



## 'This' pointer in C++

To understand 'this' pointer, it is important to know how objects look at functions and data members of a class.

1. Each object gets its own copy of the data member.
2. All-access the same function definition as present in the code segment.

Meaning each object gets its own copy of data members and all objects share a single copy of member functions.

Then now question is that if only one copy of each member function exists and is used by multiple objects, how are the proper data members are accessed and updated?

The compiler supplies an implicit pointer along with the names of the functions as 'this'.

The 'this' pointer is passed as a hidden argument to all nonstatic member function calls and is available as a local variable within the body of all nonstatic functions. 'this' pointer is not available in static member functions as static member functions can be called without any object (with class name).

For a class X, the type of this pointer is 'X\*'. Also, if a member function of X is declared as const, then the type of this pointer is 'const X\*' (see [this GFact](#))

In the early version of C++ would let 'this' pointer to be changed; by doing so a programmer could change which object a method was working on. This feature was eventually removed, and now this in C++ is an r-value.

C++ lets object destroy themselves by calling the following code :

```
delete this;
```

As Stroustrup said 'this' could be the reference than the pointer, but the reference was not present in the early version of C++. If 'this' is implemented as a reference then, the above problem could be avoided and it could be safer than the pointer.

Following are the situations where 'this' pointer is used:

### 1) When local variable's name is same as member's name

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

/* local variable is same as a member's name */
class Test
{
private:
```

```

    int x;
public:
    void setX (int x)
    {
        // The 'this' pointer is used to retrieve the object's x
        // hidden by the local variable 'x'
        this->x = x;
    }
    void print() { cout << "x = " << x << endl; }
};

int main()
{
    Test obj;
    int x = 20;
    obj.setX(x);
    obj.print();
    return 0;
}

```

Output:

```
x = 20
```

For constructors, [initializer list](#) can also be used when parameter name is same as member's name.

## 2) To return reference to the calling object

```

/* Reference to the calling object can be returned */
Test& Test::func ()
{
    // Some processing
    return *this;
}

```

When a reference to a local object is returned, the returned reference can be used to **chain function calls** on a single object.

```

#include<iostream>
using namespace std;

class Test
{
private:
    int x;
    int y;

```

```

public:
    Test(int x = 0, int y = 0) { this->x = x; this->y = y; }
    Test &setX(int a) { x = a; return *this; }
    Test &setY(int b) { y = b; return *this; }
    void print() { cout << "x = " << x << " y = " << y << endl; }
};

int main()
{
    Test obj1(5, 5);

    // Chained function calls. All calls modify the same object
    // as the same object is returned by reference
    obj1.setX(10).setY(20);

    obj1.print();
    return 0;
}

```

Output:

x = 10 y = 20

### Exercise:

Predict the output of following programs. If there are compilation errors, then fix them.

#### Question 1

```

#include<iostream>
using namespace std;

class Test
{
private:
    int x;
public:
    Test(int x = 0) { this->x = x; }
    void change(Test *t) { this = t; }
    void print() { cout << "x = " << x << endl; }
};

int main()
{
    Test obj(5);
    Test *ptr = new Test (10);
}

```

```
obj.change(ptr);
obj.print();
return 0;
}
```

## Question 2

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

class Test
{
private:
    int x;
    int y;
public:
    Test(int x = 0, int y = 0) { this->x = x; this->y = y; }
    static void fun1() { cout << "Inside fun1()"; }
    static void fun2() { cout << "Inside fun2()"; this->fun1(); }
};

int main()
{
    Test obj;
    obj.fun2();
    return 0;
}
```

## Question 3

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

class Test
{
private:
    int x;
    int y;
public:
    Test (int x = 0, int y = 0) { this->x = x; this->y = y; }
    Test setX(int a) { x = a; return *this; }
    Test setY(int b) { y = b; return *this; }
    void print() { cout << "x = " << x << " y = " << y << endl; }
```

```
};

int main()
{
    Test obj1;
    obj1.setX(10).setY(20);
    obj1.print();
    return 0;
}
```

#### Question 4

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

class Test
{
private:
    int x;
    int y;
public:
    Test(int x = 0, int y = 0) { this->x = x; this->y = y; }
    void setX(int a) { x = a; }
    void setY(int b) { y = b; }
    void destroy() { delete this; }
    void print() { cout << "x = " << x << " y = " << y << endl; }
};

int main()
{
    Test obj;
    obj.destroy();
    obj.print();
    return 0;
}
```