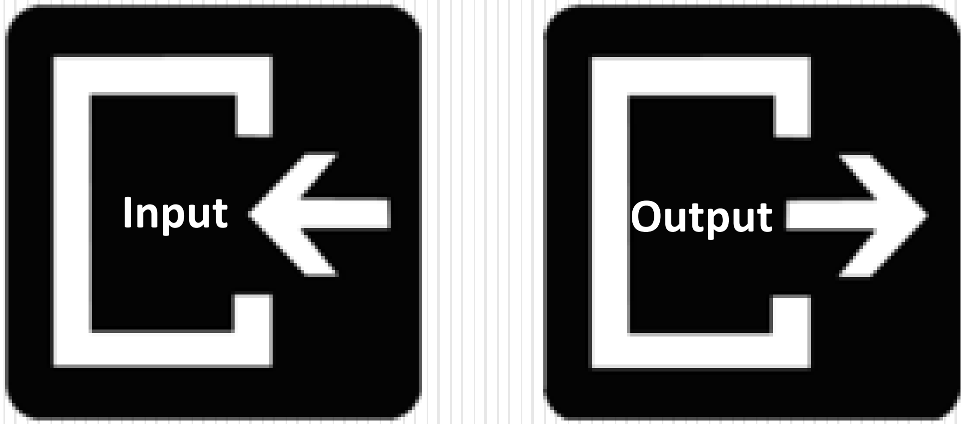
Programming in C

Chapter 15
File Input/Output



Standard File Pointers

- Assigned to console unless redirected
- Standard input = stdin
  - Used by scan function
  - Can be redirected: `cmd < input-file`
- Standard output = stdout
  - Used by printf function
  - Can be redirected: `cmd > output-file`
- Standard error = stderr
  - Can be specified in fputs function instead of stdout
  - Can be redirected: `cmd 2> output-file`

Files

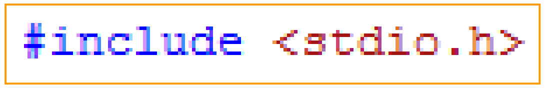
- A collection of related data treated as a unit
- Two types
  - Text
  - Binary
- Stored in secondary storage devices
- Buffer
  - Temporary storage area that holds data while they are being transferred to or from memory.

Text Files

- Data is mainly stored as human-readable characters
- Each line of data ends with a newline character
  - ↵ = `\n`

```
C666666666 20 8.55↵
A222222222 50 12.5↵
F333333333 45 8.5↵
B444444444 50 9↵
G555555555 30 6↵
E111111111 40 10↵
H777777777 40 12↵
D888888888 40 11.11↵
I999999999 45 15↵
```

User File Steps



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

1. Declare a file pointer variable
   - Program connection to external user file
2. Open the file
   - Creates a structure to store information needed for processing file and buffer area(s)
   - Makes file pointer connection to structure
3. Use functions for input and/or output
   - Handles movement of data between program and buffer and between buffer and external device
4. Close the file
   - Writes the buffer to file if necessary
   - Frees up memory associated with file

1. File Pointer Declaration

```
FILE * variable-name-list;
```

- Defines variables of type FILE*, file pointer
- Pointer is undefined unless initialized
  - If not initialized to another value, initialize to NULL
- Examples:

```
FILE * scores_in = NULL;    // Input file
FILE * scores_out = NULL;   // Output file
```

- Following slides will use `fp` for file pointer

2. fopen

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

- Parameters
  - filename – string that supplies the name of the file as known to the external world
    - Default path is current directory
  -

mode	Meaning
r	Open file for reading • If file exists, the marker is positioned at beginning • If file does not exist, error returned
w	Open text file for writing • If file exists, it is emptied • If file does not exist, it is created
a	Open text file for append • If file exists, the marker is positioned at the end • If file does not exist, it is created

fopen

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

- Return
  - If successful, file pointer
  - If not successful, NULL
  - Always check return
    - If not successful, print error message and exit or some other corrective action

fopen

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

- Examples

```
// Define and then open scores.txt for input
FILE * scores_in = NULL;
scores_in = fopen("scores.txt", "r");
if (scores_in == NULL) {
   printf("Unable to open scores.txt\n");
   exit(1);
}

// Define and open newscores.txt for output
FILE * scores_out = fopen ("newscores.txt", "w");
if (scores_out == NULL) {
   printf("Unable to open newscores.txt\n");
   exit(1);
}
```

