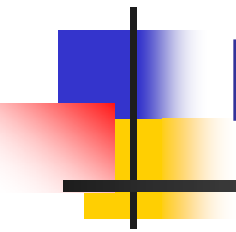
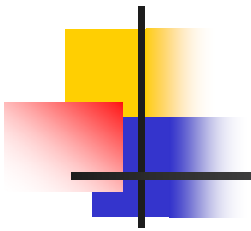


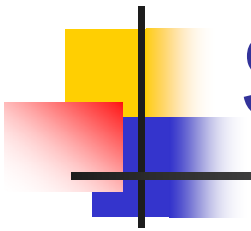
Chapter 5: Standard I/O Library

CMPS 105: Systems Programming
Prof. Scott Brandt
T Th 2-3:45
Soc Sci 2, Rm. 167



Introduction

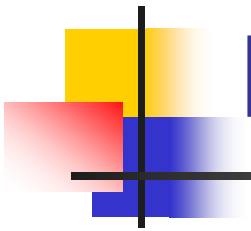
- The Standard I/O library
  - Is a library of user-level functions
  - Runs as part of application programs
  - Serves as a layer between apps and the OS system calls
  - Implements read and write buffering
  - Deals with details like block sizes



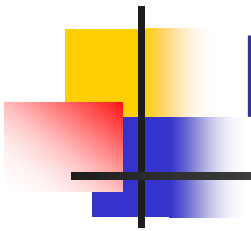
Streams and FILE Objects

- File I/O (from Ch. 3)
  - System calls
  - Uses file descriptors to identify which file
  - No buffering – direct file access
- Standard I/O
  - User-level library
  - Uses streams
  - Uses FILE pointers to identify files
  - Buffered

Standard Input, Output, and Error

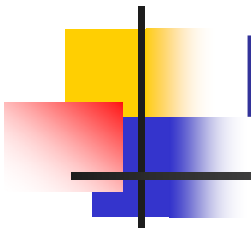


- Three predefined streams
  - Standard Input: stdin
  - Standard Output: stdout
  - Standard Error: stderr
- Defined in <stdio.h>
- These refer to the same “files” as the three file descriptors
  - STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO



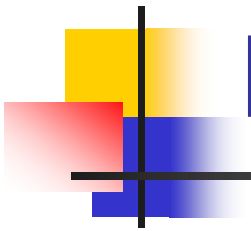
Buffering

- Goal: Reduce number of read() and write() system calls
  - And thereby reduce I/O overhead
- Buffering automatic for each I/O stream
- Three types of buffering
  - Fully buffered
  - Line buffered
  - Unbuffered



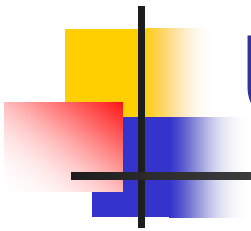
Fully Buffered

- Typically used with files
- Actual I/O occurs when
  - Reading: buffer is empty and needs to be filled
  - Writing: buffer is full and needs to be emptied
- *Flushing*: writing the buffer to disk
  - Automatically – when the buffer is full
  - Manually – when fflush() is called



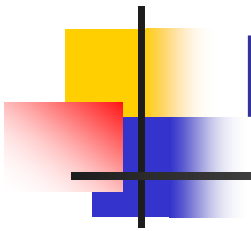
Line Buffered

- Typically used with terminal devices
- Actual I/O occurs
  - When a newline character is encountered on input or output
  - Allows for character-at-a-time application output without excessive I/O overhead
- Caveats:
  - Since buffer size is fixed, output might occur before newline
    - If buffer fills up, it has to be written
  - All line-buffered output buffers are flushed whenever input is requested from either
    - an unbuffered stream, or
    - a line-buffered stream (that requires data to be requested from the kernel



Unbuffered

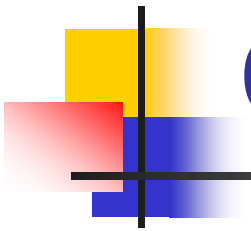
- Used with standard error stream
  - Causes output to be displayed immediately
- May be used elsewhere
- No buffering is performed



