Announcements

- Program #1B
  - Due one week from today
- Reading
  - Chapter 15 (today)
    - skip 15.10.1 and 15.10.2 (scanf)
  - Chapter 8 & 9 (next Tuesday)
    - skip 8.1.4
    - skip 9.3

Project 1b - Notes

- Start NOW !!
  - it is long
  - start simple
    - write just assembly with a couple of commands
    - write just assembly with no errors
    - add labels
    - add errors in input
  - use disassemble to check output of assemble function
- Helpful Functions
  - int strncmp(char *s1, char *s2, int max);
  - int strcpy(char *s1, char *s2);
  - int strncpy(char *s1, char *s2, int max);
  - int atoi(char *);

I/O

- void perror(char *message);
  - print out the most recent error encountered by a library
  - prints message: <error description>
  - Example:
    - We tried to do something with the file foo.c and it didn't exist
    - perror("foo.c") would print foo.c: File Not Found
- Terminating execution
  - void exit(int status);
  - ends the program at that spot (no need to return from main)
  - status is available to other programs to indicate why the program ended
    - exit (0) is typical for no error
    - exit(-1) or some other non-zero value for an error
      - negative is common

Standard I/O Library

- #include <stdio.h>
  - includes prototypes for standard I/O routines
- Most I/O is stream based
  - buffered to allow efficient operations
    - mostly to disks or networks
  - interactive I/O is normally flushed at useful points
    - printf("foo\n"); /* \n causes a flush */
    - or you can explicitly flush

FILE type

- used to describe an active I/O connection
- three default ones supplied on entry to main:
  - stdin
    - an input stream for default input
    - often keybaord
    - < redirection in UNIX
  - stdout
    - default output stream
    - often terminal window
    - > redirection in UNIX
  - stderr
    - default output stream for errors
    - often terminal window
    - >& redirection in UNIX

Opening Files

- FILE *fopen(char *name, char *mode)
  - name specifies the name of the file to open
  - mode specifies the access mode:
    - "r" - read (file must exist)
    - "w" - write (write, existing file deleted)
    - "a" - append (write, existing file left alone)
  - Check return code:
    - A Null return implies and error
    - use perror(…) to report the cause of the error
- Example:

```
FILE *fp;
fp = fopen("foo", "r");
if (!fp) {
   perror("foo");
   exit(-1);
}
```

Operations on FILE

- int fclose(FILE *fp);
  - close a file (flushing any output if needed)
  - on success, returns 0
    - **CAN FAIL!:** On AFS (e.g., WAM) files not written to server until close.
- Character I/O
  - int fgetc(FILE *stream);
    - return the next character from input (as an int)
    - returns EOF on end of file
      - EOF doesn't fit into a char
      - EOF is a #defined constant
      - check for EOF **before** assigning result to char
  - int getchar(void);
    - like fgetc
    - but reads always from **`stdin`**

Character I/O (cont.)

- int fputc(int character, FILE *stream);
  - write character to the destination named in stream
  - return EOF on failure, 0 on success
- int putchar(int character);
  - like fputc
  - but always uses **`stdout`**
- int ungetc(int character, FILE *stream);
  - if you read something, and want to put it back
  - next read will get this character
  - all implementations will support at least one ungetc
    - may support more, but not guaranteed
  - moving the file pointer will discard these
    - see fseek, etc. later in this lecture

Line I/O

- Often useful to read and process an entire line
- char *fgets(char *buffer, int bufferSize, FILE *stream);
  - read a line into buffer
    - at most bufferSize-1 characters
    - null byte added at end of buffer
  - reads from stream named as 3rd argument
  - returns NULL on error or end of line
  - on success returns buffer
- **gets: never use this function!**
  - **it leads to buffer overflow problems**
  - **class assignments will fail if you use it**
- int fputs(char *buffer, FILE *stream);
  - write a line of output to stream
  - returns EOF on error, non-negative value on success

Converting Input

- When you read data --- it is characters (in strings)
- int atoi(const char *str);
  - Convert characters in str into an integer
  - Handles sign
  - Ignores leading white space characters
  - Ignores non-numbers after integer
- long atol(const char *str);
  - Converts characters in str into a long integer
  - Handles sign
  - Ignores leading white space characters
  - Ignores non-numbers after integer
- double atof(const char *str);
  - Converts characters in str into a double
  - Handles sign
  - Ignores leading white space characters

Printf (the full story)

- int fprintf(FILE *stream, char *format, …);
  - sends output to stream
- printf(char *format, …);
  - sends output to stdout
- snprintf(char *buffer, int limit, char *format, …);
  - sends output to the character buffer
  - prints at most limit-1 characters to buffer
    - adds null to end of string
- **never! use sprintf (due to buffer overflow)**

printf format codes

- %c print argument as unsigned character
- %d print as decimal integer
- %u print as unsigned integer
- %x print as hex (use X for capital A-F in format)
- %e print in exponent form (6.02300e23)
- %f print in floating point format
- %s print as a string (null terminated character array)
- %% print a % (so %% prints on %)

Printf – Flags and Field Widths

- Flags
  - -
    - left justify the field (default is right)
  - +
    - for signed numbers force the + to be printed
  - #
    - for x and X ensures a leading 0x or 0X
- Precision
  - How many characters will be printed
  - Format is x.y
    - For int types
      - x is total width and y is numeric digits
      - Leading 0's are added to bring numeric digits up to y
    - For floating points numbers
      - X is total width and y is digits right of decimal point
  - Different ways of handling the when there isn't enough space

Format code examples (. indicates space)

Format Code	A	ABC	ABCDEFGH
%s	A	ABC	ABCDEFGH
%5s	….A	..ABC	ABCDEFGH
%.5s	A	ABC	ABCDE
%5.5s	….A	..ABC	ABCDE
%-5s	A….	ABC..	ABCDEFGH

Format	1	-12	12345	123456789
%d	1	-12	12345	123456789
%6d	…..1	…-12	.12345	123456789
%.4d	0001	-0012	12345	123456789
%6.4d	..0001	.-0012	.12345	123456789
%-4d	1…	-12.	12345	123456789
%04d	0001	-012	12345	123456789
%+d	+1	-12	+12345	+123456789

Binary I/O

- Write out data structures in raw format
  - Tends to be non-portable
  - for example: writes exact representation of the integer into the file
  - very fast to read and write this way
- size_t fread(void *buffer, size_t size,
  size_t count, FILE *stream);
  - Read into buffer, count items of size “size” from stream
  - Returns the **number of items** read
  - ret = fread(mystuff, sizeof(item), 42, stdin);
- size_t fwrite(void *buffer , size_t size,
  size_t count, FILE *stream);
  - Write to stream, count items of size “size” starting at buffer
  - Return the **number of items** written

Other FILE Operations

- int fflush(FILE *);
  - Write any buffered output to the destination
- long ftell(FILE *);
  - Return the current position (offset) of the stream
  - Return value is in number of bytes
- int fseek(FILE *stream, long offset, int from);
  - Change the position of the stream
  - Where depends on value of from
    - SEEK_SET - offset bytes from beginning of file
    - SEEK_CUR - offset bytes from the current position
    - SEEK_END - offset bytes from the end of the file
    - offset may be positive or negative

Misc. FILE Functions

- Error and Status
  - int feof(FILE *stream);
    - Returns true if currently at end of file
  - int ferror(FILE *stream);
    - Returns true if any error condition is true
  - void clearerr(FILE *stream);
    - Reset the error indication for the file
- Temporary Files
  - Sometimes need a file that is temporary
    - But it needs to be unique
  - FILE *tmpfile(void);
    - Create a new file that, it will be deleted on close