4. fclose

```
int fclose(FILE *fp)
```

- Used to close a file when no longer needed
- Prevents associated file from being accessed again
- Guarantees that data stored in the stream buffer is written to the file
- Releases the FILE structure so that it can be used with another file
- Frees system resources, such as buffer space
- Returns zero on success, or EOF on failure

fclose

- Examples:

```
fclose(scores_in);
fclose(scores_out);
```

- To go back to beginning without fclose then fopen:

```
void rewind(FILE *fp)
```

3. Input/Output Functions

- Formatted Input
  - fscanf
- Formatted Output
  - fprintf
- String Input
  - fgets
- String Output
  - fputs

Formatted Input Functions

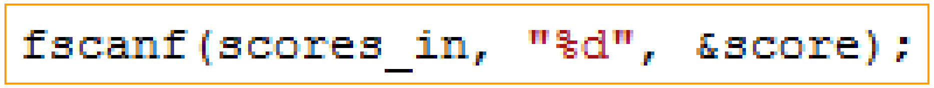
- Read and convert a stream of characters and store the converted values in a list of variables found in the address list
- scanf

```
scanf("format string", address list);
```

  - Reads text data from standard input
- fscanf

```
fscanf(fp, "format string", address list);
```

  - Reads input from the specified file



```
fscanf(scores_in, "%d", &score);
```

Formatted Output Functions

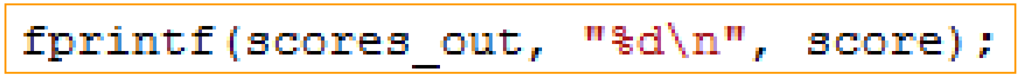
- Displays output in human readable form
- printf

```
printf("format string", value list);
```

  - Writes to standard output or standard error file
- fprintf

```
fprintf (fp, "format string", value list);
```

  - Writes to the specified file



```
fprintf(scores_out, "%d\n", score);
```

String Input

- Reminder: Watch size of string
  - Must be large enough to hold largest input string
    - Plus \n perhaps
    - Plus \0 perhaps
  - C generally gives no warning of this issue

```
char input_string[MAX_INPUT_LENGTH+2];
```

- Standard Input
  - getchar: Read one character and return value as int

```
int getchar()
```

  - gets(): Read line & convert \n to \0, no size check

```
char *gets (char *strPtr)
```

String Input: fgets

```
char *fgets (char * strPtr, int size, FILE *fp)
```

- Inputs characters from the specified file pointer through \n or until specifed size is reached
- Puts newline (\n) in the string if size not reached!!!
- Appends \0 at the end of the string
- If successful, returns the string & places in argument

```
const int MAX_LINE = 100;
char line_in[MAX_LINE + 2];
int line_len;
FILE * text_in = fopen("data.txt", "r");
// Should also check open return
fgets(line_in, MAX_LINE, text_in);
// Check for \n
line_len = strlen(line_in);
if (line_in[line_len-1] == '\n')
   line_in[line_len-1] = '\0';
```

String Output

- Standard Output
  - putchar: Write one character

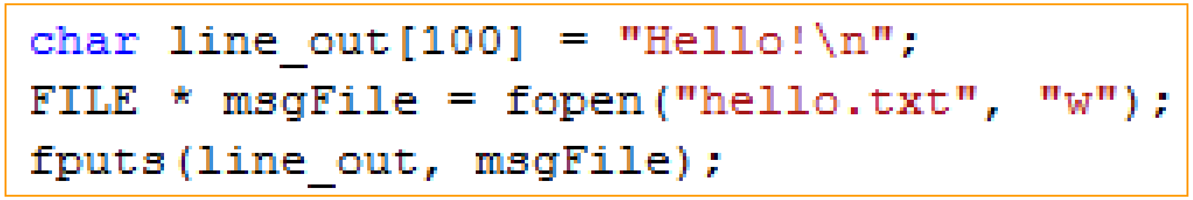
    ```
    int putchar(int outChar)
    ```
  - puts(): Write line & converting \0 to \n

    ```
    int puts (const char *strPtr)
    ```

String Output: fputs

```
int fputs (const char *strPtr, FILE *fp)
```

- Takes a null-terminated string from memory and writes it to the specified file pointer
- Drops \0
- Programmer's responsibility: Make sure the newline is present at the appropriate place(s)



```
char line_out[100] = "Hello!\n";
FILE * msgFile = fopen("hello.txt", "w");
fputs(line_out, msgFile);
```

End of File Controlled Loops

- feof

  `int feof(FILE *fp)`

  - Function to check if end of file has been reached.
  - For an end of file controlled loop
    - Read before the loop
    - Test for end of file: **`while (!feof(fp))`**
    - Inside loop:
      - Process
      - Read at the bottom of the loop

Programming in C

Chapter 15

File Input/Output

THE END