C++ classes
Tutorial 10

Katja Mankinen katja.mankinen@hep.lu.se

10-11 October 2017

Aim

The aim of the tutorials this week is to make you familiar with object oriented programming in C++. This week you will create and use classes, and play with inheritance and polymorphism. After this week, you will have a program that uses three classes, `Student, Teacher` and `Person`. Later you will also get familiar with Makefiles: `make` tool builds program from source code by reading files called Makefiles which specify how to derive the target program.

You can immediately start doing the exercises at your own pace. If you have any problems or questions, just ask!

1. Quick Quiz Time

Let's start by answering a short Quiz! Correct answers are at the end of this document but please try to think them before checking the answers.

1. What is the difference between object-oriented programming and procedural programming?
2. What is the difference between public and private data members?
3. What is the `protected` keyword used for?
4. What should you have in a header file?
5. What should you have in a source file?
6. If you declare two Dog objects, can they have different values in their itsAge member data?
7. Syntax time: how do you invoke a data member or member function in your program?
8. Syntax time: how do you define a member of a class outside the class?
9. Syntax time: what does `class Apple : public Fruit {/*code*/};` mean?

2. Students

2.1 Student class

Create a new header file, `Student.h`. Declare a class called `Student` with these data members: `age, GPA,` and student `ID`. Make the data members private, and provide public accessor methods to get and set each of the data members. In addition, declare a function `void ShowProfile();`.

Create a source file where you implement the functions defined in the header. Write a program with the Student class that makes two students, Carl and Emma, and sets their age, GPA and student ID. You can choose the values yourself. Your program should also print the values (age, GPA, ID) using `void ShowProfile()`.

2.1.1 Reminder: Constructors and Destructos

As you know, there are two ways to define a variable of type integer:

```
int variable; // define a variable
variable = 10; // assing a value to a variable
//OR:
int variable = 10; // define and initialize variable to 10
```

But how do we initialize the member data of a class? With constructors! Constructors are just member functions that are automatically called when an object of the class is declared.

- Constructors must have the same name as the class
- Constructors do not have a return value: not even `void`
- Constructors can take parameters if needed
  - default constructor: takes no parameters or has parameters that all have default values. It is called if no user-provided initialization values are provided.
  - if the `Class` constructor takes one parameter, you may define a `Class` object by `Class ExampleName(23);`
  - if the `Class` constructor takes two parameters, you may define a `Class` object by `Class ExampleName(23, 100);`
  - if the `Class` constructor takes no parameters, you may define a `Class` object by `Class ExampleName;`, so leaving off the parantheses
- If your class has no other constructors, there will be automatically generated public default constructor for you.

Remember to declare destructors, too! Destructors free any memory you might have allocated. A destructor has the same name of the class, preceded by a tilde ~ . They take no arguments and have no return value.

We can initialize our class member data in the constructor using the assignment operator. In the header you would write

```
class ClassName
{
  public:
  ClassName()
  {
  // these are assignments, not initializations. When the constructor is executed, _var1 and
    _var2 are created and the member data variables are assigned to values
  _var1 = 0;
  _var2 = 10;
  }
  private:
  int _var1;
  int _var2;
}
```

This is problematic approach when you have constant and reference variables that must be initialized on the line they are declared. You cannot assing a value to a const variable:

```
const int _var1;
_var1 = 5; // not constant!
```

To solve the problem we can use member initializer/initialization list! You will need them when using inheritance. The initializer list is in this form:

```
Class::Class() : member(value) /* additional members separated by commas */ {}
```

Writing the above code using initialization list:

```
class ClassName
{  ...
  public:
  ClassName() : _var1(0), _var2(10) // initialize member variables. Syntax: initializer list
   after the constructor parameters, begin with a colon : and then list each variable and
   value to be initialized. Note: no semicolon after the list!
  {
  // nothing here, no need for assignment
  }
  private:
  int _var1;
  int _var2;
   ...
}
```

