CSI 333 – Lecture 6

Introduction to C: Part V (Files)
A Quick Review of Standard I/O

Recall that

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

allows use of `printf` and `scanf` functions.

**Example:**

```c
int i;
scanf("%d", &i);
printf("Value of i = %d\n", i);
```

**Buffered I/O:**

```c
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  ..
**Buffering**: Used for efficiency.

- Buffer has finite capacity.
- Output written to device when buffer is full.
- Reduces synchronization overhead between CPU and device.

**Flushing the buffer:**

```c
printf("Reached Point A.\n"); fflush(stdout);
```

**Note**: The output device `stdout` is buffered.
Ref: Section 8.5 of Deitel & Deitel.

(a) Function getchar:

- Prototype: `int getchar(void);`
- Fetches the next character from `stdin`.
- Returns `EOF` if no character is available. (EOF is a special `int` value defined in `stdio.h`.)
- Return type for `getchar` is `int` because of EOF.
(b) **Function** putchar:

- Prototype: `int putchar(int c);`

- Parameter `c`: An int valued expression representing the ASCII code of the character to be output.

- Converts parameter to `unsigned char` and writes the value to `stdout`.

- If successful, returns the value of the parameter as an `int`; otherwise returns `EOF`.

**Note:** Functions `getchar` and `putchar` are implemented as macros in `stdio.h`.

**Program Examples:** See Handouts 6.1 and 6.2.
Ref: Chapter 11 of Deitel & Deitel.

Notes:

- A file is a sequence of bytes.
- File operations are normally buffered for efficiency reasons.

Declaring file variables:

```c
FILE *f, *g;
```

Remarks:

- Variables f and g are “file pointers”.
- The type FILE is defined in stdio.h.
Opening a file:

- Function `fopen` with the following prototype:

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

- Some common modes:
  - "r" : Read.
  - "w" : Write.
  - "a" : Append.

- Function returns pointer to file if successful; otherwise, returns NULL.

Important Note:

Always check the pointer value returned by `fopen` for NULL.
Descriptions of modes

(a) Read:
- The file is opened for reading; can’t be written.
- The file must exist; otherwise, the call to fopen fails.

(b) Write:
- The file is opened for writing; it can’t be read.
- If the file exists, it is truncated; otherwise, a new (empty) file is created.

(c) Append:
- Allows for writing at the end of the file.
- If the file exists, it is opened for appending; otherwise, a new (empty) file is created.
```c
#include <stdio.h>
#include <stdlib.h>

FILE *f;
if ((f = fopen("in.dat", "r")) == NULL){
   /* File open failed. */
   printf("Can’t open in.dat.\n"); exit(1);
}
else {
   /* File opened successfully. */
   .
   .
}

Note: We need <stdlib.h> to use the exit function.
```
Function `fclose`.

Prototype:

```c
int fclose (FILE *f);
```

Flushes the file buffer before closing it.

Function returns 0 if successful; otherwise, returns EOF.

**Good Programming Practice**

In your programs, close each file explicitly.
(a) **Function** `getc`:

- Similar to `getchar` (used for `stdin`).
- Prototype: `int getc(FILE *infp);`
- Fetches the next character from the file specified by the pointer `infp`.
- The file specified by `infp` must be open for reading.
- Returns `EOF` if no character is available (i.e., end of file has occurred) or there is an error.
(b) **Function putc:**

- Similar to putchar (used for stdout).
- **Prototype:** `int putc(int c, FILE *outfp);`
- Writes the character given by parameter `c` to the file specified by the pointer `outfp`.
- The file pointer is the **second** parameter.
- The file specified by `outfp` must be open for writing.
- If successful, returns the value of the parameter `c` as an `int`; otherwise returns `EOF`.

**Programming Examples:** Handouts 6.3 and 6.4.
Functions `fscanf` and `fprintf`

- Used to read/write other data types (int, float, etc.) from/to files.

(a) Function `fscanf`:

- Similar to `scanf` (used for reading from `stdin`).
- Parameters:
  - First parameter: Pointer to an input file. (The file must be open for reading.)
  - Second parameter: String specifying formats (as in `scanf`).
  - Subsequent parameters: Pointers to variables into which values must be read from the file (as in `scanf`).
- Returns `EOF` if end of file or an error occurs before any value is read; otherwise, returns the number of values read.
(a) **Function fprintf:**

- Similar to `printf` (used for writing to stdout).

- Parameters:
  - First parameter: Pointer to an output file. (The file must be open for writing.)
  - Second parameter: String specifying formats (as in `printf`).
  - Subsequent parameters: Expressions whose values must be written to the file (as in `printf`).

- Returns a negative integer if an error occurs while writing to the file; otherwise, returns the number of characters written to the output file.

**Program Example:** Handout 6.5. (Sorting example)
(a) Flushing the buffer for files:

- Write operations to files are buffered for efficiency reasons.
- We can use `fflush` for files also.

Example:

```c
int x; FILE *outfile;
.
.
fprintf(outfile, "Output = %d\n", x);
fflush(outfile);
.
.
```
Writing to stderr:

- Use fprintf to write to stderr.

Example:

```c
#define MAX_LEN 20
char fname[20];

fprintf(stderr, "Fatal: Can’t open file = %s\n", fname);
exit(1);
```
Command Line Arguments in Unix

- Allow a user to specify file names at run time.
- **Ref:** Section 14.4 of Deitel & Deitel.

**Example:**

```
%% prog in1.dat out1.dat
```

Unix passes command line arguments to `main` provided `main` is written in the following manner:

```c
int main (int argc, char *argv[]) {

   ...
   ...

}
```

Here,

- `argc` : No. of arguments (3 in the example).
- `argv` : Array of strings.
Command Line Arguments ... (continued)

Note: The program can use strings argv[0], argv[1] and argv[2].
Example:

```c
FILE *ifp, *ofp;
if ((ifp = fopen(argv[1], "r")) == NULL) {
    /* Produce error message and exit. */
}
if ((ofp = fopen(argv[2], "w")) == NULL) {
    /* Produce error message and exit. */
}
/* Here, all the opened files can be used. */

Program Example: Handout 6.6.