

CS245 Lab 0: Programming in C++ Scheme & HERA

Due Wednesday of the 3rd week of classes (14 Sept 2009); handed out 1st Tuesday.

This lab is designed to introduce you to the Eclipse I.D.E. (at a deeper level than was used in CS105/206), and to C++, Scheme, and HERA assembly language. It has three projects, one in each language (named `Simple-C++`, `Simple-Scheme`, and `Simple-HERA`). Each project starts with a program to compute factorials, and you are to change or add to it to make it do exponentiation. Your C++ and Scheme functions should work for positive integer exponents and integer bases (they may optionally work for other values as well). If you wish, you may use the $O(\log(n))$ algorithm rather than the $O(n)$ algorithm for finding x^n , but this is not required.

You will need to start by configuring Eclipse and connecting to your course repository, as for other computer science courses — if this is your first Haverford CS course, or you have forgotten since last time, see <http://cs.haverford.edu/resources/H110/settingUpEclipse.html>.

For CMSC 245, there is an additional Eclipse set-up step, because some steps of some projects will involve editing files with programs other than Eclipse. To make this work, engage Eclipse’s “auto-refresh” feature: Choose “Preferences” from Eclipse’s “Window” menu, click the *triangle* by “General”, click on the *word* “Workspace” (the last entry indented below “General”), and click the *check-box* for “refresh automatically” (and then “apply” and “OK”).

Once you have configured Eclipse, obtain the three projects and do the following (the web page <http://cs.haverford.edu/resources/H110/startingALabInEclipse.html> has instructions on getting lab files, but you’ll switch into C++ perspective rather than PyDev to work):

Simple-C++:

- Run the program to make sure you have correctly obtained the files and to ensure that Eclipse is configured properly for you to use C++. You should be able to just select the C++ perspective, right-click on the Simple-C++ executable in the Debug folder, and select “Run Local C/C++ Application”; see the `README.txt` file if you have trouble.
- Before starting any serious work, it’s time you learned more about using version control in Eclipse. Change `factorial.cc` by adding the text “This text represents a dreadful mistake and now my program will not compile” somewhere in the middle of the factorial function, without using comment notation. Confirm that the program no longer compiles. Use Team->Commit to submit this broken program. Make a further edit to make the file even more of a problem, and Team->Commit that. Then explore the history viewing and version-comparison features on the Team menu in Eclipse. Finally, use the history mechanism to replace the current version with the original version you got when you started the project (this is probably 1.1.1.1), add a comment “// thank goodness that is over”, confirm you can once again compile the program, and Team->Commit.
- Now for the real work ... create *two* new functions to raise a number to a power, one using iteration and the other “pure functional” recursion, and also test function. Create new files for your exponentiation functions, rather than using the `factorial.h` and `factorial.cc` files. Put your test function in `main.cc`, and have `main` call your new test. Run your test to make sure your function works, and remember to submit your answer (and, if you like, earlier intermediate versions) by using Team->Share.
- Set a breakpoint in your power function and capture an image of your Eclipse window showing a debugging session for the pure functional code— you should have at least three levels of recursion (e.g., computing 3^6 while working on 3^8). To capture a window in the X11 window system, you can use the `xwd` utility — open a command-line terminal window (under Applications → Accessories) and type `xwd > ~/cs245/Workspace/Simple-C++/Lab1-C++-debug-example.xwd` — at this point the cursor should change to a cross-hair, and you should click on the window you want to capture.

- Add a call to the `dump_stack` function from the base case of your recursion, and copy the output from `dump_stack` when your program has at least three levels of recursion (e.g., finding 3^1 while working on 3^3) into a file `Lab1-dump-example.txt`, and add comments identifying as many of the values as you can.
- Remember to use Team->Commit in Eclipse to submit your work when you're done.

Simple-Scheme:

- Run the `main.scm` program to make sure you have correctly obtained the files and to ensure that Eclipse is configured properly for you to use Scheme. Full details are given in the `README.txt` file.
- Create *one* new function to raise a number to a power using “pure functional” recursion and a test function. Create a new file for your exponentiation function and put your test function in `main.scm`, and have `main` call your new test.
- Run your test to make sure your function works.
- Capture an image of a window showing a debugging session with your recursive Scheme function, as described in the C++ assignment.
- **Optional:** learn about assignment (via `set!`) and loops in Scheme, and write an iterative exponentiation function (only pure functional elements of Scheme will be needed for this course, so this is only for those who just want to explore more of Scheme).
- Remember to use Team->Commit in Eclipse to submit your work when you're done.

Simple-HERA:

- Run the program to make sure you have correctly obtained the files and to ensure that Eclipse is configured properly for you to use HERA. You should be able to run it the same way you ran the Simple-C++ project.
- Either change the existing program or add to it to make it compute 3^9 (you should get 19683). You must generate your answer by doing multiplications starting with 3, but don't have to do anything sophisticated: You may “cheat” as is done in the factorial function, simply multiplying out nine threes, or you may be a bit more clever about how to group the multiplications; if you have prior experience with assembly language programming, you *may* write a loop (or, if you're *really* feeling bold, a recursive function) to do general exponentiation, but **this is not required**. All of your HERA instructions should be in the `HERA_main` function in the `HERA_main.cc` file. Make sure to have your program output the final result of the computation.
- Run your test to make sure your function works.
- Run the program in the debugger and capture an image of the window showing the program and the values of the registers, right before the final multiplication is done.
- **Optional:** If you have taken CMSC240 and 356, demonstrate the use of “Hassem” and your HERA microprocessor for a classmate who hasn't had these courses (if you didn't choose to implement the optional multiplication instruction, demonstrate finding 3^9 by adding up nine 3's).

Or, if you haven't taken those courses, find a classmate who has and ask for a demo.

- Remember to use Team->Commit in Eclipse to submit your work when you're done.

REMEMBER to submit *all three projects* when you're done (it won't hurt to submit repeatedly, so feel free to do extra submits if you're not sure whether or not you did).