CSI 402 – Lecture 2

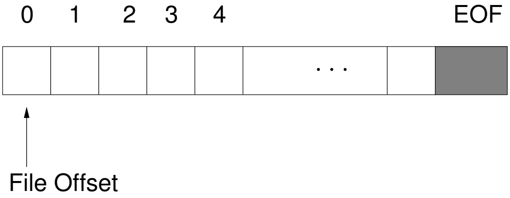
(More on Files)

Files – A Quick Review

- Type for file variables: `FILE *`
- File operations use functions from `stdio.h`.
- Functions `fopen` and `fclose` for opening and closing files.
- Functions `getc` and `putc` for reading and writing characters from/to files.
- Functions `fscanf` and `fprintf` for reading and writing other data types from/to files.

Positioning in Files

Ref: Chapter 11 of Deitel & Deitel.



Input files:

- File offset gives the number of the byte to be read next.
- It is set to zero when file is opened (using `"r"` mode).
- Value of file offset increases as bytes are read from file.

Positioning in Files (continued)

Output files:

- File offset gives the number of the byte to be written next.
- It is set to zero when file is opened (with mode `"w"`).
- File offset increases as bytes are written to file.

Note: For both input and output files, the current value of file offset can be obtained using `ftell` function.

Library Function `ftell`

- Part of `stdio.h`.
- **Prototype:** `long  ftell(FILE *fp)`
- Returns the offset for the file specified by `fp`; returns `-1L` in case of error.

Example:

```
    FILE *fp;  long  pos;
        .
        .
   /* Open file, etc. */
        .
        .
   pos = ftell(fp);
   printf("Offset = %ld\n", pos);
```

Library Function `fseek`

- Also part of `stdio.h`.
- To “move around” in a file.
- Prototype:

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

- `fp` specifies the (input or output) file.
- `offset` (which may be negative) specifies the amount of movement.
- How `offset` is used depends on the parameter `origin`.

Library Function `fseek` (continued)

- Parameter `origin` can have any of the following three values (constants).
  - `SEEK_SET`: `offset` specified relative to the beginning of the file.
  - `SEEK_CUR`: `offset` specified relative to the current position.
  - `SEEK_END`: `offset` specified relative to the end of the file.
- Function `fseek` returns 0 if successful and a non-zero value otherwise.

A Related Function: `rewind`

- Part of `stdio.h`.
- **Prototype:** `int  rewind (FILE *fp)`
- Sets file offset to 0 (i.e., gets us back to the beginning of a file).
- `rewind(fp)` is equivalent to

```
fseek(fp, 0L, SEEK_SET);
```

Program Example: Handout 2.1.

Moving Outside File Boundary

- Function `fseek` allows any offset value; it doesn't check whether specified move is within the file.
- For illegal moves, effect is implementation dependent.
- On most Unix systems:
  - Function `fseek` does not move the offset value below the beginning of the file.
  - File offset can be changed to a value beyond the end of file. However, trying to read from a non-existent position produces EOF.
  - For an output file, `fseek` allows "forward jumps"; positions where nothing was written contain `'\0'`.

Random Access Files

Random Access: Access time is independent of position.

Example:

Array : Provides random access.
List : Does not provide random access.
(Provides sequential access.)

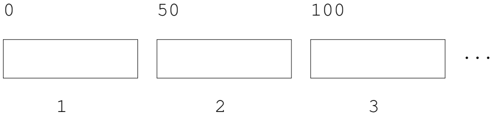
For files:

- Random access: Fast access.
- Applications: Airline reservation systems, Banking systems, etc.

Random Access Files in C

- No explicit support. (Functions `fread` and `fwrite` from `stdio.h` are used.)
- **Common method:** Make all records to be of the same size.

Example:



Size of each record = 50 bytes

Starting position of Record i $=$ $(i-1)$ $\times$ Size of record.

