Contents

1 Documentation Détaillée des Headers et Fonctions

1.1 Table des Matières

1.2 Headers de Base

1.2.1 `minishell.h` - Header Principal

Fichier : `include/minishell.h`

Description : Header principal qui inclut toutes les dépendances système et du projet.

```
#include <dirent.h>          // Opérations sur répertoires
#include <errno.h>           // Codes d'erreur
#include <fcntl.h>           // Contrôle fichiers
#include <readline/history.h> // Historique readline
#include <readline/readline.h> // Interface readline
```

```c
#include <signal.h>          // Gestion signaux
#include <stdbool.h>         // Type booléen
#include <stddef.h>          // Définitions standard
#include <stdio.h>           // Entrées/sorties standard
#include <stdlib.h>          // Fonctions standard
#include <string.h>          // Manipulation strings
#include <sys/ioctl.h>       // Contrôle I/O
#include <sys/stat.h>        // Statistiques fichiers
#include <sys/types.h>       // Types système
#include <sys/wait.h>        // Attente processus
#include <termios.h>         // Contrôle terminal
#include <unistd.h>          // Appels système POSIX
```

1.2.1.1 Inclusions Système

```c
#include "ast.h"          // Structures AST
#include "command.h"      // Types de commandes
#include "libft.h"        // Bibliothèque custom
#include "list.h"         // Structures de listes
#include "posix.h"        // Extensions POSIX
#include "token.h"        // Gestion tokens
#include "user_info.h"    // Informations utilisateur
#include "builtin.h"      // Commandes intégrées
#include "cleaner.h"      // Nettoyage mémoire
#include "env.h"          // Gestion environnement
#include "execution.h"    // Exécution commandes
#include "ms_signal.h"    // Gestion signaux
#include "parsing.h"      // Parsing et tokenisation
#include "redirection.h" // Gestion redirections
#include "syntax.h"       // Vérification syntaxe
#include "utils.h"        // Utilitaires
#include "wildcard.h"     // Expansion wildcards
```

1.2.1.2 Inclusions Projet

```c
#define PROMPT "minishell> "     // Prompt par défaut
#define PATH_MAX 4096            // Taille max chemin
#define ERROR -1                 // Code erreur
#define SUCCESS 0                // Code succès
```

1.2.1.3 Constantes Globales

```c
extern volatile sig_atomic_t g_received_signal;  // Signal reçu
```

1.2.1.4 Variables Globales

1.3 Headers de Structures

1.3.1 `ast.h` - Abstract Syntax Tree

Fichier : `include/ast.h`

Description : Définit les structures pour l'arbre syntaxique abstrait et les contextes associés.

```
typedef struct s_ast
{
    t_enum_cmd      type;               // Type de nœud
    char            **argv;             // Arguments commande
    char            *infile;            // Fichier entrée
    char            *outfile;           // Fichier sortie
    int             append_mode;        // Mode append (>>)
    int             stderr_to_devnull;  // Redirection stderr
    char            *heredoc_delimiter; // Délimiteur heredoc
    pid_t           child;              // PID processus enfant
    char            **full_cmd;         // Commande complète
    char            *ful_path;          // Chemin complet exécutable
    struct s_ast    *left;              // Nœud enfant gauche
    struct s_ast    *right;             // Nœud enfant droit
} t_ast;
```

1.3.1.1 Structure Principale AST

Utilisation : - **type** : Détermine le type de nœud (SIMPLE, PIPE, AND, OR, etc.) - **argv** : Tableau des arguments pour commandes simples - **left/right** : Pointeurs pour structure arborescente - **infile/outfile** : Gestion des redirections - **child** : Suivi des processus enfants

```
typedef struct s_expand_ctx
{
    char            **env;          // Variables d'environnement
    int             exit_status;    // Code de sortie précédent
} t_expand_ctx;
```

1.3.1.2 Contexte d'Expansion

Utilisation : Passe le contexte nécessaire pour l'expansion des variables lors du parsing.

```
typedef struct s_parse_word_ctx
{
    const char      *s;             // String à parser
    int             i;              // Index courant
    t_token         *token;         // Token en cours
    t_expand_ctx    expand_ctx;     // Contexte expansion
} t_parse_word_ctx;
```

