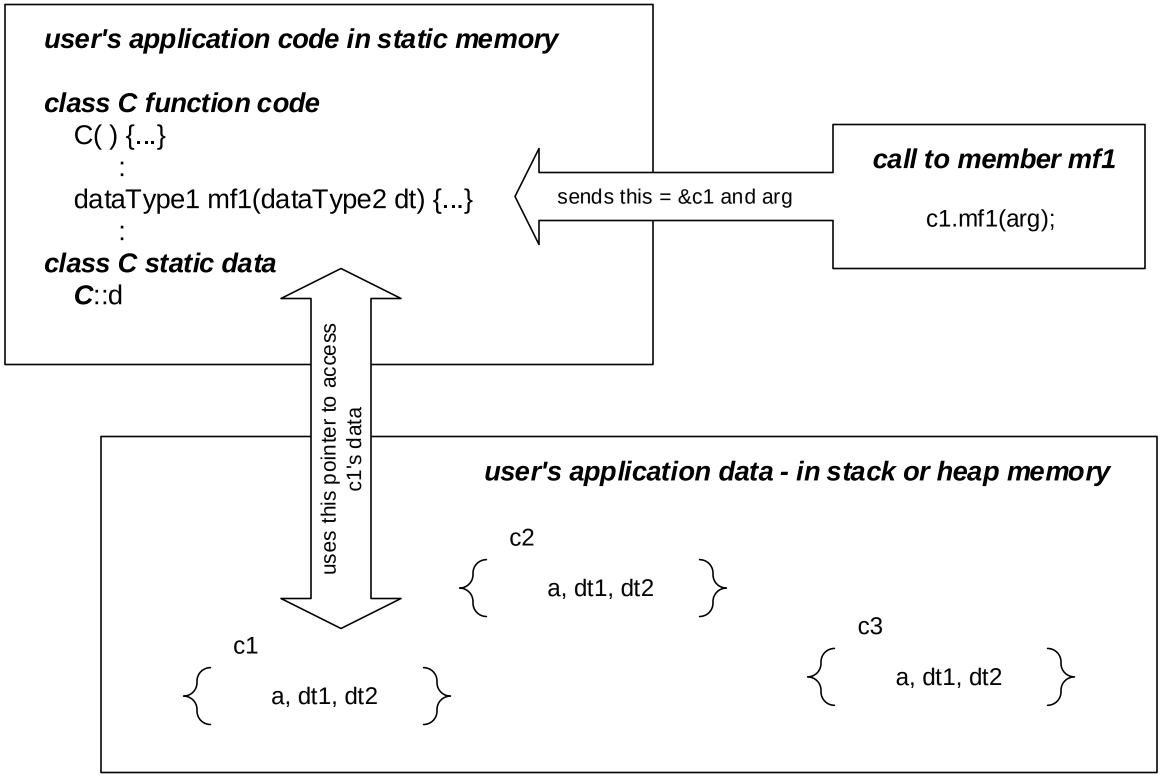
Class Diagram - Class Relationships

class C

```
Attribute:
   A a;
   static double d;
   dataType1 dt1;
   dataType2 dt2;
         :

Operation:
   C( );                          // void constructor
   C(dataType1 dt);               // promotion constructor
   C(const C &c);                 // copy constructor
   virtual ~C( );                 // destructor
   C& operator=(const C& c);
                       // assignment operator
   virtual dataType1 mf1(dataType2 dt);
                       // redefinable member function
   dataType3 mf2(dataType4 dt);
                       // non-redefinable member function
   static void mf3(dataType5 dt);
                       // non-redefinable static member function
```



user's application code in static memory

class C function code

C( ) {...}

:

dataType1 mf1(dataType2 dt) {...}

:

class C static data

C::d

call to member mf1

c1.mf1(arg);

sends this = &c1 and arg

uses this pointer to access c1's data

user's application data - in stack or heap memory

c2 { a, dt1, dt2 }

c1 { a, dt1, dt2 }

c3 { a, dt1, dt2 }

- C provides a set of member functions which operate on its data. Only these member functions are given access to that data.
- This is a very powerful property of the object oriented design approach. It means that member data will always be modified in consistent and meaningful ways.
- When several objects of the class are defined by a user's application code, class C's member functions need a way to determine which of the several objects should be acted on for any given invocation.
  - to handle this C++ silently passes the address of the calling object, in this case the address of c1, from the user's code to the class C member function invoked, in this case mf1(...).
  - this address has the identifier "this", so you will occasionally see "this" used in the code bodies of member functions
  - mf1's code uses "this" to access c1's data to perform its operations before returning control to the user's application code
- Static member functions, like mf3, may only access static member data like d. A client calls a static member function using the class name rather than an object's name. The client may invoke C::mf3(arg) even if no C objects currently exist.