'this' pointer

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 a constant pointer that holds the memory address of the current object. ‘*this'* pointer is not available in static member functions as static member functions can be called without any object (with class name).

For class X, the type of this pointer is 'X* const'. Also, if a member function of X is declared as const, then the type of *'this'* pointer is 'const X *const'.

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

When local variable's name is same as member's name

```
#include <bits/stdc++.h>
using namespace std;

class Employee {
    public:
        string id, name;
        int years;

        //'this' keyword here retrieves the object's
        //instance variables: id, name, years hidden
        //by their same-name local counterparts
        Employee(string id, string name, int years) {
            this->id = id;
            this->name = name;
            this->years = years;
        }

        //here we don't need to use 'this' keyword
        //explicitly as their are no local variables
        //with the same name. So, compiler automatically
        //deduces it as instance variables
        void printDetails() {
            cout << "ID: " << id
                 << ", Name: " << name
                 << ", Experience: " << years;
        }
};

int main()
{
    Employee emp("GFG123", "John", 4);

    emp.printDetails();
    return 0;
}
```

Output:

```
ID: GFG123, Name: John, Experience: 4
```

For constructors, initializer list can also be used when parameter name is same as member's name.

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 <bits/stdc++.h>
using namespace std;

class Employee {
    private:
        string id, name;
        int years;

    public:
        Employee setId(string id) {
            this->id = id;
            return *this;
        }

        Employee setName(string name) {
            this->name = name;
            return *this;
        }

        Employee setYears(int years) {
            this->years = years;
            return *this;
        }

        void printDetails() {
            cout << "ID: " << id
                 << ", Name: " << name
                 << ", Experience: " << years;
        }
};

int main()
{
    Employee emp;
    emp.setId("GFG123").setName("John").setYears(4).printDeta

    return 0;
}
```

Output:

```
ID: GFG123, Name: John, Experience: 4
```

In the above code, each time we call the setter methods, the employee instance we are referring to is returned using the

'this' pointer. We can thus re-use this instance to chain more method calls.