README.md

```
0x11. C - printf team project

Group Project:

0. I'm not going anywhere. You can print that wherever you want to. I'm 
here and I'm
 a Spur for life
Write a function that produces output according to a format.


1. Education is when you read the fine print. Experience is what you 
get if you don'
t
Handle the following conversion specifiers:

2. With a face like mine, I do better in print
Handle the following custom conversion specifiers:

3. What one has not experienced, one will never understand in print
Handle the following conversion specifiers:

4. Nothing in fine print is ever good news
Use a local buffer of 1024 chars in order to call write as little as 
possible.

5. My weakness is wearing too much leopard print
Handle the following custom conversion specifier:

6. How is the world ruled and led to war? Diplomats lie to journalists 
and believe t
hese lies when they see them in print
Handle the following conversion specifier: p.

7. The big print gives and the small print takes away
Handle the following flag characters for non-custom conversion 
specifiers:

8. Sarcasm is lost in print
Handle the following length modifiers for non-custom conversion 
specifiers:

l
h
Conversion specifiers to handle: d, i, u, o, x, X

9. Print some money and give it to us for the rain forests
Handle the field width for non-custom conversion specifiers.
```

10. The negative is the equivalent of the composer's score, and the print the perfor
mance
Handle the precision for non-custom conversion specifiers.

11. It's depressing when you're still around and your albums are out of print
Handle the 0 flag character for non-custom conversion specifiers.

12. Every time that I wanted to give up, if I saw an interesting textile, print what
 ever, suddenly I would see a collection
Handle the - flag character for non-custom conversion specifiers.

13. Print is the sharpest and the strongest weapon of our party
Handle the following custom conversion specifier:

14. The flood of print has turned reading into a process of gulping rather than savo
ring
Handle the following custom conversion specifier:

15. *
All the above options work well together.

==

_printf.c CODE

```
#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 CODE

```
#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 CODE

```
#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 CODE

```
#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 CODE

```
#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);
}
```

===

get_size.c CODE

```
#include "main.h"

/**
 * get_size - Calculates the size to cast the argument
 * @format: Formatted string in which to print the arguments
 * @i: List of arguments to be printed.
 *
 * Return: Precision.
 */
int get_size(const char *format, int *i)
{
	int curr_i = *i + 1;
	int size = 0;

	if (format[curr_i] == 'l')
		size = S_LONG;
	else if (format[curr_i] == 'h')
		size = S_SHORT;

	if (size == 0)
		*i = curr_i - 1;
	else
		*i = curr_i;

	return (size);
}
```

==

get_width.c CODE

```
#include "main.h"

/**
 * get_width - Calculates the width for printing
 * @format: Formatted string in which to print the arguments.
 * @i: List of arguments to be printed.
 * @list: list of arguments.
 *
 * Return: width.
 */
int get_width(const char *format, int *i, va_list list)
{
	int curr_i;
	int width = 0;

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

	*i = curr_i - 1;

	return (width);
}
```

==

handle_print.c CODE

```
#include "main.h"
/**
 * handle_print - Prints an argument based on its type
 * @fmt: Formatted string in which to print the arguments.
 * @list: List of arguments to be printed.
 * @ind: ind.
 * @buffer: Buffer array to handle print.
 * @flags: Calculates active flags
 * @width: get width.
 * @precision: Precision specification
 * @size: Size specifier
 * Return: 1 or 2;
 */
int handle_print(const char *fmt, int *ind, va_list list, char
buffer[],
	int flags, int width, int precision, int size)
{
	int i, unknow_len = 0, printed_chars = -1;
	fmt_t fmt_types[] = {
		{'c', print_char}, {'s', print_string}, {'%',
print_percent},
		{'i', print_int}, {'d', print_int}, {'b', print_binary},
		{'u', print_unsigned}, {'o', print_octal}, {'x',
print_hexadecimal},
		{'X', print_hexa_upper}, {'p', print_pointer}, {'S',
print_non_printable},
		{'r', print_reverse}, {'R', print_rot13string}, {'\0', NULL}
	};
	for (i = 0; fmt_types[i].fmt != '\0'; i++)
		if (fmt[*ind] == fmt_types[i].fmt)
			return (fmt_types[i].fn(list, buffer, flags, width,
precision, size));

	if (fmt_types[i].fmt == '\0')
	{
		if (fmt[*ind] == '\0')
			return (-1);
		unknow_len += write(1, "%%", 1);
		if (fmt[*ind - 1] == ' ')
			unknow_len += write(1, " ", 1);
		else if (width)
		{
			--(*ind);
			while (fmt[*ind] != ' ' && fmt[*ind] != '%')
				--(*ind);
			if (fmt[*ind] == ' ')
				--(*ind);
			return (1);
		}
		unknow_len += write(1, &fmt[*ind], 1);
```

```
			return (unknow_len);
		}
		return (printed_chars);
}
```

