More on Classes and Objects

Lecture 35
COP 3014 Spring 2017

April 17, 2017

Aggregation / Composition: Objects as Class Members

- **Aggregation** (or composition) is a relationship between objects
  - implemented by embedding an object of one class type (or a pointer or reference) as member data of another class type
  - This is the idea of objects embedded inside other objects (components making up the "aggregate")
- Some developers use the term composition to refer to a stronger form of aggregation, where the embedded objects (the components) would typically not exist independent of the container object
- Often known as the "has-a" relationship:
  - We might place an Engine object inside a Car object as member data, because a "car has an engine"
  - We could place 52 Card objects as member data of class Deck, because a "deck has 52 cards"
- Promotes the idea of "tool building".
  - A class is a new type. Objects of this type can now be used as components inside other classes

Constructors for embedded objects

- When an object is created, its constructor runs, but also must invoke the constructors of any embedded objects.
  - If nothing special is done, it will invoke the default constructor, if there is one.
  - To invoke a constructor with parameters for an embedded object, use the initialization list
- Assume you have a Point Class with data members x and y, with a parameterized constructor.
- Assume a class called Location contains 2 members - string name and a Point object loc.
- Then, the constructor for the Location call would be

```
Location(string n, int x1, int y1):  loc(x1,y1)
{
     name=n;
}
```

Arrays of Objects - Declaring

- Recall that an array is an indexed collection of data elements of the same type (where the indexing runs from 0 through size-1)
- In addition to building arrays of built-in types, we can have arrays of objects.

  ```
  Fraction rationals[20]; // array of 20 Fraction
  objects
  Complex nums[50]; // an array of 50 Complex
  objects
  Giraffe herd[25]; //all the Giraffes
  ```
- In an array of objects, each array position is a single object.
  - For instance, given the above declaration of the "rationals" array, there are 20 Fraction objects, named `rationals[0]`, `rationals[1], ... , rationals[19]`

Initialization

- Normally the constructor initializes an object. But how to invoke the appropriate constructor for each object in an array?
- The normal array declaration style uses the default constructor for each object in the array (if the class has a default constructor)

  ```
  Point list[4];
  // builds 4 points using default constructor
  ```
- To specify different constructors for different array items, an initializer set can be used. Since there are no literals for class types, use explicit constructor calls:

  ```
  Point list[3] = { Point(2,4) , Point(5,-1) ,
  Point() };
  // this allocates an array of 3 points,
  initialized to (2,4), (5,-1), and (0,0)
  ```

Using

- Indexing works the same as with regular arrays
- The dot-operator works the same as with single names: `objectName.memberName`
- Just remember that the name of such an object is now: `arrayName[index]`
- Examples:

```
Point points[20];
...
points[2].print(); //displays the third point
object
```

Multi File Compilation

- Ideally, each major class is written in a separate file, in order to allow for easy inclusion into other programs.
- When this is done, the class declaration and definition are written in different files.
- The class declaration is placed in a header file, usually called <classname>.h
- The class definition is placed in the .cpp file of the *same name*
- When we need to include the class in another program, we just include the .h file, but we use quotes, instead of angle brackets.

  ```
  #include "point.h"
  ```
- This assumes the files are all in the same folder

Mutli File Compilation

- We would also like to compile the files separately. Compiling a project is done in 2 steps.
  1. We compile the individual files first. Use the -c flag for this. Compile each class file:
     ```
     g++ -c point.cpp
     g++ -c triangle.cpp
     g++ -c main.cpp
     ```
     This generates the file point.o, triangle.o and main.o
  2. Link all files together into one executable.
     ```
     g++ -o project point.o triangle.o main.o
     ```
- If the object files (.o files) for a class have been generated previously, we can just import the class into our program and then just compile the current program, without having to go through the 2 step process.
- **Never compile a .h file**. This will result in huge .gch object files that will occupy space on the server and not serve any purpose.