Introduction to Computer Science and Programming in C

Session 10: October 2, 2008

Columbia University

Announcements

- Homework 2 is out. Due 10/14 before class
- Midterm Review on 10/16, exam on 10/21

Review

- struct - data structures containing multiple variable fields
- union - variables of different types that share the same memory
- typedef - define a new type
- enum - type representing discrete symbols
- Programming tips

Today

- File I/O

Review of Basic I/O

- `printf("formatted text", arg1, arg2,...);`
  - Related to `fprintf(), sprintf()`
- `fgets(string, sizeof(string), stdin);`
  - Related to `fgetc()`
- `sscanf(string, "formatted text", &var1, &var2...);`
  - Related to `fscanf(), scanf()`

File I/O

- So far, we have received input from the keyboard and output to the terminal.
- Bigger programs perform computation on larger amounts of input and output.
- We need to be able to read from and write to files.

File I/O Examples

- “pico hello.c” – pico reads hello.c as output, then when you save, outputs the text to a file.
- “gcc hello.c -o hello” – gcc reads hello.c, converts to machine language, and outputs to hello

FILE

- stdio.h defines a special type, FILE
- Variables of type FILE are always declared with an asterisk * before its name
- FILE *input_file;
- (This is because they are pointers. More on that after the midterm)

fopen()

- FILE variables have a special initialization function: fopen();
- `fopen(<file name>, <mode>);`
- <file name> – the filename as you would reference it in Unix (i.e. “hello.c”)
- <mode> is a code for how you will use file: read (r), write (w), binary (b)

fopen() Examples

- Read a list of words from “names.txt”:
  - `input_file = fopen(“names.txt”, “r”);`
- Write the solution to Hanoi with 10 discs:
  - `output_file = fopen(“hanoi10.txt”, “w”);`
- Replace “teh” with “the” in “essay.txt”
  - `text_file = fopen(“essay.txt”, “wr”);`

fclose()

- When done with a FILE, call fclose() to tell the Operating System we're done.
- fclose(<FILE variable>);

Writing to output

- `fprintf(<target file>, “formatted text”, arg1,...);`
- format text just like printf() with placeholders
- `fprintf(stdout, “text”, args);`

  is equivalent to

  `printf(“text”, args);`

Writing to output

- `fputc(<character>, <file>);`
  - writes a character to a file
- `fputs(<string>, <file>);`
  - writes a string to a file
- Why would we use these instead of fprintf()?

Reading from input

- `fscanf(<file>, “formatted text”, &arg1, &arg2,...);`
  - returns an int: number of arguments successfully converted, or End of File (EOF)
- `fgets(<file>, sizeof(<string>), <string>);`
  - Just like before; we used **stdin** as the file
- `fgetc(<file>);`
  - Returns an int (EOF or convert to char)

Buffered Output

- OS often stores output in buffer before writing to the actual file.
- Writing to file is relatively slow, writing to buffer is fast (buffer is in RAM)
- Force OS to write buffer to file using fflush()

```
fflush(<file>);
```

Summary of Functions

Name	Input	Output
fprintf()	formatted text + args	file
printf()	formatted text + args	stdout
sprintf()	formatted text + args	string
fputc(), fputs()	char, string	file
fscanf()	file	formatted text + args
scanf()	stdin	formatted text + args
sscanf()	string	formatted text + args
fgetc(), fgets()	file	(char) int, string

File Formats

- Standardized format for organizing data in a file
- Simple example, homework 2, problem 4:

```
<float>
<float>
<float>
<float>
...
```

File Formats

- Data files representing more complicated data structures can require more complicated formats
- Often files have **headers**. For example, storing a 2-d array in a file:

```
rows: <number of rows>
cols: <number of columns>
<float> <float> <float>...
<float> <float> <float>...
...
```

File Formats

- Ideally, format should be readable by humans and by computer programs
- Computer programs are not very robust, so must be specific (i.e. tab versus spaces)
- When you have huge amounts of data, you can give up on human-readability and use binary format for efficiency (read about it in the book)

Reading

- Practical C Programming, Chapter 14
- The C Programming Language, Section 7.5