Text Editors and Using the Debugger

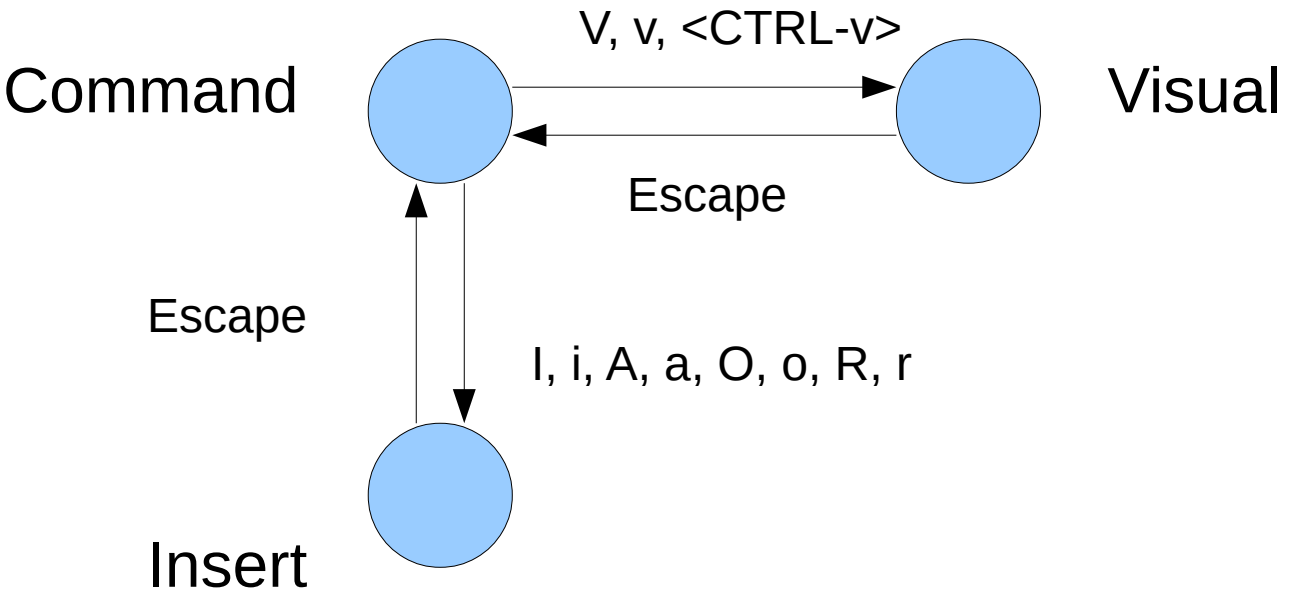
EECS 665 Compiler Construction

Some Common Editors

- Lightweight UI, limited features
  - pico, nano
- Heavier UI, limited features
  - gedit, kate
- Heavier UI, good features, customizable
  - Eclipse, Anjuta
- Lightweight UI, good features, customizable
  - emacs, vim (these slides will focus only on vim)

Vim: Modes of Operation

- Command Mode
- Insert Mode
- Visual Mode



Using Vim to Create & Edit a File

- Start a session

  > vim start_me.c
- Press 'i' to enter insert mode
  - Now you can type any text you want
- 'Esc' to enter command mode

Essential Commands

:e *file*

- Open and edit a different or new *file*

:w

- Save any modifications to the current file

:q

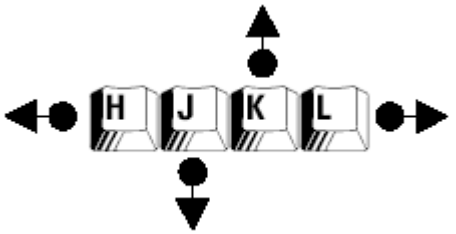
- Quit Vim. If you have modifications you do not want to save, use :q!

:wq

- Write changes and exit the session

Command Mode: Navigation

- Use j, k, l, and h to navigate around the file as seen below. This may take awhile to get used to, but is very nice once you have it down. The arrow keys may also be used in the same manner.



- For faster page scrolling, use <CTRL-b> and <CTRL-f> for page up and page down. The page up and page down keys may also be used in the same manner.

Insert Mode

- The following commands switch to insert mode
  - i – characters inserted just before the cursor position
  - I – characters inserted at the beginning of the line
  - a – characters inserted just after the cursor position
  - A – characters appended to the end of the line
  - o – characters inserted in a new line below the cursor
  - O – characters inserted in a new line above the cursor
  - C – often overlooked, deletes the line after the cursor position and start inserting characters as this point
- After your done editing a file, press Escape to go back to command mode, and :w to write the changes

Common Editor Commands

- Cut/copy/paste in command mode:
  - dd – cut a line of text
  - yy – copy (or “yank”) a line of text
  - P/p – paste a line of text above/below the cursor position
- Commands in Vim can be applied to multiple lines by typing the number of lines you want before the command:
  - '12dd' cuts 12 lines of text
  - '4j' moves the cursor down 4 lines

Searching

/word – search for occurrences of *word*

- Cursor jumps to the next occurrence of *word*
- n/N – jump to the next/previous occurrence of *word*
- *?word* – search initially jumps to previous occurrence of *word*

