CS246 lab Notes #2 gdb, Pipes, and Redirection

- Compiling your "c" programs
  - "g++ –g –Wall –o <output file> <source file>
  - Beware of the following command!!!
    - g++ hw1.cpp –o hw1.cpp
- Executing your "c" programs
  - ./<output program name>
  - or just the program name, if current directory is in the search path. The course config file should set this up for your account
- gdb – the Gnu DeBugger
  - gdb <executable file> – start gdb on a given program (note the executable must be compiled with the –g flag)
  - run (short r) – start program execution
  - list (short l) – list source code with line numbers
  - break (short b) linenumber/functionname – set a break point at the specified line number or function
  - continue (short c) – continue to the next break point
  - step (short s) – execute next program line, step INTO functions
  - next (short n) – execute next program line, step OVER functions
  - print (short p) varname – print the value of the specified variable
  - watch varname – track the value of specified variable at every step
  - quit (short q) – quits gdb
  - Refer to your reference card for more advanced options of gdb
  - You can run gdb inside Emacs with command M-x gdb
- gdb exercise (Exercise 1)
  - `make a copy of /rd/cs246s2016/shared/lab02/lab02.cpp`
  - `compile lab02.cpp, say you named your executable lab02`
  - `gdb lab02`
  - `l`
  - `b 15`
  - `run`
  - `next`
  - `step through the loops and print out any variables of interest`
  - `use continue to skip to the next outer loop iteration`
  - `q when finished`
- Streams revisited
  - The file pointer argument to the above functions is considered a stream
  - Also, we've seen three "standard" streams before
    - stdin
      - Standard Input
      - External input to a function
      - What cin takes by default

        - stdout
            - Standard output
            - Normal output from a function
            - What cout uses by default
        - stderr
            - Standard error
            - A different output from a function
            - Separate from stdout
            - What cerr uses by default
- Piping
    - UNIX has the capability to put a program's output somewhere other than the terminal, such as feed it into another program.
        - Basically, UNIX forms a link between the stdout of one program and the stdin of another program.
    - Example: suppose you have a file hw1.cpp in your directory. Type cat hw1.cpp | more
        - Remember that cat displays a file to stdout without pause
        - cat with no arguments will take input from stdin and display it to stdout
        - cat with multiple arguments (filenames) will concatenate all files to stdout
    - This puts the output of cat hw1.cpp into the functionality of the more program.
    - The advantages of piping are more pronounced when you use some of the more specialized UNIX utilities.
    - Note: you may pipe as many times as you want. A chain of programs piping to each other is called a pipeline.
        - Try: cat | cat | cat | cat | cat
        - What happens?
    - pipe both stderr and stdout into the next program's stdin. (Different for csh/tcsh vs bash
        - **csh/tcsh:** |&
        - **bash:** must talk about redirection first
- Redirection
    - Think about the beginning and end of a pipeline. The stdin is the user input to the terminal, and the stdout is what is printed to the terminal.
    - But what if you want the input from another source, like a file?
    - Remember that stdin, stdout and stderr are just special files setup by the system with specific names. Redirection allows you to use a file in place of any of the three stream locations.
    - wc
        - Counts characters, lines, words in a file
        - By default displays all; wc is equivalent to wc –clw
        - -c – Characters
        - -l – Lines
        - -w – Words

- stdin from file
  - Use < after the command to use the file following it as the input file.
  - Example: try the following three operations.
    - wc hw1.cpp
    - wc < hw1.cpp
    - cat hw1.cpp | wc
- stdout to file
  - Use > after the command to use the file following it as the output file.
  - Example:
    - ls –l > ls.txt
  - That's how we got the long file from the first recitation
- stderr to file (csh/tcsh can not do this)
  - In general, stdout is considered the "first" output stream and stderr the "second", and thus they are represented by 1 and 2 respectively
  - Example:
    - bash: cat blah 2> error.txt
- stdout to stderr (csh/tcsh can not do this)
  - stdout of a program to is written combined to its stderr
  - Example (only works in bash)
    - cat blah 1>&2
    - cat blah 2> error.txt 1>&2
    - can you pipe this?
- stderr to stdout (csh/tcsh can not do this)
  - stderr of a program to is written combined to its stdout
  - Example (only works in bash)
    - cat blah 2>&1
    - cat blah 2>&1 | wc –l
    - cat blah >error.txt 2>&1
- both stdout and stderr to file
  - Example
    - **bash and csh/tcsh:** cat blah >& error.txt
    - **bash:** mv blah gunk 1>&2 2> error.txt
- How to suppress messages:
  - Unix has a file called /dev/null
  - It is always empty, even if you redirect to it
  - Thus to suppress error messages in bash, just do <command> <args> 2> /dev/null
  - Or to suppress all output (both bash and csh/tcsh): <command> <args> >& /dev/null

- Exercise 2:
  - Write a C++ program that prints something to stdout and stderr.
  - Redirect program out:
    - stdout pipe to wc -l
    - stdout only to file
    - stderr only to file
    - stderr to stdout then pipe to wc -l
    - stderr to stdout then to file
    - stdout to stderr then to file
    - all output to file