CS 251: [Code Warrior:] Lab 09 Profiling and Documenting (Outlab)

- Handed out: 10/02 7:00 PM - Due: 10/05 11:00 PM
- Please write (only if true) the honor code. If you used any source (person or thing) explicitly state it. You can find the honor code on the web page.

Objective

A profiler is used to identify the execution time of (each part of) your code. This might help in identifying those parts of your code that can be optimized – you want to spend maximum energy to optimize that part of code which is likely to have most impact. `gprof` is the profiler we are going to use. (Also note: A profiler comes in after you have identified and implemented the best algorithm, i.e., it is not a substitute for better algorithm design.)

`Doxygen` (pronounced "docs-ee-gen") is a documentation generator, a tool for writing software reference document. It is desirable that you document your code while you are writing your program. The code should be "self documenting" – Doxygen is invaluable here. (Also see: https://en.wikipedia.org/wiki/Literate_programming)

Background

- Visit http://www.thegeekstuff.com/2012/08/gprof-tutorial/ and read about `gprof`.
- The `doxygen` manual is available at http://www.stack.nl/~dimitri/doxygen/manual/index.html. Use `doxygen -g <config-file>` to create a simple config file. Also read about Special Comment Blocks in the Doxygen manual to get an idea about how the comments are used.
- Terminology: **Base code**: The CS 251 base code which has dominoes, and the one you used in Lab 03. **External code**: The Box2D software package including makefiles, CMakefile.lst, etc. The biggie. **Your Project:** Your Box2D-based project that uses the External code and the Base code.

Tasks

Tasks for this lab is to be done in sequence. Do not alter the sequence by having some one else do part 3 in parallel with part 1.

1. `[Doxygen]`. In this part, you will add documentation about the base code so that someone new to the base code can understand it better. (Documentation about Box2D External code already exists, as you probably know). You will also add your changes in your project so that new documentation about your project is also available. [*20 marks*]

   (a) Doxygen requires a configuration file that is by default called `Doxyfile`. The base code in Lab09 will now contain a sample Doxyfile in the base code `doc` directory. This is a simple text file. Execute `make doc` in the `cs251_base_code` folder to generate html documentation for the existing base code. When doxygen is run, a folder called

`html` is created in the `doc` folder. Open `index.html` file in the html folder to see the existing documentation of the base code. You can figure out the namespaces in the code, the classes, which classes contain what kind of members and other interesting information.

(b) Now inspect the base code source and see how it is commented. Read about **Special Comment Blocks** in the `Documenting the Code` section of the Doxygen manual.

(c) Now that you have you have to add Doxygen compatible comments to `dominos.cpp`. The `dominos_t` class sets up the Box2D simulation you see when the code is run. Document the `dominos_t()` constructor function. For this you will have to read through the external Box2D documentation and learn what each of the external Box2D functions are doing. There are 10 top level Box2D blocks in the constructor function corresponding to 10 elements in the simulation. You have to document these in detail. Document what every variable in each block is doing, what is its datatype, and what is its value. Make sure your comments generate meaningful Doxygen documentation.

(d) Now recall (3) (b) in Lab03 where you made the box on the inclined plank move to the right. Add your modified code and document this. Also add **3** new Box2D elements (blocks) to the existing simulation, to make it more interesting. Document these blocks like the ones above.

(e) (Change Makefile). After all this, when you run `make doc` to generate your documentation, the status of execution should be displayed as

```
"Doxygen Documentation of Group-XX ... "
```

and the new html doc should appear in the doc directory.

2. `[gprof warmup]`. Get familiar with `gprof` before you profile your Box2D code. [*5 marks*]

(a) Download the two files `test_gprof.c` and `test_gprof.c` from the lab directory.

(b) Compile them using `-pg` flag (Why?) and generate its executable (say `a.out`).

(c) Run the executable to generate `gmon.out`.

(d) Now run the executable with `gprof` to generate the profiling data. Store this into some text file (say `warmup.txt`).

(e) Now pass this data file to `gprof2dot.py` to convert it into `.dot` format. `dot` is part of the graphviz package.

(f) Run `dot` with the previous output to generate a nice looking picture of the call graph (say `output.png`). `Call graph` is a graph showing the callee-caller relationship between functions, annotated with different information on the edges.

3. `[gprof]` After introducing the doxygen related comments in the base code, and your project code, use gprof to profile your Box2D system using `gprof` and create a report with your observations. [*20 marks*]

(a) Create the Box2D library as before (explained in the `cs251_README.txt` file in the `external/src` folder) with a couple of changes. Build the external library by performing line 4 differently; use

```
cmake  -DCMAKE_BUILD_TYPE=Debug ../
```

It is very important that you do this step correctly.

Compile your code with this library. Make sure to remove optimizing flags such as `-O2` or `-O3` flags (Why?) present in the Makefile. Also look for `-g` flag in the Makefile. Use `CPPFLAGS` in the Makefile for this.

It is very important that you do this step correctly. Otherwise the whole point of this lab might be lost.

Run your simulation until the 5th small spheres down into the container. Profile this code using `gprof` and save the profiled data into some text file.

(b) Generate a `call graph` for the profile of your code.

Use the `gprof2dot` python script to produce the call graph as an image.

(c) Study the call graph to see why your code may be running slower than before. Apply necessary fixes to the code (if necessary). Ask on Piazza for a hint if you cannot fix, but don't provide the solution on Piazza since everybody's changes might be different. If your code is working without slowdown, report that this is the case.

(d) If your code is working fine, and only if you are satisfied, move to this step. Make a report with your observations and include the Debug call graph image. (So at this time, you might have run gprof multiple times, at least two times).

(e) Now, clean your Box2d by running `make clean` and `make distclean`. (But before that make sure you have the generated profile data files with you for your submission.)

(f) Now, re-compile the Box2D library with

```
cmake -DCMAKE_BUILD_TYPE=Release ../
```

Change the makefile to a new makefile called `makefile.release`. Make sure there is a `-O3` or `-O2` compiler option defined for every compile target, i.e., all .o files and the final binary. Use the CPPFLAGS in the Makefile for this. What about `-g` flag? Now profile this code and save the profiled data into some other text file.

(g) Generate a `call graph` for this new profile of your code as you did it last time. Now compare these profiled data. Look at the 5 top functions in both of the profiled data file. Mention their names in your report for both the cases. Try to find some more interesting observations among those two profiles. Do include both of your call graphs in your report.

Submission Guidelines:

Submit the following documents. Don't give us something that we already have.

1. **Task 1:** Place the documented source files and the `Doxyfile` in the `cs251_base_code` folder. Make sure you have run `make clean` in it and that it contains no executables, no libraries, no object files and no pdf. Also, make sure you delete the `Box2D` folder from `external/src`. We want to see only the new source files, and new documentation.

2. **Task 2:** Write the percentage of time taken in each function in readme.txt file.

3. **Task 3:** Place your call graph outputs `debugVersion.png` and `releaseVersion.png` in the `doc` directory.

4. **Task 3:** Place the `report.tex` file containing both the call graphs and observations of your profiled data in the `doc` folder present in base code folder. Also, place the gprof produced text files, which contain your profiled data, in `doc` folder.

Do not forget to put `readme.txt` file in a folder. The folder and its compressed version should both be named `lab09_groupXY_final`. Hence, you submit a tar.gz named `lab09_group03_final.tar.gz` if your group number is 3.