File IO

C Programming and Software Tools

N.C. State Department of Computer Science

Computer Science
NC STATE UNIVERSITY

`<stdio.h>` … (cont'd)

- Every C program begins execution with 3 streams
  - `stdin`, `stdout`, and `stderr`
- The program does not need to open or close these streams; happens automatically

<stdio.h> fopen()

```
FILE* fopen(const char *filename,
            const char *mode)
```

Establishes a connection between a file or device and a stream

Returns pointer to object of type **FILE**, records information for controlling stream

- returns **NULL** on failure

```
FILE* infile;
infile = fopen("/tmp/testfile.txt", "r");
if (infile == NULL)
     { printf("Error.\n"); return -1;}
```

`<stdio.h>` `fopen()` (cont'd)

- Mode
  - "**`r`**" - open for reading
  - "**`w`**" - create file for writing (discard previous contents)
  - "**`a`**" - append to existing file or create for writing
  - (+ some others, less important)
- If '**`b`**' appended to above modes, file is opened as binary file

Computer Science
NC STATE UNIVERSITY

<stdio.h> Binary Files

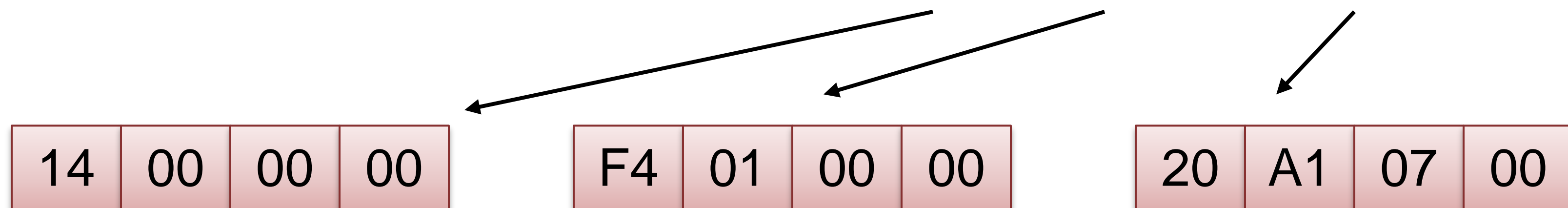
- Needed if
  - non-ASCII data, or
  - need to handle differences between outputs produced by different platforms (e.g., Windows ↔ Linux)
- Examples of binary files
  - images: .bmp, .gif, .jpg, .tif
  - audio: .wav, .ac3
  - video: .avi
  - word processing: .rtf
  - encrypted files
  - etc.

`<stdio.h>` Byte-Ordering

- Different architectures store the bytes of a word in different orders
- What's an *architecture*? Type of processor
  - Ex.: Intel, PowerPC, ARM, VIA, CELL, etc.
- What's a *word*? Primitive datatypes of a language
  - Ex.: **`int`**, **`short int`**, **`float`**, **`double`**, …

`<stdio.h>` The Problems This Causes

- Your program, executing on an Intel PC, writes the (4-byte) **`int`** values **`20`**, **`500`**, **`500000`** to a file



3 integer values, each shown as 4 bytes, in hexadecimal

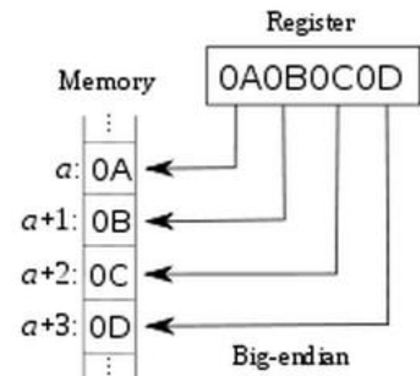
Another program, executing on a PowerPC, reads the (4-byte) **`int`** values from this file and interprets them as **335544320**, **4093706240**, and **547424000**

Same byte values, but interpreted differently!