Buffering requirements

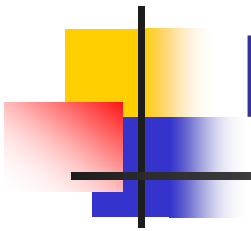
- ANSI C
  - Standard input and output are fully buffered, if and only if they do not refer to an interactive device (like a terminal)
  - Standard error is never fully buffered
- SVR4 and 4.3+BSD
  - Standard error is always unbuffered
  - Terminal device streams are line buffered
  - All other streams are fully buffered

Changing buffering



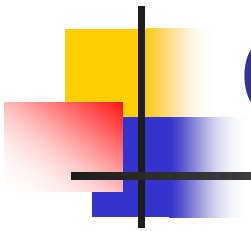
- #include <stdio.h>
- void setbuf(FILE **fp*, char **buf*);
  - Toggles buffering (i.e. turns buffering on or off)
  - Usually all we need
- int setvbuf(FILE **fp*, char **buf*, int *mode*, size_t *size*);
  - Sets buffering to a particular type:
    - Fully buffered: _IOFBF
    - Line buffered: _IOLBF
    - Unbuffered: _IONBF

Flushing the buffers

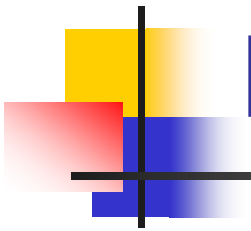


- #include <stdio.h>
- int fflush(FILE *_fp_);
  - Flushes specified stream
  - If fp = NULL, flushes all output streams
- Can be used to force data to be written
  - Timely output (must be output now)
  - Output with a required order
  - Critical output (must be output before program continues)

Opening a stream



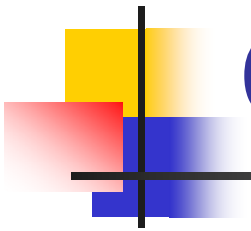
- #include <stdio.h>
- FILE *fopen(const char **pathname*, const char **type*);
  - Open the specified file
- FILE *freopen(const char **pathname*, const char **type*, FILE **fp*);
  - Open the specified file using the specified stream
- FILE *fdopen(int *fildes*, const char **type*);
  - Create a stream to correspond to the specified file descriptor (obtained from an open() call)



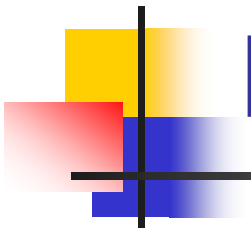
Details

- Types: r (read), w (write) , a (append), r+ (read/write), w+ (truncate/read/write), a+ (seek to end/read/write)
- When a file is opened for reading and writing
  - Input cannot directly follow output without an intervening fflush(), fseek(), fsetpos(), or rewind()
  - Output cannot directly follow input without an intervening fseek(), fsetpos(), rewind(), or an input operation that encounters an end of file
  - Otherwise, data can be lost
- Note: can't specify file permissions
  - They default to RW for user, group, and other
- Buffering can be changed only after open and before first access

Closing a stream



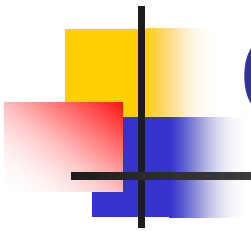
- #include <stdio.h>
- int fclose(FILE *_fp_);
- When a process exits normally, all streams are automatically closed
  - But not when a process crashes
- When an output stream is closed, all buffered data is flushed
  - When a process crashes, buffered output data is lost



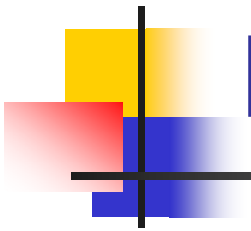
Reading and Writing Streams

- Three types of unformatted I/O
  - Independent of buffering options!
  - Character at a time I/O
    - Read/write one character at a time
  - Line at a time I/O
    - Read/write one line at a time
  - Direct I/O
    - Read/write one or more objects at a time

Character at a time input



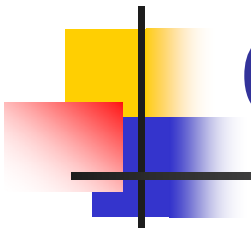
- #include <stdio.h>
- int getc(FILE *_fp_);
  - MACRO: Get one character from the specified stream
- int fgetc(FILE *_fp_);
  - FUNCTION: Get one character from the specified stream
- int getchar(void);
  - Get one character from stdin