:set ic – ignore uppercase and lowercase in search (the default is set to not ignore the case)

:nohls to turn off highlighting from last search

Undo/Redo & Find/Replace

- Undo and redo changes
  - u – undo the last action
  - U – undo all the latest changes that were made to the current line
  - <CTRL-r> – redo
- Find and replace
  - :rs/*search_for*/*replace_with*/a
  - The range (r) can be nothing (work on current line only), a number (work on the line whose number you give), and % (work on the whole file)
  - Arguments (a) can be g (replace all occurrences in the line, without this Vim will replace only the first occurrence in each line), i (ignore case for the search pattern), I (don't ignore case), and c (confirm each substitution)

Split Screens

- Vim allows you to edit multiple files in one session
  - <CTRL-w> v to split the screen vertically
  - <CTRL-w> s to split the screen horizontally
  - <CTRL-w> w to switch to the other screen

Vim Resources

- Vim Tips Wiki:
  - http://vim.wikia.com/wiki/Main_Page
- Vim Cookbook
  - http://www.oualline.com/vim/vim-cook.html
- For everything else, just use Google.

Debugger

- A powerful tool that supports examination of your program during execution
- Idea behind debugging programs
- Creates additional symbol tables that permit tracking program behavior and relating it back to the source files
- Some common debuggers for UNIX/Linux
  - gdb, ddd, sdb, dbx, etc.

GDB

- gdb is a tool for debugging C & C++ code
- You can run a program, stop it on any line, and examine various types of information like values of variables, sequence of function calls & change values of variables (during execution)
- You can call a function and trace the execution
- gdb is most effective when it is debugging a program that has debugging symbols
  - Code must be compiled with the -g option

    > gcc -g -o program file.c

Invoking and Quiting gdb

- Invoking a gdb session is easy

  > gdb

- Specify the program you would like to debug when starting a gdb session

  > gdb ./program

- Type quit (q) or <CTRL-d> to exit your session

Running a Program in gdb

- Use the run command to start your program under gdb

  > gdb ./program

  (gdb) run (r)

- Specify the arguments to give your program as the arguments of the run command

  (gdb) r *arg1 arg2* ...

Breakpoints

- One reason the debugger is so powerful is because it allows you to stop your program before it terminates. If you encounter an error, the debugger allows you stop execution before the error occurs and investigate
- break (b)
  - Sets a breakpoint in program execution
  - tbreak (tb) sets a temporary breakpoint that exists until it is hit for the first time
- Breakpoint syntax
  - b *line-number*
  - b *function-name*
  - b *line-or-function* if *condition*
  - b *filename*:*line-number*
- info breakpoints – gives information on all active breakpoints
- delete (d) *breakpoint-number*
  - Deletes the specified breakpoint number (e.g., d 1)

Watchpoints

- Set a watchpoint to stop execution whenever the value of an expression changes, without having to predict a particular place where this may happen
- watch *expr* – set a watchpoint for an expression. gdb will break when *expr* is written into by the program and its value changes
- rwatch *expr* – set a watchpoint that will break when *expr* is read by the program
- awatch *expr* – set a watchpoint that will break when *expr* is either read or written into by the program
- info watchpoints – the same as info break

Control Flow

- Navigating the program is also very useful
- continue (c) – continue until the next breakpoint is reached, the program terminates, or any errors occur
- next (n) – execute one instruction, step over function calls
- step (s) – execute one instruction, step into function calls
- kill (k) – kills the program being debugged (does not exit gdb – preserves everything else from the session, i.e., breakpoints)

Examining the Stack

- The call stack is divided up into contiguous pieces called stack frames; each frame is the data associated with one call to one function
- The frame contains the arguments given to the function, the function's local variables, and the address at which the function is executing
- Each time a function returns, the frame for that function invocation is eliminated and you can no longer see the information for that frame
- info args – prints the arguments passed into the current frame
- info locals – prints the local arguments

Examining the Stack (cont.)

- A backtrace is a summary of how your program got to where it is. The current frame will be displayed at level 0, followed by its caller, and so on
- backtrace (bt) – prints a backtrace of the entire stack
- backtrace (bt) n – prints only the innermost n frames
- up n – select the frame n levels up in the call stack (towards main)
- down n – select the frame n levels down in the call stack
- After you select a new frame, use info as described in the previous slide to display information about the frame

Examining Source Files

- To print lines from a source file, use the list (l) command. By default, 10 lines are printed.
- Variations of the list command
  - list (l) *line-number* – prints lines centered around line number *line-number* in the current source file
  - list *function* – prints lines centered around the beginning of *function* function
  - list *first*, *last* – prints lines from *first* to *last*

Examining Data

- The most common way of examining data in gdb is the print command. It evaluates and prints the value of an expression
- print (p) *expr* – prints the value of some variable
- whatis *expr* – prints the type of *expr*
- You can also use the call command for any functions linked to your program
  - call *function* – allows you to execute functions in the middle of execution

GDB References

- The Unix manual is a good quick reference for common GDB commands:

  > man gdb

- While running GDB, help will give you any information you need for any command:

  (gdb) help (h) *command*

- The official GDB documentation for users is located at: https://sourceware.org/gdb/current/onlinedocs/gdb/index.html
- And finally, Google is always a good source