Disk Files

a short? interlude

Text Files for Input and Output

- Files on disk come in two formats
  - Text files
  - Binary files
- Text files
  - Contents are simply ASCII strings
  - Typically divided into lines of text
    - *Newline* separator depends on operating system
- Binary files
  - Contents are block of arbitrary bytes

Accessing a Disk File

- File must be **fopen**ed before it can be used
- File should be **fclose**d after you're done with it
- File *must* be closed after writing to it, or the written material may be lost
- Access **mode** must be specified
  - "r", "w", "a"
  - add "b" for binary access

Reading a Text File

- **fopen()** returns a *handle*
  - similar to stdin
- Use with input functions
  - fgets(), fgetc(), fscanf()
- Lines of text may be Windows-style or Linux-style
  - What about Macs?
- **fclose()** closes the handle

Example

```
        .
        .
        .
    char buffer[1024];
    FILE *h = fopen("filename", "r");
    while( NULL != fgets(buffer, 1024, h) ) {
        len += strlen(buffer);
        fputs(buffer, stdout);
    }
    fclose(h);
        .
        .
        .
```

Writing a Text File

- Use fopen() again, but with mode "w"
  - or mode "a"
- Use output functions with handle in place of stdout
  - fputs, fputc, fprintf
- fclose() must be used to close the handle at the end
- **fflush()** can be used to *flush* the output buffer

Example

```
        .
        .
        .
    FILE *h = fopen("filename", "w");
    fputs("This is a test.\n", h);
    fputs("This is only a test.\n", h);
    fprintf(h, "%d + %d = %d\n", 7, 8, (7+8));
    fclose(h);
        .
        .
        .
```

The File Name and the Command Line

- Any text string can be used as a file name
- Command-line options in argv[] array allow user-supplied filenames conveniently
- String operations can be used to build filenames

Example A – copy part of a file

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

#define NLEN 1024
#define BUFLEN 100
```

```
int main(int argc, char **argv)
{
    char out[NLEN], buffer[BUFLEN];

    strncpy(out, argv[1] , NLEN);
    strncat(out, ".out" , NLEN);

    FILE *hr = fopen(argv[1], "r");
    FILE *hw = fopen(out, "w");

    fgets(buffer, BUFLEN, hr);
    fputs(buffer, hw);
    fflush(hw);
    fclose(hr);
    fclose(hw);
    return 0;
}
```

Example B – fix up filename

```
int main(int argc, char **argv)
{
    char *p, in[NLEN], out[NLEN], buffer[BUFLEN];

    if (argc > 1)
        strncpy(in, argv[1] , NLEN);
    else
        strncpy(in, "dummy.in" , NLEN);

    strncpy(out, in , NLEN);
    p = strrchr(out, '.');
    *p = '\0';
    strncat(out, ".out" , NLEN);

    printf("Filenames: %s,  %s\n", in, out);

        ⋮
```

Opening a Binary File

- Open in binary mode:
  - FILE *handle = fopen("foobar.data", **"rb"**);
  - FILE *handle = fopen("foobar.data", **"wb"**);
- *Windows* –
  binary mode doesn't convert newlines, text mode does
- *Linux* –
  no difference between binary mode and text mode

Reading From a Binary File

- fread(ptr, item-size, nitems, handle)
  - ptr – address of destination
  - item-size – # of bytes in an item
  - nitems – # of items to read
    - ptr should point to an array, if this is > 1
  - handle – the file handle
- Can be intermixed with fscanf(), fgets(), fgetc(), ungetc()

Writing To a Binary File

- fwrite(ptr, item-size, nitems, handle)
  - ptr – address of source
  - item-size – # of bytes in an item
  - nitems – # of items to write
    - ptr should point to an array, if this is > 1
  - handle – the file handle
- Can be intermixed with fprintf(), fputs(), fputc()

Position In a Binary File

- long ftell(handle)
  - returns current position with file
- fseek(handle, offset, *whence*)
  - changes position within the file
  - *whence*: SEEK_SET, SEEK_CUR, or SEEK_END
- rewind(handle)
  - returns file to its beginning
  - same as "fseek(handle, 0, SEEK_SET);"

Low-Level Accesses

- Alternate approach to file manipulation uses *file descriptors* instead of *FILE **
- From <unistd.h> :
  - open("filename", flags);
  - creat("filename", mode);
  - read(fd, *buffer, count);
  - write(fd, *buffer, count);
  - lseek(fd, offset, *whence*);
- We won't consider low-level accesses anymore.