===

main.h CODE

```
#ifndef MAIN_H
#define MAIN_H
#include <stdarg.h>
#include <stdio.h>
#include <unistd.h>

#define UNUSED(x) (void)(x)
#define BUFF_SIZE 1024

/* FLAGS */
#define F_MINUS 1
#define F_PLUS 2
#define F_ZERO 4
#define F_HASH 8
#define F_SPACE 16

/* SIZES */
#define S_LONG 2
#define S_SHORT 1

/**
 * struct fmt - Struct op
 *
 * @fmt: The format.
 * @fn: The function associated.
 */
struct fmt
{
	char fmt;
	int (*fn)(va_list, char[], int, int, int, int);
};


/**
```

```
 * typedef struct fmt fmt_t - Struct op
 *
 * @fmt: The format.
 * @fm_t: The function associated.
 */
typedef struct fmt fmt_t;

int _printf(const char *format, ...);
int handle_print(const char *fmt, int *i,
va_list list, char buffer[], int flags, int width, int precision, int
size);

/****************** FUNCTIONS ******************/

/* Funtions to print chars and strings */
int print_char(va_list types, char buffer[],
      int flags, int width, int precision, int size);
int print_string(va_list types, char buffer[],
      int flags, int width, int precision, int size);
int print_percent(va_list types, char buffer[],
      int flags, int width, int precision, int size);

/* Functions to print numbers */
int print_int(va_list types, char buffer[],
      int flags, int width, int precision, int size);
int print_binary(va_list types, char buffer[],
      int flags, int width, int precision, int size);
int print_unsigned(va_list types, char buffer[],
      int flags, int width, int precision, int size);
int print_octal(va_list types, char buffer[],
      int flags, int width, int precision, int size);
int print_hexadecimal(va_list types, char buffer[],
      int flags, int width, int precision, int size);
int print_hexa_upper(va_list types, char buffer[],
      int flags, int width, int precision, int size);

int print_hexa(va_list types, char map_to[],
char buffer[], int flags, char flag_ch, int width, int precision, int
size);

/* Function to print non printable characters */
int print_non_printable(va_list types, char buffer[],
      int flags, int width, int precision, int size);

/* Funcion to print memory address */
int print_pointer(va_list types, char buffer[],
      int flags, int width, int precision, int size);

/* Funciotns to handle other specifiers */
int get_flags(const char *format, int *i);
int get_width(const char *format, int *i, va_list list);
int get_precision(const char *format, int *i, va_list list);
int get_size(const char *format, int *i);

/*Function to print string in reverse*/
int print_reverse(va_list types, char buffer[],
      int flags, int width, int precision, int size);

/*Function to print a string in rot 13*/
int print_rot13string(va_list types, char buffer[],
      int flags, int width, int precision, int size);

/* width handler */
int handle_write_char(char c, char buffer[],
      int flags, int width, int precision, int size);
int write_number(int is_positive, int ind, char buffer[],
      int flags, int width, int precision, int size);
int write_num(int ind, char bff[], int flags, int width, int precision,
      int length, char padd, char extra_c);
int write_pointer(char buffer[], int ind, int length,
      int width, int flags, char padd, char extra_c, int padd_start);

int write_unsgnd(int is_negative, int ind,
char buffer[],
      int flags, int width, int precision, int size);

/****************** UTILS ******************/
int is_printable(char);
int append_hexa_code(char, char[], int);
int is_digit(char);

long int convert_size_number(long int num, int size);
long int convert_size_unsgnd(unsigned long int num, int size);

#endif /* MAIN_H */
```

