

Common Modifications of Class Members

Steven Zeil

July 13, 2013

Contents

1	Static	2
1.1	Static Data Members	2
1.2	Static Function Members	9
2	const	10
2.1	const Pointers and References	11
2.2	Is your class const-correct?	12

1 Static

static

Class members can be marked `static` to show that they are "shared" by all variables of that class type.

.....

1.1 Static Data Members

Static Data Members

A *static* data member is a single, shared data value that can be accessed by all variables of the class.

- Only one copy exists in memory
 - If it gets changed, *all* variables see the change
- Often used for shared constants
- Other uses include counters & shared pools of resources

.....

Example: Shared Constants

In our auction program, we have collections like this one:

```
struct BidCollection {
    int MaxSize;
    int size;
    Bid* elements; // array of bids

    /**
     * Create a collection capable of holding the indicated number of bids
     */
    BidCollection (int MaxBids = 1000);

    ~BidCollection ();

    /**
     * Read all bids from the indicated file
     */
    void readBids (std::string fileName);
};
```

Common Modifications of Class Members

We now know a bit more about how ADTs in C++ are supposed to look, so let's bring this up to date a bit.

.....

Bids Collection

```
class BidCollection {
    int MaxSize;
    int size;
    Bid* elements; // array of bids

public:
    /**
     * Create a collection capable of holding the indicated number of bids
     */
    BidCollection (int MaxBids = 1000);

    ~BidCollection ();

    // Access to attributes
    int getMaxSize() const {return MaxSize;}

    int getSize() const {return size;}

    // Access to individual elements
    const Bid& get(int index) {return elements[i];}

    // Collection operations

    void addInTimeOrder (const Bid& value);
    // Adds this bid into a position such that
    // all bids are ordered by the time the bid was placed
    //Pre: getSize() < getMaxSize()

    void remove (int index);
    // Remove the bid at the indicated position
    //Pre: 0 <= index < getSize()
```

Common Modifications of Class Members

```
/**
 * Read all bids from the indicated file
 */
void readBids (std::string fileName);
};
```

Note that it is impossible for the application code to do anything that would get the bids out of order.

.....

Sharing a Constant

As currently defined, all BidCollections have distinct max sizes.

- Suppose that we wanted a single common max size for all the collection objects (e.g., to make it easier to make copies of collections).
 - Make MaxSize a static data member - that one value would be shared by all BidCollections.

```
class BidCollection {
    static const int MaxSize;
    int size;
    Bid* elements; // array of bids

public:
    /**
     * Create a collection capable of holding the indicated number of bids
     */
    BidCollection ( /* int MaxBids = 1000 */ );

    ~BidCollection ();

    // Access to attributes
    int getMaxSize() const {return MaxSize;}

    int getSize() const {return size;}

    // Access to individual elements
    const Bid& get(int index) {return elements[i];}

    // Collection operations
```

Common Modifications of Class Members

```
void addInTimeOrder (const Bid& value);
// Adds this bid into a position such that
// all bids are ordered by the time the bid was placed
//Pre: getSize() < getMaxSize()

void remove (int index);
// Remove the bid at the indicated position
//Pre: 0 <= index < getSize()

/**
 * Read all bids from the indicated file
 */
void readBids (std::string fileName);
};
:
// in the .cpp file:
const int BidCollection::MaxSize = 1000;
```

.....

Example 2: Assigning Unique IDs

```
class Bid {
    std::string bidderName;
    double amount;
    std::string itemName;
    Time bidPlacedAt;
public:
    Bid (std::string bidder, double amt,
        std::string item, Time placedAt);

    std::string getBidder() {return bidderName;}
    double getAmount() {return amount;}
    std::string getItem() {return itemName;}
    Time getTimePlacedAt() {return bidPlacedAt;}
```

Common Modifications of Class Members

```
};  
:  
Bid::Bid (std::string bidder, double amt,  
          std::string item, Time placedAt);  
: bidderName(bidder), amount(amt),  
  itemName(item), bidPlacedAt(placedAt)  
{  
}
```

Suppose that we wished to assign a unique ID number to every bid

- Create a counter
- Increment each time a bid is constructed
- During construction, use the value of that counter as the newly constructed Bid's ID

.....

