

10. File Input / Output

ENEE 140

Prof. Tudor Dumitraş
Associate Professor, ECE
University of Maryland, College Park



<http://ter.ps/enee140>

1

Today's Lecture

- Where we've been
 - Scalar data types
 - Arrays and strings
 - Functions
 - Random number generation
 - Control flow
 - Structuring complex programs
- Where we're going today
 - File Input/Output
 - Project 2 review
- Where we're going next
 - No lecture next week (Thanksgiving break)

2

2

Text File I/O

- Declaring and manipulating file variables

```
#include <stdio.h>
FILE *file;                                declare the file variable
```

- Opening

```
file = fopen("filename.txt", "r");         open file for reading
```

- Mode "r": open existing file for reading
- Mode "w": open file for writing and erase existing content
- Mode "a": open file for writing and append after existing content
- Opening a file in modes "a" or "w" will create the file if it doesn't already exist
- The `fopen()` function returns NULL if there is an error

- Closing

```
fclose(file);                               close file
```

- **Frequent mistake: Not closing all the files you have opened**

4

4

Text File I/O – continued

- Declaring and manipulating file variables

```
#include <stdio.h>
FILE *file;                                declare the file variable
int i;
char line[256];
```

- Reading

```
fscanf(file, "%d", &i);                    like scanf()
i = getc(file);                             like getchar()
fgets(line, 256, file);                     read an entire line
```

- Writing

```
fprintf(file, "%d", i);                    like printf()
putc(i, file);                             like putchar()
fputs(line, file);                         write an entire line
```

- The file must be open in order to read or write

5

5

Review: Reading a File Line-by-Line

```

#include <stdio.h>

char line[MAX_LINE];
int a, b;
FILE *file;                                variable representing the file

file = fopen("myfile.txt", "r");           open file for reading

if (file == NULL) {                         fopen() failed
    printf("Could not open the myfile.txt file.\n");
    exit (-1);
}

...
fgets(line, MAX_LINE, file);               read a line of text from the file
sscanf(line, "%d %d", &a, &b);           parse line with sscanf()
...

fclose(file);                               close file

```

6

6

Position in the File

- When operating on a file, you read/write data sequentially
- You can change the current position in the file

```

rewind(file);                               go back to the beginning

fseek(file, 0, SEEK_END);                   go to the end of the file
  offset  whence
  ↙       ↘
- whence==SEEK_SET: move offset bytes after the beginning of the
  file
- whence==SEEK_CUR: move offset bytes after the current position
- whence==SEEK_END: move offset bytes after the end of the file
  (offset may be negative)

```

7

7

Special Files

- `stdin`, `stdout`, `stderr`

<code>fscanf(stdin, "%d", &i);</code>	read from standard input
<code>fprintf(stdout, "%d", i);</code>	write to standard output
<code>fprintf(stderr, "%d", i);</code>	write to standard error stream
- You don't have to open or close these special files
- By default, they are associated with the console
 - You can redirect them from the command line

<code>prog <infile.txt</code>	<code>stdin</code> redirected to <code>infile.txt</code>
<code>prog >outfile.txt</code>	<code>stdout</code> redirected to <code>outfile.txt</code>
<code>prog 2>errfile.txt</code>	<code>stderr</code> redirected to <code>errfile.txt</code>
<code>prog1 prog2</code>	pipe <code>stdout</code> of <code>prog1</code> into <code>stdin</code> of <code>prog2</code>

8

8

`printf/scanf` Operate on the Standard Output/Input

`printf("Hello %s\n", "world");` is equivalent to
`fprintf(stdout, "Hello %s\n", "world");`

