Input/output in C

Computing Lab

https://www.isical.ac.in/~dfslab

Indian Statistical Institute

Basic output to the terminal

- `putchar(int c)` : print `c`, cast to an unsigned char, to stdout (terminal)
- `puts(const char *s)` : print string `s` + newline to stdout
- `printf(const char *format , ...)` : contains zero or more *conversion specifiers*

optional

`printf`: conversion specifiers

flags	width	.precision	length modifier	type

- flags
  - `0` : right align, with zero padding on the left
  - `-` : left align
- width: *minimum* field width
- precision: usually, number of significant digits (for floating point numbers) /
  maximum number of characters to be printed (for string)
- length modifier: `l`, `ll`, `L`, `z`
- type: `d`, `i`, `c`, `f`, `s`

section no. in manual

See man pages for detailed documentation, e.g., `$ man 3 printf`

Reading from the “terminal”

Character at a time:

- Functions: `getchar()` OR `fgetc(stdin)` (equivalent)
- Return value: reads the next character and returns it as an `unsigned char` cast to an `int`, or `EOF` on end of file or error.
- Typical usage: `while (EOF != (c = fgetc(fp))) ...`
- Caution: Do not forget to declare `c` as `int` type.

  If `c` is of type `char`:
  - reading from the terminal: will probably work without any problem
  - reading from a file: in some (rare) cases, problems could occur

Reading from the “terminal”

Line at a time:

- Function: `fgets(s, n, stdin)`
- Return value
  - reads at most `n-1` characters or one line (whichever is shorter), stores input in character array `s` and terminates `s` using `'\0'`
  - if a newline is read, it is stored in `s`
  - returns `s` or `NULL` on end of file (i.e., there is nothing to be read) or error
- Typical usage: `while (NULL != fgets(s, n, stdin)) ...`
- Caution: **Without exception, do not use `gets()`!**

Reading from the terminal: `scanf`

Format string:

`" "`, `"\t"`, `"\n"` or any sequence of these characters	matches any amount of white space, including none
`"%d"`, `"%ld"`, `"%lld"`	read an `int`, `long int` or `long long int`, possibly with leading `+` or `-` sign
`"%u"`, `"%lu"`, `"%llu"`	read an `unsigned int`, `unsigned long int` or `unsigned long long int`
`"%f"`, `"%lf"`, `"%llf"`	read a `float`, `double` or `long double`, possibly with leading `+` or `-` sign
`"%c"`	read a single character (*including white-space*)

Review questions

1. Write a program that reads the given file (`getc-input.txt`) one character at a time using `fgetc`. After each character is read, print it along with its ASCII value to the screen.

   Try storing the return value of `fgetc` in an `int` type variable and a `char` type variable in turn, and report any observed difference in the behaviour of your program.

   NOTE: Do NOT open `getc-input.txt` in the browser, and copy-paste the content into a local file. Right-click on the link, save the file locally, and run your program on the downloaded file using input redirection (`./a.out < getc-input.txt`).

2. Compile and run the programs `basic-io.c` and `robust-scanf.c` on various kinds of mixed input including characters, white-space, digit sequences, punctuation marks, etc.

   NOTE: For the first part of `basic-io.c`, you may use `test-input-14082024.txt` as a test input file.

Practice problems

1. Write a program that reads text typed at the terminal, and counts
   - the number of occurrences of vowels in the input text;
   - the number of words in the input text. Assume that any contiguous sequence of letters and / or digits forms one word.