To call this in your program, you can simply write `ClassName InstanceName;`, for example `Dog Snoopy;`.
You can of course also write a constructor like

```
ClassName(int value1, int value2) : _var1(value1), _var2(value2)
```

and call it in your program

```
ClassName InstanceName(3, 5); // value1 = 3, value2 = 3
```

What if you want to have some default values in case the user did not give any? Easy:

```
ClassName(int value1, int value2=5) : _var1(value1), _var2(value2)

ClassName InstanceName(3); // value1 = 3, value2 gets the default value 5
```

Note that you are allowed to initialize const variables but not assign to them, so the following is legal:

```
class ClassName
{ ...
  public:
  ClassName() : _var1(100)
  {
  }
  private:
  const int _var1;
 ...
}
```

It is recommended to initialize variables in the initialization list in the same order in which they are declared in your class.

You can have more than one constructor in your class. For example

```
Class::Class(int value1, int value2) : _var1(value1), _var2(value2)
{
 // do something
}

Class::Class(int value1)
```

```
{
 // do something
}

Class::Class() : _var1(value1), _var2(value2)
{
}

int main() {
Class Instance1(2, 10); // use first constructor
Class Instance2(5); // use second constructor
Class Instance3; // use third constructor without arguments
}
```

After this long text, make sure your `Student` class has at least one constructor.
Test what happens if you just call `Student name1;` in your main().
Test what happens if you call `Student name1();` in your main().

2.2 Adding inheritance and polymorphism

Create another class `Teacher`. This time you can be creative: what member data and functions should this class have? Declare a class in a new header file, and implement necessary functions in a new source file. Test that your new class is working. Remember: you should not have more than one `main()` at this step.

In addition to classes `Student` and `Teacher`, you may create another class representing persons in general: class `Person`. Here inheritance comes to the stage! All students and teachers share the properties of this class - students and teachers are people, right? What properties all students and teachers have in common? Of course, a Person can have a name and a personal number. What is then still specific for students only, and for teachers only?

Define a class `Person`, implement member functions, and modify your existing classes to inherit from it. Make sure to implement `void ShowProfile()` - now both your derived class `Student` and the base class `Person` have the same function but different members. We want `Student` profile to print student-related information, so we need to override this method in a base class to display members that are special about students. How to do it?

When you test your program, you may get the following error: `error: redefinition of class ClassName` or `error: previous definition of class ClassName`. It means you are including same header files multiple times. To prevent it, add the following lines to your header files:

```
#ifndef NAMEOFHEADERFILE_H
#define NAMEOFHEADERFILE_H

// Existing code goes here

#endif
```

Next time we will use Makefiles, i.e. learn how to make your life easier if you have multiple files to compile.

3. Answers to quiz

1. Procedural programming focuses on functions separate from data. OOP combines data and functions together into objects, and focuses those objects and interaction between them.

2. Public data members can be accessed by everyone who is aware of the class. Private data members can be accessed only by member functions of the class.

3. A protected member of a base class can only be accessed by the members of itself and by the members of classes derived from it.

4. In your header file (`.h` file): specify what member functions (methods) and data members the class will have.

5. In your source file (`.cpp` file): implement the code for functions defined in the header file

6. Yes, of course. Each object of a class has its own data members: own ages, names, and so on, depending on the class.

7. Invoking a data member or member function in your program: `instanceName.memberName`. For example Snoopy is an instance of the class Dog, and it has a member Woof(). Now we can just write `Snoopy.Woof()`.

8. We define a member of a class outside the class by using the scope operator ::, for example in our constructor `Dog::Dog(int initialAge)`. Similarly, if we have other functions we want to define outside the class, we can do it by using the same syntax:

```
void ClassName::NameOfMethod()
{
  //do whatever what you want
}
```

9. It means members in class `Fruit` are inherited by `Apple` and are also public in `Apple`.