1.3.1.3 Contexte de Parsing de Mots

```c
typedef struct s_match_ctx
{
    const char      *pattern;       // Pattern à matcher
    const char      *dir;           // Répertoire à explorer
    char            **matches;      // Résultats trouvés
    int             *count;         // Nombre de matches
} t_match_ctx;
```

1.3.1.4 Contexte de Matching Wildcards

1.3.2 `token.h` - Gestion des Tokens

Fichier : `include/token.h`

Description : Structures pour la tokenisation de l'input utilisateur.

```c
typedef struct s_token
{
    int     type;    // Type du token (voir énumérations)
    char    *value;  // Valeur string du token
} t_token;
```

1.3.2.1 Structure Token **Types de tokens** : - Mots (commandes, arguments) - Opérateurs (|, &&, ||, >, <, », «) - Parenthèses (,) - Quotes (gérées dans la valeur)

```c
typedef struct s_tokenize_context
{
    char    **env;          // Environnement pour expansion
    t_list  **tokens;       // Liste des tokens créés
    int     exit_status;    // Code sortie pour $?
} t_tokenize_context;
```

1.3.2.2 Contexte de Tokenisation

1.3.3 `command.h` - Types de Commandes

Fichier : `include/command.h`

Description : Énumérations et structures pour les différents types de commandes.

```c
typedef enum e_enum_cmd
{
    SIMPLE,         // Commande simple (ls, echo, etc.)
    WORD,           // Mot/argument
    PIPE,           // Pipe |
    AND,            // Opérateur logique &&
    OR,             // Opérateur logique ||
    REDIR_IN,       // Redirection entrée <
    REDIR_OUT,      // Redirection sortie >
```

```
    REDIR_APPEND,   // Redirection append >>
    HEREDOC,        // Here document <<
    LPAREN,         // Parenthèse ouvrante (
    RPAREN,         // Parenthèse fermante )
    SUBSHELL,       // Sous-shell
    CMD_INVALID     // Commande invalide
} t_enum_cmd;
```

1.3.3.1 Énumération Types de Commandes

```
typedef int (*t_builtin_func)(t_user_info *, t_list *, int *);

typedef struct s_builtin
{
    char            *name;          // Nom de la commande
    t_builtin_func  func;           // Pointeur vers fonction
} t_builtin;
```

1.3.3.2 Structure Builtin

1.3.4 `list.h` - Structures de Listes

Fichier : `include/list.h`

Description : Structure générique de liste chaînée.

```
typedef struct s_list
{
    void            *content;   // Contenu du nœud
    struct s_list   *next;      // Pointeur vers suivant
} t_list;
```

1.3.4.1 Structure Liste **Utilisation** : - Stockage des tokens - Listes d'arguments - Gestion générique de collections

1.3.5 `user_info.h` - Informations Utilisateur

Fichier : `include/user_info.h`

Description : Structure principale contenant l'état du shell.

```
typedef struct s_user_info
{
    char    **env;              // Variables d'environnement
    int     last_exit_code;     // Code de sortie précédent
    pid_t   shell_pid;          // PID du shell principal
    // Autres champs selon implémentation
} t_user_info;
```

1.4 Headers de Modules

1.4.1 `parsing.h` - Module de Parsing

Fichier : `include/parsing.h`

Description : Déclarations de toutes les fonctions de parsing, tokenisation et expansion.

```c
// Fonction principale de tokenisation
int tokenize(const char *input, t_list **tokens, char **env, int exit_status);

// Utilitaires de reconnaissance
int is_space(char c);                    // Vérifie si caractère est espace
int is_operator(char c);                 // Vérifie si caractère est opérateur
int is_digit_redirection(const char *s, int i);  // Redirection avec fd
int is_quote(char c);                    // Vérifie si caractère est quote

// Traitement des tokens
t_token *alloc_token(void);              // Alloue nouveau token
int process_token(char *cleaned, int i, t_tokenize_context *ctx, int *new_index);
int tokenize_loop_t(char *cleaned, t_tokenize_context *ctx);
```

