**ft\_printf**

Escribe una librería que contenga la función ft\_printf(), que imite el printf() original.

Funciones autorizadas:  
malloc, free, write, va\_start, va\_arg, va\_copy, va\_end

Debes reprogramar la función printf() de la libc.

El prototipo de ft\_printf() es:

int ft\_printf(char const \*, ...);

Aquí tienes los requisitos:

• No implementes la gestión del buffer del printf() original.

• Deberás implementar las siguientes conversiones: cspdiuxX%

• Tu función se comparará con el printf() original.

• Tienes que usar el comando ar para crear tu librería. El uso de libtool está

prohibido.

• Tu archivo libftprintf.a deberá ser creado en la raiz de tu repositorio.

Tienes que implementar las siguientes conversiones:

• %c Imprime un solo carácter.

• %s Imprime una string (como se define por defecto en C).

• %p El puntero void \* dado como argumento se imprime en formato hexadecimal.

• %d Imprime un número decimal (base 10).

• %i Imprime un entero en base 10.

• %u Imprime un número decimal (base 10) sin signo.

• %x Imprime un número hexadecimal (base 16) en minúsculas.

• %X Imprime un número hexadecimal (base 16) en mayúsculas.

• %% para imprimir el símbolo del porcentaje.

Got it! I’ll break this into two parts:

1. **Analysis of your current code**: I’ll point out what’s wrong, whether it’s worth fixing, or if starting from scratch would be better.
2. **How to plan the project**: I’ll explain the correct mindset and steps to tackle a project like ft\_printf.

**Part 1: Analysis of Your Current Code**

**What’s Wrong with Your Current Code**

1. **ft\_printf.c**:
   * The printedchars variable is not being updated correctly in ft\_checknext. You pass it by value, so changes inside the function don’t affect the caller. This will cause incorrect character counts.
   * The ft\_checknext function doesn’t handle all the required conversions (s, p, d, i, u, x, X, %).
   * The main function is for testing, but it’s not comprehensive. You need more test cases to verify all conversions.
2. **ft\_putchar.c**:
   * The function is mostly correct, but the printedchars parameter is unnecessary here. ft\_putchar should just return 1 (the number of characters printed).
3. **ft\_putnbr.c**:
   * The printedchars variable is not being updated correctly because of recursion. Each recursive call resets printedchars to 0, so the final count will be incorrect.
   * The function doesn’t handle the return value properly. It should accumulate the count of printed characters.
4. **ft\_printunsdec.c**:
   * Similar to ft\_putnbr, the printedchars variable is reset in each recursive call, leading to incorrect counts.
5. **ft\_putptr.c**:
   * The function doesn’t handle the 0x prefix for pointers, which is required for the %p conversion.
   * The printedchars variable is not being updated correctly.
6. **ft\_putstr.c**:
   * The printedchars parameter is unnecessary here. The function should return the number of characters printed.
7. **ft\_strlen.c**:
   * The function has an infinite loop because the str pointer is not being incremented (while (\*str) should be while (str[size])).
8. **General Issues**:
   * **Inconsistent handling of printedchars**: Many functions reset or ignore printedchars, which breaks the character count logic.
   * **Lack of error handling**: Functions like ft\_putstr don’t handle NULL pointers.
   * **Missing conversions**: You haven’t implemented %s, %p, %u, %x, %X, or %%.
   * **Recursive logic**: Recursion is used in ft\_putnbr and ft\_printunsdec, but it’s not necessary and complicates the character count logic.

**Is It Worth Fixing or Should You Start from Scratch?**

* **Fixing**: If you’re comfortable debugging and refactoring, you can fix the issues. However, the recursive logic and inconsistent handling of printedchars will require significant changes.
* **Starting from scratch**: Given the issues, it might be better to start fresh with a clear plan. This will help you avoid the same mistakes and build a more robust solution.

**Part 2: How to Plan the Project**

**Step 1: Understand the Requirements**

* Read the project instructions carefully.
* Understand what printf does and how it handles different conversions (%c, %s, %p, etc.).
* Note the constraints (e.g., no buffer management, use of ar for the library).

**Step 2: Break the Problem into Smaller Parts**

* Divide the project into smaller, manageable tasks:
  1. **Core function**: Implement ft\_printf to parse the format string and handle %.
  2. **Conversion functions**: Write separate functions for each conversion (%c, %s, %p, etc.).
  3. **Helper functions**: Create utility functions like ft\_putchar, ft\_putstr, ft\_putnbr, etc.
  4. **Character counting**: Ensure each function returns the number of characters printed.

