SCIT-EIS-UOW
CSCI251/CSCI851 Advanced Programming
Autumn 2019

Laboratory Exercise 9 (Week 10)

Note that lab exercises marked with a * are effectively extension exercises.

I expect most students will want to spend most of this lab working on the assignment, and that fine as far as getting the lab mark goes.

1 Task One: Warm–up exercises

1. Debugging: `Debug-A.cpp`. Apply the principle of least privilege in fixing this. This is again based on an exercise taken from

   `Joyce Farrell, Object Oriented Programming Using C++, 3rd Edition, Thomson Learning.`

2. In that debugging code we overloaded the prefix operator for the class `Student`.

   (a) How do prefix and postfix operations differ?

   (b) How do you overload the postfix ++ operator so the compiler recognises it as such? Why does this make sense?

2 Task Two: Profiling

Several executables have been provided from assignments for CSCI251/CSCI851/CSCI262. These have been compiled ready for profiling.

```
Infect : Scott Mackenzie : CSCI262 Spring 2014.
Soccer-Aakesh: Aakesh Deep: CSCI851 Spring 2017.
Soccer-Ben: Ben Brown: CSCI251 Spring 2017.
Soccer-Zoe: Zoe Hodgson: CSCI251 Spring 2017.
Soccer-Rhiannon: Rhiannon Bolton: CSCI251 Spring 2017.
RPA-Anon: Student didn't want to be named: CSCI251 Spring 2018.
RPA-Helena: Helena Ibro: CSCI851 Spring 2018.
RPA-Megan: Megan Moss: CSCI251 Spring 2018.
RPA-William: William Wood: CSCI251 Spring 2018.
```

The syntax for the various programs are as follows:

```
./Infect

./Soccer 20 2 4 4 4 3 3
1st argument number of games. Others correspond to number
of attackers, midfields, defenders for the two teams.
prof Soccer-Aakesh | head -100 | more
```

```
./RPA 30 10
1st argument number of days for the adventure.
2nd argument for the chance of an encounter per day.
```

You can run a program and then use prof on it , with the exception of `RPA-Helena` for which gprof should be used. Try and identify the functions that are used a lot.

```
prof Infect
prof Soccer-Aakesh
prof RPA-Anon
...
gprof RPA-Helena
```

3 Task Three: Libraries

In lecture set S5b we looking at creating libraries. These are some exercises relating to doing just that.

1. Using a prebuilt static libary: The file `mash.h` is the header file associated with the static library in `libmash.a`. Write a program `mainMash.cpp` that includes the header to give access to the function mash.
   (a) In your `main()` you should apply mash to each command line argument, including the first.
   (b) As per the instructions in the lecture notes you can compile to link the library in using the following:

   ```
   $ CC mainMain.cpp libmash.a -o Mash
   ```

   (c) What does mash appear to do?
2. The files `main.cpp`, `mylibrary.cpp`, and `mylibrary.h` form a program.
   (a) Produce an executable from those files.

   ```
   $ CC main.cpp mylibrary.cpp -o M1
   ```

   (b) Convert the non–driver cpp file into a static library `libcode.a`.

   ```
   $ CC -c mylibrary.cpp
   $ CC -xar -o libcode.a mylibrary.o
   ```

   Generate an executable.

   ```
   $ CC main.cpp libcode.a -o M2
   ```

   (c) Convert the non—driver cpp file into a shared (dynamically linked) library.

   ```
   $ LD_LIBRARY_PATH=.
   $ export LD_LIBRARY_PATH
   $ CC -Kpic -G -o libcode.so -h libcode.so mylibrary.cpp
   $ CC -I. -L. main.cpp -lcode -o M3
   ```

   Generate an executable.
3. Can each of the executables run without the corresponding library being present?