C Lab 1

Narahari (probably a long time ago)
Shepherd, Preisner, and Coplan (2018)
Morin (2020)

Makefiles

If you run `make` , this program will look for a file named `makefile` in your directory, and then execute it.

The basic makefile is composed of:

```
target: dependencies
[tab] system command
```

Example `makefile`:

```
all:
    g++ helloWorld.c -o helloWorld
```

Makefiles

Sometimes is useful to use different targets. This is because if you modify a single file in your project, you don't have to recompile everything, only what you modified.
Here is an example:

```
all: hello

hello: main.o factorial.o hello.o
	g++ main.o factorial.o hello.o -o hello

main.o: main.cpp
	g++ -c main.cpp

factorial.o: factorial.cpp
	g++ -c factorial.cpp

hello.o: hello.cpp
	g++ -c hello.cpp

clean:
	rm *o hello
```

Now we see that the target all has only dependencies, but no system commands. In order for make to execute correctly, it has to meet all the dependencies of the called target (in this case all).

Each of the dependencies are searched through all the targets available and executed if found.

In this example we see a target called clean. It is useful to have such target if you want to have a fast way to get rid of all the object files and executables.

C - File input/output (I/O)

- The standard I/O functions are declared in the ***<stdio.h>*** header file
- **putchar** - displays an ASCII character to the screen
- **getchar** - reads an ASCII character from the keyboard
- **printf** - displays a formatted string
- **scanf** - reads a formatted string
- **fopen** - open/create a file for I/O
- **fprintf** - writes a formatted string to a file
- **fscanf** - reads a formatted string from a file

Text Streams/Buffers

- All char-based I/O in C is performed with text streams
- A stream is a sequence of ASCII characters
- Characters are processed in the order in which they were added to the stream
- Standard input stream (keyboard) is called **stdin**
- Standard output stream (monitor/display) is called **stdout**

Character I/O

- putchar(c) - adds one ASCII character (c) to stdout
- getchar() - reads one ASCII character from stdin
- These functions deal with “raw” ASCII characters; no type conversion is performed
  - `Char c = ‘h’;`
  - `putchar(c);`
  - `putchar(‘h’);`
  - `putchar(104);`
- Each of the above calls prints ‘h’ to the screen

Buffered I/O

- Keyboard input stream
  - Characters are added to the buffer only when the newline character (the “Enter” key) is pressed
  - This allows user to correct input before confirming with Enter
- Output stream
  - Characters are not flushed to the output device until the newline character is added

Input Buffering

- `printf("Input character 1:\n");`
- `Char inChar1 = getchar();`
- `printf("Input character 2:\n");`
- `Char inChar2 = getchar();`
- After seeing the first prompt and typing a single character, nothing happens
- Expect to see the second prompt, but the character is not added to stdin until Enter is pressed
- When enter is pressed, newline is added to the stream and is consumed by second getchar() so in
  Char2 is set to '\n'

Output Buffering

- `putchar('a');`
- `/* generate some delay */`
- `For (i = 0; i < DELAY: i++) sum += i;`
- `putchar('b');`
- `putchar('\n');`
  - User doesn't see any character output until after the delay
  - 'a' is added to the stream, but the stream is not flushed (displayed) until '\n' is added

Formatted I/O

- Printf and scanf allow conversion between ASCII representations and internal data types
- Format string contains text to be read/written, and formatting characters that describe how data is to be read/written
  - %d signed decimal integer
  - %f signed decimal floating-point number
  - %x unsigned hexadecimal number
  - %b unsigned binary number
  - %c ASCII character
  - %s ASCII string

Special Character Literals

- Certain characters cannot be easily represented by a single keystroke
- Some of these are represented as shown:
  - \n newline
  - \t tab
  - \b backspace
  - \\ backslash
  - \’ single quote
  - \” double quote

scanf

- Reads ASCII characters from stdin
- `char name[100];`
- `int bMonth, bDay, bYear;`
- `double gpa`
- `scanf("%s %d/%d/%d %lf", name, &bMonth, &bDay, &bYear, &gpa);`

File I/O

- A file can be thought of as a sequence of ASCII characters stored on some device
- Each file is associated with a stream
  - May be input stream, output stream, or both
- The type of stream is a “file pointer,” declared as:
  - `FILE *infile;`
- FILE type is defined in stdio.h

fopen

- The fopen function associates a physical file with a stream
  - `FILE *fopen(char* name, char* mode);`
- Name - the name of the physical file (if it's within your present working directory) or the location of it on your computer - formatting is dependent on the underlying OS
- Mode - how the file will be used
  - 'r' - read from the file
  - 'w' - write, starting at the beginning of the file
  - 'a' - write, starting at the end of the file (append)

Fprintf and fscanf

- Once a file is opened, it can be read or written using fscanf() and fprintf(), respectively.
- These are just like scanf() and printf() except an additional argument specifies a file pointer
  - `fprintf(outfile, "The value read is %d\n", x);`
  - `fscanf(infile, "%s %d/%d/%d %lf", name, &bMonth, &bDay, &bYear, &gpa);`

Simple File I/O Example

- `FILE *inptr; // contains an int at first char`
- `FILE *outptr;`
- `int a;`
- `Inptr = fopen("input-filename.txt", "r");`
- `outptr = fopen("output-filename.txt", "w");`
- `fscanf(inptr, "%d", &a);`
- `fprintf(outptr, "Value of A is %d \n", a);`

String functions

- C library string.h has functions that perform operations on strings
- Strings are null terminated arrays of characters

strsep

- Run “man strsep” in the terminal
- **Key Takeaways:**
  - **#include <string.h>**
  - char * **strsep**(char **stringp, const char *delim);
- Pass in a pointer to your string and a delimiter (i.e. “,” for a string such as “one,two,three”)
- Strsep will return a pointer to each portion of your string separated by the delimiter, and NULL when there are no portions left.
- For **strsep(“one,two,three”, “,”)**, it will return “one”, “two”, “three”, NULL for the first 4 calls to the function
- Note that strsep will manipulate the stringp pointer argument

String functions

- Strcmp: string compare
  - `int strcmp(char *a, char *b)`
  - Compares string a with string b
    - Returns 0 if they are equal
    - Returns value greater than 0 if A is lexicographically greater than B
    - Returns values less than 0 if A is lexicographically smaller than B
- Strcpy: to copy string a to string b

Lab Exercise

1. Download lab11.zip and unzip it
    a. There should be 5 files: Makefile, fileIO-ex1.c, strsep.c, file1.txt, file2.txt
2. Examine the Makefile
    a. What are the targets and do they do?
    b. Run "`make build`" What happened?
    c. Run "`make clean`" What happened?
3. File I/O Example
    a. Read the code in fileIO_example.c and the contents of file1.txt and file2.txt
    b. Run "`make fileIO`"
    c. What happened? Did any of the files change? Are there any new files?
4. String function Example
    a. Read the code in strsep_example.c
    b. Run "`make strsep`"
    c. What happened?

Project Time