ALX Software Enginering Printf Team Project

This team project is a custom made printf function for the C programming language called _printf. It has been optimized to take various inputs and optional arguments based exactly on how the standard library function printf works. We submitted this as part of the ALX software engineering course requirement for grading.

Synopsis

This function **_printf()** writes output to stdout, the standard output stream with the format and options without making use of any of the standard library files. It was written to use a local buffer of 1024 bytes when printing although it can print larger sets of data.

The _printf() function returns the total number of characters printed to the stdout(excluding the null byte at the end of strings) after a successful execution.

If an output error is encountered, a negative value of -1 is returned.

The prototype of this function is: **int _printf(const char** ***format, ...);***

This means that it has one mandatory format argument, and an extra number of arguments that can be none, or many.

Format of the format string

The format string is a character string starting and ending with double quotes. The format string is composed of zero or more directives; ordinary characters (not %), and conversion specifications, each of which results in fetching zero or more subsequent arguments.

Each conversion specification is introduced by the character % and ends with a **conversion specifier**. In between there may be (in this order):

> Zero or more **flags**
>
> An optional field **width**
>
> An optional **precision** modifier
>
> An optional **length** modifier

The flag characters

Flag	Description
#	For **o** conversions the first character of the output string is made zero (by prefixing a 0 if it was not zero already). For **x** and **X** conversions, a nonzero result has the string "**0x**" or "**0X**" respectively added.
0	(Not implemented yet) The value should be zero padded. For **d**, **i**, **o**, **u**, **x**, and **X** the converted value is padded on the left with zeros. If the 0 and - flags both appear,the **0** flag is ignored. If a precision is given with a numeric conversion, the **0** flag is ignored.
-	(Minus sign, not implemented yet) The converted value is to be left adjusted on the field boundary, (Default is right justification) and padded with blanks in the right rather than on the left with blanks or zeros. This flag overrides **0** if both are given.
' '	(Blank Space) The argument is padded with a single blank space before a positive number or empty string produced by a signed conversion.
+	A sign (+ or -) should always be placed before a number produced with a signed conversion. By default, only negative numbers have this sign.

The field width

An optional decimal digit string (with nonzero first digit) specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left if the flag - is not present, and on the right if it is present. A character * can be used instead of a decimal string. In this case, an argument passed to the function will be taken as the width value.

```
printf("%5d", num);
```

or

```
printf("%*d", width, num);
```

The precision

An optional precision, in the form of a period ('.') followed by an optional decimal digit string. A negative precision is taken as if the precision were omitted. This gives the minimum number of digits to appear for d, i, o, u, x, and X conversions, or the maximum number of characters to be printed from a string for s and S conversions. A character * can be used instead of a decimal string. In this case, an argument passed to the function will be taken as the precision value.

```
printf("%.3d", num);
```

or

```
printf("%.*d", precision, num);
```

The length modifiers

Modifier	Description
l	An integer conversion to a **long int** or **unsigned long int** argument.
h	An integer conversion to a **short int** or **unsigned short int** argument.

The conversion specifier

Specifier	Description
d, i	The argument **int** is converted to a signed decimal notation. If precision is present,it gives the minimum number of digits that must appear; if the converted value requires fewer digits, then it is padded with zeros on the left. Default precision is 1.
o, u, x, X	The argument is converted to unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexamedical (**x** and **X**) notation. The letters abcdef are used for x conversion and the letters ABCDEF are used for X conversion. If precision is present, it will give the minimum number of digits that must appear; if the converted value requires fewer digits, then it will be padded with zeros. By default the precision is 1.
c	The int argument is converted to an unsigned char and the resulting character is written. The representation of characters is based off the ASCII coding.
s	The argument received is expected to be a pointer type char * to an array of characters. Characters from this array are printed up to (but not including) a null byte (**'\0'**). If precision is specified, then this will determine how many characters are taken into account for printing.
p	A void * pointer argument is printed as hexadecimal in lower caps representing an adress in memory.
%	A ' % ' character is written and no conversion is made. The specification is as follows: %%.
b	The argument is converted to an unsigned int value and then operated to get its binary representation (base 2).
S	The argument received is expected to be a pointer type char * to an array of characters. Characters from this array are printed up to (but not including) a null byte ('\0'). Non printable characters (0 < ASCII value < 32 or >= 127) are printed this way: \x, followed by the ASCII code value in hexadecimal (upper case - always 2 characters).
r	The argument received is expected to be a pointer type char * to an array of characters. Characters from this array are printed in reverse order up to (but not including) a null byte ('\0').
R	The argument received is expected to be a pointer type char * to an array of characters. Characters from this array are encoded to ROT13 and printed in order up to (but not including a null byte ('\0').

Author

Yonas Leykun