ECS 36A, May 24, 2023

Announcements

1. Midterm statistics: mean, 60.89; median, 62; max, 94; min, 27; standard deviation 13.14
2. ***Do not panic!*** Even though the midterm grades are not curved, the final course grade will be, and the curving method will be independent of your class standing; it will solely depend on *your* grade.

Background

- System calls: interfaces to operating system functions
- Example: some Linux system calls
  - I/O: reading, writing, networking, etc.
  - Files: chown, chgrp, stat, etc.
  - Resource usage: ulimit, getrlimit, etc.
  - Timing: gettimeofday, time
- Library functions provide system-independent interface to them
  - Also provide other features

C Library Functions

- The C library provides many functions that do useful things
  - Standard I/O C library
  - Math library
- Character type
- String to integer or float/double types
- Handling options
- Time
- Random numbers
- String and memory manipulation

Standard I/O Functions

- Implements open, read, write, close, and others
- *Requires* #include <stdio.h>
- Basis: streams or files
  - Usually FILE * types
  - Buffers input, output
  - Predefined streams: stdin (input), stdout (output), stderr (error output)

Buffering

- For efficiency; goal is to reduce number of read, write system calls
- On read, the library reads a block of data
  - The number of bytes in a block here depends on the system
  - This is *not* the same thing as a block in a program; it's a chunk of data
- The library then returns the amount of data requested, and keeps the rest in memory
- On next library call, it returns the next byte *without* doing another call to system
- This explains why *ungetc*() can only guarantee one char of pushback

Full Buffering in Standard I/O Library

- Typically used when reading/writing files
- Read: call to system call fills buffer; next call is when a read occurs and buffer is empty
- Write: call to system call empties buffer; next call is when a write occurs and the buffer is full
- Flushing: emptying the buffer; as noted, done automatically
  - Use *fflush*() to do this manually
- On exit or return from *main*(), all buffers are flushed

Line Buffering in Standard I/O Library

- Typically used with line-oriented devices such as terminals
- Buffers flushed when newline encountered *or* buffer is full
  - Doesn’t matter if buffer is for reading or for writing
  - Also output is flushed when process reads from a line-buffered or unbuffered stream
- Idea is to act like fully buffered I/O, except that reading/writing in blocks is infeasible, as process can’t read a terminal beyond what has been typed
- On exit or return from *main*(), all buffers are flushed

Unbuffered Streams in Standard I/O Library

- Don’t buffer anything
- On input, byte *immediately* made available to process
  - Terminals usually need to be put into a special mode (called ``raw’’ mode) in which no character processing is done; usual mode is called ``sane’’ or ``cooked’’
- On output, character is *immediately* written to device or file

Useful Functions: Positioning for Read/Write

- Every stream has a *read/write pointer* (*rw-pointer*) pointing to where the next byte is to be read or written
- fgetpos(*fp*, *pos*): gets current position *pos* of rw-pointer of *fp*
  - ftell(*fp*, *pos*): return position of rw-pointer of *fp*
- fgetset(*fp*, *pos*): set current position *pos* of rw-pointer of *fp*
  - rewind(*fp*): reset rw-pointer to 0 (the beginning of the file)
- fseek(*fp*, *offset*, *whence*): set current position of rw-pointer of *fp* to *offset* bytes from *whence*
  - *whence* is SEEK_SET (beginning), SEEK_CUR (current position), or SEEK_END (from the end)

Strings to Numbers

- int atoi(char *_str_), long int atol(char * _str_): convert _str_ to int or long, respectively
- double atof(char * _str_): convert _str_ to double
- No error checking
  - If any non-digit or non-floating character is found, these stop converting and return what they have converted

Strings to Numbers

- long int strtol(char **str*, char ***eostr*, int *base*): convert *str* to long it in base *base*; return a pointer *eostr* to first char that is not converted
  - If *eostr* is NULL, the end pointer is not returned
  - If **eostr* is '\0', there were no errors
  - If no digits, the value of *str* is put into **eostr* and returns 0
- float strtof(char **str*, char ***eostr*): convert *str* to float
- double strtod(char **str*, char ***eostr*): convert *str* to double

Processing Options

- int getopt(int *argc*, char ***argv*, char **optstr*): process the arguments in *argv* looking for those that begin with “-” or “--” and are in *optstr*
  - If character in *optstr* followed by ”:”, the option has an argument
- External variables are char **optarg*, int *optind*
- Returns the character option
  - Or –1 if there are no more options
- On return, *optind* is index of *next* option to be processed and *optarg* points to the option’s argument (if any)