

# CSC209 Week 5 Notes

Hyungmo Gu

May 14, 2020

## Files in C 1 of 5

- Opening file

- **Syntax:** `*fopen(const char *filename, const char *mode)`
- the import file should be in the same folder as 'a.out' (default)
- Mode Strings
  1. *r* - File opened for reading
  2. *w* - File opened for writing
  3. *a* - File opened for appending

```
1  #include <stdio.h>
2
3  int main() {
4      FILE *sample_file;
5
6      sample_file = fopen("example_sources/sample.txt", "r");
7      if (sample_file == NULL) {
8          fprintf(stderr, "Error opening file \n");
9          return 1;
10     }
11
12     ...
13
14     return 0;
15 }
16
```

- Closing file

- **Syntax:** `fclose(FILE *filename)`
- returns 0 if close successful

```
1  #include <stdio.h>
2
3  int main() {
```

```

4      FILE *sample_file;
5
6      ...
7
8      if (fclose(sample_file) != 0) {
9          fprintf(stderr, "fclose failed\n");
10         return 1;
11     }
12
13     return 0;
14 }
15

```

## Files in C 2 of 5

- Reading from Files

- **Syntax:** `char *fgets(char *s, int n, FILE *stream)`
- Reads data line by line
  1. *char \*s* is a pointer to memory where text can be stored
    - \* Note new var can be created here, like for loop (i.e. `for(i=0; i < 1; i++)`).
    - \* On success, `fgets` returns *s*
    - \* On failure, `fgets` returns `NULL`
  2. *int n* is the maximum upper number of characters `fgets` allowed to put in *s*

```

1  #include <stdio.h>
2
3  #define LINE_LENGTH 80
4
5  int main() {
6      FILE *sample_file;
7      int error;
8      char line[LINE_LENGTH + 1];
9
10     sample_file = fopen("example_sources/sample.txt", "r");
11
12     while (fgets(line, LINE_LENGTH + 1, sample_file) != NULL) {
13         printf("%s", line);
14     }
15
16     ...
17     return 0;
18 }
19

```

- Reading from Input

- **Syntax:** `fgets(line, LINE_LENGTH + 1, stdin)`
- Notice *stdin* is the standard input, like `input` in Python

```
1  #include <stdio.h>
2
3  #define LINE_LENGTH 80
4
5  int main() {
6      char line[LINE_LENGTH + 1];
7
8      while (fgets(line, LINE_LENGTH + 1, stdin) != NULL) {
9          printf("%s", line);
10     }
11
12     return 0;
13 }
14
```

## Files in C 3 of 5

- The `scanf` function
  - returns successfully read items
  - number of read items depends on format
  - **Syntax:** `int fscanf(FILE *stream, const char *format, type *s, type *n)`

```
1  #include <stdio.h>
2
3  #define LINE_LENGTH 80
4
5  int main() {
6      FILE *sample_file;
7      int error, score, total;
8
9      sample_file = fopen("example_sources/sample.txt", "r");
10     if (sample_file == NULL) {
11         perror("Error opening file\n");
12         return 1;
13     }
14
15     while (fscanf(sample_file, "%d %d", &score, &total) == 2) { //
16         <- ==2 means each fscan must return 2 values, one for each col.
17         printf("Score: %d, Total: %d.\n", score, total);
18     }
19
20     error = fclose(sample_file);
21     if (error != 0) {
```

```
21         perror("fclose failed on input file\n");
22         return 1;
23     }
24
25     return 0;
26 }
27
```

## Files in C 4 of 5

- Writing to Files

- **Syntax:** `output_file = fopen(const char *filename, "w")`

- \* *w* overwrites existing file

- \* *a* appends output to the end of file

```
1      #include <stdio.h>
2
3      int main() {
4          FILE *output_file;
5          int error;
6          int total = 50;
7          float small_number = 0.125;
8
9          output_file = fopen("example_sources/output.txt", "w")
10         ;
11         if (output_file == NULL) {
12             perror("Error opening file\n");
13             return 1;
14         }
15
16         fprintf(output_file, "The first line in the file\n");
17         fprintf(output_file, "The integer is %d\n", total);
18         fprintf(output_file, "The small float is %f\n",
19             small_number);
20
21         ...
22
23         return 0;
24     }
25
```

- NOTE: the output stream is first stored in memory controlled by OS before to disk



- \* OS periodically writes content from memory to disk
- \* Abnormal computer shutdown → some data may be lost

1  
2

## Files in C 5 of 5

- Bringing Everything Together

```
1  #include <stdio.h>
2
3  int main() {
4      FILE *sample_file, *output_file;
5      int error_open, error_closed, score;
6      int total = 50;
7
8      sample_file = fopen("example_sources/sample.txt", "r");
9      if (sample_file == NULL) {
10         fprintf(stderr, "Error opening file \n");
11         return 1;
12     }
13
14     output_file = fopen("example_sources/output.txt", "w");
15     if (output_file == NULL) {
16         perror("Error opening file\n");
17         return 1;
18     }
19
20     while (fscanf(sample_file, "%d %d", &score, &total) == 2) {
21         printf("Score: %d\n", score);
22         fprintf(output_file, "Score: %d\n", score);
23     }
24
25     error_open = fclose(sample_file);
26     if (error_open != 0) {
27         perror("fclose failed\n");
28         return 1;
29     }
30
31     error_closed = fclose(output_file);
```

```

32     if (error_closed != 0) {
33         perror("fclose failed\n");
34         return 1;
35     }
36
37     return 0;
38 }
39