===

utils.c CODE

```
#include "main.h"

/**
 * is_printable - Evaluates if a char is printable
 * @c: Char to be evaluated.
 *
 * Return: 1 if c is printable, 0 otherwise
 */
int is_printable(char c)
{
	if (c >= 32 && c < 127)
		return (1);

	return (0);
}

/**
 * append_hexa_code - Append ascci in hexadecimal code to buffer
 * @buffer: Array of chars.
 * @i: Index at which to start appending.
 * @ascii_code: ASSCI CODE.
 * Return: Always 3
 */
int append_hexa_code(char ascii_code, char buffer[], int i)
{
	char map_to[] = "0123456789ABCDEF";
	/* The hexa format code is always 2 digits long */
	if (ascii_code < 0)
		ascii_code *= -1;

	buffer[i++] = '\\';
	buffer[i++] = 'x';

	buffer[i++] = map_to[ascii_code / 16];
	buffer[i] = map_to[ascii_code % 16];

	return (3);
}

/**
 * is_digit - Verifies if a char is a digit
 * @c: Char to be evaluated
 *
 * Return: 1 if c is a digit, 0 otherwise
 */
int is_digit(char c)
{
	if (c >= '0' && c <= '9')
		return (1);

	return (0);
}

/**
 * convert_size_number - Casts a number to the specified size
 * @num: Number to be casted.
 * @size: Number indicating the type to be casted.
 *
 * Return: Casted value of num
 */
long int convert_size_number(long int num, int size)
{
	if (size == S_LONG)
		return (num);
	else if (size == S_SHORT)
		return ((short)num);

	return ((int)num);
}

/**
 * convert_size_unsgnd - Casts a number to the specified size
 * @num: Number to be casted
 * @size: Number indicating the type to be casted
 *
 * Return: Casted value of num
 */
long int convert_size_unsgnd(unsigned long int num, int size)
{
	if (size == S_LONG)
		return (num);
	else if (size == S_SHORT)
		return ((unsigned short)num);

	return ((unsigned int)num);
}
```

