Improve Your Workflow

EECS 281

Faster *nix Navigation

Use the Ctrl Key!

- Ctrl+p “Previous”, (up arrow)
- Ctrl+n “Next”, (down arrow)
- Ctrl+f “Forward”, (right arrow)
- Ctrl+b “Back”, (left arrow)
- Ctrl+a “home”, (stArt of line)
- Ctrl+e “End”, (End of line)

*Works on CAEN, Mac OS, cygwin

Quick *nix Edits

Use Ctrl Key!

- Ctrl+d “Delete”, (delete character to right)
- Ctrl+h “backspace”, (delete character to left)
- Ctrl+k “Kill”, (delete to end of line)

BONUS!!

- Ctrl+l “cLear screen”, (enough said)
- Ctrl+r “Reverse search”, (Reuse from history)

*Works on CAEN, Mac OS, cygwin

Faster Shell Commands

- Use the exclamation point (!) or “bang”
- Duplicate command line entries in your history

!!	The previous command
!-1	Same as !!
!-2	Two inputs back (can do -3, -4, etc.)

- Duplicate commands by keyword

!mak	The most recent input that starts with “mak”
!.	The most recent command starting with .

- Use the last term from the previous input

!$	From ls /some/path, !$ = /some/path

*Works on CAEN, Mac OS, cygwin

Reading Data from cin

- 1st rule: Don't use cin.eof, cin.good, cin.bad, cin.fail
- 2nd rule: Don't use cin.eof, cin.good, cin.bad, cin.fail
- Conversion after extracting data from an input stream behaves like a boolean, use it to control read loops in your programs

```
while   (cin >> new_value)  {
   // execute  only   if new_value   read   properly
}
while   (getline(cin,  new_line))  {
    // execute  only   if new_line  read   properly
}
```

Reading Data from cin (cont.)

- How are line endings handled?
  - Extraction operator (>>) ignores line endings
  - getline() includes line endings
  - Be careful when getline() follows >>
- Unwanted strings can be read into the same variable

```
cin >> unwanted >> value1 >> unwanted >> value2;
```

Using getopt

- Read the docs!
  http://www.gnu.org/software/libc/manual/html_node/Getopt-Long-Options.html
- Parse *longopts* (longform flags) and *shortopts* (abbreviated flags) with getopt_long()
  - Longopts are specified in a struct option variable
  - Shortopts are specified in a char * (or string)
  - Optional and required arguments must be specified in both places

Using getopt (cont)

- Use getopt_long() to make longopts flags parse like shortopts

  short_opt = getopt_long(…) // commo usage
  n

- getopt_long() reads one option at a time
  - Use with a while loop to get multiple options
  - Options with arguments store the argument value in optarg, if none is included, optarg will be a nullptr
- Beware: When executing the program, multiple shortopts can be specified at once!

  -this is the same as -t -h -i -s

getopt_long() example

```
1  // Declarations before main()
2
3  #include <getopt.h>
4  // ...
5
6  static struct option longopts[]          = {
7      {"add",      no_argument,       nullptr,  'a'},
8      {"delete",   required_argument, nullptr,  'd'},
9      {nullptr,    0,                 nullptr,  0}
10 };
11
12 //  Declare function main() with parameters:
13 //      argc: count of arguments on the command line
14 //      argv: array of char pointers to the command line arguments
15 //            (an array of C-strings)
```

getopt_long() example (cont.)

```
1  int main(int argc, char *argv[]) {
2     int idx = 0;      // getopt_long stores the option index here
3     char c;
4
5     while ((c = getopt_long(argc, argv, "ad:", longopts, &idx)) != -1)
6     {
7        switch (c) {
8           case 'a':    cout << "Option 'a' specified." << endl; break;
9           case 'd':
10             cout << "Option 'd' specified." << endl;
11             cout << "Argument for 'd': " << optarg << endl;
12             break;
13       } // switch
14    } // while
15 } // main()
```

make

- Read the docs! https://www.gnu.org/software/make/manual/
- make is **NOT** just for building executables!
- The make command reads from Makefile by default
- A make rule combines a target, prerequisites, and a recipe

```
target:  prerequisite(s)
<Tab>  recipe
```

  - The target is the file that is generated when the rule is inovoked
  - The prerequisites are the file or files that are used to build a target
  - The recipe is the step or steps that are executed when a rule is invoked
- Phony targets do not create files named after the target (eg. all, debug, clean), they are a “recipe” only
- make rules often reference other make rules

Saving Memory in a `class/struct`

- Arrange member variables by size, from largest to smallest
  - Objects (such as `string`)
  - `double`
  - `int`
  - `short`
  - `char`
  - `bool`

Staff Tips

- When using dynamic memory, always write new and delete in pairs, inserting all the rest of your code between
  - Before & after a loop
  - In class constructor and destructor
- Extra command line options can be added for testing, converters, enabling verbose output, etc.
- Write functions to avoid code duplication
- Managing program data
  - Good: Class with 15 variables
  - Bad: Function(s) with 15 parameters
  - Worse: 15 global variables

Staff Tips

- Use the compiler flag –DDEBUG and #ifdef DEBUG

```
#ifdef DEBUG
#define _(args) args
#else
#define _(args)
#endif

_(cout << "debug-only message" << endl);
```

- Use the compiler flag -DNDEBUG and #ifdef NDEBUG

Staff Tips

- **LEARN YOUR TOOLS!**
  - Editor or IDE
  - GDB
  - Valgrind
  - Version control (eg. git, mercurial, bazaar)
  - Terminal or cygwin
  - ssh, scp, sshfs
  - make
  - Other *nix utilities (eg. diff, wc, grep, rsync)
- Spending 5-10% of your time every week or every session learning more about your tools and/or refining your workflow can eliminate hours of frustration

More Tips

- Don’t exit(0) at the end of your program! It ends immediately, destructors don’t get called, and you produce a memory leak
  - Use return 0 instead