minishell

1

Generated by Doxygen 1.15.0

Chapter 1

Minishell

1.1 Introduction

Welcome to the documentation of the Minishell project for EICNAM 2025.

1.2 Features

- Command parsing and execution
- Alias management
- Batch mode
- Command history
- Internal commands (cd, pwd, echo, etc.)

1.3 Usage

See the README for usage instructions.

Chapter 2

Projet - Minishell

2.1 À propos

Ce projet, toujours réalisé dans le cadre de notre module de systèmes avancés, est une version plus légère d'un shell, appelé minishell. Il réponds à la plupart des exigences techniques demandées dans le sujet, celles-ci étant expliquées ci-dessous (lien vers le sujet à l'appui).

2.2 Compilation du projet

Le projet se compile avec le fichier Makefile situé dans le même dossier, à l'aide de la commande `make`. Le fichier exécutable "minishell" se trouvera alors dans le sous-dossier `output/`, prêt à l'emploi.

2.3 Documentation

- [Documentation générée par doxygen **A REMPLIR QUAND ON AURA LES DONNEES**]()
- `PDF généré par doxygen`
- `Sujet du projet`
- `Page de manuel linux`

2.4 Exigences techniques auxquelles nous avons répondu

2.4.1 Capacité d'exécuter des commandes simples

Le minishell réalisé permet l'exécution de commandes simples.

Exemples :

- `ls -a`
- `ps`
- `ls -alF > file.txt`
- `ps aux | grep minishell`

2.4.2 Capacité d'exécuter un sous-ensemble de plusieurs commandes

La logique de différents opérateurs a été recréé, permettant l'accès aux :

- Opérateurs de contrôle
  - Le ET logique: `&&`
  - Le OU logique: ||
- Opérateurs de redirection de flux simples :
  - Le pipe: |
  - Les chevrons: < et >
  - Les chevrons doubles: << et >>
- Exécutions en arrière-plan
  - Symbole `&`

2.4.3 L'exécution built-in de certaines commandes

La logique des commandes suivantes a été réécrite :

- **cd**
- **pwd**
- **exit**
- **echo**

2.4.4 Le maintien d'un historique

Un fichier renseignant toutes les commandes rentrées dans le minishell est disponible sous `~/minishell_↩ logs/command_history` à partir du moment où une commande est rentrée dans le terminal. La commande **history** a également été programmée, et permet la visualisation du fichier susmentionné dans le terminal.

2.4.5 Un mode de lancement "batch"

Un mode de lancement spécial "batch" a été réalisé, qui permet l'exécution de commandes par argument (qu'elles soient simples ou imbriquées avec des opérateurs).

Pour utiliser le mode batch, la syntaxe est la suivante :

- `./[path_to_minishell_executable] [OPTION] <command>`

Les différentes options possibles sont :

- `-c`, `--command <command>`: Execute the specified command and exit.
- `--help`: Display help message and exit.

2.4.6 La prise en charge d'alias temporaires

Une gestion des alias temporaires, donc disponibles durant une session du minishell, a été prévue (gestion et utilisation des alias dans les commandes lancées).

Les différentes actions possibles sur les alias sont les suivantes :

- La définition d'alias
  - `alias [nom_alias]='[command]'`
- L'affichage de la commande correspondante à un alias
  - `alias [nom_alias]`
- L'affichage de toutes les définitions d'alias
  - `alias`
- La suppression d'alias
  - `unalias [nom_alias]`
- La suppression de tous les alias
  - `unalias -a`

2.5 Page de manuel linux

Une page de manuel linux a été créée pour le minishell, et est disponible dans le dossier `doc/` sous le nom `minishell`. Elle peut être consultée en utilisant la commande suivante dans un terminal :

```
man doc/minishell.1
```

Il est possible de l'ajouter au système de manuel linux en copiant le fichier `minishell.1` dans le dossier `/usr/share/man/man1/` (nécessite les droits administrateurs) :

```
sudo cp doc/minishell.1 /usr/share/man/man1/
```

Après cela, la page de manuel pourra être consultée avec la commande :

```
man minishell
```

Si vous ne souhaitez plus avoir accès à la page de manuel, il suffit de supprimer le fichier copié dans `/usr/share/man/man1/` :

```
sudo rm /usr/share/man/man1/minishell.1
```

Chapter 3

Directory Hierarchy

3.1 Directories

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Directory Documentation

6.1 include Directory Reference

Files

- file aliases.h
- file batch.h
- file env.h
- file executor.h
- file history.h
- file internal_commands.h
- file parser.h
- file typedef.h

6.2 src Directory Reference

Directory dependency graph for src:

Files

- file aliases.c

  Implementation of alias handling for the minishell.
- file batch.c

  Handles batch mode execution for the minishell.
- file env.c

  Implementation of environment variable management for the minishell.
- file executor.c

  Implementation of command execution for the minishell.
- file history.c

  Implementation of command history handling for the minishell.
- file internal_commands.c

  Implementation of internal commands for the minishell.
- file main.c

  Main entry point for the minishell application.
- file parser.c

  Implementation of the command line parser for the minishell.