===

write_handlers.c

```
#include "main.h"

/************************* WRITE HANDLE *************************/
/**
 * handle_write_char - Prints a string
 * @c: char types.
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags.
 * @width: get width.
 * @precision: precision specifier
 * @size: Size specifier
 *
 * Return: Number of chars printed.
 */
int handle_write_char(char c, char buffer[],
      int flags, int width, int precision, int size)
{ /* char is stored at left and paddind at buffer's right */
      int i = 0;
      char padd = ' ';

      UNUSED(precision);
      UNUSED(size);

      if (flags & F_ZERO)
            padd = '0';

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

      if (width > 1)
      {
            buffer[BUFF_SIZE - 1] = '\0';
            for (i = 0; i < width - 1; i++)
                  buffer[BUFF_SIZE - i - 2] = padd;

            if (flags & F_MINUS)
                  return (write(1, &buffer[0], 1) +
                              write(1, &buffer[BUFF_SIZE - i - 1], width
- 1));
            else
                  return (write(1, &buffer[BUFF_SIZE - i - 1], width -
1) +
                              write(1, &buffer[0], 1));
      }

      return (write(1, &buffer[0], 1));
}

/************************* WRITE NUMBER *************************/
/**
 * write_number - Prints a string
 * @is_negative: Lista of arguments
 * @ind: char types.
 * @buffer: Buffer array to handle print
 * @flags:  Calculates active flags
 * @width: get width.
 * @precision: precision specifier
 * @size: Size specifier
 *
 * Return: Number of chars printed.
 */
int write_number(int is_negative, int ind, char buffer[],
```

```c
      int flags, int width, int precision, int size)
{
      int length = BUFF_SIZE - ind - 1;
      char padd = ' ', extra_ch = 0;

      UNUSED(size);

      if ((flags & F_ZERO) && !(flags & F_MINUS))
            padd = '0';
      if (is_negative)
            extra_ch = '-';
      else if (flags & F_PLUS)
            extra_ch = '+';
      else if (flags & F_SPACE)
            extra_ch = ' ';

      return (write_num(ind, buffer, flags, width, precision,
            length, padd, extra_ch));
}

/**
 * write_num - Write a number using a bufffer
 * @ind: Index at which the number starts on the buffer
 * @buffer: Buffer
 * @flags: Flags
 * @width: width
 * @prec: Precision specifier
 * @length: Number length
 * @padd: Pading char
 * @extra_c: Extra char
 *
 * Return: Number of printed chars.
 */
int write_num(int ind, char buffer[],
      int flags, int width, int prec,
      int length, char padd, char extra_c)
{
      int i, padd_start = 1;

      if (prec == 0 && ind == BUFF_SIZE - 2 && buffer[ind] == '0' &&
width == 0)
            return (0); /* printf(".0d", 0)  no char is printed */
      if (prec == 0 && ind == BUFF_SIZE - 2 && buffer[ind] == '0')
            buffer[ind] = padd = ' '; /* width is displayed with padding
' ' */
      if (prec > 0 && prec < length)
            padd = ' ';
      while (prec > length)
            buffer[--ind] = '0', length++;
      if (extra_c != 0)
            length++;
	if (width > length)
	{
		for (i = 1; i < width - length + 1; i++)
			buffer[i] = padd;
		buffer[i] = '\0';
		if (flags & F_MINUS && padd == ' ')/* Asign extra char to
left of buffer */
		{
			if (extra_c)
				buffer[--ind] = extra_c;
			return (write(1, &buffer[ind], length) + write(1,
&buffer[1], i - 1));
		}
		else if (!(flags & F_MINUS) && padd == ' ')/* extra char to
left of buff */
		{
			if (extra_c)
				buffer[--ind] = extra_c;
			return (write(1, &buffer[1], i - 1) + write(1,
&buffer[ind], length));
		}
		else if (!(flags & F_MINUS) && padd == '0')/* extra char to
left of padd */
		{
			if (extra_c)
				buffer[--padd_start] = extra_c;
			return (write(1, &buffer[padd_start], i - padd_start)
+
				write(1, &buffer[ind], length - (1 -
padd_start)));
		}
	}
	if (extra_c)
		buffer[--ind] = extra_c;
	return (write(1, &buffer[ind], length));
}

/**
 * write_unsgnd - Writes an unsigned number
 * @is_negative: Number indicating if the num is negative
 * @ind: Index at which the number starts in the buffer
 * @buffer: Array of chars
 * @flags: Flags specifiers
 * @width: Width specifier
 * @precision: Precision specifier
 * @size: Size specifier
 *
 * Return: Number of written chars.
 */
int write_unsgnd(int is_negative, int ind,
	char buffer[],
	int flags, int width, int precision, int size)
{
	/* The number is stored at the bufer's right and starts at
position i */
	int length = BUFF_SIZE - ind - 1, i = 0;
	char padd = ' ';

	UNUSED(is_negative);
	UNUSED(size);

	if (precision == 0 && ind == BUFF_SIZE - 2 && buffer[ind] == '0')
		return (0); /* printf(".0d", 0)  no char is printed */

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

	while (precision > length)
	{
		buffer[--ind] = '0';
		length++;
	}

	if ((flags & F_ZERO) && !(flags & F_MINUS))
		padd = '0';

	if (width > length)
	{
		for (i = 0; i < width - length; i++)
			buffer[i] = padd;

		buffer[i] = '\0';

		if (flags & F_MINUS) /* Asign extra char to left of buffer
[buffer>padd]*/
		{
			return (write(1, &buffer[ind], length) + write(1,
&buffer[0], i));
		}
		else /* Asign extra char to left of padding [padd>buffer]*/
		{
			return (write(1, &buffer[0], i) + write(1,
&buffer[ind], length));
		}
	}

	return (write(1, &buffer[ind], length));
}

/**
 * write_pointer - Write a memory address
 * @buffer: Arrays of chars
 * @ind: Index at which the number starts in the buffer
 * @length: Length of number
 * @width: Wwidth specifier
 * @flags: Flags specifier
 * @padd: Char representing the padding
 * @extra_c: Char representing extra char
 * @padd_start: Index at which padding should start
 *
 * Return: Number of written chars.
 */
int write_pointer(char buffer[], int ind, int length,
      int width, int flags, char padd, char extra_c, int padd_start)
{
      int i;

      if (width > length)
      {
            for (i = 3; i < width - length + 3; i++)
                  buffer[i] = padd;
            buffer[i] = '\0';
            if (flags & F_MINUS && padd == ' ')/* Asign extra char to
left of buffer */
            {
                  buffer[--ind] = 'x';
                  buffer[--ind] = '0';
                  if (extra_c)
                        buffer[--ind] = extra_c;
                  return (write(1, &buffer[ind], length) + write(1,
&buffer[3], i - 3));
            }
            else if (!(flags & F_MINUS) && padd == ' ')/* extra char to
left of buffer */
            {
                  buffer[--ind] = 'x';
                  buffer[--ind] = '0';
                  if (extra_c)
                        buffer[--ind] = extra_c;
                  return (write(1, &buffer[3], i - 3) + write(1,
&buffer[ind], length));
            }
            else if (!(flags & F_MINUS) && padd == '0')/* extra char to
left of padd */
            {
                  if (extra_c)
                        buffer[--padd_start] = extra_c;
                  buffer[1] = '0';
                  buffer[2] = 'x';
                  return (write(1, &buffer[padd_start], i - padd_start)
+
                        write(1, &buffer[ind], length - (1 - padd_start)
- 2));
```

```
			}
		}
		buffer[--ind] = 'x';
		buffer[--ind] = '0';
		if (extra_c)
			buffer[--ind] = extra_c;
		return (write(1, &buffer[ind], BUFF_SIZE - ind - 1));
}
```

```
==============================================
```