First Attempt

Question: What's wrong with this?

```
class Bid {  
    int bidCounter = 0;  
  
    int id;  
    std::string bidderName;  
    double amount;  
    std::string itemName;  
    Time bidPlacedAt;  
public:  
    Bid (std::string bidder, double amt,  
         std::string item, Time placedAt);  
  
    std::string getBidder() {return bidderName;}  
    double getAmount() {return amount;}  
    std::string getItem() {return itemName;}  
    Time getTimePlacedAt() {return bidPlacedAt;}  
  
    int getID() {return id;}  
};  
:  
:
```

Common Modifications of Class Members

```
Bid::Bid (std::string bidder, double amt,  
         std::string item, Time placedAt);  
: bidderName(bidder), amount(amt),  
  itemName(item), bidPlacedAt(placedAt)  
{  
  ++bidCounter;  
  id = bidCounter;  
}
```

.....

Common Modifications of Class Members

Answer: All bids get the same ID: 1

.....

Fixed

```
class Bid {
    static int bidCounter;

    int id;
    std::string bidderName;
    double amount;
    std::string itemName;
    Time bidPlacedAt;
public:
    Bid (std::string bidder, double amt,
        std::string item, Time placedAt);

    std::string getBidder() {return bidderName;}
    double getAmount() {return amount;}
    std::string getItem() {return itemName;}
    Time getTimePlacedAt() {return bidPlacedAt;}

    int getID() {return id;}
};
:
int Bid::bidCounter = 0;

Bid::Bid (std::string bidder, double amt,
    std::string item, Time placedAt);
: bidderName(bidder), amount(amt),
    itemName(item), bidPlacedAt(placedAt)
{
    ++bidCounter;
    id = bidCounter;
}
```

- bidCounter is initialized just once, and shared by all the Bid objects.

.....

1.2 Static Function Members

Static Function Members

Less often, we will make function members static

- Used for functions that work with the class but not with specific variables of that class
- Called as `ClassName::functionName(...)` rather than as `variable.functionName(...)`

.....

Example: Presetting the Max

Suppose we want to make sure that all `BidCollections` has the same max size, but want to be able to vary that size from one program execution to another.

- Make `MaxSize` no longer a constant
- Provide a function to set the max
 - This needs to be called *before* the first `BidCollection` is constructed
 - So it cannot be a regular member function

```
class BidCollection {
    static int MaxSize;
    int size;
    Bid* elements; // array of bids

public:
    static void setMaxSize(int max);
    // Establish the common max size used for all bid collections
    // Must be called BEFORE any collections have been constructed

    /**
     * Create a collection capable of holding the indicated number of bids
     */
    BidCollection ( /* int MaxBids = 1000 */ );

    ~BidCollection ();

    // Access to attributes
};
```

Common Modifications of Class Members

```
int getMaxSize() const {return MaxSize;}

int getSize() const {return size;}

// Access to individual elements
const Bid& get(int index) {return elements[i];}

// Collection operations

void addInTimeOrder (const Bid& value);
// Adds this bid into a position such that
// all bids are ordered by the time the bid was placed
//Pre: getSize() < getMaxSize()

void remove (int index);
// Remove the bid at the indicated position
//Pre: 0 <= index < getSize()

/**
 * Read all bids from the indicated file
 */
void readBids (std::string fileName);
};
:
// in the .cpp file:
int BidCollection::MaxSize = 1000;

void BidCollection::setMaxSize(int max)
{
    maxSize = max;
}
```

.....

2 const

What is "const"?

What does it mean if we label something as "const"?

.....

Simple Constants

```
int i = 0;
const j = 0;
:
i = i + 1; // legal
j = j + 1; // illegal
```

We can't modify const variables.

.....

Not-So-Simple Constants

- This is legal
- j is a constant
 - Even though it takes on a thousand different values

```
for (int i = 0; i < 1000; ++i)
{
    const j = i-1;
    cout << j << endl;
}
```

.....

What Does const Mean?

- Once a const has been *initialized*, its value cannot be changed.

```
for (int i = 0; i < 1000; ++i)
{
    const j = i-1;
    cout << j << endl;
}
```

.....

2.1 const Pointers and References

const pointers

```
const int *p = new int;
p = new int; // legal
*p = 0; // illegal
```

- A const pointer can be shifted to point somewhere else
- but it cannot be used to change the bits at any location to which it points

.....

References

```
int[] myArray = {...};
int& i23 = myArray[23];
i23 = 4;
```

- i23 "points to" myArray[23]
- The assignment changes the value at that location

– does *not* change where i23 "points"

.....



Moving References

```
int [] myArray = {...};  
for (int i = 0; i < 1000; ++i)  
{  
    int& x = myArray[i]; // OK  
    x = 4; // OK  
}
```

- Once a reference is initialized to point somewhere, it can never be shifted to point elsewhere

.....

const References

- Like const pointers, const references cannot be used to alter the value at the location to which they point

```
int [] myArray = {...};  
for (int i = 0; i < 1000; ++i)  
{  
    const int& x = myArray[i]; // OK  
    int y = x; // OK  
    x = 4; // illegal  
}
```

.....

