FT_PRINTF

ft_printf.pdf

1. Función `ft_printf` (archivo: `ft_printf.c`)

Esta es la función principal que reemplaza el comportamiento del `printf` estándar de C.

Flujo de trabajo:

1. **Parámetros**: Recibe como parámetro una cadena de formato (`format`) y una lista variable de argumentos (`...`).
2. **Inicialización de `va_list`**: Se crea un objeto `va_list` llamado `args` con `va_start(args, format);`, que permite acceder a los argumentos variables.
3. **Recorrido del formato**:
   - Se recorre el string `format` carácter por carácter con un ciclo `while`.
   - Si encuentra el carácter `%`, llama a `handle_format` para procesar el especificador de formato que sigue al `%`.
   - Si no es un `%`, simplemente imprime el carácter usando `ft_print_char`.
4. **Fin del ciclo**: Una vez que ha terminado de recorrer `format`, la función retorna el número total de caracteres impresos.

Pasos:

1. **Inicialización de variables:**

```
va_list args;
int num_print = 0;
int i = 0;
va_start(args, format);
```

- `va_list args`: Un objeto para acceder a los argumentos variables.
- `num_print`: Lleva la cuenta de los caracteres impresos.
- `i`: Variable de control para recorrer el `format`.
- `va_start`: Inicializa `args` para comenzar a procesar los argumentos.

2. **Recorrer el formato:**

```
while (format[i]) {
    if (format[i] == '%') {
        i++;
        handle_format(format[i], args, &num_print);
    } else {
        num_print = ft_print_char(format[i], num_print);
    }
    i++;
}
```

- Se recorre la cadena `format` carácter por carácter.
- Cuando se encuentra un `%`, se llama a `handle_format` para procesar el especificador que sigue (`c`, `d`, `s`, etc.).
- Si no es un `%`, imprime el carácter actual usando `ft_print_char`.

3. **Finalización:**

```
va_end(args);
return (num_print);
```

- `va_end(args)` libera la memoria utilizada por `args`.
- Retorna la cantidad total de caracteres impresos.

---

2. Función `handle_format` (archivo: `ft_printf.c`)

Esta función decide qué función de impresión específica llamar dependiendo del especificador de formato que sigue al `%`.

Especificadores:

- `%c`: Llama a `ft_print_char` para imprimir un carácter.

- `%s`: Llama a `ft_print_string` para imprimir una cadena de caracteres.
- `%p`: Llama a `ft_print_pointer` para imprimir una dirección de memoria.
- `%d` / `%i`: Llama a `ft_print_decimal` para imprimir enteros decimales.
- `%u`: Llama a `ft_print_unsigned` para imprimir enteros sin signo.
- `%x` / `%X`: Llama a `ft_print_hex` para imprimir enteros en formato hexadecimal (en minúsculas o mayúsculas respectivamente).
- `%%`: Imprime el carácter `%` usando `ft_print_char`.

Pasos:

1. **Condiciones para los especificadores:**

```
if (specifier == 'c')
    *num_print = ft_print_char(va_arg(args, int), *num_p
rint);
else if (specifier == 's')
    *num_print = ft_print_string(va_arg(args, char *), *
num_print);
else if (specifier == 'p')
    *num_print = ft_print_pointer(va_arg(args, void *),
*num_print);
else if (specifier == 'd' || specifier == 'i')
    *num_print = ft_print_decimal(va_arg(args, int), *nu
m_print);
else if (specifier == 'u')
    *num_print = ft_print_unsigned(va_arg(args, unsigned
int), *num_print);
else if (specifier == 'x')
    *num_print = ft_print_hex(va_arg(args, unsigned in
t), *num_print, 0);
else if (specifier == 'X')
    *num_print = ft_print_hex(va_arg(args, unsigned in
t), *num_print, 1);
else if (specifier == '%')
    *num_print = ft_print_char('%', *num_print);
```

- Dependiendo del valor de `specifier` (el carácter que sigue al `%`), se llama a la función correspondiente:
  - `%c`: Para caracteres.
  - `%s`: Para cadenas.
  - `%p`: Para punteros.
  - `%d` / `%i`: Para enteros decimales.
  - `%u`: Para enteros sin signo.
  - `%x` / `%X`: Para hexadecimales en minúsculas o mayúsculas.
  - `%%`: Para imprimir el carácter `%`.

2. **Uso de `va_arg`:**
   - `va_arg(args, tipo)` extrae el siguiente argumento del tipo específico (por ejemplo, `int`, `char *`, etc.) de la lista de argumentos `args`.
   - La función de impresión se llama con ese valor y el contador de caracteres `num_print`.

---

3. Función `ft_print_char` (archivo: `ft_print_char.c`)

Qué hace:

Imprime un solo carácter en pantalla utilizando la función `write` de Unix y retorna el número total de caracteres impresos hasta ahora, incrementando en uno.

```
int ft_print_char(char c, int num_print) {
    write(1, &c, 1);
    return (num_print + 1);
```

Pasos:

1. **Imprimir carácter:**

   ```
   write(1, &c, 1);
   ```

   - Utiliza la función `write` para escribir el carácter `c` en la salida estándar (`1` representa la salida estándar en Unix).

2. **Contar caracteres impresos:**

```
return (num_print + 1);
```

   - Incrementa el contador `num_print` en 1 (ya que se imprimió un carácter) y retorna el valor actualizado.

