C++ Pointers (this->Part III)

(https://xkcd.com/371/)

Krishna Kumar

When a member function is called, how does C++ know which object it was called on?

```
// simple.h

class Simple {

private:

    int m_nid;

public:

    Simple(int nid) { //Ctor

        set_id(nid);

    }

    void set_id(int nid) { m_nid = nid; }

    int get_id() { return m_nid; }

};
```

```
int main() {

    Simple csimple(1);

    csimple.set_id(2);

    cout << csimple.get_id();

}
```

// How does a compiler know which object called set_id(2) when it only passes one input argument (int nid)?

What you see vs what the compiler sees

What you see

- set_id(2) takes one argument.
- csimple.set_id(2);
- void set_id(int nid) {

  m_nid = nid;

  }

What a compiler sees

- set_id(2) actually takes two arguments: (2 and address of the object &csimple).
- set_id(&csimple,2);
- void set_id(Simple* const this, int nid) {

  this->m_nid = nid;

  }

this->pointer

- The compiler has automatically converted the function's declaration and definition by adding a new parameter.
- The new hidden parameter 'this' points to the class object the member function is working with.
- Every object has a special pointer "this" which points to the object itself. 'this' is immutable. 'this' can't be zero or null or declared.
- This pointer is accessible to all members of the class but not to any static members of the class, global functions and friend functions.
- Presence of this pointer is not included in the sizeof calculations. As 'this' is not part of the object.

Uses of this pointer

- If you have a constructor (or member function) that has a parameter of the same name as a member variable, you can disambiguate them by using “this”:

```
class Something {
private:
    int id; //member variable
public:
    Something(int id) {
        this->id = id; //this->id member variable ; id - parameter
    }
};
```

Returning *this

- return a reference to the object that was implicitly passed to the function by C++

```
class Calc {
private:
    int m_nValue;
public:
    Calc() { m_nValue = 0; }
    void Add(int nValue) { m_nValue += nValue; }
    void Sub(int nValue) { m_nValue -= nValue; }
    void Mult(int nValue) { m_nValue *= nValue; }
    int GetValue() { return m_nValue; }
};
```

If you wanted to add 5, subtract 3, and multiply by 4, you'd have to do this:

```
Calc objcalc;
objcalc.Add(5);
objcalc.Sub(3);
objcalc.Mult(4);
objcalc.GetValue();
```

Returning *this

```
class Calc {
private:
    int m_nValue;
public:
    Calc() { m_nValue = 0; }
    Calc& Add(int nValue) {  m_nValue += nValue; return *this; }
    Calc& Sub(int nValue) {  m_nValue -= nValue; return *this; }
    Calc& Mult(int nValue) { m_nValue *= nValue; return *this; }
    int GetValue() { return m_nValue; }
};
```

```
// Implementation
Calc obj_calc;
obj_calc.Add(5).Sub(3).Mult(4);
```

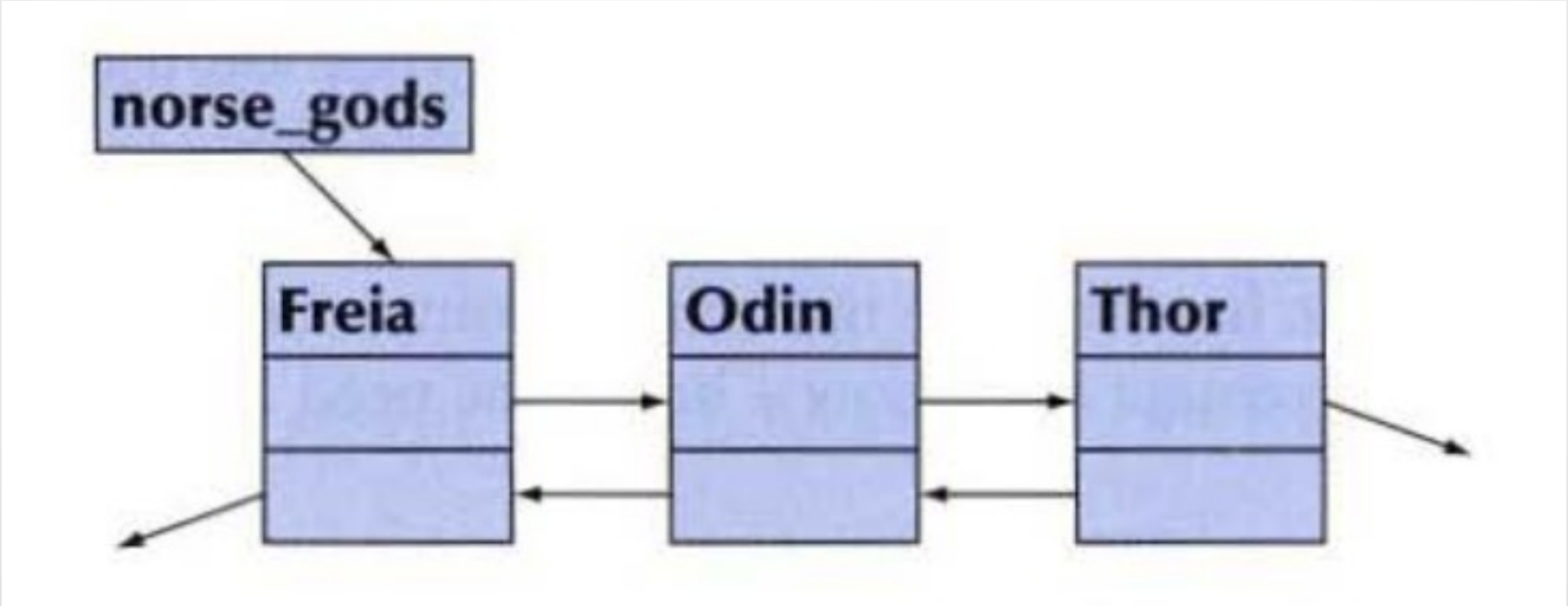
In assignment operators to reduce memory usage

```
class MyClass {
        int data1;
        int data2;
    public:
        MyClass(int data1, int data2) {
            this->data1 = data1;
            this->data2 = data2;
        }
      // Return by reference. Less memory usage
    MyClass& operator = ( MyClass& c ) {
        this->data1 = c.data1;
        this->data2 = c.data2;
        return *this;
    }
};
```

```
int main() {
    MyClass obj1(10, 20);
    MyClass obj2 = obj1;
}
```

