Basic I/O Task

The aim of this task is to become familiar with basic input and output operations from `stdin` and to `stdout`, respectively.

Step #1: Basic Input/Output

The standard C library `libc.so` provides various functions via the header files `<stdio.h>` that enables input from `stdin` (typically your keyboard) and output to `stdout` (typically the terminal program where you started the program). Some of these functions are listed in the following table.

Input	Output	Description
`getc()`, `getchar()`	`putc()`, `putchar()`	transfer a single ASCII symbol at a time
`fgets()`	`puts()`	transfer a string including the `\n` escape sequence
`scanf()`	`printf()`	format (using format modifiers) and transfer data
`fread()`	`fwrite()`	transfer a number of bytes

You'll find some examples for each of these functions at https://devdocs.io/c/io.

Task #1: Using `getc()`, `getchar()`, `putc()` and `putchar()` (i) read-in at least 5 symbols from `stdin`, (ii) store them in respective variables, and (iii) output them to `stdout`. Carefully, monitor what you type (which keys on the keyboard) and how the program behaves. Document your observations as comments in the code. Repeat the same task for the function pairs `fgets()` and `puts()` and `fread()` and `fwrite()`. The program shall also validate the input. Print respective information to `stdout`, e.g., whether the given input was a *control* symbol, a *digit*, a *lowercase* or an *uppercase* letter etc. To that end, checkout the ASCII table and/or functions provided via the `<ctype.h>` header file like `isdigit()`, `isalpha()` etc. What do you get and how to validate input for *not so common* keys like `ö`, `ß`, `@`, `\`, `§`, `esc`, or `strg`?

Put all your code in a single C-source file along with a proper comment header describing your tests and findings, name the file `yourname_task01.c` (substitute `yourname` with your family name in lowercase letters; use *ue* rather than *ü* etc.), and upload it to the *Basic I/O Task* section via the Moodle class.

Step #2: Formatted Input/Output

The input and output function families of `printf()` and `scanf()` allow for reading and writing to multiple variables and converting these values between different types. To that end, these functions provide something called *format modifiers* and a *variable argument list*. Format modifiers are written within the first argument enclosed by double quotes. For (almost) every format modifier you'll need to provide a variable (in case of `printf()`) or the address of a variable (in case of `scanf()`). The sequence and number of format modifiers and variables need to match.

The *address of a variable* is obtained by putting the `&` symbol in front of the name of a variable - see the example below.

The following table lists some of the more common format modifiers of these function families. A somewhat more complete list can be found in the documentation, e.g.: at

- https://devdocs.io/c/io/fprintf or
- https://devdocs.io/c/io/fscanf, respectively.

Modifier	Type	Description
`%c`	`char`	output the value of a variable as a symbol according to the ASCII table
`%s`	`char []`	output the values stored in a char array as ASCII symbols
`%d`	`int`	output the value of a variable as a decimal integer
`%i`	`int`	output the value of a variable as a decimal integer
`%u`	`unsigned int`	output the value of a variable as a unsigned decimal integer
`%x`	`int`	output the value of a variable as a hexadecimal number
`%f`	`float`	output the value of a variable as a floating point number
etc.		

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

int main(void) {
  int n = 0, age = 0;
  float weight = 0.0;
  char name[5];
  n = scanf("%4s %d %f", &name[0], &age, &weight);
  assert (3 == n);
  printf("Name: %s, Age: %d, Weight %2.2f\n", name, age, weight);
  return 0;
}
```

An execution sequence of this program could look like:

```
Tom 20 69.3
Name: Tom, Age: 20, Weight 69.30
```

Task #2: Create a program using `scanf()` and `printf()` to read integer, floating-point numbers, characters and strings in intermixed sequences into various variables. Add code to validate your input to some extend. Finally, output all the readings using `printf()`.

Put all your code in a single C-source file along with a proper comment header describing your tests and findings, name the file `yourname_task02.c` (substitute `yourname` with your family name in lowercase letters; use *ue* rather than *ü* etc.), and upload it to the *Basic I/O Task* section via the Moodle class.