1.4.1.1 Fonctions de Tokenisation

```c
// Parsing des différents types d'opérateurs
int parse_operator_t(const char *s, int i, t_token *token);
int parse_pipe_paren_operator(const char *s, int i, t_token *token);
int parse_redir_in(const char *s, int i, t_token *token);
int parse_double_operator(const char *s, int i, t_token *token);
int parse_redir_devnull(const char *s, int i, t_token *token);
int parse_redir_out(const char *s, int i, t_token *token);
int parse_redirection_operator(const char *s, int i, t_token *token);
int parse_single_operator(const char *s, int i, t_token *token);
```

1.4.1.2 Fonctions de Parsing d'Opérateurs

```c
// Expansion des variables
char *expand_variable(const char *str, int *i, char **env, int exit_status);
char *expand_variable_in_dquotes(const char *str, int *i, char **env, int exit_status);
char *expand_variable_improved(const char *str, int *pos, char **env, int exit_status);

// Expansion tilde
char *expand_tilde(const char *str, int *i, char **env);

// Fonction principale d'expansion
char *remove_quotes_and_expand(const char *input, char **env);

// Utilitaires
int skip_spaces(const char *s, int i);
```

1.4.1.3 Fonctions d'Expansion

```c
// Quotes simples (pas d'expansion)
int handle_single_quote(const char *s, int *i, char **value);
int handle_single_quote_content(const char *s, int *i, char **value);

// Quotes doubles (avec expansion)
int handle_double_quote(const char *s, int *i, char **value, t_expand_ctx *ctx);
int handle_double_quote_content_ctx(const char *s, int *i, char **value, t_expand_ctx *ctx);
int handle_double_quote_ctx(const char *s, int *i, char **value, t_expand_ctx *ctx);
int handle_double_quote_expansion(const char *s, int *i, char **value, t_expand_ctx *ctx);
```

1.4.1.4 Fonctions de Gestion des Quotes

```c
// Gestion du symbole $
int handle_dollar(const char *s, int *i, char **value, t_expand_ctx *ctx);
int handle_dollar_ctx(const char *s, int *i, char **value, t_expand_ctx *ctx);
int handle_dollar_in_dquotes(const char *s, int *i, char **value, t_expand_ctx *ctx);

// Gestion du tilde ~
int handle_tilde(const char *s, int *i, char **value, t_expand_ctx *ctx);
int handle_tilde_ctx(const char *s, int *i, char **value, t_expand_ctx *ctx);

// Gestion code de sortie $?
int handle_exit_status(char **value, int *i, t_expand_ctx *ctx);

// Ajout de variables expandées
int append_expanded_variable_ctx(const char *s, int *i, char **value, t_expand_ctx *ctx);
```

1.4.1.5 Fonctions d'Expansion de Variables

```c
// Construction de l'AST
t_ast *new_ast_node(t_enum_cmd type);        // Crée nouveau nœud
t_ast *parse_command(t_list *tokens);        // Parse commande complète
t_ast *parse_command_recursive(t_list *tokens);  // Parsing récursif
t_ast *parse_logical_command(t_list *tokens);    // Parse && et ||
t_ast *parse_simple_command(t_list **tokens);   // Parse commande simple
```

1.4.1.6 Fonctions de Parsing AST

```c
// Manipulation des tokens
t_token *duplicate_token(t_token *token);    // Duplique token
int find_last_pipe(t_list *tokens);         // Trouve dernier pipe
int find_last_logical_operator(t_list *tokens);  // Trouve dernier && ou ||
int find_last_l_oper_with_paren(t_list *tokens); // Avec parenthèses
```

```c
// Division des listes de tokens
void split_token(t_list *tokens, int index, t_list **left, t_list **right);
```

1.4.1.7 Utilitaires de Parsing

```c
// Remplissage des nœuds AST
void fill_cmd_argv(t_ast *node, t_list *tokens);     // Remplit argv
int fill_cmd_redir(t_ast *node, t_list *tokens);     // Remplit redirections
void fill_simple_cmd(t_ast *node, t_list *tokens);   // Remplit commande simple

// Comptage et allocation d'arguments
int count_cmd_args_b(t_list *tokens);                // Compte arguments
void fill_cmd_args_b(t_ast *node, t_list *tokens, int count);  // Remplit args
int fill_args_loop_b(char ***argv, t_list *tokens, int *arg_count, int *capacity);
int process_token_b(char ***argv, int *capacity, int *arg_count, t_token *tok);
void free_argv_on_error_b(char **argv, int arg_count);  // Nettoyage erreur
```

