Software Development

Welcome
Notes
Welcome to OO
Object Terminology
Modular Programming
Types Overloading
Dynamic Memory
Member Functions
Construction
Current Object
Member Operators
Class + Resources
Helper Functions
Input Output
Derived Classes
Derived Functions
Virtual Functions
Abstract Classes
Templates
Polymorphism
I/O Refinements
D C + Resources
Standards
Bibliography
Library Functions
ASCII Sequence
Operator Precedence
C++ and C
Workshops Workshops
Assignments
Handouts

### Part C - Encapsulation

# The Current Object

Distinguish a member function's access to instance variables and client variables
Access the current object from within a member function

*"Each member function knows what object it was invoked for and can explicitly refer to it" (Stroustrup, 1997)*

Member Function Parameters | this | Summary | Exercises

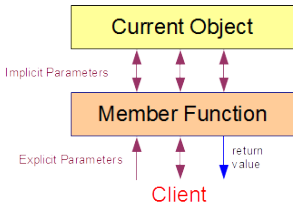
Once client code calls a member function on an object of its class and before that function returns control to the client code, that member function may need to refer to its host object. We refer to the host object from within a member function as the *current object* for that function. In other words, the current object is the region of memory that contains the data on which a member function currently operates.

This chapter describes the mechanism by which a member function accesses its current object and shows how to refer to that current object from within the function.

## MEMBER FUNCTION PARAMETERS

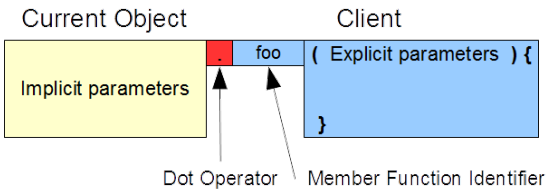
Member functions receive information through parameters and return information through a return value and possibly some of their parameters. The parameters of any member function are of two distinct kinds:

- explicit - access the client code
- implicit - access the instance variables




Explicit parameters receive information from the client code and return information to the client code. We define them explicitly in the header of a member function. Their lifetime extends from entry to the function to exit from the function. They have function scope.

Implicit parameters tie the member function to the current object.



Practice

Resources

The syntax of a normal member function call reflects this two-part mechanism. The name of the object on which the client code invokes the function identifies the implicit parameters, while the arguments that the client code passes to the function initialize the explicit parameters.

Consider the constructors and calls to the **display()** member function in the following code snippet:

```
// ...

Student::Student(int s, const float* g, int n)
{
    cout << "Entering 3-arg constructor\n";
    set(s, g, n);
}

void Student::display() const {
    if (no > 0) {
        cout << no << ":\n";
        cout.setf(ios::fixed);
        cout.precision(2);
        for (int i = 0; i < ng; i++) {
            cout.width(6);
            cout << grade[i] << endl;
        }
        cout.unsetf(ios::fixed);
        cout.precision(6);
    } else {
        cout << "no data available" << endl;
    }
}

int main () {
    float gh[] = {89.4f, 67.8f, 45.5f};
    float gj[] = {83.4f, 77.8f, 55.5f};
    Student harry(1234, gh, 3),
            josee(1235, gj, 3);
    harry.display();
    josee.display();
}
```

```
Entering 3-arg constructor
Entering 3-arg constructor
1234:
 89.40
 67.80
 45.50
1235:
 83.40
 77.80
 55.50
Entering destructor for 1235
Entering destructor for 1234
```

The constructor for **harry** receives data in its explicit parameters and copies that data to the arguments in the call to the **set()** member function on the current object, which consists of the instance variables for **harry**. The constructor for **josee** receives data in its explicit parameters and copies that data to the arguments in the call to the **set()** member function on the current object, which consists of the instance variables for **josee**.

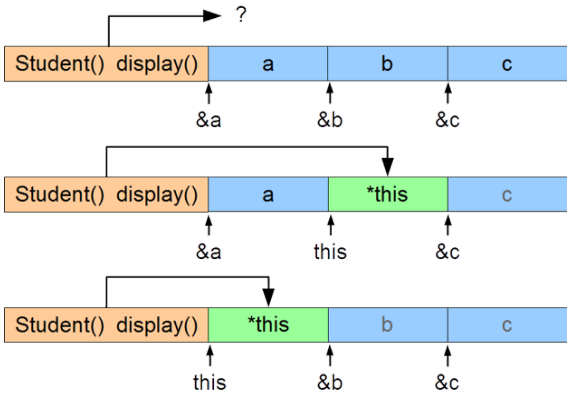
The first client call to the **display()** member function accesses **harry** through its implicit parameters. The second client call accesses **josee** through its implicit parameters.

