COSC 3750
Linux Programming
Homework 4, Write "wycat"

1 Intro

Now we write a simple utility. This is a start at writing C programs and using basic file I/O. It will also include a Makefile. This program MUST compile and run on the departments Linux machines. I do not care if if works on a Mac, or some other Linux/UNIX distribution, but it MUST work on our machines. That means you had better test it. Oh, and instead of searching for weird ways to accomplish something, read the **man** pages, the **make** manual, and ask me questions.

For this and all future programs you WILL have an introductory comment heading in each file that you turn in. This heading will look similar to

```
/*
 * wycat.c
 * Author: Kim Buckner
 * Date: Feb 18, 2020
 *
 * COSC 3750, Homework 4
 *
 * This is a simple version of the cat utility. It is designed to
 * ...
 *
 */
```

If you do not have the asterisks on all the lines that is fine. My text editor does that for me. "All files" includes the Makefile, all source code (including header files), and any text files I ask you to submit. In the Makefile, every comment line must start with a # at the left margin. Please leave at least one space between the comment markers and the text. That makes this easier to read.

2 Your job

Write a C program. This program will be called "wycat". It will be a version of the standard system utility "cat". To accomplish this you will:

- You will create a simple text file named exactly **hw4_pcode.txt** that contains a pseudocode version of the program. If you are smart, you will create this file first, instead of after you have done all the other work. Plan for programming. Pseudocode is NOT A BUNCH OF C code. It is plain text. Yes, you can use words like if and while, but plain text. Read the **Pseudocode Guide**.
- Create a Makefile, named exactly **Makefile**.
  - It will have at least one target, wycat, that is dependent on wycat.c and generates the executable, **wycat**.
  - If I type *make* with your makefile and source code in a directory, the executable **wycat** will be the only thing generated.
  - That means do not bother to create an object file first, totally unnecessary in this situation.
  - Your makefile can have targets for *clean* and maybe *tidy* but those are the only ones other than *wycat*. Besides removing the executable, "wycat", clean (or tidy if you have it) will also remove any coredump files that are generated. There will be a correct **.PHONY** target to account for *clean*, *tidy* and any other PHONY targets in the file.
  - You will correctly use the variables `${CC}$` and `${CFLAGS}`. Make sure that the flags include `-ggdb`. That and `-Wall` should be the only ones needed here.
- Create a C source code file named exactly **wycat.c**.
- The compiled program, "wycat" will
  1. If "wycat" gets no arguments on start up it will read from standard input and wite to standard output.
  2. Otherwise process, without modifying them in any way, its arguments via **argc** and **argv**.
  3. Arguments will be assumed to be filenames. If some argument cannot be opened, a suitable error message will be printed and the program will **continue**. Try running the system utility "cat" and see what it does with a bad filename.
  4. If some argument is "-" (dash) the program will process standard input. This may occur any number of times among the arguments. Processing of standard input is up to you. There are two ways to do it: read and write a line at a time (what "cat" normally does) or read and write larger blocks of data. DO NOT READ STDIN 1 CHARACTER AT A TIME!!!!!
  5. Read all files (and stdin) in their entirety. All data will be written to standard output. If reading from a file (you must use *fread()*, you will NOT read line-at-time because there is no way to know if there are any line terminations. You must read files using some reasonably sized (like 4096 bytes) buffer.
  6. You will **not** write file data with *printf()*. You must correctly use *fwrite()*.

7. Make sure that you check (not just save) the return values on all function calls. This is how you determine things like "end-of-file" correctly. The exception to this is `printf()`, that return does not really mean much. And no, you do have to check *void* functions.
8. Close all files (except for **stdin**, **stdout**, and **stderr**) opened by the process as soon as possible.

Examples:

$> `wycat f - g`
Output f's contents, then standard input, then g's contents.

$> `wycat`
Copy standard input to standard output.

NOTE: Remember that when you are typing input at the terminal for **wycat** (or cat for that matter), typing CTRL-D as the first thing on a blank tells the O/S to give the program an "end-of-input". You can do that any number of times for a process as the input is NOT actually closed.

3 What to turn in

You will upload a tar archive of the three files: **wycat.c**, **hw4_pcode.txt**, and **Makefile**.

Extracting the archive and then executing *make* in my directory containing the makefile and the source code file will:

1. generate only the executable **wycat** and
2. **not** generate **any** warnings (because you will have fixed your code) or errors.

Your tar archive will NOT CONTAIN any directories, only the 3 files specified. It will have an extension of **.tar** or if compressed using the -z option, **.tgz**.