1.4.1.8 Fonctions de Remplissage de Commandes

```c
// Expansion et ajout d'arguments
int add_expanded_args(char ***argv_ptr, int *capacity, int *count, const char *token_value);
int ensure_argv_capacity(char ***argv_ptr, int *capacity, int needed);
void add_expanded_to_argv(char ***argv_ptr, int *count, char **expanded);
```

1.4.1.9 Fonctions d'Expansion d'Arguments

1.4.2 `execution.h` - Module d'Exécution

Fichier : `include/execution.h`

Description : Déclarations pour l'exécution des commandes et gestion des processus.

```c
// Exécution de l'AST
int execute_ast(t_ast *node, t_user_info *user, int *exiter);
int execute_simple(t_ast *node, t_user_info *user, int *exiter);
int execute_pipe(t_ast *node, t_user_info *user, int *exiter);
```

1.4.2.1 Fonctions Principales d'Exécution

```c
// Recherche de commandes
char *find_command_path(char *cmd, char **envp);
char *resolve_command_path(const char *cmd, char **env);
char *find_command_in_path(const char *cmd, char **env);
char *build_full_path_path(const char *dir, const char *cmd);
```

1.4.2.2 Résolution de Chemins

```c
// Tests sur fichiers et répertoires
int file_exists_path(const char *path);
int is_executable_file_path(const char *path);
int is_executable(const char *path);
int is_directory(const char *path);
int is_builtin_command(const char *cmd);
```

1.4.2.3 Vérifications Fichiers

```c
// Fork et exécution
pid_t fork_and_exec_left(t_ast *left, int pipefd[2], t_user_info *user, int *exiter);
pid_t fork_and_exec_right(t_ast *right, int pipefd[2], t_user_info *user, int *exiter);
int wait_for_child_sim(pid_t pid, int *exiter);
int handle_child_process_s(t_ast *node, t_user_info *user);
```

1.4.2.4 Gestion des Processus

```c
// Gestion des pipes
int create_pipe_1(int pipefd[2]);
```

1.4.2.5 Utilitaires Pipes

```c
// Nettoyage processus enfants
void cleanup_child_memory_1(t_user_info *user);
void cleanup_child_memory_sim(t_user_info *user);
```

1.4.2.6 Nettoyage Mémoire

```c
// Variable underscore (_)
void update_underscore_in_pipeline_1(t_user_info *us, t_ast *left_node);
void update_underscore_var_s(t_user_info *user, t_ast *node);
```

1.4.2.7 Mise à Jour Environnement

```c
// Gestion des erreurs de commandes
void handle_cmd_path_errors_s(const char *cmd_path, const char *cmd_name, t_user_info *user);
void handle_permission_denied(const char *cmd_path, const char *cmd_name, t_user_info *user);
void handle_is_directory(const char *cmd_path, const char *cmd_name, t_user_info *user);
void handle_dot_command(const char *cmd_path, const char *cmd_name, t_user_info *user);

// Fonctions de sortie
void exit_sim(int code, char *msg, char *to_free, t_user_info *user);
void print_and_exit_perm(const char *cmd_name, t_user_info *user);
void print_and_exit_dir(const char *cmd_name, t_user_info *user);
```

1.4.2.8 Gestion d'Erreurs Exécution

1.4.3 `builtin.h` - Commandes Intégrées

Fichier : `include/builtin.h`

Description : Déclarations des commandes builtin du shell.

```
// Gestion globale des builtins
int ms_builtin(t_user_info *prompt, t_list *cmd, int *exiter);

// Commandes individuelles
int ms_echo(t_list *cmd);                          // echo
int ms_pwd(void);                                  // pwd
int ms_cd(char **args, char ***env);               // cd
int ms_exit(int argc, char **argv, char **envp);   // exit
int ms_env(char **env);                            // env

// Fonctions de gestion avancée
int handle_builtin_exit(char **args, int argc, char **envp, int *exiter);
int handle_builtin_export(t_user_info *prompt, char **args);
int handle_builtin_unset(t_user_info *prompt, char **args);
```