More character at a time input

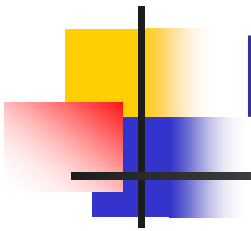
- Decoding errors
  - ferror(FILE *_fp_);
    - Returns true if the error was a real error
  - feof(FILE *_fp_);
    - Returns true if the end of file was reached
- int ungetc(int _c_, FILE *_fp_);
  - Forces one character back onto the specified stream
  - Usually used to check a character (i.e. “did we reach a space?” without consuming it)

Character at a time output



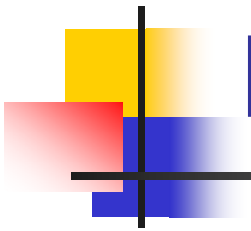
- #include <stdio.h>
- int putc(int *c*, FILE * *fp*);
  - MACRO: Put one character (note that it is an int) onto the designated output stream
- int fputc(int *c*, FILE * *fp*);
  - FUNCTION: Put one character (note that it is an int) onto the designated output stream
- int putchar(int *c*);
  - Put one character (int) onto stdout

Line at a time input



- #include <stdio.h>
- char *fgets(char **buf*, int *n*, FILE **fp*);
  - Read one line from the specified input stream
  - Read until newline or until *buf* is full (*n*-1 characters);
- char *gets(char **buf*);
  - Read one line from stdin
  - Deprecated due to buffer overflow potential

Line at a time output

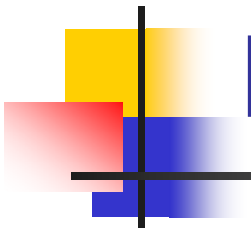


- #include <stdio.h>
- int fputs(const char *_str_, FILE *_fp_);
  - Put one line (null-terminated) onto the specified output stream
- int puts(const char *_str_);
  - Put one line (null-terminated) onto stdout

Standard I/O Efficiency

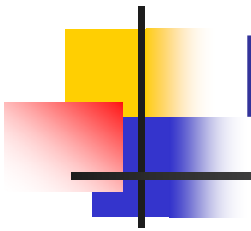
- Standard I/O can be more efficient because of buffering
- Can be less efficient because of extra code

Function	User CPU (seconds)	System CPU (seconds)	Clock Time (seconds)	Bytes of Program Text
Best file I/O	0.0	0.3	0.3	
fgets, fputs	2.2	0.3	2.6	184
getc, putc	4.3	0.3	4.8	384
fgetc, fputc	4.6	0.3	5.0	152
Worst file I/O	23.8	397.9	423.4	



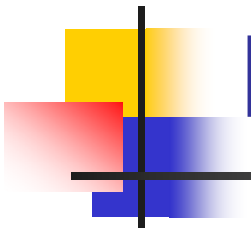
Binary I/O

- Section 5.9



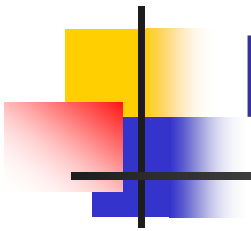
Positioning a stream

- Section 5.10



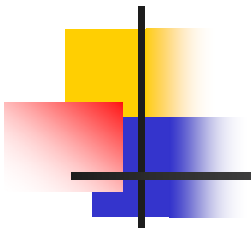
Formatted output

- Section 5.11



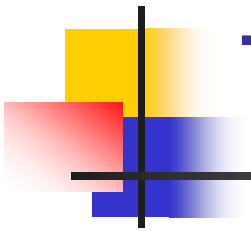
Formatted input

- Section 5.11



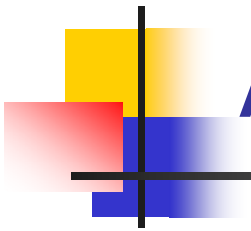
Implementation Details

- Section 5.12



Temporary Files

- Section 5.13



Alternatives to Standard I/O

- Section 5.14
- We didn’t discuss this in class