La Lógica para Determinar el `char_type` de un Símbolo

El `char_type` (el carácter que ves en la segunda columna de la salida de `nm`, como `T`, `U`, `D`, `B`, etc.) no se lee directamente del archivo ELF. Se **deriva** de una combinación de información sobre el símbolo y la sección a la que pertenece.

`nm` utiliza una serie de reglas y prioridades para asignar este carácter. Si un símbolo cumple múltiples condiciones, se aplica la regla de mayor prioridad.

Aquí están los elementos clave que influyen en el `char_type`, y cómo se usan:

1. `symbol_binding` (STB_LOCAL, STB_GLOBAL, STB_WEAK)

Este valor indica la **visibilidad o el "enlace" del símbolo**. Lo extraes del campo `st_info` del símbolo usando las macros `ELFXX_ST_BIND`.

- `STB_GLOBAL` **(Global):** Son símbolos visibles fuera de su unidad de compilación (archivo `.o` o biblioteca). Generalmente se representan con una letra **mayúscula** en la salida de `nm` (ej. `T`, `D`, `B`).
- `STB_LOCAL` **(Local):** Son símbolos visibles solo dentro de su unidad de compilación. Se representan con una letra **minúscula** (ej. `t`, `d`, `b`).
- `STB_WEAK` **(Débil):** Son símbolos que pueden ser sobrescritos por un símbolo global con el mismo nombre en otro archivo.
  - Si un símbolo débil no está definido (su `st_shndx` es `SHN_UNDEF`), se representa con `w`.
  - Si está definido, se representa con `W`.
- **Otros *bindings*:** Hay otros como `STB_GNU_UNIQUE`, `STB_LOPROC`, `STB_HIPROC`, pero para un `ft_nm` básico, los tres anteriores son los más relevantes.

Dependencia: El `symbol_binding` es crucial para decidir si el carácter final es **mayúscula o minúscula**. Si es `STB_GLOBAL` o `STB_WEAK` (y definido), será mayúscula. Si es `STB_LOCAL`, será minúscula (con algunas excepciones como `U` o `w` que siempre son minúsculas por su naturaleza de "no definido").

2. `symbol_type` (STT_NOTYPE, STT_OBJECT, STT_FUNC, STT_SECTION, STT_FILE, etc.)

Este valor describe la **naturaleza del símbolo** (qué representa). Lo extraes también del campo `st_info` usando las macros `ELFXX_ST_TYPE`.

- `STT_NOTYPE`**:** Tipo no especificado. `nm` a menudo lo representa con `?` o, si la sección lo permite, como `U` si es indefinido.
- `STT_OBJECT`**:** Símbolo que se refiere a datos (variables). Se mapea a `D` (datos inicializados), `B` (datos no inicializados/BSS), o `R` (datos de solo lectura).
- `STT_FUNC`**:** Símbolo que se refiere a una función (código ejecutable). Se mapea a `T` (código).
- `STT_SECTION`**:** Símbolo asociado a una sección del archivo. Normalmente se representa con `s` o `S`.
- `STT_FILE`**:** Símbolo que da el nombre del archivo fuente. A menudo no se imprime por `nm` o se usa internamente.
- `STT_COMMON`**:** Símbolo de "common block" (un tipo de datos no inicializados). Se mapea a `C`.
- `STT_TLS`**:** Símbolo de Thread Local Storage.
- **Otros tipos:** `STT_GNU_IFUNC`, `STT_LOOS`, `STT_HIOS`, `STT_LOPROC`, `STT_HIPROC`.

Dependencia: El `symbol_type` es la base para determinar la categoría principal del símbolo (¿es una función, un dato, etc.?).

3. `st_shndx_val` (SHN_ABS, SHN_COMMON, SHN_UNDEF, etc.)

Este valor es el **índice de la sección a la que pertenece el símbolo**. Es un campo directo de la estructura `ElfX_Sym`.

