cis15-fall2007-sklar, assignment V

instructions

- This is assignment for unit V.
- It is worth 8 points.
- **It is due on Monday November 26** and must be submitted by email (as below).
- **Follow these emailing instructions:**
  1. Create a mail message addressed to **sklar@sci.brooklyn.cuny.edu** with the subject line **cis15 hw5**.
  2. Attach ONLY the **.cpp** source code files created below.
  3. 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 — grrrr!)

program description

For this assignment, you will write four small programs that help you explore arrays of objects, pointers and references.

You will use some of what you have done in the past. In particular, you will need the `point` class that you created for assignment II and the self-contained files for it (`point.h` and `point.cpp`) that you created for assignment IV. You may find it helpful to refer to the instructions for multiple file program compilation detailed at the beginning of assignment IV.

You may find it helpful to refer to the `arrayso1.cpp` example from this unit, posted on the class web page (`http://www.sci.brooklyn.cuny.edu/~sklar/teaching/f07/cis15/examples/arrayso1.cpp`).

a. pointers to objects

(2 points)

1. Create a file called `hw4a.cpp`.
2. In this file, create a `main()` that reads two command-line arguments, the `(x,y)` values for a point. Remember to convert the command-line arguments from `char **` to `int`. Also remember to check that the user entered two command-line arguments.
3. Inside the `main()`, declare a pointer to a `point` object, like this:
   ```
   point *mypoint = new point;
   ```
4. Then, initialize the `point`'s data members to the `(x,y)` values that the user entered on the command line. Remember to use the "dash-greater-than" (`->`) pointer notation to refer to the object's members, e.g.,:
   ```
   mypoint->set( x, y );
   ```
5. Finally, also inside the `main()`, display a message that says something like "here is your point:" and then display the coordinates of the point (using the `point` object's function member `print()`).

Use what you learned about multi-file program compilation to compile `point.cpp` and `hw4a.cpp` and link them together into one executable program. Test your code to make sure it works robustly.

b. arrays of objects

(2 points)

1. Create a file called `hw4b.cpp`.
2. In this file, create a `main()` that reads one command-line argument, a number, indicating the number of points in a polygon (i.e., 3 for a triangle, 4 for a rectangle, 5 for a pentagon, 8 for an octagon, 12 for a dodecahedron, etc.). Remember to convert the command-line argument from a `char **` to an `int` so you can use it as a number. Also remember to check that the user entered a command-line argument.
3. Inside the `main()`, declare an array of `point` objects whose size is equal to the number entered on the command-line. For example, if you take the command-line argument and store its numeric value in an `int` called `N`, then you would declare your array as follows:
   ```
   point *polygon = new point[N];
   ```
4. Then, inside the `main()`, loop through the `polygon` array and initialize each of its `points` to randomly generated `x` and `y` values that are between 0 and 100,
5. Finally, also inside the `main()`, display a message that says something like "here are the points in your polygon:" and then display all the points in the polygon.

Compile, link and run your code. Test it to make sure it works robustly.

EXTRA CREDIT:
Instead of prefacing the printing of all the points in the polygon with the generic message "here are the points in your polygon", create a customized message in which the word "polygon" is replaced by the appropriate shape name, depending on the number of points the user entered (e.g., line, triangle, rectangle, pentagon, etc.). You should still use "polygon" as the default, for shapes that don't have a name.
(1 point)

c. passing object arrays to functions

(2 points)

1. Create a file called `hw4c.cpp`.
2. As in the previous step, in this file, create a `main()` that reads one command-line argument, a number, indicating the number of points in a polygon. Remember to convert the command-line argument from a `char **` to an `int` so you can use it as a number. Also remember to check that the user entered a command-line argument.
3. As in the previous step, inside the `main()`, declare an array of `point` objects whose size is equal to the number entered on the command-line.
4. Then (not inside the `main()`), create a function whose prototype looks like this:
   `void initPolygon( point *polygon, int n );` that (as in the previous step) loops through the `polygon` array and initialize each of its `points` to randomly generated `x` and `y` values that are between 0 and 100.

   Invoke this function from the `main()`, passing the polygon array you defined above and the number of points in the polygon.
5. Then (not inside the `main()`), create a function whose prototype looks like this:
   `void printPolygon( point *polygon, int n );` that (as in the previous step) displays a message that says something like "here are the points in your polygon:" and then display all the points in the polygon.

Note that if you have done the extra credit, above, you can also include the customized preamble to displaying the points.

Invoke this function from the `main()`, passing it the appropriate values (i.e., `polygon` array and number of points).

Compile, link and run your code. Test it to make sure it works robustly.

d. classes with arrays of objects

(2 points)

1. Create a file called `hw4d.cpp`.
2. Inside the file, create a class called `polygon` whose definition and function member prototypes look like this:

```
class polygon {
private:
  int n;
  point *points;
public:
  polygon( int n );
  void init();
  void print();
};
```

3. Complete the constructor, in which you set the value of the data members. The value of `n`, the size of the `points` array, simply takes on the value of the constructor's argument. The `points` array is then initialized by allocating space for the array (as you did in the previous step, but inside the constructor instead of at the time when you declare the array).
4. Complete the function member `init()`, which is essentially the same as the `initPolygon()` function you created in the previous step, except that it is now part of the `polygon` class, instead of being declared outside of any class.
5. Complete the function member `print()`, which is essentially the same as the `printPolygon()` function you created in the previous step, except that it is now part of the `polygon` class, instead of being declared outside of any class.
6. Create a `main()` that reads one command-line argument, a number, indicating the number of points in a polygon. Remember to convert the command-line argument from a `char **` to an `int` so you can use it as a number. Also remember to check that the user entered a command-line argument.
7. Inside the `main()`, declare (and construct) an object of type `polygon`, passing to the constructor the size indicated by the command-line argument.
8. Inside the `main()`, invoke the `polygon::init()` function to initialize the points to randomly generated (x, y) values that are between 0 and 100.
9. Finally, inside the `main()`, invoke the `polygon::print()` function.

Compile, link and run your code. Test it to make sure it works robustly.