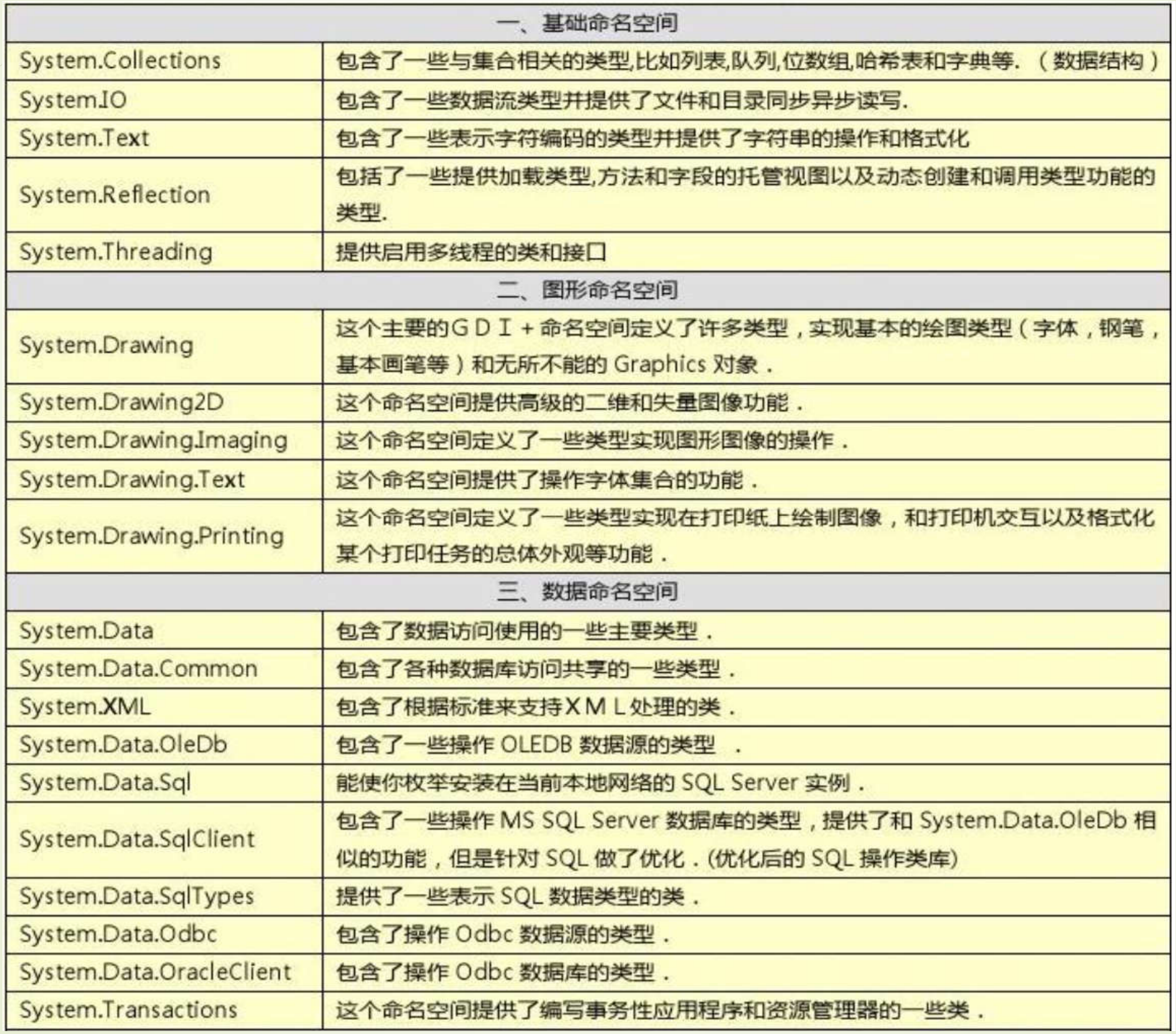
```
// ScopeResolution.cpp
namespace DB
{
    class SQL
    {
     static int i;
    public:
     void Value(int) { }
    };
    class EXCEL;
    void GetDBType( );
}

int DB::SQL::i = 9;
```

```
class DB:: EXCEL
{
    int u, v, w;
public:
    EXCEL (int i);
    int Value ();
};
DB::EXCEL::EXCEL(int i) { u=v=w=i; }
int DB::EXCEL::Value () { return w; }
void DB::GetDBType()
{
     DB::SQL  obj;    // object
     obj. Value(1);
}
int main()  {  DB::GetDBType();  return 0;}
```

10.3 Using directive

```
namespace calculator  {
          double Add(double x, double y) { return x + y; }
          void Print(double x) { cout << x << endl; }
          class Shape  {   };
}
calculator :: Shape   S1;          // Define object with namespace
using namespace calculator;    // Using Directive
int main( )  {
          Shape   S2;
          double a, b;
          cin >> a >> b;
          double = Add(a, b));
          return 0;
}
```

Namespace in .Net Framework



一、基础命名空间	
System.Collections	包含了一些与集合相关的类型,比如列表,队列,位数组,哈希表和字典等. （数据结构）
System.IO	包含了一些数据流类型并提供了文件和目录同步异步读写.
System.Text	包含了一些表示字符编码的类型并提供了字符串的操作和格式化
System.Reflection	包括了一些提供加载类型,方法和字段的托管视图以及动态创建和调用类型功能的类型.
System.Threading	提供启用多线程的类和接口
二、图形命名空间	
System.Drawing	这个主要的ＧＤＩ＋命名空间定义了许多类型，实现基本的绘图类型（字体，钢笔，基本画笔等）和无所不能的 Graphics 对象．
System.Drawing2D	这个命名空间提供高级的二维和失量图像功能．
System.Drawing.Imaging	这个命名空间定义了一些类型实现图形图像的操作．
System.Drawing.Text	这个命名空间提供了操作字体集合的功能．
System.Drawing.Printing	这个命名空间定义了一些类型实现在打印纸上绘制图像，和打印机交互以及格式化某个打印任务的总体外观等功能．
三、数据命名空间	
System.Data	包含了数据访问使用的一些主要类型．
System.Data.Common	包含了各种数据库访问共享的一些类型．
System.XML	包含了根据标准来支持ＸＭＬ处理的类．
System.Data.OleDb	包含了一些操作 OLEDB 数据源的类型　．
System.Data.Sql	能使你枚举安装在当前本地网络的 SQL Server 实例．
System.Data.SqlClient	包含了一些操作 MS SQL Server 数据库的类型，提供了和 System.Data.OleDb 相似的功能，但是针对 SQL 做了优化．(优化后的 SQL 操作类库)
System.Data.SqlTypes	提供了一些表示 SQL 数据类型的类．
System.Data.Odbc	包含了操作 Odbc 数据源的类型．
System.Data.OracleClient	包含了操作 Odbc 数据库的类型．
System.Transactions	这个命名空间提供了编写事务性应用程序和资源管理器的一些类．

Summary

- Inline Function
- Inline function in the class
- Inline function VS. #define
- Name control: namespace