- `SHN_UNDEF` **(Undefined):** El símbolo no está definido en este archivo. Es un símbolo **externo** que será resuelto por el enlazador. Siempre se representa con `U` (o `w` si es débil).
- `SHN_ABS` **(Absolute):** El valor del símbolo es absoluto y no cambia durante el enlace. Se representa con `A`.
- `SHN_COMMON` **(Common):** Indica un símbolo "common block", una forma de datos no inicializados. Se representa con `C`.
- `SHN_XINDEX`**:** Indica que el índice de sección real es demasiado grande para `st_shndx` y está en la tabla `SHN_XINDEX`. (Caso avanzado, probablemente no necesario para tu `ft_nm`).
- **Otros valores (índices de sección reales):** Si `st_shndx_val` es un número mayor que 0 y menor que `SHN_LORESERVE` (generalmente 0xFF00), significa que el símbolo está definido en la sección de la tabla de secciones con ese índice. Aquí es donde se vuelve más complejo, ya que necesitas consultar los **flags de esa sección**.

Dependencia: `st_shndx_val` tiene la **mayor prioridad** para algunos casos especiales (`U`, `A`, `C`, `w`). Si el símbolo no cae en estas categorías especiales, entonces `st_shndx_val` te indica qué

sección debes examinar para determinar el `char_type`.

4. El Tipo y los Flags de la Sección a la que Apunta `st_shndx_val`

Si `st_shndx_val` es un índice de sección real (no `SHN_UNDEF`, `SHN_ABS`, `SHN_COMMON`), entonces necesitas ir a la **tabla de cabeceras de sección (`aux->elf32_sh_table` o `aux->elf64_sh_table`)** y buscar la entrada correspondiente a ese índice (`st_shndx_val`).

Una vez que tengas esa cabecera de sección (`ElfX_Shdr`), deberás revisar dos de sus campos:

- `sh_type` **(Tipo de Sección):**
  - `SHT_PROGBITS`: Contiene datos definidos por el programa (código o datos).
  - `SHT_NOBITS`: Contiene datos que ocupan espacio en memoria pero no en el archivo (BSS).
  - `SHT_STRTAB`: Tabla de cadenas.
  - `SHT_SYMTAB`: Tabla de símbolos.
  - ... y muchos más.
- `sh_flags` **(Flags de Sección):** Estos flags indican las propiedades de la sección.
  - **`SHF_ALLOC`:** La sección ocupa memoria durante la ejecución. (Casi todas las secciones de código/datos importantes tienen esto).
  - **`SHF_EXECINSTR`:** La sección contiene código ejecutable. (Clave para `T` o `t`).
  - **`SHF_WRITE`:** La sección puede ser escrita durante la ejecución (datos modificables). (Clave para `D` o `d`, `B` o `b`).
  - **`SHF_MERGE`, `SHF_STRINGS`, etc.**

Dependencia: Esta es la parte más compleja. El `char_type` para `STT_OBJECT` y `STT_FUNC` a menudo depende de los `sh_type` y `sh_flags` de la sección en la que residen.

- Si `symbol_type == STT_FUNC` y la sección tiene `SHF_EXECINSTR` y `SHF_ALLOC`, es probable que sea `T` (o `t` si es local).
- Si `symbol_type == STT_OBJECT` y la sección tiene `SHF_ALLOC` y `SHF_WRITE` y `sh_type == SHT_PROGBITS`, podría ser `D` (o `d`).
- Si `symbol_type == STT_OBJECT` y la sección tiene `SHF_ALLOC` y `SHF_WRITE` y `sh_type == SHT_NOBITS`, es `B` (o `b`, para BSS).
- Si `symbol_type == STT_OBJECT` y la sección tiene `SHF_ALLOC` pero NO `SHF_WRITE`, y `sh_type == SHT_PROGBITS`, podría ser `R` (o `r`, para solo lectura).

Orden de Prioridad de la Lógica

La lógica para determinar el `char_type` sigue un orden de prioridad, ya que un símbolo puede cumplir varias condiciones. Por ejemplo, un símbolo puede ser `STB_WEAK` y `SHN_UNDEF` al mismo tiempo, lo cual lo convierte en `w`.

