CS 213 -- Lecture #7

"Late Night Guide to C++"
Chapter 6 pg 131 - 144
CONSTRUCTORS

Administrative...

- Assignment #3 due today
- Assignment #4 is up!

Constructors

- You may have noticed that in some class definitions we've included a member function to do initialization (`init`)
- In this member function we've done things like:
  - zero out member variables (provide initial values)
  - allocate dynamic space
- Well, C++ has a built in mechanism to doing this type of work.
- It is called a constructor.
- A constructor is a special member function which is always called immediately after space is allocated for the class instance in question.
- The member function name of the constructor is required to be the same name as the class.
- So, if we had a class named `Calculator`, we would define the constructor as follows:

Constructors (cont)

```
class Calculator
{
public:
  Calculator();  // Declare the constructor
  bool calculate(char op,float arg1,float arg2,float &result);
  int getOperationsCount() { return opCount; }
private:
  int opCount;
};

// Here's the constructor definition
Calculator::Calculator()
{
  opCount = 0;
}
```

Simple Constructors(cont)

```
class Calculator
{
public:
  Calculator();  // Declare the constructor
  bool calculate(char op,float arg1,float arg2,float
&result);
  int getOperationsCount() { return opCount; }
private:
  int opCount;
};
```

- Notice a couple of things:
  - The constructor is declared in a public section
    - Has the exact same name as the class itself
  - There is no return type. **Constructors cannot return a value!**
  - There are no arguments (parameters)
    - A simple constructor has no parameters

Simple Constructors(cont)

```
Calculator::Calculator()
{
  opCount = 0;
}
```

- Notice a couple of things:
  - The constructor is defined the same way as any other member function
    - Except, there is no return type
  - Inside the constructor we can perform necessary initializations.
- When does a Constructor get called?
  - A constructor gets called when the object is created.
    - Whether the object is created statically (local variable)
    - or dynamically (with the `new` operator)
  - You do not need to explicitly call the constructor yourself.
- Let's see an example...

Demonstration #1

A Simple Constructor

Constructors with Arguments

- You may define constructors which take arguments as well.
- Consider a simple Course class
  - –similar to the one we used last lecture

```
class Course
{
public:
  Course(string theCourseName,string theInstructor,
         int classSize);
private:
  string courseName;
  string instructor;
  int size;
};
```

- Notice how there is **no** "init" member function...

Constructors with Arguments (cont)

- We would define the Constructor as follows:

```
Course::Course(string theCourseName,string theInstructor,
               int classSize)
{
  courseName = theCourseName;
  instructor = theInstructor;
  size = classSize;
}
```

- This saves us having to define a separate "init" member function
- More importantly, this will be called automatically!
- But if a constructor takes arguments, how do we pass them?

Constructors with Arguments (cont)

- There are two ways to call a constructor with arguments:

```
int main()
{
  Course cs213("COM S 213","Ron DiNapoli",45);
  Course *aCourse = new Course("COM S 213","Ron DiNapoli",45);
  // Rest of program here
  delete aCourse;
}
```

- Again, this saves us having to write a separate "init" function
- But can you have a simple constructor declared as well?
- What happens if you do the following...

Overloaded Constructors

```
class Course
{
public:
  Course();                 // Simple Constructor
  Course(string theCourseName,string theInstructor,
         int classSize);    // Constructor with arguments
private:
  string courseName;
  string instructor;
  int size;
};
```

- Can you really have two member functions with the same name but different arguments?
- Yes, you can. It is called *Overloading*.
- The linker will make sure the right version gets called.

Overloaded Constructors (cont)

```
Course::Course()
{
  courseName = "";
  instructor = "";
  size = 0;
}
Course::Course(string theCourseName,string theInstructor,
               int classSize)
{
  courseName = theCourseName;
  instructor = theInstructor;
  size = classSize;
}
```

- If a Course object is created with no arguments specified, the simple constructor is called...

