_printf.c

```
#include "main.h"

void print_buffer(char buffer[], int *buff_ind);

/**
 * _printf - Printf function
 * @format: format.
 * Return: Printed chars.
 */
int _printf(const char *format, ...)
{
	int i, printed = 0, printed_chars = 0;
	int flags, width, precision, size, buff_ind = 0;
	va_list list;
	char buffer[BUFF_SIZE];

	if (format == NULL)
		return (-1);

	va_start(list, format);

	for (i = 0; format && format[i] != '\0'; i++)
	{
		if (format[i] != '%')
		{
			buffer[buff_ind++] = format[i];
			if (buff_ind == BUFF_SIZE)
				print_buffer(buffer, &buff_ind);
			/* write(1, &format[i], 1);*/
			printed_chars++;
		}
		else
		{
			print_buffer(buffer, &buff_ind);
			flags = get_flags(format, &i);
			width = get_width(format, &i, list);
			precision = get_precision(format, &i, list);
			size = get_size(format, &i);
			++i;
			printed = handle_print(format, &i, list, buffer,
```

```
                                flags, width, precision, size);
                        if (printed == -1)
                                return (-1);
                        printed_chars += printed;
                }
        }

        print_buffer(buffer, &buff_ind);

        va_end(list);

        return (printed_chars);
}

/**
 * print_buffer - Prints the contents of the buffer if it exist
 * @buffer: Array of chars
 * @buff_ind: Index at which to add next char, represents the length.
 */
void print_buffer(char buffer[], int *buff_ind)
{
        if (*buff_ind > 0)
                write(1, &buffer[0], *buff_ind);

        *buff_ind = 0;
}
```

functions.c

```
#include "main.h"

/************************* PRINT CHAR *************************/

/**
 * print_char - Prints a char
 * @types: List a of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: Width
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed
 */
int print_char(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	char c = va_arg(types, int);

	return (handle_write_char(c, buffer, flags, width, precision, size));
}
/************************* PRINT A STRING *************************/
/**
 * print_string - Prints a string
 * @types: List a of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width.
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed
 */
int print_string(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	int length = 0, i;
	char *str = va_arg(types, char *);

	UNUSED(buffer);
```

```
	UNUSED(flags);
	UNUSED(width);
	UNUSED(precision);
	UNUSED(size);
	if (str == NULL)
	{
		str = "(null)";
		if (precision >= 6)
			str = "      ";
	}

	while (str[length] != '\0')
		length++;

	if (precision >= 0 && precision < length)
		length = precision;

	if (width > length)
	{
		if (flags & F_MINUS)
		{
			write(1, &str[0], length);
			for (i = width - length; i > 0; i--)
				write(1, " ", 1);
			return (width);
		}
		else
		{
			for (i = width - length; i > 0; i--)
				write(1, " ", 1);
			write(1, &str[0], length);
			return (width);
		}
	}

	return (write(1, str, length));
}
/************************* PRINT PERCENT SIGN ************************/
/**
 * print_percent - Prints a percent sign
 * @types: Lista of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width.
```

```
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed
 */
int print_percent(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	UNUSED(types);
	UNUSED(buffer);
	UNUSED(flags);
	UNUSED(width);
	UNUSED(precision);
	UNUSED(size);
	return (write(1, "%%", 1));
}

/************************* PRINT INT *************************/
/**
 * print_int - Print int
 * @types: Lista of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width.
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed
 */
int print_int(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	int i = BUFF_SIZE - 2;
	int is_negative = 0;
	long int n = va_arg(types, long int);
	unsigned long int num;

	n = convert_size_number(n, size);

	if (n == 0)
		buffer[i--] = '0';

	buffer[BUFF_SIZE - 1] = '\0';
	num = (unsigned long int)n;

	if (n < 0)
```

```
	{
		num = (unsigned long int)((-1) * n);
		is_negative = 1;
	}

	while (num > 0)
	{
		buffer[i--] = (num % 10) + '0';
		num /= 10;
	}

	i++;

	return (write_number(is_negative, i, buffer, flags, width, precision, size));
}

/************************* PRINT BINARY *************************/
/**
 * print_binary - Prints an unsigned number
 * @types: Lista of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width.
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Numbers of char printed.
 */
int print_binary(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	unsigned int n, m, i, sum;
	unsigned int a[32];
	int count;

	UNUSED(buffer);
	UNUSED(flags);
	UNUSED(width);
	UNUSED(precision);
	UNUSED(size);

	n = va_arg(types, unsigned int);
	m = 2147483648; /* (2 ^ 31) */
	a[0] = n / m;
	for (i = 1; i < 32; i++)
```

```
	{
		m /= 2;
		a[i] = (n / m) % 2;
	}
	for (i = 0, sum = 0, count = 0; i < 32; i++)
	{
		sum += a[i];
		if (sum || i == 31)
		{
			char z = '0' + a[i];

			write(1, &z, 1);
			count++;
		}
	}
	return (count);
}
```