Double linked list

- Lists are the most common and useful data structures
- List is made out of links



Link Class

```
struct Link {
    std::string value;
    Link* prev;
    Link* succ;
    Link(const string& v, Link*p,
    Link* s=0) : value(v),
    prev(p), succ(s) {}
}
```

```
// Implementation
Link* norse_gods = new
Link("Thor",0,0);
norse_gods = new Link("Odin",
norse_gods,0);
norse_gods->succ->prev =
norse_gods;
norse_gods = new Link("Freia",
norse_gods,0);
norse_gods->succ->prev =
norse_gods;
```

List Class

```
Link* insert(Link* p, Link* n) {
    if (n == 0) return p;
    if (p == 0) return n;
    n->succ = p; // p comes after n
    if (p->prev)
        p->prev->succ = n;
    n->prev = p->prev; //p's predecessor becomes n's predecesor
    p->prev = n; // n becomes p's predecessor
    return n;
}
```

```
// Implementation
Link* norse_gods
                = new  Link("Thor");
norse_gods = insert(norse_gods, new Link ("Odin"));
norse_gods = insert(norse_gods, new Link ("Freia"));
```

Simplifying insert using this pointer

```
class Link {

public:

string value;

Link(const string& v, Link* p = 0, Link* s = 0) :
value(v), prev(p), succ(s) ( )

Link* insert(Link* n) ; // insert n before this object

Link* add(Link* n) ; // insert n after this object

Link* erase() ; // remove this object from list

Link* (ind(const string& s); // find s in list

const Link* find (const string& s) const; // find s in list

Link* advance(int n) const; // move n positions in list

Link* next() const { return succ;}

Link* previous() const { return prev; }

private :

Link* prev, *succ;

} ;
```

```
// Insert implementation

Link* Link::insert(Link* n) {

// insert n before p; return n

Link* p = this; // pointer to this object

if (n==0) return p; // nothing to insert

if (p==0) return n; // nothing to insert into

n- >succ = p; //p comes after n

if (p->prev) p->prev->succ = n;

n->prev = p->prev; // p's predecessor becomes n's
predecessor

p->prev = n; // n becomes p's predecessor

return n;

}
```

Link using this pointer

```
Link* Link::insert(Link* n){
// insert n before this object; return n
if (n==0) return this;
if (this==0) return n;
n->succ = this; // this object comes after n
if (prev) prev->succ = n;
n- >prev = prev; // this object's predecessor becomes n's predecessor
prev = n; // n becomes this object's predecessor
return n;
}
```

Exercise

Let's build two lists:

```
Link* norse_gods = new Link("Thor");
norse_gods = insert(norse_gods, new Link ("Odin"));
norse_gods = insert(norse_gods, new Link ("Zeus"));
norse_gods = insert(norse_gods, new Link ("Freia"));

Link* greek_gods = new Link("Hera");
greek_gods = insert(greek_gods, new Link ("Athena"));
greek_gods = insert(greek_gods, new Link ("Mars"));
greek_gods = insert(greek_gods, new Link ("Poseidon"));
```

Exercise (cont...)

1) Add functions like Add, Insert, Sort, Find, Erase & Print in the Link Class

   “Unfortunately,” we made a couple of mistakes: Zeus is a Greek god, and the Greek god of war is Ares and not Mars.

- To fix that we need a find function.

  Link* p = find(greek_gods, “Mars”);

- We need an erase function to remove a wrong entry

References

http://www.sourcetricks.com/2008/05/c-this-pointer.html

http://www.learncpp.com/cpp-tutorial/87-the-hidden-this-pointer/

http://www.dreamincode.net/forums/topic/119461-c-this-pointer-tutorial/