Big-endian vs. little-endian

- Big-endian: MOST significant byte FIRST
- Little-endian: LEAST significant byte FIRST

- Little-endian: Intel x86
- Big-endian: Everything else (almost)



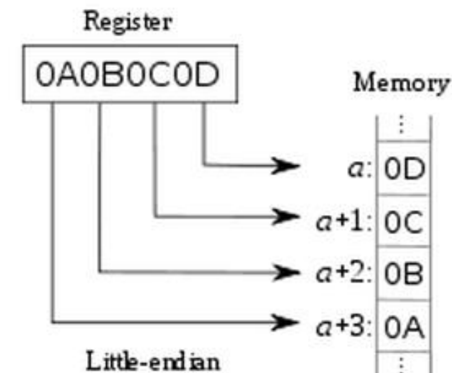


Figure from http://en.wikipedia.org/wiki/Endianness

Converting Between B-E and L-E

```
#include <stdint.h>
uint32_t swap_4byte_word(uint32_t x) {
        return
                ((x>>24) & 0x000000ff) |
                ((x>> 8) & 0x0000ff00) |
                ((x<< 8) & 0x00ff0000) |
                ((x<<24) & 0xff000000);
}
```

Other data sizes are left to the reader…

<stdio.h> fgetc()

```
int fgetc(FILE *stream)
int getc(FILE *stream)
```
← Function
← Macro

Read next character of stream as **unsigned char** (converted to **int**)

returns **EOF** if end of file or error

getchar() is equivalent to **getc(stdin)**

```
int res;
unsigned char c;
if ((res = getc(stdin)) == EOF)
    …take action here…
c = (unsigned char) res;
```


Computer Science
NC STATE UNIVERSITY

<stdio.h> fputc()

```
int fputc(int c, FILE *stream)
int putc(int c, FILE * stream)
```

← Function
← Macro

Write the character **c** (converted to **unsigned char**) to **stream**

Returns character written, or **EOF** on error

putchar(c) equivalent to **putc(c, stdout)**

```
putc('H', stdout);
putc('I', stdout);
putc('!', stdout);
```


Computer Science
NC STATE UNIVERSITY

`<stdio.h>` `ungetc()`

```
int ungetc(int c, FILE * stream)
```

Pushes **`c`** (converted to **`unsigned char`**) back onto **`stream`** !

- Clears the stream's end-of-file indicator.
- **`c`** will be read by next **`getc`** on **`stream`**

Only one character of pushback per stream is *guaranteed*

`EOF` may not be pushed back

Returns character pushed back, **`EOF`** on error

Computer Science
NC STATE UNIVERSITY

<stdio.h> fread()

```
size_t fread (void * ptr, size_t
size, size_t nobj, FILE * stream)
```

Reads up to **nobj** objects of size **size** from **stream** into array pointed to by **ptr**

Returns number of objects read, less if error

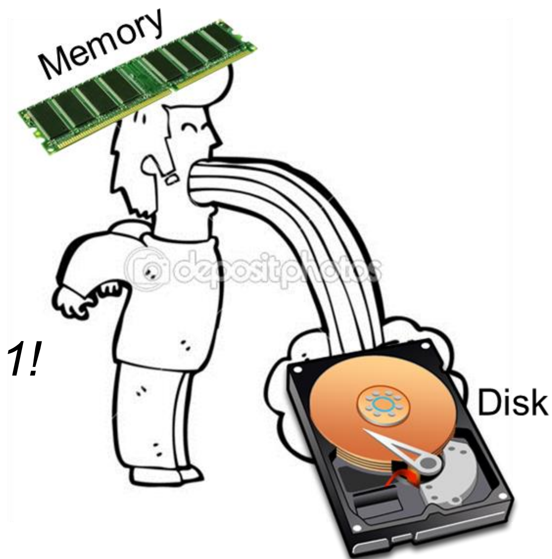
```
int nums[NUMNUMS];
size_t nr = fread((void *) nums, sizeof(int),
                  (size_t) NUMNUMS, stdin);
if (nr != NUMNUMS)
    … do something here …
```

<stdio.h> fwrite()

```
size_t fwrite (const void * ptr,
    size_t size, size_t nobj,
          FILE * stream)
```

Writes up to **nobj** objects of size **size** starting at address **ptr** to **stream**

Returns number of objects written, less than requested if error



This is the "vomit" function from lecture 1!

