Formatted Printing Function

Header file

- `stdio.h` contains the function prototype and any other definitions that are needed for the `printf` function

`int printf(const char *format, ...)`

- formats and prints its arguments as specified by the `format` string.
- Plain characters in `format` are simply copied
- Format specifications are made up of a the percent sign (`%`) followed by one of the following conversion operators, which determine what `printf` does with its arguments:
  - **%** - print a single `%` character
  - **c** - convert an `int` to an `unsigned character` and print the resulting character
  - **d** or **i** - print an `int` as a signed decimal number
  - **u** - print an `unsigned` as an unsigned decimal number
  - **o** - print an `unsigned` as an unsigned octal number
  - **x** or **X** - print an `unsigned` as an unsigned hexadecimal number (where the letters `abcdef` are used with `x` and `ABCDEF` are used with `X`
  - **e** or **E** - print a `double` using an exponential format like:
    `[-]`*`d.ddd`*`e[+-]`*`dd`*
    (where the `e` is replaced by `E` if the uppercase `E` is specified)
  - **f** - print a `double` using a decimal format like `[-]`*`ddd.ddd`*
  - **g** or **G** - print a `double` using the same decimal format used by the `f` specification unless the exponent is less than -4 or greater than or equal to the specified precision (as described below), in which case the `e` format is used (replacing the `e` with an `E` if the `G` format specification is used.) Trailing zeros are removed from the right side of the decimal point. If there would be no digits to the right of the decimal point, the decimal point is also omitted.
  - **s** - print the string pointed to by a `char *`
  - **p** - print a `void *` argument in hexadecimal *(ANSI C only)*
  - **n** - store the number of characters printed at this point in the interger pointed to by the `int *` argument. Nothing is printed. *(ANSI C only)*
- The conversion operator may be preceeded by zero or more of the following flag characters:
  - **#** specifies that the value should be converted to an alternate form:
    - For **o**, the precision (described below) is increased so that the first digit printed is a `0`
    - For **x** or **X**, a non-zero value has a `0x` prepended (or `0X` for the `X` specification)
    - For **e**, **E**, **f**, **g** or **G**, the result will always contain a decimal point, even if no digits follow it. Additionally, trailing zeros are not removed from numbers formatted with **g** or **G**
  - **0** specifies that the value printed should be padded on the left with zeros to the maximum width specified
  - **-** specifies that the value should be left justified (and padded with spaces to the right). If both **0** and **-** flags are specified, the **0** flag is ignored.
  - A space character specifies that a blank should be left before a positive number in a **d**, **e**, **E**, **f**, **g**, **G** or **i** conversion
  - **+** specifies that a plus sign should placed before a positive number in a **d**, **e**, **E**, **f**, **g**, **G** or **i** conversion. If both **+** and space character flags are specified, the space character flag is ignored.
- The flag(s) (if any) may be followed by an optional minimum field width specification, written as a decimal integer. If the value to be printed is shorter than the field width, it is padded with spaces (or `0`s if the `0` flag was specified) to the left (or, if the `-` flag was specified, to the right with spaces)
- Alternatively, the minimum field width specification may be a `*`, in which case the value to be printed is assumed to be preceeded by an `int` argument which is used to specify the minimum width.

- The flags(s) and/or field width may be followed by a precision specification, written as a period followed by a decimal integer, which specifies:
  - the minimum number of digits to be printed for **d**, **i**, **o**, **u**, **x**, and **X** conversions
  - the number of digits to the right of the decimal-point for **e**, **E**, and **f** conversions
  - the number of significant digits for **g** and **G** conversions
  - the maximum number of characters to be printed from a string for **s** conversions
- Alternatively, the precision specification may be a `*`, in which case the value to be printed is assumed to be preceeded by an `int` argument (following the `int` for the minimum field width, if `*` is specified for it as well) which is used to specify the precision.
- Some conversion operators may also be preceeded by a size specification:
  - **h** indicates that the argument associated with a **d**, **i**, **o**, **u**, **x** or **X** operator is a `short` or `unsigned short`, or that the argument to an **n** is a `short *` *(ANSI C only)*
  - **l** indicates that the argument associated with a **d**, **i**, **o**, **u**, **x** or **X** operator is a `long` or `unsigned long`, or that the argument to an **n** is a `long *`
  - **L** indicates that the argument associated with a **e**, **E**, **f**, **g** or **G** operator is a `long double` *(ANSI C only)*

Here is a demonstration program and the output it produces.

Previous, Next, Index