

# C++ Classes & Object Oriented Programming

## Overview & Terminology

# Object Oriented Programming

- Programmer *thinks* about and defines the attributes and behavior of objects.
- Often the objects are modeled after real-world entities.
- Very different approach than *function-based* programming (like C).

# Reasons for OOP

Abstraction

Encapsulation

Information hiding

Inheritance

Polymorphism

*Software Engineering Issues*

# Class: Object Types

- A C++ *class* is an object type.
- When you create the definition of a class you are defining the attributes and behavior of a new type.
  - Attributes are data members.
  - Behavior is defined by methods.

# Creating an object

- Defining a class does not result in creation of an object.
- Declaring a variable of a class type creates an object. You can have many variables of the same type (class).

*Instantiation*

# Information Hiding

- The *interface* to a class is the list of public data members and methods.
- The interface defines the behavior of the class to the *outside world* (to other classes and functions that may access variables of your class type).
- The implementation of your class doesn't matter outside the class – only the interface.

# Information Hiding (cont.)

- You can change the implementation and nobody cares! (as long as the interface is the same).
- You can use other peoples classes without fear!
- Wars will end, poverty and illness will be vanquished from earth!

# Inheritance

- It is possible to *extend* existing classes without seeing them.
- Add whatever new behavior you want.
- Example:
  - You have a class that represents a "student".
  - Create a new class that is a "good student".
    - Most of the behavior and attributes are the same, but a few are different – specialized.

# Polymorphism

- The ability of different objects to respond to the same *message* in different ways.
- Tell an int to print itself: `cout << i;`
- Now tell a double: `cout << x;`
- Now tell the Poly: `cout << poly;`

# Private vs. Public

- Public data members and methods can be accessed outside the class directly.
- The public stuff is *the interface*.
- Private members and methods are for internal use only.
  
- We will also see Protected.

# Special Member Functions

- Constructors: called when a new object is created (instantiated).
  - can be many constructors, each can take different arguments.
- Destructor: called when an object is eliminated
  - only one, has no arguments.

# Accessing Data Members

- Data members are available within each method (as if they were local variables).
- Public data members can be accessed by other functions using the member access operator "." (just like struct).

# Accessing class methods

- Within other class methods, a method can be called just like a function.
- Outside the class, public methods can be called only when referencing an object of the class.

# What happens here?

```
class foo {  
    int i;           // # elements in the array  
    int a[10];      // array 10 at most  
    // sum the elements  
    int sum(void) {  
        int i, x=0;  
        for (i=0; i<i; i++)  
            x+=a[i];  
        return(x);  
    }  
    ...  
}
```

which i is it ?

# Class Scope Operator ::

- You can solve the previous problem using the "::" operator `classname::membername`.

```
for (i=0;i<foo::i;i++)  
    x+=a[i];
```

# Method names and ::

- Sometimes we put just a prototype for a member function within the class definition.
- The actual definition of the method is outside the class.
- You have to use :: to do this.

# Example

```
class foo {  
    ...  
    int sum(void) ;  
};  
  
int foo::sum(void) {  
    int sum=0;  
    for (int i=0;i<foo::i;i++) {  
        sum += a[i];  
    }  
    return (sum) ;  
}
```

# Classes and Files

- The relationship between C++ class definitions and files depends on the compiler.
- In general you can put class definitions anywhere! Visual C++ wants one class per file.
- Most people do this:
  - class definition is in `classname.h`
  - any methods defined outside of the class definition are in `classname.cpp`

# static methods

- A static method is a class method that can be called without having an object.
- Must use the `::` operator to identify the method.
- Method can't access non-static data members! (they don't exist unless we have an object).

# Static Data Members

- It is possible to have a single variable that is shared by all *instances* of a class (all the objects).
  - declare the variable as **static**.
- Data members that are static must be declared and initialize outside of the class.
  - at global or file scope.

# Static data member example

```
class foo {  
private:  
    static int cnt;  
public:  
    foo() {  
        cnt++;  
        cout << "there are now " << cnt  
            << " foo objects << endl;  
    }  
    ...  
}
```

# A Problem

- Class definitions usually in a ".h" file.
- Can be included by many other ".cpp" files, all part of the same program.
- Can't declare and initialize a static data member in the ".h" file.

# Solution

foo.h

```
class foo{  
    static int cnt;  
    ...  
};
```

foo.cpp

```
int foo::cnt=1;
```

f1.cpp

```
#include "foo.h"  
...
```

f2.cpp

```
#include "foo.h"  
...  
int main() {  
    ...
```

# Friends

- A Class can declare other classes as "friend classes".
- A Class can declare external functions as "friends".
- friends can access private members and methods.

# Friend Declaration

```
class foo {  
private:  
    int i,j;  
...  
  
friend class fee;  
friend int printfoo( foo &f1) ;  
};
```

# Operator Overloading

- We can define what some C++ operators mean when applied to user defined objects.
- For example, you can define these operators:

```
ostream <<  
istream >>  
  
+        -  
  
          [ ]  
  
          =  
  
==      <        >
```

# Operator Overloading Restrictions

- You can't:
  - define a completely new operator
  - redefine the meaning of operators on built-in data types (like `int` or `double`).
  - Change operator precedence or associativity.

# Inheritance

- You can create a new class that *inherits* from an existing class.
- You can add new members and methods.
- You can replace methods.
- The new class is a *specialization* of the existing class.

# Terminology

- The existing class is called the *base class*.
- The new class is called the *derived class*.
- Objects of the derived class are also objects of the base class.

# Inheritance Example

- Base class is shape, represents the abstract notion of a shape.
- Derived classes:
  - rectangle
  - circle
  - triangle.
- An object that is a circle is also a shape!

# Protected Class members/methods

- We've already seen private and public.
- Protected means derived classes have access to data members and methods, but otherwise the members/methods are private.

# Public/Private/Protected Inheritance

- Usually use public inheritance.
- Private and Protected inheritance change what members/methods can be accessed outside of the class.
- Complicated rules - see [C++ FAQ - private/protected inheritance](#) for the gory details.

# Polymorphism

- You can treat derived class objects as objects of the base class.
- Example studentlist on the WWW does this.
- The problem is that when doing this we don't get access to the derived class methods!

# Virtual Functions

- A virtual function is a method in a base class that can be overridden by a derived class method.
- In the student list example, we would like each student object to "know" which print() method to use.
- Declare the base class print method virtual, and this happens!