Computer Science 62
Lab 8

Wednesday, March 24, 2010

Today's lab has two purposes: It is a continuation of the binary tree experiment from last lab and an introduction to some of the command-line tools. The Java portion of the exercise is easy ... except for the fact that we will not use Eclipse.

Getting started

You should have already read the terminal, emacs and svn sections from "our own highly-abbreviated introduction" to Terminal Windows, Emacs, Subversion and Make on the resources section of the course web page. Review this material if you've forgotten it :)

Then open a terminal window, create a directory `cs062/lab08`, and make that your working directory. Open Emacs, type the following code, and save it as `cs062/lab08/Lab08.java`.

To compile the code from the command-line, go back to your terminal window and type the following:

```
public class Lab08{
   public static void main(String[] args){
      System.out.println("Running my first command-line program!");
      System.out.println("The command-line arguments are:");

      for( int i = 0; i < args.length; i++ ){
         System.out.println(i + ": " + args[i]);
      }
   }
}
```

Save this and then compile this by typing the following into the terminal:

```
javac *.java
```

which tells the java compiler to compile all of the .java files in your current directory (which should just be Lab08.java). If you now type `ls` you'll see two files: your original Lab08.java file and the compiled java byte code Lab08.class.

Finally, run your program:

```
java Lab08
```

You should see the message printed out without any command-line arguments. Notice that to call it you just specify the classname of the main method you want to run (Lab08).
To pass in command-line arguments, you just add them after the java command:

```
java Lab08 argument argument2 argument3
```

The arguments are determined by whitespace. If you want to have an argument with spaces in it, you need to surround it by quotes. Play with your program a bit until you're comfortable with command-line arguments.

Try making some changes to your program and run it again. Notice that to make a change you change the file in Emacs, save it and then you need to recompile with the javac command (otherwise, you'll still be using the old version).

Tree heights

Now that you've got the hang of compiling at the command-line, let's experiment with binary search trees and red-black search trees. In Emacs, clear everything in your Lab08.java file and replace it with:

```
import structure5.*;

public class Lab08 {
    public static void main(String[] args) {
         BinarySearchTree<Integer> bstree = new BinarySearchTree<Integer>();
         for (int i = 0; i < 128; i++){
             bstree.add(i);
         }

         System.out.println(bstree.height());
     }
}
```

To compile this you'll need to add one extra thing to your compile command:

```
javac -classpath /common/cs/cs062/bailey.jar:.  *.java
```

The -classpath command tells the compiler where to find the data that we need to compile and/or run. If we need multiple things, we separate them with a '`:`'. In this case, we need the bailey.jar file as a resource *and* the files in the current directory, i.e. '.'.
When we run it, we similarly need to specify the classpath:

```
java -classpath /common/cs/cs062/bailey.jar:.  Lab08
```

Now that you have this working, let's play around with different types of trees:

- Construct a `RedBlackSearchTree` alongside the `BinarySearchTree` and compare their heights.
- For a more realistic comparison, calculate the heights of the two kinds of trees with randomly-generated entries (you may find your code from last time useful to look at).

SVN to the rescue

Follow the directions in "our own highly-abbreviated introduction" to create a Subversion repository for `Lab08`. Add all the files from this lab to your repository and try some of the "harmless" commands, like `ls` and `stat`.

Next, commit your files as if you were finished working. Then choose a file to delete. Pick one that is easily recoverable, like `Lab08.class`. Delete the file, and check out the files again. The deleted file should reappear.

Finally, try the version control. Modify `Lab08.java` and save it. Then revert to the earlier version and watch your changes disappear. (You will have to close and reopen Emacs to see the changes.)

Time permitting...

If you still have time think about the following:

What is the minimum height of a binary tree with 127 elements? Write code to add the integers 1 through 127 to a binary search tree so that the resulting height is minimum.