June, 3rd, 2012

1. 函数指针的声明

Typedef void (\_\_stdcall \*MyFunc)(parameters);

MyFunc pMyFunc;

\_\_stdcall calling concention is used to call Win32 API functions. Functions that use this calling convention require a function prototype.

typedef is a language construct that associates a keyword to a type.  
You use it the same way you would use the initial type, for instance

typedef int myinteger;

typedef char \*mystring;

typedef void (\*myfunc)();

using them like

myinteger i; // is equivalent to int i;

mystring s; // is the same as char \*s;

myfunc f; // compile equally as void (\*f)();

As you can see, you could just replace the typedefed keyword with its definition given above.

The difficulty lies in the pointer to functions syntax and readability in C and C++, and the typedef can improve the readability of such declarations. However, the syntax is appropriate, since functions - unlike other simpler types - may have a return value and parameters, thus the need to use more keywords and parentheses to write the function declaration.

The readability may start to be really tricky with pointers to functions arrays, and some other even more indirect flavors.

To answer your three questions

* **Why is typedef used?** To ease the reading of the code - especially for pointers to functions, or structure names.
* **The syntax looks odd (in the pointer to function declaration)** That syntax is not obvious to read, at least when beginning. Using a typedef declaration instead eases the reading
* **Is a function pointer created to store the memory address of a function?** Yes, a function pointer stores the address of a function. This has nothing to do with the typedef construct which only ease the writing/reading of a program ; the compiler just expands the typedef definition before compiling the actual code.

Exemple:

typedef int (\*t\_somefunc)(int,int);

int square(int u, int v) {

return u\*v;

}

t\_somefunc afunc = &square;

...

int x2 = (\*afunc)(123, 456); // call square() to calculate 123\*456

1. Difference between STRUCT and UNION

With a union, you are only supposed to use one of the elements, because they are all stored at the same spot. This makes it useful when you want to store something that could be one of several types.

A struct, on the other hand, has a separate memory location for each of its elements and they all can be used at once.

union foo {

int a; // can't use both a and b at once

char b;

} foo;

struct bar {

int a; // can use both a and b simultaneously

char b;

} bar;

union foo x;

x.a = 3; // OK

x.b = 'c'; // NO! this affects the value of x.a!

struct bar y;

y.a = 3; // OK

y.b = 'c'; // OK

1. C#

“IF” condition must evaluate a Boolean expression to either a true or false value.

“FOREACH” loop is used to iterate through the items in a list. It operates on arrays or collections such as ArrayList, which can be found in the System.Collections namespace.

foreach (<type> <iteration variable> in <list>) { <statements> }

The iteration vatiable is an identifier that you choose, which could be anything but should be meaningful.

While iterating through the items of a list with a foreach loop, the list is read-only.

using System;  
  
class ForEachLoop  
{  
    public static void Main()  
    {  
        string[] names = {"Cheryl", "Joe", "Matt", "Robert"};  
  
        foreach (string person in names)  
        {  
            Console.WriteLine("{0} ", person);  
        }  
    }  
}

1. C# -- Methods

There are 4 kinds of parameters a C# method can handle: out, ref, params, and value.

If a method takes a ref parameter, it means that a reference to the parameter is copied to the method. This reference still refers to the same object on the heap as the original reference used in the caller’s argument. This means any changed to the local reference’s object also changes the caller reference’s object. The code can’t change the reference, but it can changes to the object being referenced.

Out parameters are only passed back to the calling function. Because of definite assignment rules, you cannot use this variable until it has a valid value assigned. Once assigned and the program returns, the value of the out parameter will be copied into the caller’s argument variable. You must assign a value to an out parameter before your method returns.

Params parameter lets you define a method that can accept a variable number of arguments. It must be a single dimension or jagged array. The params parameter is considered an input only parameter and any changes affect the local copy only.

1. C# -- Class

In C#, there are two types of class members, instance and static.

Instance class members belong to a specific occurrence of a class. Every time you declare an object of a certain class, you create a new instance of that class.

If a class member is static, you can access it simply by using the syntax <classname>.<static class member>.

You must call static class members through their class name and not their instance name. This means that you don’t need to instantiate a class to use its static members. There is only ever one copy of s static class member. A good use of static members is when there is a function to be performed and no intermediate state is required, such as math calculations.

Another type of constructor is the static constructor. Use static constructor to initialize static fields in a class. A static constructor is called before an instance of a class is created, before a static member is called, and before the static constructor of a derived class.

1. C# -- Inheritance

The base class is specified by adding a colon, “:”, after the derived class identifier and then specifying the base class name.

C# supports single class inheritance only. Therefore, you can specify only one base class to inherit from. However, it does allow multiple interface inheritance. Because base classes are automatically instantiated before derived classes, the parent class’s constructor executed before the child class constructor.

1. C# -- Polymorphism

…..

**Listing 9-1. A Base Class With a Virtual Method: DrawingObject.cs**

using System;  
  
public class DrawingObject  
{  
    public virtual void Draw()  
    {  
        Console.WriteLine("I'm just a generic drawing object.");  
    }  
}

##### Listing 9-2. Derived Classes With Override Methods: Line.cs, Circle.cs, and Square.cs

using System;  
  
public class Line : DrawingObject  
{  
    public override void Draw()  
    {  
        Console.WriteLine("I'm a Line.");  
    }  
}  
  
public class Circle : DrawingObject  
{  
    public override void Draw()  
    {  
        Console.WriteLine("I'm a Circle.");  
    }  
}  
  
public class Square : DrawingObject  
{  
    public override void Draw()  
    {  
        Console.WriteLine("I'm a Square.");  
    }  
}

The override modifier allows a method to override the virtual method of its base class at run-time. The override will happen only if the class is referenced through a base class reference. Overriding methods must have the same signature, name and parameters, as the virtual base class method it is overriding.