Chapter 5, Standard I/O

Not UNIX ... C standard (library)

- Why?
  - UNIX programmed in C
  - stdio is very UNIX based

- #include <stdio.h>
- FILE *f;
- Standard files (FILE *varname)
  - variable: stdin
    - File Number: STDIN_FILENO
  - variable: stdout
    - File Number: STDOUT_FILENO
  - variable: stderr
    - File Number: STDERR_FILENO

Buffering in Stdio

- Buffers used by stdio
- 3 methods of buffering available
  - Fully (block) buffered
  - Line buffered
    - printf ("Hello ...");
    - printf ("Hello ...\n");
  - Unbuffered

- int setvbuf(FILE *stream, char *buf, int mode, size_t size);
  - setbuf(3), setbuffer(3), setlinebuf(3) also.

int fflush(FILE *stream);

- write stream’s buffer to OS

int fpurge(FILE *stream);

- erase any input or output in the buffer for the stream

Standard Operations:

FILE *fopen (char *path, char *mode)

- mode:
  - "r" -- read, must exist
  - "w" -- write, may create
  - "a" -- append, may create
  - "r+" -- read & write, must exist
  - "w+" -- write & read, may create
  - "a+" -- append & read, may create
  - "rb", "wb", "ab", "rb+", ... -- binary version
- returns pointer to a stream, NULL => error.

FILE *fdopen (int fd, char *mode)

int fclose(FILE *stream)

Input via Stdio

- int fgetc(FILE *stream); (function)
- int getc(FILE *stream);   (macro)
- int getchar();   (macro)

- int ungetc(int c, FILE *stream);

- char * gets(char *str); (UNSAFE!)
- char * fgets(char * restrict str, int size, FILE * restrict stream);

- int feof(FILE *stream);
- int ferror(FILE *stream);
- void clearerr(FILE *stream);
- int fileno(FILE *stream);

Output via Stdio

- int fputc(int c, FILE *stream); (function)
- int putc(int c, FILE *stream); (macro)
- int putchar(int c);   (macro)

- int fputs(const char * restrict str, FILE * restrict stream);
- int puts(const char *str);   /* adds \n */

Binary I/O

Not what one would expect.

- size_t fread(void * ptr, size_t size, size_t nmemb, FILE * stream);
- size_t fwrite(const void * ptr, size_t size, size_t nmemb, FILE * stream);

- Usage: Array based!

```
float d[10];
if (fwrite (d, sizeof(float), 10, fp) != 10 ) ....

int a[35];
if (fread (a, sizeof(int), 35, fp) != 35 ) ....
```

catstdio.c -- example use.

Positioning a stream

- int fseek(FILE *stream, long int offset, int whence);
- long int ftell(FILE *stream);
- void rewind(FILE *stream);
  - Similar to lseek(), but using stdio streams

- int fgetpos(FILE * restrict stream, fpos_t * restrict pos);
- int fsetpos(FILE * restrict stream, const fpos_t * restrict pos);
  - Added due to number of bytes "addressed" by a long int at that time (32 bits)
  - fpos_t allows arbitrary size based on OS max file size

Formatted Output

- int printf(const char * format, ...);
- int fprintf(FILE * stream, const char * format, ...);
- int sprintf(char * str, const char * format, ...);
- int snprintf(char * str, size_t size, const char * format, ...);
- format -- string to print with conversion specifiers
  - printf ("integer %d, string, %s ...\n", 10, "string");
- conversions: %[flags][fieldwidth][.precision][modifier]conv
  - %d %i (%ld %sd) -- Integer (long, short)
  - %x -- hex output of an integer
  - %o -- octal output of an integer
  - %c -- Char
  - %s -- String
  - %e, %f, ... -- Real numbers
  - %% -- print % (No conversion)

Formatted Output (page 2)

Formats: %[flags][fieldwidth][.precision][modifier]conv

- Flags
  - - Left Justify
  - + Always sign the number
  - 0 zero padded
  - (space) blank space for +

- field width
  - number That many columns
  - * uses an integer from parameters for number

- precision (floating, strings)
  - number number of digits after decimal point
  - number of characters of a string
  - * uses an integer from parameters for number

Examples

```
#include <stdio.h>
int main()
{
  int ix;
  for (ix = 0; ix < 20; ix++)
    printf ("%*c%.*s\n", 39-ix, ’ ’, ix*2+1,
      "****************************************************");
 return 0;
}
```

```
#include <stdio.h>
int main()
{
  int i;
  printf ("%12s%14s\n", "num", "square");
  for (i = 0; i<10; i++)
    printf ("%12d%14d\n", i , i*i);
  return 0;
}
```

Formatted Input

int scanf(const char * format, ...);
int fscanf(FILE * stream, const char * format, ...);
int sscanf(const char * str, const char * format, ...);

Returns number of input items assigned.

Format:

- spaces: Match white space
- Other chars: Match that char
- Conversions: %[flag][field-width][modifier]conv
  - * suppresses assignment
  - % matches %
  - s matches non-white-space characters
    - stops at space or field-width chars
  - [minimal re matching -- to a string
  - n number of characters consumed stored in int

Temporary files & names

- FILE * tmpfile(void);
  - /usr/tmp/aaaa00738

```
char *
tmpnam(char *str);
```

```
char *
tempnam(const char *tmpdir, const char *prefix);
```