Un orden común de evaluación (de mayor a menor prioridad) es:

1. **Símbolos Indefinidos (`SHN_UNDEF`):**
   - Si `st_shndx_val == SHN_UNDEF`:
     - Si `symbol_binding == STB_WEAK`: `char_type = 'w'`
     - En cualquier otro caso: `char_type = 'U'`
       (Estas son generalmente las primeras comprobaciones)
2. **Símbolos Comunes (`SHN_COMMON`):**
   - Si `st_shndx_val == SHN_COMMON`: `char_type = 'C'`
3. **Símbolos Absolutos (`SHN_ABS`):**
   - Si `st_shndx_val == SHN_ABS`: `char_type = 'A'`
4. **Símbolos Débiles Definidos (`STB_WEAK`):**
   - Si `symbol_binding == STB_WEAK` (y ya sabemos que NO es `SHN_UNDEF`): `char_type = 'W'`
5. **Símbolos de Sección (`STT_SECTION`):**
   - Si `symbol_type == STT_SECTION`: `char_type = 'S'`
6. **Símbolos de Archivo (`STT_FILE`):**
   - Si `symbol_type == STT_FILE`: `nm` a menudo ignora estos o los deja con un tipo `?`.
7. **Símbolos en Secciones Específicas (Objetos y Funciones):**
   - Aquí es donde necesitas la función auxiliar que te devuelva la `ElfX_Shdr` dado un `st_shndx_val`.
   - Si la sección es una de "bits de programa" (`SHT_PROGBITS`) y tiene el flag `SHF_EXECINSTR`: `char_type = 'T'` (para funciones, etc.).
   - Si es `SHT_PROGBITS` y tiene `SHF_WRITE` (y `SHF_ALLOC`): `char_type = 'D'` (datos inicializados).

- Si es `SHT_NOBITS` y tiene **`SHF_WRITE`** (y `SHF_ALLOC`): `char_type = 'B'` (datos no inicializados, BSS).
- Si es `SHT_PROGBITS` y tiene **`SHF_ALLOC`** pero NO `SHF_WRITE`: `char_type = 'R'` (datos de solo lectura).
- También hay otros casos para `N` (debug sections), `V` (versión), etc.

8. **Conversión a minúscula (Local vs. Global):**
   - Después de determinar el carácter base (`U`, `A`, `C`, `T`, `D`, `B`, `R`, `W`, `S`), si `symbol_binding == STB_LOCAL` (y el carácter no es `U` o `w` que ya son minúsculas), entonces conviertes el carácter a minúscula.

Lo que Necesitas Implementar

1. **Función para Obtener Cabecera de Sección:**
   Necesitas una función auxiliar, por ejemplo:
   ```
   Elf32_Shdr *get_shdr32(uint16_t shndx_val, Elf32_Shdr *sh_table, uint16_t sh_num);
   Elf64_Shdr *get_shdr64(uint16_t shndx_val, Elf64_Shdr *sh_table, uint16_t sh_num);
   ```
   Estas funciones te darían el puntero a la cabecera de sección para el índice dado, permitiéndote examinar `sh_type` y `sh_flags`.

2. **La Lógica Completa en `extr_detc_symbol_type`:**
   Dentro de tu bucle que itera sobre `aux_sym`, aplicarás toda la cadena de `if/else if` basada en las prioridades y combinaciones de `symbol_binding`, `symbol_type`, `st_shndx_val`, y los flags de sección.

3. **Manejo de Errores/Por Defecto:** Si después de todas las comprobaciones no se ha asignado un `char_type`, un `?` suele ser el valor por defecto para símbolos de tipo desconocido.

Este paso, `extr_detc_symbol_type`, es una función clave que iterará sobre tu `aux_file->symbol_list` y rellenará el campo `char_type` de cada `t_symbol_info` con el carácter adecuado.