MANUAL MINISHELL

1. **Arquitectura general del programa**

**Este programa está compuesto por varios módulos cada uno con un cometido determinado. A continuación vamos a describir cada uno de los módulos.**

* 1. **Inicio del programa. Main.c**

**En el main vamos a inicializar las estructuras principales del programa. En ella, vamos a ejecutar un bucle principal del programa en el que vamos a leer y ejecutar las ordenes que se nos envíe. Además nos va a permitir el manejo de señales como, por ejemplo, Ctrl+C**

**Las funciones claves del main son las siguientes:**

* **Inicializar el estado global con la estructura (t\_data)**
* **Realizar el bucle operativo:**
  1. **Mostrar el prompt.**
  2. **Leer la entrada. (Uso de la función read\_l)**
  3. **Construir los tokens. (A través del lexer)**
  4. **Parsear los tokens para el AST. (A través del parser)**
  5. **Ejecutar los comandos. (A través del ejecutor del AST)**
  6. **Liberar la memoria.**
* **Detectar una posible salida a través de (check\_exit)**
  1. **Análisis léxico. Lexer.c**

**El cometido del análisis léxico es convertir el string de entrada en una secuencia de tokens. Los tokens hacen referencia a cada uno de los posibles inputs que nos pueden enviar, como comandos, pipes, redirecciones, palabras....**

**Las funciones claves del lexer son las siguientes:**

* **Orquestar el análisis a través de la función lex( ).**
* **Identificar los tokens uno por uno a través de la función tokenize( ).**
* **Generar una estructura (t\_tok) para los tokens (tipo, contenido, posición)** 
  1. **Análisis sintácticos (AST). Parser.c**

**El cometido del análisis sintáctico es convertir la lista de tokens en un Árbol de Sintaxis Abstracta (o AST por sus siglas en inglés). El Árbol representa la estructura lógica de comandos: tuberías, secuencias, condiciones, redirecciones...**

**Las funciones claves del análisis sintáctico son las siguientes:**

* **Construcción del árbol con la función parse( ).**
* **Generación de las funciones para tratar cada unos de los tokens, parse\_pipeline( ), parse\_command()...**
* **Generar una estructura (t\_node) para representar cada nodo del AST. Los tipos de nodos generados son:**
  + **NODE\_COMMAND. Comando simple.**
  + **NODE\_PIPE. Pipe entre dos comandos.**
  + **NODE\_AND, NODE\_OR. Condicionales && y ||.**
  + **NODE\_REDIR. Nodos para las redirecciones.**

* 1. **Ejecución de comandos. Executor.c**

**El cometido del ejecutor es recorrer el AST y ejecutar los comandos en el orden correspondiente. En el ejecutor manejamos los forkings, los piping, las redirecciones y la ejecución de builtins.**

**Las funciones claves del ejecutor de comandos son las siguientes:**

* **Ejecutar el árbol completo a través de la función exec\_ast( ).**
* **Ejecutar los nodos de tipo a través de la función exec\_command( ).**
* **Manejar múltiples procesos conectados simultáneamente a través de pipes.**
* **Configurar las redirecciones antes del ejecutor a través de la función apply\_redirections( ).**
  1. **Comandos internos. Builtins.c**

**El cometido de los comandos internos es la implementación de los comandos integrados de la minishell, a saber: cd, pwd, exit, echo, env.**

**Las funciones claves de los Builtins son las siguientes:**

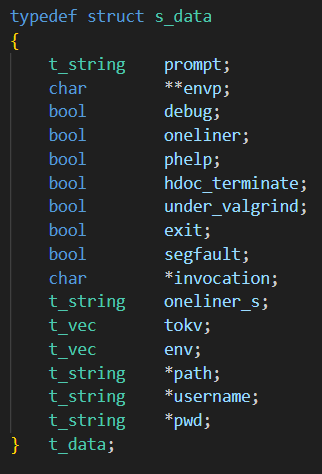
* **Cambiar el directorio de trabajo a través de la función sh\_\_builtin\_cd( ).**
* **Mostrar el texto bajo condiciones específicas a través de la función sh\_\_builtin\_echo( ).**
* **Mostrar la ruta absoluta del directorio actual a través de la función sh\_\_builtin\_pwd( ).**
* **Cerrar la sesión actual de la minishell o salir de un script a través de la función sh\_\_builtin\_exit( ).**
* **Mostrar las variables de entorno a través de la función sh\_\_builtin\_env( ).**

**La minishell verifica previamente si el comando introducido es un Builtin antes de realizar un fork(). En caso de ser un Builtin lo ejecuta directamente en el proceso padre (por ejemplo, el Builtin cd debe modificar el entorno actual).**

* 1. **Utilidades auxiliares. Utils/**

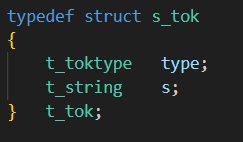
**El cometido de las funciones o utilidades auxiliares es la manipulación de cadenas, memoria, listas, o estructuras dinámicas (como t\_vec ó t\_string).**

**Las funciones claves de las utilidades auxiliares son las siguientes:**

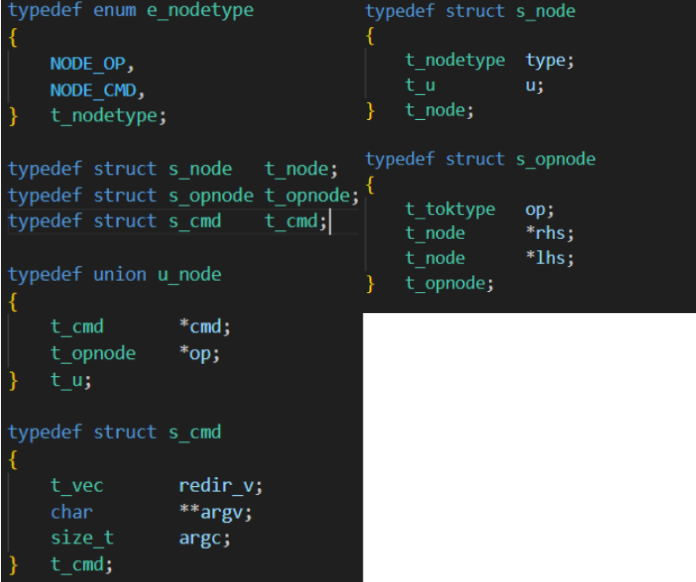
* **Manejo de string dinámicos a través de las funciones (ft\_tstr\_from\_cstr( ), ft\_tstr\_trim( ), ft\_tstr\_free())**
* **Manejo de vectores dinámicos a través de las funciones (ft\_vec\_\*( ))**
* **Manejo del historial de comandos a través de la función (add\_history( ))**
  1. **Estructuras de datos principales.**

**Las estructuras de datos principales utilizadas son las siguientes:**

* **t\_data: Muestra el estado global de la minishell.**
  + **Pwd: Directorio actual.**
  + **Exit: Flag para terminar la minishell.**
  + **Envp: Entorno de la minishell.**
* **t\_tok: Estructura par los tokens.**

****

* **t\_node: Estructura para los nodos del AST.**

****