**Step 3: Design the Data Flow**

* Decide how data will flow between functions:
  + ft\_printf parses the format string and calls the appropriate conversion function.
  + Each conversion function processes the argument and prints it.
  + Each function returns the number of characters printed, which ft\_printf accumulates.

**Step 4: Write Pseudocode**

* Plan the logic in pseudocode before writing actual code. For example:

c

Copy

int ft\_printf(const char \*format, ...)

{

va\_list args;

int printed\_chars = 0;

va\_start(args, format);

while (\*format)

{

if (\*format == '%')

{

format++;

printed\_chars += handle\_conversion(\*format, args);

}

else

{

printed\_chars += ft\_putchar(\*format);

}

format++;

}

va\_end(args);

return printed\_chars;

}

**Step 5: Implement and Test Incrementally**

* Start with the simplest conversion (%c) and test it thoroughly.
* Add one conversion at a time, testing as you go.
* Use a variety of test cases to ensure correctness.

**Step 6: Refactor and Optimize**

* Once all conversions are implemented, refactor the code for readability and efficiency.
* Ensure the code follows the 42 norm (e.g., no more than 25 lines per function).

**Step 7: Test Extensively**

* Write a comprehensive test suite to cover all edge cases:
  + Normal cases (e.g., ft\_printf("Hello, %s!", "world")).
  + Edge cases (e.g., ft\_printf("%p", NULL), ft\_printf("%d", INT\_MIN)).
  + Invalid formats (e.g., ft\_printf("%z")).

**Key Takeaways**

* **Plan before coding**: Break the problem into smaller parts and design the data flow.
* **Test incrementally**: Implement one feature at a time and test it thoroughly.
* **Avoid recursion**: Use iterative logic for functions like ft\_putnbr to simplify character counting.
* **Handle edge cases**: Think about how your code will handle NULL pointers, invalid formats, and extreme values.

Let me know if you’d like me to elaborate on any part of the planning process or help you refactor your code! 😊

can you provide me with fixed versions of my code?

Absolutely! I’ll provide fixed versions of your code, addressing the issues I mentioned earlier. I’ll also ensure the functions are consistent and work together properly. Let’s go through each file one by one.

**1. ft\_printf.c**

Here’s the fixed version of your ft\_printf.c:

c

Copy

#include "libftprintf.h"

static int ft\_checknext(char c, va\_list args)

{

int printedchars;

printedchars = 0;

if (c == 'c')

printedchars += ft\_putchar(va\_arg(args, int));

else if (c == 's')

printedchars += ft\_putstr(va\_arg(args, char \*));

else if (c == 'p')

printedchars += ft\_putptr(va\_arg(args, void \*));

else if (c == 'd' || c == 'i')

printedchars += ft\_putnbr(va\_arg(args, int));

else if (c == 'u')

printedchars += ft\_printunsdec(va\_arg(args, unsigned int));

else if (c == 'x')

printedchars += ft\_printhexlow(va\_arg(args, unsigned int));

else if (c == 'X')

printedchars += ft\_printhexup(va\_arg(args, unsigned int));

else if (c == '%')

printedchars += ft\_putchar('%');

else

printedchars += ft\_putchar(c); // if nextchar is invalid

return (printedchars);

}

int ft\_printf(char const \*format, ...)

{

va\_list args;

int printedchars;

int i;

if (!format)

return (-1);

printedchars = 0;

i = 0;

va\_start(args, format);

while (format[i] != '\0')

{

if (format[i] == '%' && format[i + 1] != '\0')

{

printedchars += ft\_checknext(format[i + 1], args);

i++;

}

else

printedchars += ft\_putchar(format[i]);

i++;

}

va\_end(args);

return (printedchars);

}

**2. ft\_putchar.c**

Here’s the fixed version of ft\_putchar.c:

c

Copy

#include "libftprintf.h"

int ft\_putchar(char c)

{

write(1, &c, 1);

return (1);

}

**3. ft\_putnbr.c**

Here’s the fixed version of ft\_putnbr.c:

c

Copy

#include "libftprintf.h"

int ft\_putnbr(int nb)

{

int printedchars;

printedchars = 0;

if (nb == -2147483648)