---

4. Función `ft_print_string` (archivo: `ft_print_string.c`)

Qué hace:

Imprime una cadena de caracteres (`char *`) en pantalla. Utiliza la función `ft_print_char` para imprimir cada carácter de la cadena uno por uno.

- Recorre la cadena con un ciclo `while (*str)`, imprimiendo carácter a carácter hasta llegar al final (cuando encuentra `\0`).

Pasos:

1. **Recorrer la cadena:**

```
while (*str)
{
    num_print = ft_print_char(*str, num_print);
    str++;
}
```

   - Se recorre la cadena `str` carácter por carácter hasta llegar al final (`\0`).
   - Cada carácter de la cadena se imprime con `ft_print_char` y se actualiza el contador `num_print`.

2. **Retornar contador:**

   - Devuelve el número total de caracteres impresos.

---

5. Función `ft_print_decimal` (archivo: `ft_print_decimal.c`)

Qué hace:

Imprime un número entero en formato decimal (`%d` o `%i`). Si el número es negativo, imprime primero el signo `-` y luego convierte el número a positivo.

Detalles:

- Si el número es el valor mínimo (`2147483648`), se maneja explícitamente porque no se puede convertir directamente a positivo.
- Para números mayores a 9, se descompone el número en dígitos usando recursión. Primero imprime los dígitos más significativos dividiendo por 10, y luego imprime el dígito final (el resto de la división).

Pasos:

1. **Caso especial para el mínimo valor de `int`:**

```
if (num == -2147483648)
    return (ft_print_string("-2147483648", num_print));
```

   - El valor mínimo de un entero de 32 bits no se puede convertir directamente a positivo, por lo que se maneja de forma especial imprimiendo directamente la cadena `"-2147483648"`.

2. **Manejo de números negativos:**

```
if (num < 0) {
    num_print = ft_print_char('-', num_print);
    num = -num;
}
```

   - Si el número es negativo, imprime el signo `-` y convierte el número a positivo.

3. **Impresión recursiva:**

```
if (num >= 10)
    num_print = ft_print_decimal(num / 10, num_print);
num_print = ft_print_char((num % 10) + '0', num_print);
```

   - Si el número es mayor o igual a 10, se divide por 10 y se llama recursivamente para imprimir los dígitos más significativos primero.
   - Luego, se imprime el dígito más bajo (el resto de la división).

---

6. Función `ft_print_unsigned` (archivo: `ft_print_unsigned.c`)

Qué hace:

Imprime un número entero sin signo (`%u`). Similar a `ft_print_decimal`, pero no maneja números negativos. Utiliza recursión para imprimir el número descomponiéndolo en dígitos.

Pasos:

1. **Impresión recursiva:**

```
if (num >= 10)
    num_print = ft_print_unsigned(num / 10, num_print);
num_print = ft_print_char((num % 10) + '0', num_print);
```

   - Si el número es mayor o igual a 10, se divide por 10 y se llama recursivamente para imprimir los dígitos.
   - Luego imprime el dígito más bajo usando `ft_print_char`.

---

7. Función `ft_print_hex` (archivo: `ft_print_hex.c`)

Qué hace:

Imprime un número en formato hexadecimal (`%x` o `%X`). Para hacerlo:

- Usa una función auxiliar `ft_puthex` para descomponer el número en dígitos hexadecimales y luego imprimirlos uno por uno.
- Si el número es mayor o igual a 16, lo divide y llama recursivamente para imprimir los dígitos en orden.

Pasos:

1. **Función auxiliar `ft_puthex`:**

```
if (num >= 16)
    ft_puthex(num / 16, uppercase, num_print);
*num_print = ft_print_char(hex[num % 16], *num_print);
```

   - Si el número es mayor o igual a 16, se divide por 16 y se llama recursivamente para imprimir los dígitos más significativos primero.
   - Luego imprime el dígito más bajo correspondiente, basado en la tabla `hex` (en mayúsculas o minúsculas dependiendo del formato).

2. **Selección de mayúsculas o minúsculas:**

```
if (uppercase)
    hex = "0123456789ABCDEF";
else
    hex = "0123456789abcdef";
```

---

8. Función `ft_print_pointer` (archivo: `ft_print_pointer.c`)

Qué hace:

Imprime una dirección de memoria (`%p`), que es un puntero. Para esto:

- Convierte el valor del puntero a una representación hexadecimal, precedida por `0x` (el prefijo estándar de direcciones de memoria en hexadecimal).
- Usa una función auxiliar `ft_puthex` para convertir el puntero a formato hexadecimal.

Pasos:

1. **Prefijo `0x`:**

   ```
   num_print = ft_print_string("0x", num_print);
   ```

2. **Conversión a hexadecimal:**
   - Se convierte la dirección de memoria (puntero) a un número hexadecimal usando la función auxiliar `ft_puthex`.
3. **Impresión del valor hexadecimal del puntero:**
   - El valor del puntero se imprime recursivamente en formato hexadecimal.

Conclusión del flujo general:

1. El programa comienza llamando a `ft_printf`.
2. `ft_printf` recorre el formato e imprime directamente caracteres regulares o usa `handle_format` cuando encuentra un `%`.
3. Dependiendo del especificador, se llama a la función correspondiente para imprimir el valor (carácter, cadena, número, etc.).

4. Al final, `ft_printf` devuelve el número total de caracteres impresos, igual que la función `printf` estándar.