在C++中，struct关键字定义和class关键字定义的唯一区别是：struct关键字默认的访问级别为public，而class关键字默认的访问级别为private。

在C++程序开发中，一般情况下把类的定义和其成员函数的实现分开，分别放在头文件和源文件中。

类的声明：class A;

变量的声明：extern int n;

函数的声明：void func();

类的作用域与可见性：

每个类都定义了自己的作用域，两个类具有不同的作用域。

类的作用域指：类定义和相应的成员函数定义范围。注意：成员函数的返回类型不一定在类的作用域中。例：

程序class\_test1

头文件：

#ifndef A\_H

#define A\_H

#include <string>

class A

{

public:

A();

~A();

private:

typedef std::string::size\_type Index;

public:

// typedef std::string::size\_type Index;

Index func();

private:

Index cursor\_;

};

#endif

源文件：

#include "a.h"

A::A()

{

}

A::~A()

{

}

A::Index A::func()

{

return cursor\_;

}

this指针：

在类的成员函数（static静态成员函数除外）中都具有一个附加的this指针。一般并不显示使用this指针。this指针的最常用的使用如下：

例：程序class\_test2

头文件：

#ifndef B\_H

#define B\_H

class B

{

public:

B();

public:

B& SetHeight(int height);

B& SetWidth(int width);

private:

int height;

int width;

};

#endif // B\_H

源文件：

#include "b.h"

B::B()

{

}

B& B::SetHeight(int height)

{

this->height = height;

return \*this;

}

B& B::SetWidth(int width)

{

this->width = width; // 通过this指针确定赋值运算符左边的width为类定义的，而右边的width为函数的形参

return \*this; // 函数返回调用该函数的类对象的引用。

}

B b;

b.SetHeight(30).SetWidth(40); // 形成一个链式操作

对于普通的非const成员函数，this指针是一个const指针（指针常量），即this指针指向的值可以改变（可以继续调用非const成员函数），但指针本身不能改变，this指针存储的地址为调用该函数的对象的地址。对于const成员函数，this指针是一个指向const类对象的const指针（只能继续调用const成员函数），即this指针指向的对象不能改变，指针本身也不能改变。

类中的const成员变量和const成员函数：

头文件：

#ifndef C\_H

#define C\_H

class C

{

public:

C();

~C();

public:

int GetHeight() const;

void PrintValue();

C& SetHeight(int height);

private:

int height;

const int width;

// const int width = 0; // error：不能在这里进行初始化

};

// const int n = 0; // ok，可以在头文件中初始化const变量，const变量默认被看成一个局部变量，每一个包含了该头文件的源文件都有一个const变量n，其值全部是相同的。

#endif // C\_H

源文件：

#include "c.h"

#include <iostream>

C::C():width(10)

{

// width = 10; // error，const成员变量必须在构造函数的初始化列表中初始化

}

C::~C()

{

}

int C::GetHeight() const // 这里的const修饰符不能去掉

{

// height = 100; //error： const函数不能改变类的成员变量

// PrintValue(); //error：const函数不能调用类的非const成员函数

return height;

}

void C::PrintValue()

{

// GetHeight(); // ok，非const成员函数可以调用const成员函数

std::cout << "height = " << height << " width = " << width << std::endl;

}

C& C::SetHeight(int height)

{

this->height = height;

return \*this;

}

C c1;

c1.SetHeight(100);

c1.GetHeight(); // ok，非const对象可以调用const成员函数

c1.PrintValue();

const C c2;

// c2.SetHeight(300); // error，const对象只能调用const成员函数

c2.GetHeight(); // ok，可以调用const成员函数

在C++中，只有构造函数和析构函数从不定义为常量成员函数，但可被常量对象调用（被自动调用）。

基于const的重载：

头文件：

#ifndef D\_H

#define D\_H

class D

{

public:

D& Display();

const D& Display() const; // 这样可以构成函数的重载

void func();

};

#endif // D\_H

源文件：

#include "d.h"

#include <iostream>

D& D::Display()

{

std::cout << "Display" << std::endl;

return \*this;

}

const D& D::Display() const

{

std::cout << "const Display" << std::endl;

return \*this;

}

void D::func()

{

}

D d;

d.Display().func(); // 调用的是非const的Display函数

const D d1;

d1.Display(); // 调用的是const的Display函数

// d1.Display().func(); // error，返回的this指针是一个指向const对象的const指针，不能调用非const成员函数

以上关于const对象和const成员函数的限制，如果希望突破此限制，可以使用关键字mutable来实现。

<http://blog.csdn.net/wuliming_sc/article/details/3717017>