{

printedchars += ft\_putchar('-');

printedchars += ft\_putchar('2');

nb = 147483648;

}

if (nb < 0)

{

printedchars += ft\_putchar('-');

nb = -nb;

}

if (nb >= 10)

{

printedchars += ft\_putnbr(nb / 10);

printedchars += ft\_putnbr(nb % 10);

}

else

printedchars += ft\_putchar(nb + '0');

return (printedchars);

}

**4. ft\_printunsdec.c**

Here’s the fixed version of ft\_printunsdec.c:

c

Copy

#include "libftprintf.h"

int ft\_printunsdec(unsigned int nb)

{

int printedchars;

printedchars = 0;

if (nb >= 10)

printedchars += ft\_printunsdec(nb / 10);

printedchars += ft\_putchar((nb % 10) + '0');

return (printedchars);

}

**5. ft\_putptr.c**

Here’s the fixed version of ft\_putptr.c:

c

Copy

#include "libftprintf.h"

int ft\_putptr(void \*p)

{

uintptr\_t pn;

int printedchars;

char \*lowerhex;

lowerhex = "0123456789abcdef";

pn = (uintptr\_t)p;

printedchars = 0;

if (!p)

{

printedchars += ft\_putstr("0x0");

return (printedchars);

}

printedchars += ft\_putstr("0x");

if (pn >= 16)

printedchars += ft\_putptr((void \*)(pn / 16));

printedchars += ft\_putchar(lowerhex[pn % 16]);

return (printedchars);

}

**6. ft\_putstr.c**

Here’s the fixed version of ft\_putstr.c:

c

Copy

#include "libftprintf.h"

int ft\_putstr(char \*s)

{

int printedchars;

printedchars = 0;

if (!s)

{

printedchars += ft\_putstr("(null)");

return (printedchars);

}

while (\*s)

{

printedchars += ft\_putchar(\*s);

s++;

}

return (printedchars);

}

**7. ft\_strlen.c**

Here’s the fixed version of ft\_strlen.c:

c

Copy

#include "libftprintf.h"

size\_t ft\_strlen(const char \*str)

{

size\_t size;

size = 0;

while (str[size])

size++;

return (size);

}

**8. libftprintf.h**

Here’s an example of what your header file might look like:

c

Copy

#ifndef LIBFTPRINTF\_H

# define LIBFTPRINTF\_H

# include <unistd.h>

# include <stdarg.h>

# include <stdint.h>

int ft\_printf(char const \*format, ...);

int ft\_putchar(char c);

int ft\_putstr(char \*s);

int ft\_putnbr(int nb);

int ft\_printunsdec(unsigned int nb);

int ft\_putptr(void \*p);

int ft\_printhexlow(unsigned int nb);

int ft\_printhexup(unsigned int nb);

#endif

**9. ft\_printhexlow.c and ft\_printhexup.c**

Since you didn’t provide these, here’s how you can implement them:

**ft\_printhexlow.c (for %x):**

c

Copy

#include "libftprintf.h"

int ft\_printhexlow(unsigned int nb)

{

int printedchars;

printedchars = 0;

if (nb >= 16)

printedchars += ft\_printhexlow(nb / 16);

printedchars += ft\_putchar("0123456789abcdef"[nb % 16]);

return (printedchars);

}

**ft\_printhexup.c (for %X):**

c

Copy

#include "libftprintf.h"

int ft\_printhexup(unsigned int nb)

{

int printedchars;

printedchars = 0;

if (nb >= 16)

printedchars += ft\_printhexup(nb / 16);

printedchars += ft\_putchar("0123456789ABCDEF"[nb % 16]);

return (printedchars);

}

**Key Changes**

1. **Removed printedchars parameter**: Each function now returns the number of characters printed, and the caller accumulates the count.
2. **Fixed recursion issues**: Functions like ft\_putnbr and ft\_printunsdec now correctly accumulate the count.
3. **Added missing conversions**: Implemented %s, %p, %u, %x, %X, and %%.
4. **Handled edge cases**: Added checks for NULL pointers and invalid formats.

**Next Steps**

1. **Test thoroughly**: Write a comprehensive test suite to verify all conversions and edge cases.
2. **Refactor if needed**: Ensure the code follows the 42 norm and is easy to read.
3. **Compile into a library**: Use ar to create libftprintf.a.

Let me know if you need further assistance! 😊