CS246 Lab Notes #2 gcc, gdb, more unix and emacs

- Compiling your "c" programs
  - "gcc –g –Wall –o <output file> <source file>
  - Beware of the following command!!!
    - gcc hw1.c –o hw1.c
- 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
  - make a copy of ~dxu/handouts/cs246/lab02.c
  - compile lab02.c, say you named your executable lab02
  - gdb lab02
  - l
  - b 15
  - step through the loops and print out any variables of interest
  - use continue to skip to the next outer loop iteration
  - q when finished
- A look at scanf
  - Remember always use the & notation with variables. This will be explained in more detail when we get to pointers.
  - Uses the same escape sequences as printf
  - Remember to use %% for the % sign.
  - scanf is not that good about error checking!
    - Thus use scanf when you are relatively confident about not receiving bad input (example: two computer programs talking to each other is OK, user input is risky)
  - fflush(stdin)
    - Flushes the input buffer (where all keyboard input is collected until processed).

        - Special buffers stdin, stdout and stderr
            - stdin – standard input (keyboard)
            - stdout – standard output (screen)
            - stderr – standard error (also screen)
            - These are actually special files, and fflush is a file operation. We will see more of these when we cover files.
        - Useful in situations where more than required input is given, i.e. asked for 1 int, the user enters 5, etc
    - scanf vs. getchar: which should you use?
        - getchar is much safer: you write your own error handling.
        - On the other hand, you have to write all the handling functions for getchar. scanf is often more convenient for quick-and-dirty applications.
- Unix
    - Permissions
        - The Unix file permissions system is simpler than Windows permissions, yet extremely robust
        - Users, Groups, and World permissions
        - ls –l, on the right column you can see:
        - -rw-r—r—
            - 1st byte = file (-) or directory (d). Other possibilities are link, setuid, etc.
        - chmod
            - Syntax “absolute mode” and “symbolic mode”
            - absolute mode:
                - chmod ### <files>
                - sets all files to the specified permissions. Each number represents the three types of permissions: first user, then group, then world. A value of 4 represents read, 2 for write, and 1 for execute. So 7 means “read, write, and execute”, 5 means “read and execute but no write”

                Example: chmod 644 test.txt – sets the file so the owner can read and write it, but everyone else can only read it.
    - config tcsh
        - There are many different options that can be configured for your shell. We have a fairly well developed configuration in the course account.
        - Make sure you are on the toplevel of your home directory (cd with no args)
        - cp ~dxu/handouts/cs246/cs246-cshrc ~/.cshrc
        - source .cshrc
            - These two commands will bring the sample shell config into your main directory and add it to the current shell. “source” reads a file and treats it as shell configuration.

        - rehash
            - This resets your environment variables to the defaults, as specified by your shell configuration.
        - command completion
            - Press “tab” to finish file names in the directory of the current argument.
            - This works even for directories you’re not working in – for example if you typed “/home1/c/cse123/hando”, then pressed “tab” it would complete it to handouts.
            - This also works inside Emacs for the Meta commands.
        - arrow keys
            - Use up and down to cycle through your command history. Up means go back a command, Down means go forward a command.
        - !
            - History command, that doesn’t require you to go back using the arrow keys
            - !n – the “nth” command
            - !-n – then “nth” command before the current command
            - !! – the previous command.
            - !str – the most recent command that begins with the string “str”
            - !?str? the most recent command that contains the string str
    - wildcards
        - Syntax to allow for multiple file names of a given format.
        - ? – can stand for any single character
        - * - can stand for any number of characters (including zero)
    - file naming conventions
        - You should name files with whatever extension makes sense for organization. For example, use .txt for text files, .c for C files, .sh for shell scripts, etc. This helps keep your files organized so that you know what everything is, and tells Emacs to use what formatting instructions it knows for that format. Note: binary executables don’t normally have an extension, but you know they’re executable from the executable permission.
    - processes and jobs
        - Ctrl + c – quit the current job.
        - Ctrl + d – equivalent to “end of file” , use at the end when entering into standard input from the command prompt.
        - Ctrl + z - suspend a currently running process.
        - ps – display all processes you have running at the moment
        - & - use after a command to make it run in the background (in other words, run the program and immediately give you the command prompt back. For example “emacs <file> &”. Usually used in conjuction with “bg”

        - jobs – different from “ps”, this displays all currently running jobs, because sometimes executing one command at the prompt could start more than one process. More on that in later recitations.
        - fg – run a specified process in the foreground, and unsuspend it if it has been suspended. You will lose your shell prompt after this until you suspend or end the program.
        - bg – tell a program that’s suspended or running in the foreground (the second case usually doesn’t come up with a single terminal) to run in the background.
        -

- Emacs
    - Page up/Page down
        - C-v = Page down
        - M-v = Page up
    - Open a new file
        - C-x C-f
        - You will be prompted for a file name
    - Insert a file
        - M-x insert-file
        - You will be prompted for a file name
    - goto-line
        - M-x goto-line
        - You will be prompted for a number. Emacs then goes to that line.
    - Query replace
        - M-%
            - Replaces the prompted for string interactively
            - Space – “replace and go to next”
            - Delete – “don’t replace”
            - ! – “replace all”
            - - - “back up to previous match”
            - Return – “exit query replace”
    - goto beginning of file
        - M-<
    - goto end of file
        - M->
    - indentations (need to name files with .c)
        - When you press TAB in the middle of a line, it will correctly indent that line for you.
    - compiling within Emacs
        - M-x compile
        - You will be prompted for a compilation command, default is ‘make –k’. Enter appropriate gcc command
        - Alternatively, use “M-x shell” to spawn a shell inside emacs. Then you can call gcc from within Emacs.
    - go to line of compilation error
        - C-x backquote will move the cursor to the line of the first error

        - Alternatively, note the line number and use M-x goto-line
    - finding
        - C-s – “search forward”
        - C-r – “search backwards”