CIS 15 Fall 2007, Assignment IV

Instructions

- This is assignment for Unit IV.
- It is worth 10 points.
- **It is due on Tuesday April 22nd** and must be submitted by email (as below).
- **Follow these emailing instructions:**
  1. Create a mail message addressed to **parsons@sci.brooklyn.cuny.edu** with the subject line **cis15 hw4**.
  2. Write your name, that is the name under which you registered for the course, in the email. When I get an email from `deathmetal@aol.com` or `pinkprincess@yahoo.com`, I can usually guess whose program it is, but that is not as good as *knowing* whose program it is.
  3. Attach ONLY the files I tell you to send me (see below).
  4. Failure to follow these instructions will result in points being taken away from your grade. The number of points will be in proportion to the extent to which you did not follow instructions . . . (which can make it a lot harder for me to grade your work)

Program description

For this assignment, you will explore two new aspects of programming. One of these is the writing of *multiple file programs*. The other is the re-use of code that you (or someone else) wrote previously.

The idea behind writing "multiple file programs" is that instead of writing a long program in one big file, you split the program up into a number of smaller files. The contents of each file is divided logically, so that you have the definition of one class in each file. By convention, each file is named with the name of the class whose definition is inside it, followed by the `.cpp` extension. In order to make the file's contents usable by other files, you also need to define a *header* file (named `<class>.h`) that contains the definition of the class name and member names, while the `.cpp` file contains the definition of the content of the class members. For example, here is how it would work for the `point` class:

```
// filename: point.h

class point {
private:
  int x, y;
public:
  int  getX() const;
  int  getY() const;
  void set( int x, int y );
  void print() const;
};
```

```
// filename: point.cpp
#include <iostream>
using namespace std;

#include "point.h"

int point::getX() const { return x; }

int point::getY() const { return y; }

void point::set( int x, int y ) {
  this->x = x;
  this->y = y;
}

void point::print() const {
  cout << "(" << x << ", " << y << ")" << endl;
}
```

You can compile `point.cpp` even though it doesn't have a `main()` function by invoking g++ with the `-c` switch, which says: compile but don't link. *Compiling* is the process of turning source code into object code (i.e., files

that end in .o). *Linking* is the process of turning object code into executable code.
So for example, to compile but not link point.cpp, do:

```
unix-prompt> g++ -c point.cpp -o point.o
```

Next, we create a second source code file that will be used to test the point class, e.g.:

```
// filename: testpoint.cpp

#include <iostream>
using namespace std;

#include "point.h"

int main() {
  point p;
  p.set( 1, 2 );
  p.print();
}
```

which is compiled using:

```
unix-prompt> g++ -c testpoint.cpp -o testpoint.o
```

and linked together with point.o using:

```
unix-prompt> g++ testpoint.o point.o -o testpoint
```

and finally we can run the test program using:

```
unix-prompt> ./testpoint
```

A. Write multi-file programs and test programs for rabbit5.cpp

For each of the classes defined in the rabbit5.cpp program discussed in class and available on the class web page, create an individual C++ source (.cpp) file and header (.h) file and a test program:

1. point class (just like above) → point.cpp, point.h, testpoint.cpp
2. living class → living.cpp, living.h (NO TEST PROGRAM!)
3. plant class → plant.cpp, plant.h, testplant.cpp
4. carrot class → carrot.cpp, carrot.h, testcarrot.cpp
5. animal class → animal.cpp, animal.h, testanimal.cpp
6. rabbit class → rabbit.cpp, rabbit.h, testrabbit.cpp
7. fox class → fox.cpp, fox.h, testfox.cpp
8. world class → world.cpp, world.h, testworld.cpp

Do them in the order listed above. NOTE that you CANNOT create a test program for living.cpp because it is an abstract class. Remember, you cannot instantiate an object of an abstract class—you can only extend the abstract class, create definitions for its virtual function(s) and then instantiate objects of that type.

Note that you will need to include the header files for superclasses in the subclass definitions. In other words, since rabbit is a subclass of animal, you will need to include "animal.h" and "rabbit.h" in rabbit.cpp, and when you create a test program for rabbit.cpp, you will need to link both rabbit.o and animal.o, as well as the test program testrabbit.o to create an executable testrabbit.

While this may become tedious once you get the hang of it, being able to write code like this has distinct advantages of (1) "unit testing" each class as you write it and (2) creating modular classes that can be re-used without having to re-write code.

B. Put the pieces together for rabbit5.cpp's main

Now take the `main()` function from `rabbit5.cpp` and place it in a file called `hw4-1.cpp`. Make sure to include the necessary header files and then link this file along with all the superclass and subclass files to create the final executable:

```
unix-prompt> g++ -c hw4-1.cpp -o hw4-1.o
unix-prompt> g++ hw4-1.o point.o living.o plant.o
             carrot.o animal.o rabbit.o fox.o world.o -o hw4
```

Now zip up all the **.cpp** and **.h** files that you have created into **hw4-1.zip** and send this to me.

C. Reuse the code

Now, we'll see how code reuse works, by writing another small simulation that uses some of the same classes.

This simulation is built using:

- A class `person`, which is a subclass of `living`.
- A class `dog`, which is a subclass of `living`.
- An abstract class `building`, which has a location, and which is not a subclass of `living`.
- A class `house`, which is a subclass of `building` and which has a string member `address`.
- A class `factory`, which is a subclass of `building` and which has a string member `product`.
- A class `world`, which includes a `person`, a `dog`, a `house` and a `factory`.

You should have a .cpp file and a .h file for each of these classes (plus any classes that you reuse from the previous questions), plus a test program for each class.

Now, have the simulation:

1. Create a world object in which the house and the factory are at random locations, and in which the person and the dog are at the same location as the house.
2. Set up the dog so that:
   (a) it moves like the rabbit in `rabbit5.cpp`; and
   (b) when it is in the same location as the house, it says "I'll just sleep by the fire".
3. Set up the person so that:
   (a) she moves, one square at a time, towards the factory (rather like the robot in the soccer playing program);
   (b) when she is in the same location as the house, she says "home, sweet home" and the house prints out its address.
   (c) when she is in the same location as the factory, she says "ah, the mighty wheels of capitalism continue to turn" and the factory prints out the product.

To do this, you will need to make some modifications to the class definitions, and you'll also need to write a program with a `main()` function — you cand call this program **hw4-2.cpp**.

Once the simulation is working as described, zip up all of the files (and you won't get full credit unless you give me a seperate .cpp and .h file for each class as well as a hw4-2.cpp with the `main()`) into **hw4-2.zip** and send this to me.

D. Marking rubric

This assignment is worth 10 points. The breakdown is as follows:

A cpp, header and test files for each class (except abstract classes):
`living` = *0.5 point*
`plant` = *0.75 point*
`carrot` = *0.75 point*
`animal` = *0.75 point*
`rabbot` = *0.75 point*
`fox` = *0.75 point*
`world` = *0.75 point*

B hw4.1 main program file = *0.5 point*

C factory example:
cpp, header and test files for each class (except abstract classes):
`person` = *0.75 point*
`dog` = *0.75 point*
`building` = *0.5 point*
`house` = *0.75 point*
`factory` = *0.75 point*
`world` = *0.75 point*
hw4.2 main program file = *0.5 point*

(Yes, this is fractionally more than 10 points)