Due Thursday, 28 Oct 1999, at the beginning of class.

1 Reading

Read Sections 4.2, 5.1–5.3.

2 Compilation Using Xemacs Tip

Last week, we told you about compiling programs using commands such as `make hello`. This week, we talk about the Xemacs support for compiling programs while still using Xemacs, rather than in the shell.

To compile a program inside Xemacs, type `ESC` than `x` and then `compile`. Then type the desired file, e.g., `hello`. (This assumes you created a Makefile as described in last week's compilation tip.)

If the program has no errors, an executable program called `hello` will be created. Otherwise, one can walk through the error and warning messages using Control-x ‘. (That is a backtick near the upper left of the keyboard.) The cursor will move to the offending line in the program.

3 Problems

When submitting a program, please submit a printed copy of the program's code prepended with comments briefly describing its input and output. Please indent your code to make it readable. See Section 2.5 of the textbook for style hints. If you desire, you can also submit examples demonstrating your program's correctness.

1. In this problem, we will construct a program similar to the UNIX `grep` command. Given a pattern, i.e., a string, and a filename, grep prints all the lines that contain the pattern. For example, given a pattern of `1` and this file

   ```
   1776
   hello, world
   21478756282
   ```

   The program should print the first and third lines because these lines contain the pattern `1`.

   Your program should ask the user for a pattern and a filename. Then it should print each line that matches the pattern.

   When possible, define a function for each subtask. We have not yet learned how to pass `ifstream` s and `ofstream` s as function arguments so this restricts the functions you can write.[1] It may be useful to declare a function that determines whether the pattern occurs in the string. Be sure to check for the case that the pattern is longer than the string.

[1] The difficulty with passing an `ifstream` or `ofstream` as an argument is that a copy of the stream is created. However, this is not permitted. Suppose it was permitted. A function would use a copy of the `ifstream` to read the characters. After the function returns, the original `ifstream` would yield exactly the same characters!

You may assume that each line of the file, even the last line, ends with a newline character `'\n'`.

Clarification (Added 1999 October 25)

Assume the pattern contains no whitespace.

This program is similar to the "find" function on a word processor. You choose find, type a pattern, and then the word processor shows you all the matches. grep is similar except it shows you all lines that contain the pattern.

Suppose we have a file named `foo` that contains the lines

```
1776
hello, world
21478756282
```

Here are some examples how the program should run:

```
janus00> a.out
What is the pattern? 1
What is the file? foo
1776
21478756282
janus00> a.out
What is the pattern? hello
What is the file? foo
hello, world
janus00> a.out
What is the pattern? tr
What is the file? foo
janus00> a.out
What is the pattern? wor
What is the file? foo
hello, world
janus00> a.out
What is the pattern? 21
What is the file? foo
21478756282
```

The program asks for a pattern and a name of a file. Every line in the file that matches the pattern should be printed out. For example, the pattern "hello" occurs in the middle line of the file `foo`. For example, the pattern "tr" does not occur in any of the lines.

The phrases "What is the pattern? " and "What is the file? " should be printed by the program. Everything else on that line is typed by the program's user.

2. Currency Conversion Revisited

   In a previous homework we wrote a simple currency-conversion program. Its user interface was less than ideal, in part because we did not have the tools to deal with text strings or obtaining input from more than one source. For this homework, you are to write an improved currency-conversion program that will read a similarly-formatted list of currencies and conversion factors from a file (filename to be provided by the user) and then read a sequence of conversions from standard input. The program should work as follows:

(a) Prompt the user for the name of the file containing the list of exchange rates. As in the previous homework, each line of this file will contain a "from" currency, a "to" currency, and an exchange rate, but this time the currency names will be strings. For example, the file might contain these lines:

```
dollars     pounds      0.25
pounds      dollars     4.0
dollars     rubles      1000.0
yen         dollars     10.0
```

(b) Prompt the user repeatedly to enter one of the following commands:

- `s` to show all exchange rates in the file.
- `c` to perform a conversion.
- `q` to quit the program.

The program should quit in response to a `q`, print out all exchange rates, one per line, in response to a `s`, and in response to a `c`, it should do the following:

i. Prompt the user for "from" and "to" currencies.
ii. Look up the corresponding exchange rate in the list.
iii. If found, prompt the user for an amount to convert, do the conversion, and print the result. If not found, print an appropriate error message.

The program should not make *any* assumptions about the number of exchange rates. (Hence it should not use arrays to store all of the exchange rates. It is acceptable to use arrays to store strings.)

Hints, tips, and other help

- *Hint:* There is nothing that prevents a program from reading the same file over and over.
- In response to a `s` command, your program should print all the exchange rates, one per line. The format of this output does not need to be fancy; for the input file described previously, the following output is perfectly okay:

```
dollars pounds 0.25
pounds dollars 4.0
dollars rubles 1000.0
yen dollars 10.0
```

  Making things line up in columns, as in the input file, is an interesting challenge if you have time but is not required.

- You are free to obtain data for the table of exchange rates from any source you like, or you may use file http://www.cs.trinity.edu/~joldham/1320/hw/hw7/sampleRates (data for which was obtained from Yahoo's Finance Site).