2.2 Is your class const-correct?

Is your class const-correct?

In C++, we use the keyword `const` to declare constants. But it also has two other important uses:

1. indicating what formal parameters a function will look at, but promises not to change
2. indicating which member functions don't change the object they are applied to

These last two uses are important for a number of reasons

- This information often helps make it easier for programmers to understand the expected behavior of a function.
- The compiler may be able to use this information to generate more efficient code.
- This information allows the compiler to detect many potential programming mistakes.

.....

Common Modifications of Class Members

definition

A class is *const-correct* if

1. Any formal function parameter that will not be changed by the function is passed by copy or as a const reference (const &).
 2. Every member function that does not alter the object it's applied to is declared as a const member.
-

Const and Member Functions

```
class Point {
public:
    double x;
    double y;

    double distanceFrom (Point p);
};
```

.....

Question: How many parameters does distanceFrom have?

How Many Parameters?

```
class Point {
public:
    double x;
    double y;

    double distanceFrom (Point p);
};
```

- When we call it, we do so like this:

```
double dist = p1.distanceFrom(p2);
```

which shows that there are clearly two parameters, one on the left and one inside the parentheses.

.....

this is Interesting!

```
class Point {
public:
    double x;
    double y;

    double distanceFrom (/* Point* this ,*/ Point p);
};
```

Common Modifications of Class Members

- The implicit parameter is called `this` and has data type "pointer-to-whatever-class-this-is":

.....

Protecting Parameters

```
class Point {  
public:  
    double x;  
    double y;  
  
    double distanceFrom (/* Point* this, */ Point p);  
};
```

- If we want to promise that `distanceFrom` will not change `p`, we do so by passing `p` by value or by `const` reference:

```
    double distanceFrom (/* Point* this, */  
                        const Point& p);
```

- How do we promise not to change `*this`?

.....

Protecting `*this`

```
class Point {  
public:  
    double x;  
    double y;  
  
    double distanceFrom (/* Point* this, */  
                        const Point& p) const ;  
};
```

- Would like to declare `this` as `const Point*`
- But there's no place to do so
 - So the C++ language designers decided to stick it on to the end
- Called a *const member function*

.....



Common Modifications of Class Members

Const member Functions

As a general rule, any member functions that, logically, do not change the object they are applied to, should be marked as const.

.....

Const Correctness: BidCollection

Passed **by copy**, passed **by const ref**, & **const member functions**

```
class BidCollection {
    static int MaxSize;
    int size;
    Bid* elements; // array of bids

public:
    static void setMaxSize( int max );
    // Establish the common max size used for all bid collections
    // Must be called BEFORE any collections have been constructed

    /**
     * Create a collection capable of holding the indicated number of bids
     */
    BidCollection ();

    ~BidCollection ();

    // Access to attributes
    int getMaxSize() const {return MaxSize;}

    int getSize() const {return size;}

    // Access to individual elements
    const Bid& get( int index ) const {return elements[i];}

    // Collection operations

    void addInTimeOrder (const Bid& value );
    // Adds this bid into a position such that
```

Common Modifications of Class Members

```
// all bids are ordered by the time the bid was placed
//Pre: getSize() < getMaxSize()

void remove ( int index );
// Remove the bid at the indicated position
//Pre: 0 <= index < getSize()

/**
 * Read all bids from the indicated file
 */
void readBids ( std::string fileName );
};
:
// in the .cpp file:
int BidCollection::MaxSize = 1000;

void BidCollection::setMaxSize( int max )
{
    maxSize = max;
}
```

.....

Exercise

Question: What changes would you make so that Bid would be const-correct?

```
class Bid {
    static int bidCounter;

    int id;
    std::string bidderName;
    double amount;
    std::string itemName;
    Time bidPlacedAt;
public:
    Bid (std::string bidder, double amt,
        std::string item, Time placedAt);

    std::string getBidder() {return bidderName;}
    double getAmount() {return amount;}
```


Common Modifications of Class Members

```
std::string getItem() {return itemName;}
Time getTimePlacedAt() {return bidPlacedAt;}

    int getID() {return id;}
};
    :
int Bid::bidCounter = 0;

Bid::Bid (std::string bidder, double amt,
          std::string item, Time placedAt);
: bidderName(bidder), amount(amt),
  itemName(item), bidPlacedAt(placedAt)
{
  ++bidCounter;
  id = bidCounter;
}
```

.....