ENGG1003 - Tuesday Week 7

File I/O
More Pointers

Brenton Schulz

University of Newcastle

April 8, 2019

Che C Documentation

- Linux systems have a program called "man"
  - Short for "manual"
- It is used to display a wide variety of documentation called "man pages"
- To install it type this in the terminal:

```
sudo apt update
sudo apt install man
```

  and press `y` (or `<enter>`) when prompted to confirm installation
- Afterwards, C documentation can be accessed by typing `man <topic>`

Che C Documentation

- For example, all library functions have a `man` page you can read by typing:
  `man <function name>`
- eg, try:
  - `man fopen`
  - `man printf`
  - `man sin`
  - `man string`
  - etc..

File I/O

- A stream is kept in a variable of type `FILE *`
  - Read as "pointer to `FILE`" or "`FILE`-star"
- Three already exist in your C programs:
  - `stdin`
  - `stdout`
  - `stderr`
- Additional streams are declared like other variables, eg:

```
1 FILE *input, *output;
```

Correction: String Initialisation

- This is totally fine:

```
1 char string[] = "initial value";
```

- The compiler copies the string literal into `string[]`
- The length is automatically calculated
  - You may specify a length *longer* than necessary:

```
1 char string[1024] = "initial value";
```

Correction: String Initialisation

- This is totally fine:

```
1 char string[] = "initial value";
```

- The compiler copies the string literal into `string[]`
- The length is automatically calculated
  - You may specify a length *longer* than necessary:

```
1 char string[1024] = "initial value";
```

- A constant string is created with:

```
1 char *str = "some string";
```

  - We will study this *pointer* syntax later

File I/O - Quick Review

- Before a file can be accessed you must *open* it with the `fopen()` function
- In order to open files you need two pieces of information:
  - The file's name
  - The data direction (mode)
    - Reading
    - Writing
    - Both

File I/O

- `fopen()`'s function prototype is:

```
1 FILE *fopen(const char *name, const char *mode);
```

- `const char *name` is a string holding the file's name
- `const char *mode` is a string describing the desired data direction
- Both of these can be passed as variable strings or hard-coded

File I/O

- The `*mode` argument can be one of the following:
  - `"r"` (reading)
  - `"r+"` (reading and writing)
  - `"w"` (writing)
  - `"w+"` (reading and writing, file truncated)
  - `"a"` (appending)
  - `"a+"` (reading and appending)
- Read documentation for details
- `fopen()` example:

```
1 FILE *input;
2 input = fopen("data.txt", "r");
```

`fopen()` Errors

- The return value of `fopen()` is `NULL` on error
- Check it! Attempting to access a `NULL` stream will result in a segmentation fault!

```
1 FILE *input;
2 input = fopen("data", "r");
3 if(input == NULL) {
4   perror("fopen()");
5   return;
6 }
```

- `perror()` prints a user-friendly error message

File I/O

- Once opened, a file can be accessed with:
  - `fscanf()`
  - `fprintf()`
- These functions behave just like `scanf()` and `printf()` except they take an extra argument:

```
1 int fscanf(FILE *stream, const char *format,
    ...);
```

- The first argument is a `FILE *`
- The rest is identical to `printf()` and `scanf()`

File I/O - Position Indicators

- Concept: bytes in files have an address known as a *position indicator*
- The address is the number of bytes, starting at zero, from the start of the file
- Unless otherwise controlled, files are only read from and written to *sequentially*
- The position indicator automatically increments when a byte is read or written

File I/O - Position Indicators

- Some useful functions:
  - `ftell()` - Returns the position indicator
  - `fseek()` - Sets the position indicator
  - `feof()` - Returns TRUE if the position indicator is at the end of the file
- For example, to process data until the end of file is reached:

```
1 FILE *stream;
2 // open file etc
3 while(!feof(stream)) {
4   // Read from file
5   // Do stuff
6 }
```

File I/O Example

Write a C program which opens a file, `test.txt`, and prints its contents to `stdout`, reading and writing one character at a time.

File I/O Example

Write a C program which opens a file, `test.txt`, and prints its contents to `stdout`, reading and writing one character at a time.

- Declare `FILE *input;`

File I/O Example

Write a C program which opens a file, `test.txt`, and prints its contents to `stdout`, reading and writing one character at a time.

- Declare `FILE *input;`
- Use `fopen()` to open it for reading

File I/O Example

Write a C program which opens a file, `test.txt`, and prints its contents to `stdout`, reading and writing one character at a time.

- Declare `FILE *input;`
- Use `fopen()` to open it for reading
- Write a loop which reads and writes characters until the whole file has been read
  - Read with: `fscanf(input, "%c", &c);`
  - Write with: `fprintf("%c", c);`

File I/O Example 1

Write a C program which opens a file, `input.txt`, then reads and prints each character to the console on a new line, indicating the position indicator's value *after* reading each character.

File I/O Example 2

Write a C program which copies a file, `input.txt`, into a new file, `output.txt`. While copying, the program should count how many spaces there are in the input and print the final count to the terminal before exiting.

File I/O Example 3

Write a C program which opens a file, `input.txt`, and counts the number of times the string `"the"` appears.

The program should include a function, `isThe()`, which tests if a string is equal to `"the"` or not.

Pointers