Using Pointers to Member Functions

David Kieras, EECS Dept.
Prepared for EECS 381, Winter 2001

Pointers to member functions are not like regular pointers to functions, because member functions have a hidden "this" parameter, and so can only be called if you supply an object to play the role of "this", and use some special syntax to tell the compiler to set up the call using the hidden “this” parameter.

Declaring pointers-to-member-functions
You declare a pointer-to-member-function just like a pointer-to-function, except that the syntax is a tad different: it looks like the verbose form of ordinary function pointers, and you qualify the pointer name with the class name, using some syntax that looks like a combination of scope qualifier and pointer.

Declaring a pointer to an ordinary function:
```
return_type (*pointer_name) (parameter types)
```

Declaring a pointer to a member function:
```
return_type (class_name::*pointer_name) (parameter types)
```

The odd-looking `"::*"` is correct.

Setting a pointer-to-member-function
You set a pointer-to-member-function variable by assigning it to the address of the class-qualified function name, similar to an ordinary function pointer.

Setting an ordinary function pointer to point to a function:
```
pointer_name = function_name;  // simple form
pointer_name = &function_name; // verbose form
```

Setting a member function pointer to point to a member function:
```
pointer_name = &class_name::member_function_name;
```

Using a pointer-to-member-function to call a function
You call a function with a pointer-to-member-function with special syntax in which you supply the object or a pointer to the object that you want the member function to work on. The syntax looks like you are preceding the dereferenced pointer with an object member selection (the “dot” operator) or object pointer selection (the “arrow” operator).

Calling an ordinary function using a pointer to ordinary function:
```
result = pointer_name(arguments); // short form, allowed
```
or
```
result = (*pointer_name)(arguments);  // the more verbose form
```

Calling the member function on an object using a pointer-to-member-function
```
result = (object.*pointer_name)(arguments);
```

or calling with a pointer to the object
```
result = (object_ptr->*pointer_name)(arguments);
```

Again, the odd looking things are correct: `".*"` and `"->*"`. The parentheses around the whole

pointer-to-member construction are required because of the operator precedences. Of course, if the function returns void, you won't have the "result =" part.

Calling a member function from another member function using pointer to member
This seems confusing but actually is just an application of the pointer to member syntax with "this" object playing the role of the hidden this parameter. If you want a member function f of Class A to call another member function g of class A through a pointer to member function, it would look like this:

```
class A {
      void f();
      void g();
};


void A::f()
{
      // declare pmf as pointer to A member function,
      // taking no args and returning void
      void (A::*pmf)();
      // set pmf to point to A's member function g
      pmf = &A::g;

      // call the member function pointed to by pmf points on this object
      (this->*pmf)();   // calls A::g on this object
}

// using a typedef to preserve sanity - same as above with typedef

// A_pmf_t is a pointer-to-member-function of class A
typedef void (A::*A_pmf_t)();

void A::f()
{
      A_pmf_t p = &A::g;

      (this->*p)();     // calls A::g on this object
}
```