# MyPrintf

Deadline: September 22nd, 2014

## 1 Instructions

You must implement the following function:

```
int st_printf(st_t *, const char *fmt, ...);
```

Constraints:

- each function must be implemented in a `.c` file of its own, named after the function it contains. The function prototypes must be declared in a `.h` file, in accordance with the C coding standard. The submitted archive may (but needs not) include a test program.
- you must include a `Makefile` which properly places the function(s) in `libminic.a`.
- you must not include any standard/system header in your code, except for `<stdarg.h>`, `<stddef.h>` and `<stdlib.h>`.
- you must not use any function from the standard C library, except for `malloc`/`realloc`/`free`; however, you may use functions from a previous assignment, by including their source in your submission.

## 2 Function semantics

`st_printf` is based on the previous assignment (MyStream). It must format its arguments and output them to the stream identified by its first argument, buffering the output as needed.

The format string is composed of zero or more directives: ordinary characters (not "%"), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent argument. `st_printf` must recognize the following conversions:

| Conversion | Meaning |
|---|---|
| `%c` | An `int` argument is converted to `unsigned char`, and the resulting character is written. |
| `%s` | A `const char*` argument is written as a nul-terminated string. |
| `%p` | The value of a pointer is written in hexadecimal with a `0x` prefix, or the string "`null`" if the pointer is equal to 0. |

... continued on next page

| Conversion | Meaning |
|---|---|
| `%d` / `%ld` | A `int`/`long` argument is written in decimal. |
| `%u` / `%lu` | A `unsigned`/`unsigned long` argument is written in decimal. |
| `%x` / `%lx` | An `unsigned`/`unsigned long` argument is written in hexadecimal. |
| `%%` | A single "`%`" is written; no argument is converted. |

The function must return the number of characters written/buffered, or 0 if no character could be written.

You may implement the following for a higher grade:

- an optional "minimum width" in decimal between the "`%`" character and the format. For example, `st_printf(st, "%10s", "hello")` prints 5 spaces followed by "`hello`".
- the modifier "`-`" to align the field on the left instead of the right. For example, `st_printf(st, "%-10s", "hello")` prints "`hello`" followed by 5 spaces.
- the modifier "`0`" to pad a number with zeroes. For example, "`st_printf(st, "%06d", 123)`" would print "`000123`".
- a variable "minimum width" with `*`, where the width is determined by an `int` positional argument. For example, `st_printf(st, "%*s", 6, "hello")` prints 1 spaces then "`hello`".
- the function:

  ```
  size_t my_snprintf(char* buf, size_t n, const char* fmt, ...);
  ```

  which formats its arguments like `st_printf` but outputs the characters to the buffer `buf`, with a maximum of `n`-1 characters, terminated with a nul character. `my_snprintf` must return the number of characters written (excluding the terminating nul) if `n` is large enough, or the number of characters that would have been written otherwise.

# 3 Grading

- 0.75 point per conversion correctly implemented in the mandatory list (7.50 points in total);
- +0.5 if constant minimum width and alignment modifiers are properly implemented.
- +0.5 if zero-padding is properly implemented.
- +0.5 if variable minimum width is properly implemented.
- +0.5 if `st_printf` does not use `malloc`/`realloc`/`free`.
- +0.5 if `my_snprintf` is properly implemented.