## THIS

The keyword **this** returns the address of the current object. That is, **this** holds the address of the region of memory that contains all of the data stored in the instance variables of current object. ***this** refers to the current object itself; that is, to the complete set of its instance variables.

We use the **this** keyword within a member function to refer to the complete set of instance variables that that member function is currently accessing through its implicit parameters.

The figure below illustrates usage of the keyword. The top memory map shows two member function and three **Student** objects. The question mark indicates that the data on which the member function logic operates is unknown. The middle memory map identifies the data when the member functions are called on object **b**. The bottom memory map identifies the data when the member functions are called on object **a**.




For example, for the `display()` member function to return a copy of its host object, we write:

```
Student Student::display() const {

    // ...

    return *this;
}
int main() {
    float gh[] = {89.4f, 67.8f, 45.5f};
    Student harry(1234, gh, 3), backup;
    backup = harry.display();
    backup.display();
}
```

```
Entering 3-arg constructor
1234:
 89.40
 67.80
 45.50
Entering destructor for 1234
Entering destructor for 1234
Entering destructor for 1234
Entering destructor for 1234
```

The keyword `this` has no meaning outside a member function.

## Reference to the Current Object

We can improve this definition of `display()` by returning an unmodifiable reference to the current object rather than a copy of the object. This would improve performance if the object was large, since copying all of its instance variables would be compute intensive. Returning a reference only copies the object's address, which is typically a 4-byte operation:

```
const Student& Student::display() const {

    // ...

    return *this;
}
```

The `const` qualifier on the return type prevents client code from placing the call to the member function on the left side of an assignment operator and thereby enabling a change to the instance variables themselves.

## Assigning to the Current Object

To copy the values of the instance variables of one object into those of the current object, we dereference the keyword and use `*this` as the left operand in an assignment expression:

```
*this =             ;
```

### Example - Validated Input

Let us introduce a member function to our `Student` class called `read()` that

- extracts data from standard input

- stores that data in a temporary **Student** object
- copies the temporary object to the current object only if the temporary object is not empty
- leaves the current object unchanged if the temporary object is empty

To avoid duplicating validation logic, we

- construct a local **Student** object passing the input data to the three-argument constructor
- let the internal logic determine whether to accept the data or place the object in a safe empty state
- assign the local object to the current object if the temporary object accepted the data

```
void Student::read() {

    int no;          // will hold the student number
    int ng;          // will hold the number of grades
    float grade[NG]; // will hold the grades

    cout << "Enter student number : ";
    cin >> no;
    cout << "Enter number of grades : ";
    cin >> ng;
    if (ng > NG) ng = NG;
    for (int i = 0; i < ng; i++) {
        cout << "Enter student grade : ";
        cin >> grade[i];
    }

    // construct the temporary object
    Student temp(no, grade, ng);
    // if data is valid, the student number is non-zero
    if (temp.no != 0)
        // copy the temporary object into the current object
        *this = temp;
}
```

Since the temporary object (**temp**) and the current object are instances of the same class, this member function can access each object's instance variables directly.

## SUMMARY

- the current object is the current host object for the member function
- a member function's parameters consist of implicit and explicit parameters
- a member function's explicit parameters receive information from the client code
- a member function's implicit parameters bind the function to the instance variables of current object
- the keyword **this** holds the address of the current object
- ***this** refers to the current object itself
- the keyword **this** is meaningless outside a member function

## EXERCISES

- Complete the Handout on Overloading Constructors

print this page Top
Previous: Construction and Destruction Next: Member Operators

ICT ICT Home Outline Outline Timeline
Notes IPC Notes IPC Notes MySeneca MySeneca Workshops Workshops Assignments Instructor

Designed by Chris Szalwinski

Copying From This Site

Last Modified: 06/20/2022 06:25Last Modified: 05/20/2017 11:50