Formatted and Unformatted Files

Formatted Files:

- Also called **text** files; they can be viewed/edited using a standard text editor.
- Can be produced by a C program using “formatted write” (i.e., using `fprintf`).

Example for formatted write:

```
FILE *ofp;  int  num = -25;
  .
  .
fprintf(ofp, "%d", num);
```

No. of bytes written to the file = 3.

Note: The number of bytes written to the output file depends on the value of the integer.

Formatted and Unformatted Files (continued)

Unformatted files:

- Also called **binary** files; they cannot be viewed/edited using standard text editors.
- To produce unformatted files, C program must use "unformatted write" using the `fwrite` function.

Function `fwrite`:

- `Prototype:`

```
size_t  fwrite (const void *p, size_t  size,
                size_t  nent,  FILE *fp)
```

- Writes bytes from memory to a file.
- `p`: Gives the starting address in memory.

Description of `fwrite` (continued)

- `size`: Gives the size (i.e., number of bytes) of each entry.
- `nent`: Gives the number of entries to be written.
- `fp`: Pointer to the output file.
- Writes the specified number of entries (starting from the specified memory address) to the output file.
- Returns the number of entries written. (If this value is less than `nent`, it is an indication of error.)

Description of fwrite (continued)

Some Technicalities:

```
FILE  *ofp;  int  num = -25;
```

- `&num`: Starting address of `num` (Type: `int *`).
- `(const void *) &num`: Type casts address to `const void *` .
- `sizeof(num)`: Size of the entry (i.e., no. of bytes) to be written.
- No. of entries to be written: 1.

Now, the call to `fwrite` is as follows:

```
fwrite((const void *) &num,  sizeof(num), 1,  ofp);
```

Note: We must check the return value of `fwrite` to ensure that no errors occurred.

Difference Between fprintf and fwrite

Example: (Assume int uses 4 bytes.)

```
int  num = -2017;   FILE *out_f1, *out_f2;

-- Open file out_f1 (out1.fmt)  --
-- Open file out_f2 (out2.ufmt) --

/* Formatted write. */
fprintf(out_f1, "%d", num);

/* Unformatted write. */
fwrite((const void *) &num, sizeof(num), 1, out_f2);

-- Close files. --
```

File out1.fmt:

- Size = 5 bytes.
- A text file: can be examined/edited using a text editor.
- Can be read using fscanf.

Difference Between `fprintf` and `fwrite` (continued)

File `out2.ufmt`:

- Size = 4 bytes.
- A binary file: cannot be examined using a text editor.
- Can be read from using `fread` (a function for reading unformatted files).

Formatted Read: Uses `fscanf`.

```
FILE *ifp;  int  num;
   .
   .
fscanf(ifp, "%d", &num);
```

Unformatted read: Uses `fread`.

Description of Function `fread`

- **Prototype:**

```
size_t  fread (void *p,  size_t  size,
               size_t  nent,  FILE *fp)
```

- Reads bytes from file into memory.
- `p`: Gives the starting address for reading into memory.
- `size`: Gives the size (i.e., number of bytes) of each entry to be read.
- `nent`: Gives the number of entries to be read.
- `fp`: Pointer to the input file.
- Reads the specified number of entries from the input file into memory starting from the specified memory address.
- Returns the number of entries read. (If this value is less than `nent`, it is an indication of error.)

Program Examples

1. Creating a random access file: Handout 2.2.
2. Writing to a random access file: Handout 2.3.
3. Reading from a random access file: Handout 2.4.

Examples of Binary Files:

- Compiled versions of C programs (i.e., files with extension "`.o`").
- Executable versions of C programs (e.g. file "`a.out`").
- Compressed files.
- File archives (e.g. files created using `tar` command in Unix).

Suggested Exercises

1. Study Handout 2.1 carefully to understand the use of functions `fseek` and `ftell`.

2. Study Handouts 2.2, 2.3 and 2.4 carefully to understand the use of functions `fread` and `fwrite`.

3. Study the other program examples in Chapter 11 of Deitel & Deitel.