```

Listing 1: files\_example\_5.c

## Strings in C 1 of 6

- Introduction to Strings
  - String is an array of chars with ‘\0’ at the end
    - \* i.e. ‘hello’ = [‘h’,‘e’,‘l’,‘l’,‘o’,‘\0’]
    - \* without ‘\0’, undesired output is included, i.e. hello?[BT]

## Strings in C 2 of 6

- Initializing Strings and String Literals
  - There are two ways
    1. Using array

```

1     #include <stdio.h>
2
3     int main() {
4         char text[20] = {'h','e','l','l','o','\0'};
5
6         printf("%s\n", text);
7         return 0;
8     }
9

```

\* The following is how array looks like after initialization

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
h	e	l	l	o	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

2. Using array of chosen size and double quoted string

```

1  #include <stdio.h>
2
3  int main() {
4      char text[20] = "hello";
5
6      printf("%s\n", text);
7      return 0;
8  }
9

```

\* The following is how array looks like after initialization

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
h	e	l	l	o	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

\* Note: `char text[5] = "hello";` causes error, since `'\0'` is not included.

### 3. Using array of unchosen size and double quoted string

```

1  #include <stdio.h>
2
3  int main() {
4      char text[] = "hello";
5
6      printf("%s\n", text);
7      return 0;
8  }
9

```

\* Here, the size of string is just enough for characters plus `'\0'`.

### 4. Using pointers

```

1  #include <stdio.h>
2
3  int main() {
4      char *text = "hello";
5
6      printf("%s\n", text);
7      return 0;
8  }
9

```

## Strings in C 2 of 6

- Initializing Strings and String Literals

- There are two ways

1. Using array

```

1  #include <stdio.h>
2
3  int main() {
4      char text[20] = {'h','e','l','l','o','\0'};
5
6      printf("%s\n", text);
7      return 0;
8  }
9

```

\* The following is how array looks like after initialization

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
h	e	l	l	o	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

## 2. Using array of chosen size and double quoted string

```

1  #include <stdio.h>
2
3  int main() {
4      char text[20] = "hello";
5
6      printf("%s\n", text);
7      return 0;
8  }
9

```

\* The following is how array looks like after initialization

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
h	e	l	l	o	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

\* Note: `char text[5] = "hello";` causes error, since `'\0'` is not included.

## 3. Using array of unchosen size and double quoted string

```

1  #include <stdio.h>
2
3  int main() {
4      char text[] = "hello";
5
6      printf("%s\n", text);
7      return 0;
8  }
9

```

\* Here, the size of string is just enough for characters plus `'\0'`.

## 4. Using pointers

```

1  #include <stdio.h>
2
3  int main() {
4      char *text = "hello";
5
6      printf("%s\n", text);
7  }

```



```
7     return 0;
8 }
9
```

## Strings in C 3 of 6

- Size and Length

- strlen

- \* **Syntax:** size\_t strlen(const char \*s)
    - \* is from c-string library, i.e. '<string.h >'
    - \* is the recommended way to determine the length of string

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char weekday[10] = "Monday";
6      printf("Length of string: %lu\n", strlen(weekday)); // <-
Returns 6
7      ...
8      return 0;
9  }
10
```

Listing 2: strings\_example\_3.c

- sizeof

- \* returns total size of array
    - \* not a good way to measure the length of string

## Strings in C 4 of 6

- Copying Strings

- strncpy

- \* **Syntax:** char \*strncpy(char \*s1, const char \*s2, int n);
    - \* **strncpy:** is a function from c-string library
    - \* **s1:** is destination
    - \* **s2:** is source
    - \* **n** is max number of characters to be copied from source

\*  $n$  is determined based on the string of lesser size

```

1      #include <stdio.h>
2      #include <string.h>
3
4      int main() {
5          char s1[5];
6          char s2[32] = "University of";
7          strncpy(s1,s2, sizeof(s1));
8          s1[4] = '\0';
9          printf("%s\n", s1);
10         printf("%s\n", s2);
11         return 0;
12     }
13

```

Listing 3: strings\_example\_4.c

· Note: ‘\0’ is added at the end of  $s1$  to ensure the copied string is safe.

– strcpy

- \* Don't do it.
- \* This is not safe.

## Strings in C 5 of 6

- Concatenating Strings

– strcat

- \* **Syntax:** `char *strncat(char *s1, const char *s2, int n);`
  - **s1:** is the destination
  - **s2:** is the source
  - **n:** is the max number of characters without null terminator to be copied from  $s2$  to  $s1$ .

This is usually `sizeof(s1)`

```

1      #include <stdio.h>
2      #include <string.h>
3
4      int main() {
5          char s1[30];
6          char s2[14] = "University of";
7          char s3[15] = "C Programming";
8
9          strncpy(s1,s2, sizeof(s1));
10         s1[sizeof(s1)-1] = '\0';

```

```
11     strcat(s1, s3, sizeof(s1) - strlen(s1) - 1); // -1 is
    for \0.
12     printf("%s\n", s1);
13     printf("%s\n", s2);
14     printf("%s\n", s1);
15     return 0;
16 }
17
```

Listing 4: strings.example\_5.c

- Note:  $\text{sizeof}(s1) - \text{strlen}(s1) - 1$  is the remaining space in  $s1$  excluding the null character.

– strcat

- \* This is not safe.
- \* Don't use it. No No!!