CMSI 284

COMPUTER SYSTEMS ORGANIZATION/ SYSTEMS PROGRAMMING

Spring 2010

Assignment 0318

This assignment provides some intermediate-level C programming exercises.

Not for Submission

Review the "Introduction to C" notes from Dr. Toal's web site as needed. A direct link can be found on the course web site.

By March 9

Work on at least one or two of the exercises so that, if needed, you can ask questions during Tuesday's class.

By March 16

You should be putting finishing touches to your programs at this point. This class is your last opportunity to ask questions before the due date.

For Submission

1. Write a C program that takes, as an argument, a 26-character string that represents a "cipher" for letters of the alphabet. The program then "encodes" standard input according to this cipher and produces the "encrypted" version. Upper- and lowercase characters should encrypt to the same cipher symbol.

   For example, an argument (cipher) of `p!q@o#w$i%e^u&r*t(yans<>/;` would encode `Hello world` as `$o^^r <r(^@`.

   Call this program *secret-encoder-ring.c*.

2. Write a C program that takes the same argument and standard input as the previous one, but *decrypts* the input according to the cipher. Use a BULLET character to indicate symbols that were not part of the cipher. Call this program *secret-decoder-ring.c* (duh).

3. Write a C program that takes an integer as an argument (positive or negative) and expects a sequence of chords separated by spaces (`C F G Dm Am` etc.) as standard input. The program *transposes* the chord sequence according to the interval indicated by the integer. Use `#` for sharps and `b` (lowercase "B") for flats.

   For example, an argument of –2 would transpose `A D F#m C#m Bm E` into `G C Em Bm Am D`. Call this program *transposer.c*.

4. Reimplement your *calculator.c* program, calling it *calculator-plus.c*, so that it uses a "library" of operators using these definitions:

```
typedef int (*BinaryOperation)(int, int);
struct OpDefinition {
    char *operatorSymbol;
    BinaryOperation function;
};
```

   Instead of an *if* or *switch* statement, this new calculator iterates through an array of *OpDefinition*s until it finds one that matches the operator given in the arguments. It then uses the *function* attribute of the *OpDefinition* to perform the calculation. Standardize the output to look like this:

```
operand1 operatorSymbol operand2 = result
```

5. Implement a *dictionary* "module" that creates a dictionary from a text file consisting of individual words. Include functions for adding, removing, and looking up words.

   Provide a test program and dictionary file that verifies the functionality of the module. Call it *spelling-dictionary-test.c*.

   Finally, provide a "spell-checker" program that takes a "dictionary file" as an argument, then checks standard input for words in the dictionary. If a word is not in the dictionary, the program should ask the user if this word should be added to the dictionary. Subsequent encounters of that word should then pass the spell-check. Call this program *spell-checker.c*.

   Include a *Makefile* to facilitate easy building of your module and accompanying programs.

Commit the first four programs to CVS under the directory */homework/cmsi284/c-major*, using the given filenames. The fifth program, with its multiple files, should go in */homework/cmsi284/spelling*.