CSE 333 13su Homework #0

Out: Tuesday, June 25, 2013

Due: Friday, June 28, 2013 by **11:00 pm**.

[summary | part a | part b | part c | part d | part e | how to submit | grading]

Summary.

For homework #0, you will fetch a source distribution that we have prepared. Next, you will trivially modify a simple hello world C application, use "make" to compile it, and run it to learn the magic code. You will also run a tool that checks your code for memory leaks and another that checks for potential style problems. Finally, you will package up and submit your code.

As you can probably tell, the real goal of this homework is to make sure that all of the course infrastructure is working for you.

Part A -- Fetch the code

Overview

For this quarter, you'll fetch code from the course web site. The code is packaged as a "gzip'ed tar archive" so you'll need to un-gzip and un-tar it to produce your working directory. Once you've done that, you'll have the homework files and can edit them locally until ready to turn them in. We'll use the course drop box for turn-in.

You may want to use some source control system to help manage your files, and to provide a crude form of backup for them. Popular options are git and subversion.

Fetch the code

Create a directory to contain your cse333 projects. Click (or right-click if needed) on this hw0.tar.gz link to download the compressed archive containing the starter code and store it in that directory.

Expand the tar file

First, have a look at what is about to happen:

```
bash% tar tzf hw0.tar.gz
clint.py
hw0/
hw0/hello_world.c
hw0/Makefile
bash%
```

The files listed will be created when we expand the tar file. The paths are relative to the current working directory. So, subdirectory `hw0` and the files it contains will be created in the current working directory.

```
bash% # expand the tar file
bash% tar xzf hw0.tar.gz
bash% cd hw0
```

Part B -- edit, compile, and run hello_world

In the hw0 directory, type `make`. This command will use the instructions in file `Makefile` to compile the hello_world application using the gcc compiler. Run hello_world by typing the command `./hello_world`. You should see one line of output that looks like:

```
bash% ./hello_world
The magic code is: <xxxxx>
bash%
```

for some `<xxxxx>`. Now, edit `hello_world.c` using your favorite Unix-based editor (mine is emacs), and change the printf so that it instead says:

```
bash% ./hello_world
The magic word is: <xxxxx>
bash%
```

Execute `make` to rebuild, and then re-run `hello_world` to make sure it does what you expect.

Part C -- experience the `gdb` debugger

You're unlikely to have any run time bugs in this homework, but let's use the debugger so that you know what it can do. Here is a session that launches the debugger, sets a breakpoint on source line 27 of `hello_world.c`, prints the values of a couple of variables and an expression, prints a "backtrace" of the stack (showing the sequence of procedure calls that led to executing line 27 of `hello_world.c`), and then continues execution. Type the italicized lines.

```
bash$ gdb ./hello_world
GNU gdb (GDB) Fedora (7.4.50.20120120-54.fc17)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
```

```
This is free software: you are free to change and
redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type
"show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see: ...
Reading symbols from
/home/auser/cse333/hw0/hello_world...done.
(gdb) b 27
Breakpoint 1 at 0x400511: file hello_world.c, line 27.
(gdb) r
Starting program: /home/auser/cse333/hw0/hello_world

Breakpoint 1, main (argc=1, argv=0x7fffffffe0c8) at
hello_world.c:27
27  printf("The magic code is: %X\n", a + b);
(gdb) p argv[0]
$1 = 0x7fffffffe394 "/home/auser/cse333/hw0/hello_world"
(gdb) p a
$2 = -889262067
(gdb) p /x a
$3 = 0xcafef00d
(gdb) p a+b
$4 = -559038737
(gdb) bt
#0  main (argc=1, argv=0x7fffffffe0c8) at hello_world.c:27
(gdb) c
Continuing.
```

```
The magic code is: <xxxxxxxx>
[Inferior 1 (process 6760) exited normally]
(gdb) q
bash$
```

Notes: For `gdb` to be useful, you must give the `-g` switch to `gcc` when compiling. The make file we distributed does that. If you have already modified your file for part B you will see the new output message after "Continuing."

Part D -- verify you have no leaks

Throughout the quarter, we'll also be testing whether your code has any memory leaks. We'll be using Valgrind to do this. Try out Valgrind for yourself, to make sure you can run it:

```
bash% valgrind --leak-check=full ./hello_world
```

Note that Valgrind prints out that no memory leaks were found.

Part E -- check for style issues

Another requirement during the quarter is that your code must be written in good style. Although there are many opinions about what constitutes "good style", in this course we will generally follow the Google C++ Style Guide for both C and C++ code. The distribution file for this assignment included a Python script `clint.py` to check C source files looking for style issues. Try it yourself to be sure you can run it:

```
bash% ../clint.py hello_world.c
```

No style-checking tool is perfect, but you should try to clean up any problems that clint detects.

Turn in

When you're ready to turn in your assignment, do the following:

1. Create a README.TXT file in directory `hw0` that contains your name, student number, and UW email address. As well, write a sentence that reveals the magic word you learned from Part B.
2. Execute the following commands:

```
3. bash$ ls
4. hello_world  hello_world.c  hello_world.o  Makefile README.TXT
5. bash$ make clean
6. /bin/rm -f *.o *~ hello_world
7. bash$ ls
8. hello_world.c  Makefile README.TXT
9. bash$ cd ..
```

10. **bash$** tar czf hw0_<*username*>.tar.gz hw0 # substitute your CSE user name
11. **bash$**
12. Verify that what you're turning in is what you expect to be turning in:
13. $ tar tzf hw0_<*username*>.tar.gz
14. hw0/
15. hw0/hello_world.c
16. hw0/README.TXT
17. hw0/Makefile
18. Turn in file hw0_<*username*>.tar.gz using the course dropbox linked on the main cse333 web page.

Grading

For hw0, we'll give you 3 points if you learn the magic code and correctly submit your work.