Today’s code was pushed to your repository as directory CypherExample/

These slides will be posted after class
Stuff

- clint.py has been banished
- **ex03 has been delayed** (but it’s available in close to final form if you want to look early)
- Thank you for your cooperation and patience with the discussion board changes
  - I’m quite optimistic about it
  - We’ll actually implement soon, so please send/post suggestions
- Right now:
  - No assignment discussion, except as relayed from email questions by us
  - A social thread, for whatever
  - A “C language” thread for general (not assignment) discussion
Module Overview

- I’m going to design and build an app in multiple phases
- Each phase is intended to improve upon the previous one in some way
- We’ll have a few slides explaining the app and then sit in an editor looking at code while I explain what’s going on
- The code will be too complicated to completely absorb during the presentation, but will be available somehow shortly
- This module has no bounds
  - It’s about everything having to do with C programming...
  - Even though it doesn’t try to show “everything”
Module Overview (cont.)

- This module has no bounds
  - It’s about everything having to do with C programming...
  - Even though it doesn’t try to show “everything”
- Ask questions...
  - There are no bounds on questions either

- Claim: you now know pretty close to everything there is to know to understand any C program
  - You can understand them more easily with more experience
- The major challenge is to stay dumb
  - C isn’t complicated; it’s doing very simple things; it wants them to look like complicated things, and that’s what gets you into trouble/confusion
- When confused, ask yourself “what is the type of this subexpression?”
The Application: Substitution Cypher

- I want to write C code that will encode a text string by substituting every occurrence of each letter with some other letter
- For example, ‘a’ -> ‘,’ and ‘B’ -> ‘a’ and ...
- My application needs two things
  - The string to be encoded
  - The mapping from source character to encoded character
- “Project 1” develops a sequence of decisions about where those two things come from, and implements them
- “Project 2” re-implements the functionality of the final Project 1 design, but in a much nicer way
- “Project 3” addresses not design or coding, but build
The Basic Idea

Input string

Translation Table

Output string
Project 1, Version 1

- Major decisions:
  - The string to be encoded will be embedded in the software as a literal
  - The translation table (a char [256] array) will be initialized by code

- Implications
  - Easy to write!
  - To translate their own input string, a user would have to modify and code!
  - To translate my string but with a different translation table, a user would have to modify and build C code!
Project 1, Version 1

Encode.c

main.c
cypher
Input string la la

Decorated output
Project 1, Version 2

Encode.c

FileToString.c

File contents

Input file

main.c

cypher

Decorated output
Project 1, Version 3

Encode.c

FileToString.c

CreateCypher.c

main.c

Input file

Cypher file

Decorated output
Project 2

- Project 1 V3 source but build infrastructure

![Project 3 Structure]

- App
  - `makefile`
  - `main.c`

- Cypher
  - `makefile`
  - `Cypher.c`
  - `Cypher.h`

- FileToString
  - `makefile`
  - `FileToString.c`
  - `FileToString.h`
Project 3

- “Object oriented C”
- Don’t read entire file into memory!

```
Project 3

makefile

App

makefile
main.c

Cypher

makefile
Cypher.c
Cypher.h
```
Project 4

- gnu implementation of \textit{tr} utility
  - TR(1) \hspace{1cm} User Commands \hspace{1cm} TR(1)

- NAME
  - tr - translate or delete characters

- SYNOPSIS
  - tr [OPTION]... SET1 [SET2]

- DESCRIPTION
  - Translate, squeeze, and/or delete characters from standard input, writing to standard output.

- Source Code file: \textit{tr.c}