构造(创建)对象有很多种复杂的情况。

不同情况又对应着不同的对象调用。

何时调用何种构造函数，请见下面的示例。建议你自己运行一下。

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

class C  {
public:
    C(int i) {          //带参构造函数
        std::cout<<"\t parameterized ctor" << std::endl;
    }
    C() {               //无参（默认）构造函数
        std::cout << "\t default ctor" << std::endl;
    }
    C(const C& c) {     //拷贝构造函数
        std::cout << "\t copy ctor" << std::endl;
    }
    C& operator =(const C& c) {  //重载了类C的赋值运算符，观察main()中的对象赋值现象
        std::cout << "\t assignment operator" << std::endl;
        return *this;
    }
};

int main() {
    cout << "C c1;" << endl;
    C c1;           //调用无参(默认)构造函数
    cout << "C c2(2);" << endl;
    C c2(2);        //调用有参构造函数
    cout << "C c3 = c1;" << endl;
    C c3 = c1;      //调用拷贝构造函数
    cout << "C c4 = C();" << endl;
    C c4 = C();     //编译器将之等价为 C c4;
    cout << "C c5 = C(5);" << endl;
    C c5 = C(5);    //编译器将之等价为 C c5(5);
    cout << "C c6(c1);" << endl;
    C c6(c1);       //调用拷贝构造函数
    cout << "C c7(C(7));" << endl;
    C c7(C(7));     //编译器将之等价为 C c7(7);
    cout << "C c8[4] = {C(), C(18), c7};" << endl;
    C c8[4] = {C(), C(18), c7}; //c8中的4个元素初始化时，是看做4个独立的对象处理：
                                //C c80 = C();       调默认构造函数
                                //C c81 = C(18);     调有参构造函数
                                //C c82 = c7;        调拷贝构造函数
                                //C c83 = C();       调默认构造函数
    cout << "C c9; c9 = C(9);" << endl;
    C c9;           //调用默认构造函数
    c9 = C(9);      //先调用有参构造函数构造一个匿名对象C(9)，然后调用赋值运算符将该匿名对象赋值给c9

    return 0;
}
```

运行结果(Dev-C++ 5.6.0, MinGW GCC 4.8.1 32-bit Debug)：

```
C c1;
         default ctor
C c2(2);
         parameterized ctor
C c3 = c1;
         copy ctor
C c4 = C();
         default ctor
C c5 = C(5);
         parameterized ctor
C c6(c1);
         copy ctor
C c7(C(7));
```

```
         parameterized ctor
C c8[4] = {C(), C(18), c7};
         default ctor
         parameterized ctor
         copy ctor
         default ctor
C c9; c9 = C(9);
         default ctor
         parameterized ctor
         assignment operator
```