More on classes: Constructors and inheritance

Additional notes for tutorial 6b

Katja Mankinen

20 October 2017

Questions and answers

Yesterday many students had problems and questions with inheritance and constructors.

Read Chapter 15 from course book for more examples and throughout information!

Syntax

A base class called `Base`. A derived class called `Derived` can inherit from `Base`:

```
#include "Base.h"

class Derived : public Base{
//members
};
```

Constructors

- default constructor: a constructor that can be called with no arguments. A default constructor is usually declared without any parameters, but it is also possible for a constructor with parameters to be a default constructor if all of those parameters are given default values.

```
myClass() // default constructor, the user cannot set the initial values
```

- overloaded constructors: depending on the program, various constructors each with a different purpose can be created. A large class would have many data members, some of which may not be defined when an object is instantiated.

```
myClass(int a, int b=0) // construction with a certain 'a', b=0; or allows user to provide
   both a and b
myClass(int a, int b) // overloaded constructor, the user must give both values
```

Usage:

```
myClass NameForClass;   // default constructor
myClass NameForClass(10);  // overloaded constructor; set private member values to a = 10, b =
   0
myClass NameForClass(10,20);  // overloaded constructor; set private member values to a = 10,
   b = 20
```

Note that there's a difference between initialization and assignment:

```
class Example
{
private:
    int itsVar1;
    double itsVar2;

public:
    Example()
    {
        // These are all assignments, not initializations!
        itsVar1 = 1;
        itsVar2 = 2.0;
    }
};

int main()
{
    Example SomeName;
}
```

When the class' constructor is executed (`Example SomeName`), members itsVar1 and itsVar2 are created. Then the body of the constructor is run, and the member data variables are assigned values.

Now let's write the same using **initializer list**:

```
class Example
{
private:
    int itsVar1;
    double itsVar2;
public:
    Example() : itsVar1(1), itsVar2(2.0) // directly initialize our member variables
    {
    // No need for assignment here
    }
};

int main()
{
    Example SomeName;
}
```

We can be even smarter and pass in the initialization values:

```
class Example
{
private:
    int itsVar1;
    double itsVar2;
public:
    Example(int val1, int val2) : itsVar1(val1), itsVar2(val2) // directly initialize our
   member variables
    {
    // No need for assignment here
    }
};

int main()
{
    Example SomeName(95,12);
}
```

The member initializer list is inserted after the constructor parameters. It begins with a colon (:), and then lists each variable to initialize with the value for that variable, separated by a comma. The initializer list does not end with a semicolon.

How to create constructors for Base and Derived classes?

Base class initializer:

- member-initializer syntax
- can be provided in the derived class constructor
- otherwise base class's default constructor called

Derived class constructor

- calls the constructor for its base class first, to initialize base class members of it
- if the derived-class constructor is omitted, its default constructor calls the base class' default constructor

The object of Derived class is constructed in two stages: first, the Base constructor is invoked and then the Derived constructor is invoked. Each class should need to initialize things that belong to it, not things that belong to other classes: the Derived class should not construct the portion of it that belongs to the Base class. The Derived class may depend on members in Base class when initializing its own members. Thus, the constructor of the Base class needs to be called before the Derived class' constructor. In addition, all of the objects that belong to classes should be initialized so that the constructor can use them if it needs to.

What do we do if we want to take arguments to Base class' constructor? We can use initialization lists.

```
class Derived : public Base
{
        Derived() : Base( "arg" ) // initialization list
        {
                // you must include a body, even if it's empty; i.e. curly brackets matter here
        }
};
```

```
#include <iostream>
class Base
{
    public:
        Base( int x )
        {
        std::cout << "Base constructor: " << x << std::endl;
        }
};

class Derived : public Base
{
    public:
        Derived() : Base( 10 )  // construct the Base part of Derived
        {
                std::cout << "Derived constructor" << std::endl;
        }
};

int main()
{
        Derived derivation;
}
```

```
Base constructor: 10
Derived constructor
```

You also need initializer lists when you don't have any default constructors, and thus you must specify which constructor you wish to use. In addition, if you have a members that are references, you also must initialize them in the initialization list.

So now how do we properly initialize members in Base when creating a Derived class object?

So far when we instantiate a Derived class object, the Base class portion has been created using the default Base constructor. We can create another Base constructor, i.e. not use the default one!

```
#include <iostream>

class Base
{
public:
    int baseInt_;

    Base(int baseInt=0)
        : baseInt_(baseInt)
    {
    }

    int getBaseInt() {
  std::cout << "baseint: "<< baseInt_<< std::endl;
        return baseInt_;
  }
};


class Derived: public Base
{
public:
    double derivedVar_;
    int derivedVar2_;

    Derived(double derivedVar=0.0, int derivedVar2=0)
        : Base(derivedVar2), // Call Base(int) constructor with value derivedVar2!
          derivedVar_(derivedVar)
    {
    }

    double getDerVar() {
  std::cout << "dervar: "<< derivedVar_ << std::endl;
        return derivedVar_;
  }
};


int main()
{
    Derived derivation(1.,2);
    derivation.getBaseInt();
    derivation.getDerVar();

    return 0;

}
```

So what you probably were missing yesterday was something like that.

A small help:

```
class Person
{
private:
    std::string itsName;
    int itsPersonNumber;

public:
    Person(std::string name = "", int PersonNumber = 0)
        : itsName(name), itsPersonNumber(PersonNumber )
    {
    }
    ~Person(){};
    std::string getName() const { return itsName; }
    int getPersonNumber() const { return itsPersonNumber; }

};
```

```
class Student : public Person
{
private:
    int itsAge;
    double itsGPA;
    int itsID;

public:
    Student(std::string name = "", int PersonNumber = 0,
        int age=0, double GPA = 0.0, int ID = 0)
        : Person(name, PersonNumber), // call Person(std::string, int) to initialize these
            itsAge(age), itsGPA(GPA), itsID(ID)
    {
    }
    ~Student(){};
    double getAge() const { return itsAge; }
    int getID() const { return itsID; }
    int getGPA() const { return itsGPA; }
};
```

```
int main()
{
  Student pekka("Pekka", 123456, 52, 3.5, 9874);
  std::cout << pekka.getName() << "\n"; //getName() in Person!
  std::cout << pekka.getAge() << "\n"; //getAge() in Student!
  std::cout << pekka.getID() << "\n";
  std::cout << pekka.getGPA() << "\n";
  //Pekka
  //52
  //9874

}
```

Usual errors

No matching function for call to ..constructor

Probably due to missing parameters to initialize the base class, as was described above.

Redefinitions: redefinition of class Name

Add include guardians to your header files:

```
#ifndef NAMEOFHEADER_H
#define NAMEOFHEADER_H
//your code..
#endif
```

Undefined reference to...

- Did you remember to include all necessary header files?
- In case you have multiple .cpp files, did you remember to use them all when compiling?

Naming conventions with classes

Different people and projects use different naming conventions for classes and data members. **Whatever choice you make, pick one style and stay with it**.

For example, the following conventions can be used:

- int val_;
- int m_val;
- int mVal;
- int itsVal;

Same for class names: to start or not to start with a capital letter depends on the project, company and your fellow programmers.