<stdio.h> fseek()

King, Section 22.7

```
int fseek (FILE *stream, long
offset, int origin)
```

Sets file position (for subsequent reading or writing) to `offset` from `origin`

`origin` may be `SEEK_SET` (beginning of file), `SEEK_CUR` (current position), or `SEEK_END` (end of file)

Mainly for binary streams

Returns non-zero on error

Computer Science
NC STATE UNIVERSITY

<stdio.h> fseek() … (cont’d)

```
int res = fseek(infile, (long) 1000, SEEK_SET);
c = getc(infile);   /* now read 1001st byte */

int res = fseek(infile, (long) -5, SEEK_END);
c = getc(infile);   /* read 5th byte from end */
```

`<stdio.h>` `fflush()`

King, p. 549

```
int fflush(FILE *stream)
```

Causes any buffered data to be immediately written to output file

Helpful if you don't want to wait for **'\n'** to see output

```
fflush(stdout);
```

`<stdio.h>` `fclose()`

King, p. 545

```
int fclose(FILE * stream)
```

Actions

- flush any unwritten data to output file or device
- close the stream (cannot be read or written after)

```
fclose(outfile);
```

<stdio.h> remove()

King, p. 551

```
int remove(const char *filename)
```

- Delete the named file, return 0 if successful

```
if (remove("/tmp/testfile.txt"))
    …error, take action here…
```

<stdio.h> fscanf()

King, p. 558

```
int fscanf(FILE *stream,
           const char *fmt, …)
```

- Like **scanf**, but specify stream to be read from
  - **scanf(fmt, args…)** is same as **fscanf(stdin, fmt, args…)**

```
int sscanf(char * s,
           const char *fmt, …)
```

- Like **scanf**, but … scans from a string instead of a file!

`<stdio.h>` `fprintf()`

King, p. 552

```
int fprintf(FILE *stream,
                const char *fmt, …)
```

- Like **printf**, but specify stream to be written to
  - **printf(fmt, args…)** is same as **fprintf(stdin, fmt, args…)**

```
int sprintf(char * s,
   const char *fmt, …)
```

- Like **printf**, but … prints to a string instead of a file!

`<stdio.h>` I/O Error Functions

King, p. 564

```
int feof(FILE *stream)
```

- Returns true if **EOF** for **stream** has been reached

```
int ferror(FILE *stream)
```

- Returns true if error indicator for **stream** is set

```
void clearerr(FILE *stream)
```

- Clears previously set error indicator for **stream**
  - errors are not cleared unless programmer explicitly uses **clearerr**

Normal IO workflow

fopen
Check for **NULL**!

fclose

Read	Write	Data
fgetc/getc	fputc/putc	One character at a time
fscanf	fprintf	ASCII tokens
fread	fwrite	Binary data

fseek: Move around the file.
feof: Check for EOF.
ferror: Check for error.
fflush: Force output to go out.

Computer Science NC STATE UNIVERSITY

Example: mycp.c (1)

```
#include <stdio.h>
#include <stdlib.h>

#define BUF_SIZE 4096

void pdie(char* msg) {
    perror(msg);
    exit(1);
}
```

```
int main(int argc, char* argv[])
{
    if (argc != 3) {
        printf("Usage: mycp <src> <dest>\n");
        exit(1);
    }
    FILE* fp_in  = fopen(argv[1],"rb");
    if (!fp_in) {
        pdie(argv[1]);
    }
    FILE* fp_out = fopen(argv[2],"wb");
    if (!fp_out) {
        pdie(argv[2]);
    }
    char buffer[BUF_SIZE];
    // CONTINUED NEXT SLIDE...
```

Example: mycp.c (2)

```
  while (1) {
      int bytes_read = fread(buffer, sizeof(char), BUF_SIZE, fp_in);
      if (ferror(fp_in)) {
          pdie(argv[1]);
      }
      fwrite(buffer, sizeof(char), bytes_read, fp_out);
      if (feof(fp_out) || ferror(fp_out)) {
          pdie(argv[2]);
      }
      if (feof(fp_in)) {
          break;
      }
  }
  fclose(fp_in);
  fclose(fp_out);
}
```

Not **BUF_SIZE**!

Computer Science NC STATE UNIVERSITY

Exercises 11a and 11b

Writing bytes

- **11a**) Write a program to generate the test input from HW3 called “test_allbytes.dat” – a 256-byte file consisting of bytes 0x00 through 0xFF. Use a loop and **`fputc`** in your solution.

- **11b**) Write a program that does the same thing, except now do it by building a char array with the bytes and write it out with a single **`fwrite`** call.

Reminder: Go to course web page for link to exercise form.
Paste code into ideone.com and submit the link.

Computer Science
NC STATE UNIVERSITY

Any Questions?