1.4.3.1 Commandes Builtin Principales

```
// Wrappers pour compatibilité avec interface générale
int ms_cd_wrapper(t_user_info *prompt, t_list *cmd, int *exiter);
int ms_env_export_wrapper(t_user_info *prompt, t_list *cmd, int *error_code);
int ms_unsetenv_wrapper(t_user_info *prompt, t_list *cmd, int *exiter);
int ms_env_wrapper(t_user_info *prompt, t_list *cmd, int *exiter);
int ms_exit_wrapper(t_user_info *prompt, t_list *cmd, int *exiter);
int ms_echo_wrapper(t_user_info *prompt, t_list *cmd, int *exiter);
int ms_pwd_wrapper(t_user_info *prompt, t_list *cmd, int *exiter);
```

1.4.3.2 Wrappers pour Interface Uniforme

```
// Utilitaires pour cd
char *get_target_directory(char **args, char **env);
int update_env_pwd(char ***env, char *oldpwd);

// Utilitaires pour export
void ft_export_list(char **env);                    // Affiche variables
void print_export_error(const char *arg);           // Affiche erreurs
int is_valid_identifier_export(const char *str);    // Valide identifiant
int handle_export_args_2(char ***envp, char **args);
int handle_export_arg(char ***envp, const char *arg);
void free_new_env(char **new_env, int j);
```

```c
// Substitution de variables
void substitute_env_vars(char **args, char **env);
```

1.4.3.3 Utilitaires Builtin

1.4.4 `env.h` - Gestion Environnement

Fichier : `include/env.h`

Description : Gestion des variables d'environnement.

```c

// Validation des noms de variables
int ms_is_valid_var_char(char c);      // Caractère valide dans nom
int ms_is_valid_var_name(char *s);     // Nom de variable valide
```

1.4.4.1 Validation Variables

```c

// Accès aux variables
char *ms_getenv(char **env, const char *key, int n);   // Récupère variable
char *get_env_var(const char *name, char **env);       // Alternative

// Modification de l'environnement
char **ms_setenv(char **env, const char *key, const char *value);  // Définit
char **ms_unsetenv(char **env, const char *key);                   // Supprime
char **ms_env_add(char **env, char *entry);                        // Ajoute

// Export spécialisé
char **ms_env_export(char **env, const char *arg, int *error_code);
```

1.4.4.2 Manipulation Variables

```c

// Initialisation
char **ms_env_init(char **envp);           // Copie environnement
t_list *ms_env_list_init(char **envp);     // Version liste
```

1.4.4.3 Initialisation Environnement

1.4.5 `libft.h` - Bibliothèque Custom

Fichier : `include/libft.h`

Description : Fonctions de base réimplémentées.

```c

// Fonctions de base
size_t ft_strlen(const char *s);
char *ft_strcpy(char *dest, const char *src);
char *ft_strncpy(char *dest, const char *src, size_t n);
char *ft_strcat(char *dest, const char *src);
```

```c
char *ft_strdup(const char *s1);

// Comparaison et recherche
int ft_strcmp(const char *s1, const char *s2);
int ft_strncmp(const char *s1, const char *s2, size_t n);
char *ft_strchr(const char *s, int c);
char *ft_strrchr(const char *s, int c);
int ft_strchr_i(const char *s, int c);

// Manipulation avancée
char *ft_strjoin(char const *s1, char const *s2);
char *ft_strjoin_3(char const *s1, char const *s2, char const *s3);
char *ft_strjoin_free(char *s1, char *s2);
char *ft_substr(char const *s, unsigned int start, size_t len);
size_t ft_strlcpy(char *dst, const char *src, size_t dstsize);
```

1.4.5.1 Manipulation de Strings

```c
// Classification caractères
int ft_isalpha(int c);          // Alphabétique
int ft_isalnum(int c);          // Alphanumérique
int ft_is_numerique(char *str); // String numérique
```

1.4.5.2 Tests de Caractères

```c
// Conversion nombres
char *ft_itoa(int n);           // Int vers string
long ft_atol(const char *str);  // String vers long
```

1.4.5.3 Conversion

```c
// Opérations sur listes
void ft_lstadd_back(t_list **lst, t_list *new);
void ft_lstclear(t_list **lst, void (*del)(void*));
void ft_lstdelone(t_list *lst, void (*del)(void*));
int ft_lstsize(t_list *lst);
t_list *ft_lstget(t_list *lst, int index);
```