Demonstration #2

Overloaded Constructors

Initialization Shorthand

- Sometimes it is tedious to write out all of the initializations like we do below:

```
Course::Course(string theCourseName,string theInstructor,
               int classSize)
{
  courseName = theCourseName;
  instructor = theInstructor;
  size = classSize;
}
```

- There is a "shorthand" we can use to simplify this:

Initialization Shorthand (cont)

- Initialization shorthand:

```
Course::Course(string theCourseName,string theInstructor,
               int classSize):courseName(theCourseName),
               instructor(theInstructor),size(classSize)
{
}
```

- Any member variable may be initialized in any constructor for the same class in this manner.
- The format is to append the following expression after the parameter list:

```
: member-name(expression) {, member-name(expression)}
```

Constructors -- Quick Summary

- A *default constructor* is a constructor which takes no arguments
  - If you declare additional constructors you may need to provide a default constructor which does nothing (if you haven't defined one already)
  - Otherwise you may get "Can't construct class" errors when trying to create an instance of the class without passing arguments.
- Other constructors may be added which take arguments
  - This is called *constructor overloading*.
    - A specific form of *function overloading*, which we'll discuss a little later
  - The linker will make sure the right one is called, depending on the arguments passed (or lack thereof)
- A shorthand way to initialize member variables in a Constructor's definition is to follow the parameter list with a colon followed by a comma separated list of member variable names and their initial values in parenthesis.

Constructors and Resource Allocation

- Another common use of constructors is to allocate system resources
  - Memory, GUI objects (Windows, Menus, etc.)
  - Other dynamic structures/classes
- Consider a modification to the Course class from last lecture which allows us to store a dynamic array of Students as a member variable.

```
class Course
{
public:
  Course();
  Course(string theCourse,string theInstructor,int classSize);
private:
  string courseName;
  string instructor;
  int size;
  Student *studentList;
  int nextStudent;
};
```

Constructors and Resource Allocation (cont)

```
Course::Course()
{
  courseName = "unknown";
  instructor = "unknown";
  size = nextStudent = 0;
  studentList = NULL;
}
Course::Course(string theCourseName,string theInstructor,
               int classSize):courseName(theCourseName),
               instructor(theInstructor),size(classSize),
               nextStudent(0)
{
  studentList = new Student[size];
}
```

- It's OK to move the initializations back into the body of the constructor if you're starting to make a mess!

Constructors and Inheritance

- Remember our `Person` class from last lecture?

```
class Person
{
public:
  void setInfo(string Name,string Addr,string Phone);
  virtual void printInfo();
  virtual void printClassification() = 0;  // Pure Virtual
private:
  string name;
  string address;
  string phone;
};
```

- We could apply what we've learned about constructors to do the following:

Constructors and Inheritance (cont)

- Let's provide a constructor to initialize name,address and phone:

```
class Person
{
public:
  Person(string Name,string Addr,string Phone):name(Name),
         address(Addr),phone(Phone){}
  void setInfo(string Name,string Addr,string Phone);
  virtual void printInfo();
  virtual void printClassification() = 0;  // Pure Virtual
private:
  string name;
  string address;
  string phone;
};
```

- Oh yeah, constructors can be defined in the class definition too!

Constructors and Inheritance (cont)

- Oh, wait. `Person` is an abstract class (has one or more pure virtual functions).
- That means that we can never create a "standalone" instance of `Person`.
- Hmmm, can we do something with the constructors of the derived classes?
- Let's look at our Student class again and add a constructor their too...

Constructors and Inheritance (cont)

```
class Student: public Person
{
public:
  Student(string Name,string Addr,string Phone,int id);
  void printInfo();
  int  getId() { return studentID; }
  void printClassification();
private:
  int  studentID;
};

Student::Student(string Name,string Addr,string Phone,int id):
  name(Name),address(Addr),phone(Phone),studentID(id)
{
}
```

- Will it work?

Demonstration #3

Constructors and Inheritance

Constructors and Inheritance (cont)

- Nope, it didn't work.
- You can't access the private members of `Person`, even from the derived class!
- However, you *can* call `Person's` constructor!

```
Student::Student(string Name,string Addr,string Phone,int id):
  Person(Name,Addr,Phone),studentID(id)
{
}
```

- Let's verify that this does indeed work...

Demonstration #4

Constructors and Inheritance II

Copy Constructors

- Consider a constructor which takes an object of same type

```
class Point
{
public:
  Point(){}
  Point(Point anotherPoint);
  void setXY(int newX,int newY) {x =newX; y=newY; }
  void getXY(int &curX,int &curY) {curX=x; curY = y; }
private:
  int x,y;
};

Point::Point(Point anotherPoint)
{
  anotherPoint.getXY(x,y);
}
```

Copy Constructors (cont)

- In a pass by value situation you are actually creating a copy of a given argument on the stack.
- If the argument is a class and has a constructor, it will be called.
- If the parameter to the "copy constructor" is declared pass by value, it will be called
- You can see this would produce infinite recursion!
- Thus, for a copy constructor, the argument *must* be passed by reference.
- Let's verify...

Demonstration #5

Copy Constructors

Final Thoughts

- Remember, prelim exam October 14
  - In class, closed book
  - 25 "short answer" questions