functions1.c

```
#include "main.h"

/************************ PRINT UNSIGNED NUMBER ************************/
/**
 * print_unsigned - Prints an unsigned number
 * @types: List a of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed.
 */
int print_unsigned(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	int i = BUFF_SIZE - 2;
	unsigned long int num = va_arg(types, unsigned long int);

	num = convert_size_unsgnd(num, size);

	if (num == 0)
		buffer[i--] = '0';

	buffer[BUFF_SIZE - 1] = '\0';

	while (num > 0)
	{
		buffer[i--] = (num % 10) + '0';
		num /= 10;
	}

	i++;

	return (write_unsgnd(0, i, buffer, flags, width, precision, size));
}

/************* PRINT UNSIGNED NUMBER IN OCTAL  ****************/
/**
 * print_octal - Prints an unsigned number in octal notation
```

```
 * @types: Lista of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed
 */
int print_octal(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{

	int i = BUFF_SIZE - 2;
	unsigned long int num = va_arg(types, unsigned long int);
	unsigned long int init_num = num;

	UNUSED(width);

	num = convert_size_unsgnd(num, size);

	if (num == 0)
		buffer[i--] = '0';

	buffer[BUFF_SIZE - 1] = '\0';

	while (num > 0)
	{
		buffer[i--] = (num % 8) + '0';
		num /= 8;
	}

	if (flags & F_HASH && init_num != 0)
		buffer[i--] = '0';

	i++;

	return (write_unsgnd(0, i, buffer, flags, width, precision, size));
}

/************* PRINT UNSIGNED NUMBER IN HEXADECIMAL *************/
/**
 * print_hexadecimal - Prints an unsigned number in hexadecimal notation
 * @types: Lista of arguments
 * @buffer: Buffer array to handle print
```

```
 * @flags:  Calculates active flags
 * @width: get width
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed
 */
int print_hexadecimal(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	return (print_hexa(types, "0123456789abcdef", buffer,
		flags, 'x', width, precision, size));
}

/************* PRINT UNSIGNED NUMBER IN UPPER HEXADECIMAL **************/
/**
 * print_hexa_upper - Prints an unsigned number in upper hexadecimal notation
 * @types: Lista of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed
 */
int print_hexa_upper(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	return (print_hexa(types, "0123456789ABCDEF", buffer,
		flags, 'X', width, precision, size));
}

/************** PRINT HEXX NUM IN LOWER OR UPPER **************/
/**
 * print_hexa - Prints a hexadecimal number in lower or upper
 * @types: Lista of arguments
 * @map_to: Array of values to map the number to
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @flag_ch: Calculates active flags
 * @width: get width
 * @precision: Precision specification
 * @size: Size specifier
 * @size: Size specification
 * Return: Number of chars printed
```

```
 */
int print_hexa(va_list types, char map_to[], char buffer[],
	int flags, char flag_ch, int width, int precision, int size)
{
	int i = BUFF_SIZE - 2;
	unsigned long int num = va_arg(types, unsigned long int);
	unsigned long int init_num = num;

	UNUSED(width);

	num = convert_size_unsgnd(num, size);

	if (num == 0)
		buffer[i--] = '0';

	buffer[BUFF_SIZE - 1] = '\0';

	while (num > 0)
	{
		buffer[i--] = map_to[num % 16];
		num /= 16;
	}

	if (flags & F_HASH && init_num != 0)
	{
		buffer[i--] = flag_ch;
		buffer[i--] = '0';
	}

	i++;

	return (write_unsgnd(0, i, buffer, flags, width, precision, size));
}
```

