Today's topic: creating objects in C++

- class definitions
- constructors
- overloading the "<<" operator

Topics for future lectures

- inheritance
- virtual functions
- abstract base classes

C++ Class Definitions

- C++ introduced an additional structuring mechanism

```cpp
class NameRecord {
  char name[20];
  char address[20];
  int phoneNumber;
}
```

- Note:
  - coding standard uses initial capitals for class names
  - typedef not needed; the class automatically becomes a new type
  - the semicolon is required

History: struct

- C has a type called a struct
- Use it to create a data structure containing one or more simpler items

```cpp
struct {
  char name[20];
  char address[20];
  int phoneNumber;
} x;
x.name = "George Dubya";
```

In this example struct is used to define the type of a single variable, x.

typedef

- Structures are commonly introduced with a typedef

```cpp
typedef string char[80];
```

```cpp
typedef name_record struct _name_rec_ {
  ...
};
```

```cpp
name_record x;
name_record myAddressBook[100];
```

- Type definitions should be placed in header files

Named Structures

- A struct can have a name

```cpp
struct name_record {
  char name[20];
  char address[20];
  int phoneNumber;
};
```

```cpp
struct name_record x;
struct name_record myAddressBook[100];
```

- The structure is introduced by something similar to a function prototype (note the semicolon)
- This struct has an id (name_record) and the id is used to declare variables (x, myAddressBook).

C++ Class Definitions

```cpp
class NameRecord {
  char name[20];
  char address[20];
  int phoneNumber;
}
```

```cpp
NameRecord x, myAddressBook;
```

- Note:
  - coding standard uses initial capitals for class names
  - typedef not needed; the class automatically becomes a new type
  - the semicolon is required

Class Members

- Elements of a class are called members
- Members can be functions as well as data

```cpp
class Counter {
  int num;
  void inc(int n) { num += n; }
  void dec(int n) { num -= n; }
};
```
Information Hiding

✓ Programmers can restrict access to class members
  - public members are accessible outside the class
  - private members are accessible only to code inside the class
  - protected members are accessible by derived classes

✓ The same definitions are used in Java
  - The default is private

Using Classes

✓ Objects (instances of classes) can be
  - local variables
  - global variables
  - allocated on the heap and accessed via pointers

✓ If Counter is a class,
  Counter a, b; // a and b are instances
  Counter *pa = new Counter; // pa points to a Counter

✓ In Java, objects are only created by calling new

Accessing Members

✓ Suppose Counter is a class with a public member inc
  Counter a, b;
  Counter *pc = new Counter;
  Use the same notation as Java to access the inc member of instances a and b
  a.inc();
  There are two ways to call inc for an object on the heap
  (*pc).inc(); // but not *pc.inc()
  pc->inc();

Constructors

✓ As in Java, object instances are initialized when they are created
✓ The initialization is done by a constructor
✓ C++ classes have three types of constructors
  - default constructor
  - parameterized constructors
  - copy constructor
✓ All are declared within the class as function members
  - same name as the class
  - no return type

Default Constructor

✓ The default constructor has no parameters
  class Player {
    Player() {
      // initialize data members
    }
  }

✓ Called when an object declared with no parameters
✓ Required if you define an array of objects
  Player x;
  Player basketballTeam[5];
Parameterized Constructors

Parameterized constructors provide different ways of initializing objects:

```cpp
class Player {
    Player(string s) {...}
    Player(string s, int ht, int wt) {...}
};
```

Each constructor should have a different combination of arguments.

Copy Constructor

The copy constructor has one parameter, a reference to an instance of the class:

```cpp
class Player {
    Player(const Player &p) {...}
};
```

Called when an object is passed by value to a function:

```cpp
Counter a;
foo(a);  // calls the copy constructor
```

Destructor

The destructor for a class is called when an object is deallocated:

```cpp
class X {
    ~X() {...}
};
```

The destructor has no parameters.

- Define a destructor if an instance contains pointers to items on the heap.

Automatically Generated Members

- The compiler will create several member functions for you:
  - default constructor, generated if you have no other constructors
  - copy constructor, generated if you do not define one
  - "shallow copy" of all data members
  - destructor, generated if you do not define one
  - assignment operator (shallow copy), generated if you do not define one; allows you to copy objects:
    ```cpp
    Counter a, b;
    a = b;
    ```

Class Organization

When defining member functions you have two choices:

- define the member "in-line" as part of the class declaration
- put a function prototype in the class declaration and define the function itself later

```cpp
class Counter {
    void inc();   // prototype for inc
};
```

```cpp
void Counter::inc() {   // define inc
}
```

Class Declarations

The recommended method for organizing classes is to declare all member functions with prototypes.

- Define members in the following order:

```cpp
class X {
    public:
    // constructor and destructor prototypes
    // public member function prototypes
    private:
    // instance variables
    // private member function prototypes
};
```
Class Header Files

- Put the class declaration in a header file
- Define the constructors and member functions in a separate implementation file

The header will be included by:
- the implementation file, which defines the class members
- all programs that use the class

Printing Objects

- It is very common to overload the << operator so you can print objects of your class
- Inside the class, define a print method:
  ```cpp
  void print(ostream &s) {
    s << "...";
  }
  ```
- Outside the class, overload the << operator:
  ```cpp
  ostream &operator <<(ostream &s, Player &P) {
    P.print(s);
    return s;
  }
  ```
- (We’ll look at operator overloading in detail later...)

Printing Objects (cont’d)

- Once you have defined how << handles objects of your class, you can use it in your application
- ```cpp
  Player p;
  cout << "p = " << p << endl;
  ```

Next:

Inheritance in C++