CS 246 Fall 2015 - Tutorial 10

November 20, 2015

1 Summary

- GDB
- Standard Template Library (STL)

2 GDB

- As we write more complex programs, errors start to crop up all over the place
- Sometimes these errors are easy to identify, but often they aren't.
- There are a variety of ways to try to find errors
  - A common debugging tool is the print statement
  - Throwing a bunch of print statments into your code that print out variable values can often find the problem
  - ...But not always (especially in concurrent code, where this will often fix the problem)
- Other times we need a tool that allows us to step through the execution of a program
- In first year, you might have used DrRacket's stepper.
- `gdb` is something like that for C/C++
- `gdb` allows you to print variables, set variables, watch variables, set breakpoints, step through execution, etc
- To use `gdb`, we need to compile our program with the `-g` option which provides debugging information to the debugger
  - For example, it keeps variable and function names, line numbers, etc
- Some common commands include:

Command	Description
run [args]	run the program until it crashes or completes
backtrace|bt	print trace of current stack (list of called routines)
print var-name	print value of specified variable
break routine|[filename:]line-no	set breakpoint at routine or line of file
step [n]	execute next n lines (into routines)
continue [n]	skip next n breakpoints
watch var-name	print a message every time var-name is changed
quit	exit gdb

- To start a gdb session you run the command: `gdb <executable name>`
- You will then be prompted to run the program, set breakpoints, etc
- By default, run will run the program until completion or a crash. So it is wise to set breakpoints before you begin.
- See `gdbexp0.cpp`, `gdbex1.cpp`, `gdbex2.cpp` for examples of buggy programs.

3 STL

- The Standard Template Library is a useful collection of classes and functions to accomplish common tasks.
- You've already been using the STL for streams, strings, and other useful classes.
- In addition to those, the STL most importantly provides a number of common data structures as well as algorithms for interacting with them.
- Exercise: Build a simple debt calculator.
  - Input should be accepted on stdin. Each line should take the following form:

    ```
    person1 amount person2
    ```

    to signifies that `person1` owes `amount` to `person2`. Each `person` should be a string without whitespace, and `amount` should be an int.
  - The output should be the name of the person who owes the most, as follows:

    ```
    person owes the most, to a grand total of $amount
    ```

  - You may find the following STL headers and functions/classes useful:
    - `std::map<K,V>` in `<map>`, which allows storage of values of type `V` associated with an arbitrary (not necessarily integer) key of type `K`.
    - `std::accumulate(std::iterator<T> start,std::iterator<T> end,T init,T (*foldfn)(T,T))` in `<numeric>`, which behaves very similar to `foldr` from functional languages.
    - `std::pair<T,U>` in `<utility>`, which allows the easy storage of pairs with two possibly unrelated types.
- One of the most common and useful parts of the STL is the family of iterators, which act like pointers but can access non-sequential data (for example, if `*it` is the first element in a linked list, then `*(it+1)` would be the second, even if the two elements are not stored sequentially in memory).
- The `<algorithm>` library provides a number of functions for working with iterators to solve common problems, such as searching, sorting, folding, mapping, filtering, reversing, and so forth.
- Other useful headers: `<vector>` for vectors (arbitrarily sized arrays), `<list>, <deque>, <queue>, <stack>` and so forth for their respective data structures, and many more!