functions2.c

```
#include "main.h"

/****************** PRINT POINTER ******************/
/**
 * print_pointer - Prints the value of a pointer variable
 * @types: List a of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed.
 */
int print_pointer(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	char extra_c = 0, padd = ' ';
	int ind = BUFF_SIZE - 2, length = 2, padd_start = 1; /* length=2, for '0x' */
	unsigned long num_addrs;
	char map_to[] = "0123456789abcdef";
	void *addrs = va_arg(types, void *);

	UNUSED(width);
	UNUSED(size);

	if (addrs == NULL)
		return (write(1, "(nil)", 5));

	buffer[BUFF_SIZE - 1] = '\0';
	UNUSED(precision);

	num_addrs = (unsigned long)addrs;

	while (num_addrs > 0)
	{
		buffer[ind--] = map_to[num_addrs % 16];
		num_addrs /= 16;
		length++;
	}
```

```
	if ((flags & F_ZERO) && !(flags & F_MINUS))
		padd = '0';
	if (flags & F_PLUS)
		extra_c = '+', length++;
	else if (flags & F_SPACE)
		extra_c = ' ', length++;

	ind++;

	/*return (write(1, &buffer[i], BUFF_SIZE - i - 1));*/
	return (write_pointer(buffer, ind, length,
		width, flags, padd, extra_c, padd_start));
}

/************************* PRINT NON PRINTABLE *************************/
/**
 * print_non_printable - Prints ascii codes in hexa of non printable chars
 * @types: Lista of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Number of chars printed
 */
int print_non_printable(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	int i = 0, offset = 0;
	char *str = va_arg(types, char *);

	UNUSED(flags);
	UNUSED(width);
	UNUSED(precision);
	UNUSED(size);

	if (str == NULL)
		return (write(1, "(null)", 6));

	while (str[i] != '\0')
	{
		if (is_printable(str[i]))
			buffer[i + offset] = str[i];
		else
```

```
                        offset += append_hexa_code(str[i], buffer, i + offset);

                i++;
        }

        buffer[i + offset] = '\0';

        return (write(1, buffer, i + offset));
}

/************************* PRINT REVERSE *************************/
/**
 * print_reverse - Prints reverse string.
 * @types: Lista of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Numbers of chars printed
 */

int print_reverse(va_list types, char buffer[],
        int flags, int width, int precision, int size)
{
        char *str;
        int i, count = 0;

        UNUSED(buffer);
        UNUSED(flags);
        UNUSED(width);
        UNUSED(size);

        str = va_arg(types, char *);

        if (str == NULL)
        {
                UNUSED(precision);

                str = ")Null(";
        }
        for (i = 0; str[i]; i++)
                ;
```

```
	for (i = i - 1; i >= 0; i--)
	{
		char z = str[i];

		write(1, &z, 1);
		count++;
	}
	return (count);
}
/************************ PRINT A STRING IN ROT13 ************************/
/**
 * print_rot13string - Print a string in rot13.
 * @types: Lista of arguments
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width
 * @precision: Precision specification
 * @size: Size specifier
 * Return: Numbers of chars printed
 */
int print_rot13string(va_list types, char buffer[],
	int flags, int width, int precision, int size)
{
	char x;
	char *str;
	unsigned int i, j;
	int count = 0;
	char in[] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
	char out[] =
"NOPQRSTUVWXYZABCDEFGHIJKLMnopqrstuvwxyzabcdefghijklm";

	str = va_arg(types, char *);
	UNUSED(buffer);
	UNUSED(flags);
	UNUSED(width);
	UNUSED(precision);
	UNUSED(size);

	if (str == NULL)
		str = "(AHYY)";
	for (i = 0; str[i]; i++)
	{
		for (j = 0; in[j]; j++)
```

```
                {
                        if (in[j] == str[i])
                        {
                                x = out[j];
                                write(1, &x, 1);
                                count++;
                                break;
                        }
                }
                if (!in[j])
                {
                        x = str[i];
                        write(1, &x, 1);
                        count++;
                }
        }
        return (count);
}
```

get_flags.c

```
#include "main.h"

/**
 * get_flags - Calculates active flags
 * @format: Formatted string in which to print the arguments
 * @i: take a parameter.
 * Return: Flags:
 */
int get_flags(const char *format, int *i)
{
	/* - + 0 # ' ' */
	/* 1 2 4 8  16 */
	int j, curr_i;
	int flags = 0;
	const char FLAGS_CH[] = {'-', '+', '0', '#', ' ', '\0'};
	const int FLAGS_ARR[] = {F_MINUS, F_PLUS, F_ZERO, F_HASH,
F_SPACE, 0};

	for (curr_i = *i + 1; format[curr_i] != '\0'; curr_i++)
	{
		for (j = 0; FLAGS_CH[j] != '\0'; j++)
			if (format[curr_i] == FLAGS_CH[j])
			{
				flags |= FLAGS_ARR[j];
				break;
			}

		if (FLAGS_CH[j] == 0)
			break;
	}

	*i = curr_i - 1;

	return (flags);
}
```

get_precision.c

```
#include "main.h"

/**
 * get_precision - Calculates the precision for printing
 * @format: Formatted string in which to print the arguments
 * @i: List of arguments to be printed.
 * @list: list of arguments.
 *
 * Return: Precision.
 */
int get_precision(const char *format, int *i, va_list list)
{
	int curr_i = *i + 1;
	int precision = -1;

	if (format[curr_i] != '.')
		return (precision);

	precision = 0;

	for (curr_i += 1; format[curr_i] != '\0'; curr_i++)
	{
		if (is_digit(format[curr_i]))
		{
			precision *= 10;
			precision += format[curr_i] - '0';
		}
		else if (format[curr_i] == '*')
		{
			curr_i++;
			precision = va_arg(list, int);
			break;
		}
		else
			break;
	}

	*i = curr_i - 1;

	return (precision);
```

```
}
```