1.4.5.4 Gestion Listes

```c
// Opérations sur tableaux de strings
char **ft_split(char const *s, char c);    // Split string
int ft_matrix_len(char **matrix);          // Longueur matrice
int ft_countchar(const char *s, char c);   // Compte caractères
```

1.4.5.5 Manipulation Matrices

```c
// Tokenisation manuelle
char *ft_strtok(char *str, const char *delim);
```

1.4.5.6 Tokenisation

```c
// Fonctions d'affichage
void ft_putchar_fd(char c, int fd);
void ft_putstr_fd(char *s, int fd);
void ft_putendl_fd(char *s, int fd);
```

1.4.5.7 Sortie

1.5 Index des Fonctions

1.5.1 Fonctions par Catégorie

1.5.1.1 Parsing et Tokenisation

Fonction	Description	Header
`tokenize()`	Tokenise input utilisateur	parsing.h
`parse_command()`	Parse commande en AST	parsing.h
`expand_variable()`	Expanse variables $VAR	parsing.h
`handle_single_quote()`	Gère quotes simples	parsing.h
`handle_double_quote()`	Gère quotes doubles	parsing.h

1.5.1.2 Exécution

Fonction	Description	Header
`execute_ast()`	Exécute AST	execution.h
`execute_simple()`	Exécute commande simple	execution.h
`execute_pipe()`	Exécute pipeline	execution.h
`find_command_path()`	Trouve chemin commande	execution.h
`fork_and_exec_left()`	Fork côté gauche pipe	execution.h

1.5.1.3 Builtins

Fonction	Description	Header
`ms_echo()`	Commande echo	builtin.h
`ms_cd()`	Commande cd	builtin.h
`ms_pwd()`	Commande pwd	builtin.h
`ms_env()`	Commande env	builtin.h
`handle_builtin_export()`	Commande export	builtin.h
`handle_builtin_unset()`	Commande unset	builtin.h

1.5.1.4 Environnement

Fonction	Description	Header
`ms_getenv()`	Récupère variable env	env.h
`ms_setenv()`	Définit variable env	env.h
`ms_unsetenv()`	Supprime variable env	env.h
`ms_env_init()`	Initialise environnement	env.h

1.5.1.5 Libft

Fonction	Description	Header
`ft_strlen()`	Longueur string	libft.h
`ft_strdup()`	Duplique string	libft.h
`ft_strjoin()`	Joint strings	libft.h
`ft_split()`	Divise string	libft.h
`ft_lstadd_back()`	Ajoute à liste	libft.h

1.6 Diagrammes de Dependencies

1.6.1 Dépendances Headers

```
graph TB
    A[minishell.h] --> B[ast.h]
    A --> C[command.h]
    A --> D[libft.h]
    A --> E[list.h]
    A --> F[token.h]
    A --> G[user_info.h]
    A --> H[parsing.h]
    A --> I[execution.h]
    A --> J[builtin.h]
    A --> K[env.h]

    B --> C
    B --> F
    H --> B
    H --> E
    H --> F
    I --> B
    I --> G
    J --> E
    J --> G
    K --> E
    C --> E
    C --> G
```

1.6.2 Flux d'Inclusion

1. **minishell.h** inclut tous les headers système et du projet
2. **ast.h** définit les structures de l'AST et dépend de command.h et token.h
3. **parsing.h** dépend de ast.h, list.h et token.h pour le parsing
4. **execution.h** dépend de ast.h et user_info.h pour l'exécution
5. **builtin.h** dépend de list.h et user_info.h pour les commandes
6. **env.h** dépend de list.h pour la gestion environnement

Cette architecture garantit une séparation claire des responsabilités et évite les dépendances cycliques.

1.7 Conclusion

Cette documentation détaillée des headers fournit une vue complète de l'architecture du projet Minishell. Chaque header a un rôle spécifique et bien défini, permettant une maintenance et une extension faciles du code.

1.7.1 Points Clés

1. **Modularité** : Chaque header gère un aspect spécifique
2. **Séparation** : Headers de structures vs headers de fonctions
3. **Dépendances** : Hiérarchie claire sans cycles
4. **Documentation** : Chaque fonction est documentée avec son rôle
5. **Maintenabilité** : Structure facilitant les modifications

Cette organisation permet à n'importe quel développeur de comprendre rapidement la structure du projet et de contribuer efficacement au code.