`scanf("%d\n", &a);` is equivalent to
`fscanf(stdin, "%d\n", &a);`

9

9

Review: Formatted Input

- You can read from stdin, from a file or from a string


```
FILE *file;
int read;
char string[256];
read = scanf(format, vars);           read from standard input
read = fscanf(file, format, vars);    read from file
read = sscanf(string, format, vars);  read from string
```
- These functions allow you to read scalar data types (format specifiers (%d, %u, %f, etc.) and strings (format specifier %s)
 - Remember to put an & before each scalar variable you are reading, e.g. `scanf("%d", &a);`
- The `Xscanf()` functions return the number of variables read
 - Return is 0: the input did not match the format provided
 - Return is EOF: the end-of-file was reached

10

10

Review: Formatted Output

- You can write to stdout, to a file, or to a string


```
FILE *file;
int read;
char s[MAX_S];
printf(format, vars);           print to standard output
fprintf(file, format, vars);    print to file
sprintf(s, format, vars);       print to string
```
- `format` uses the same specifiers as the `Xscanf` functions
 - Additionally, may specify the width and precision, e.g. `"%4.2f"`
 - Width or precision may be specified as `*`: read it from next argument


```
printf("%. *s", MAX_S, s);    print at most MAX_S chars from s
```

 - For `Xscanf`, there is no modifier like `*` for `Xprintf`
 - For all specifiers and modifiers, see Chapter 7.2 or type `man printf`
- With `sprintf`, you must be careful not to exceed the size of the string!**

11

Pushing Back Characters

- We've seen: character I/O

<code>c = getc(file);</code>	read a character from file
<code>putc(c, file);</code>	write a character to file
- Can also push a character back to the input stream

<code>ungetc(c, file);</code>	c will be returned by the next read operation
-------------------------------	---
- The formatted I/O functions (fscanf, fprintf) are implemented using the character I/O functions
 - Ability to push back characters is needed when reading formatted numbers
 - You know that you have all the digits of the number when you read a non-digit character
 - But that character may be part of the next formatted input requested (you've read one character too far) => push it back to the stream

12

12

Status of File Streams

- File operations interact with hardware devices
 - These operations may fail
 - You must be able to distinguish between these errors and reaching EOF during normal file operations
- You can check the status of your **FILE*** stream

<code>FILE *file;</code>	
<code>if (ferror(file)) {...}</code>	check if an error occurred
<code>if (feof(file)) {...}</code>	check if you reached EOF
<code>rewind(file);</code>	rewind clears the EOF and error flags

13

13

Error Checking

- If you receive an error, you can print an error-specific message

```
#include <stdio.h>
FILE *file;
if ( (file=fopen("my_file.txt","r")) == NULL) {
    perror("Cannot open file");  prints a message describing the error
    exit(-1);
}
```

- `perror()` appends an error-specific message to the text provided and prints it to `stderr`
 - You may also print additional error messages to `stderr` with `fprintf(stderr, ...)`
- **Good programming practice: check the return values of all the functions you invoke – an error may have occurred!**

14

14

Error Checking: Examples

```
#include <stdio.h>

FILE *file;
unsigned options;

if ( (file=fopen("my_file.txt","r")) == NULL) {
    perror("Cannot open file for reading");
    exit(-1);                                cannot proceed: file is not opened
}

if ( fscanf(file, "%u", &options) < 1 ) {
    fprintf(stderr, "File must start with an unsigned int");
}

printf("Read %u from the file\n", options);

if ( ferror(stdout) ) {
    perror ("Error writing to stdout");
}
```

15

15

Review of Lecture

- What did we learn?

16

16

Next Steps

- Next week
 - Thanksgiving recess (no lecture, no discussions)
- Next lecture (after Thanksgiving)
 - Sorting
- Assignments for this week
 - **Project 2**: partial implementation due on Friday at 11:59 pm
 - Read **K&R Chapters 6.8, 8.1, 8.2, 8.3, 8.4, 5.11** (how to use the library function `qsort`). Re-read **K&R Chapter B11 and 3.5** (try to understand how the shellsort implementation works)
 - **No quiz**
 - Weekly challenge: **selection_sort.c** (after Thanksgiving)
 - Homework: **lab10.pdf** (on <http://ter.ps/enee140>), due on Friday at 11:59 pm

17