Chapter 7

Data Structure Documentation

7.1 Alias Struct Reference

```
#include <typedef.h>
```

Data Fields

- char ∗ name
- char ∗ command

7.1.1 Detailed Description

Structure for alias management

7.1.2 Field Documentation

7.1.2.1 command

```
char* Alias::command
```

7.1.2.2 name

```
char* Alias::name
```

The documentation for this struct was generated from the following file:

- include/typedef.h

7.2 BackgroundProcess Struct Reference

```
#include <typedef.h>
```

Collaboration diagram for BackgroundProcess:

Data Fields

- int count
- Jobs processes [MAX_BG_PROCESSES]

7.2.1 Detailed Description

Structure to manage background processes

7.2.2 Field Documentation

7.2.2.1 count

```
int BackgroundProcess::count
```

7.2.2.2 processes

```
Jobs BackgroundProcess::processes[MAX_BG_PROCESSES]
```

The documentation for this struct was generated from the following file:

- include/typedef.h

7.3 Command Struct Reference

```
#include <typedef.h>
```

Data Fields

- char ∗ command
- size_t length
- char ∗∗ args
- int arg_count
- char ∗ input_redirect
- char ∗ output_redirect
- char ∗ heredoc_delimiter
- bool append_output

7.3.1 Detailed Description

Structure to hold a single command and its details

7.3.2 Field Documentation

7.3.2.1 append_output

```
bool Command::append_output
```

7.3.2.2 arg_count

```
int Command::arg_count
```

7.3.2.3 args

```
char** Command::args
```

7.3.2.4 command

```
char* Command::command
```

7.3.2.5 heredoc_delimiter

```
char* Command::heredoc_delimiter
```

7.3.2.6 input_redirect

```
char* Command::input_redirect
```

7.3.2.7 length

```
size_t Command::length
```

7.3.2.8 output_redirect

```
char* Command::output_redirect
```

The documentation for this struct was generated from the following file:

- include/typedef.h

7.4 Commands Struct Reference

```
#include <typedef.h>
```

Collaboration diagram for Commands:

Data Fields

- Command commands [MAX_COMMANDS]
- char ∗ operators [MAX_COMMANDS]
- int command_count

7.4.1 Detailed Description

Structure to hold multiple commands and their operators

7.4.2 Field Documentation

7.4.2.1 command_count

```
int Commands::command_count
```

7.4.2.2 commands

```
Command Commands::commands[MAX_COMMANDS]
```

7.4.2.3 operators

```
char* Commands::operators[MAX_COMMANDS]
```

The documentation for this struct was generated from the following file:

- include/typedef.h

7.5 EnvVar Struct Reference

```
#include <typedef.h>
```

Collaboration diagram for EnvVar:

Data Fields

- char ∗ name
- char ∗ value
- struct EnvVar ∗ next

7.5.1 Field Documentation

7.5.1.1 name

```
char* EnvVar::name
```

7.5.1.2 next

```
struct EnvVar* EnvVar::next
```

7.5.1.3 value

```
char* EnvVar::value
```

The documentation for this struct was generated from the following file:

- include/typedef.h

7.6 Jobs Struct Reference

```
#include <typedef.h>
```

Data Fields

- pid_t pid
- char ∗ command

7.6.1 Detailed Description

Structure to hold background job details

7.6.2 Field Documentation

7.6.2.1 command

```
char* Jobs::command
```

7.6.2.2 pid

```
pid_t Jobs::pid
```

The documentation for this struct was generated from the following file:

- include/typedef.h

Chapter 8

File Documentation

8.1 include/aliases.h File Reference

```
#include "typedef.h"
#include <limits.h>
```

Include dependency graph for aliases.h: This graph shows which files directly or indirectly include this file:

Macros

- #define MAX_ALIASES 20

Functions

- void handle_alias (char ∗∗args)

  Handles the alias command and its various functionalities.
- void handle_unalias (char ∗∗args)

  Handles the unalias command and its various functionalities.
- void add_alias (Alias new_alias)

  Adds a new alias to the alias list.
- void remove_alias (const char ∗name)

  Removes an alias from the alias list from its name.
- int is_alias (const char ∗name)

  Checks if a given name corresponds to an existing alias.
- void list_aliases ()

  Lists all currently defined aliases. Used when the alias command is called without arguments.
- void free_aliases ()

  Frees all allocated memory for aliases and resets the alias count.
- char ∗ get_alias_command (const char ∗name)

  Retrieves the command associated with a given alias name.

8.1.1 Macro Definition Documentation

8.1.1.1 MAX_ALIASES

```
#define MAX_ALIASES 20
```

8.1.2 Function Documentation

8.1.2.1 add_alias()

```
void add_alias (
             Alias new_alias)
```

Adds a new alias to the alias list.

Parameters

new_alias	The Alias struct containing the name and command of the new alias.

8.1.2.2 free_aliases()

```
void free_aliases ()
```

Frees all allocated memory for aliases and resets the alias count.

8.1.2.3 get_alias_command()

```
char * get_alias_command (
             const char * name)
```

Retrieves the command associated with a given alias name.

Parameters

name	The name of the alias.

Returns

The command string if the alias exists, NULL otherwise.

8.1.2.4 handle_alias()

```
void handle_alias (
             char ** args)
```

Handles the alias command and its various functionalities.

Parameters

args	The different arguments passed to the alias command.

8.1.2.5 handle_unalias()

```
void handle_unalias (
             char ** args)
```

Handles the unalias command and its various functionalities.

Parameters

args	The different arguments passed to the unalias command (an alias name or -a).

8.1.2.6 is_alias()

```
int is_alias (
             const char * name)
```

Checks if a given name corresponds to an existing alias.

Parameters

name	The name to check.

Returns

EXIT_SUCCESS if the name is an alias, EXIT_FAILURE otherwise.

8.1.2.7 list_aliases()

```
void list_aliases ()
```

Lists all currently defined aliases. Used when the alias command is called without arguments.

8.1.2.8 remove_alias()

```
void remove_alias (
             const char * name)
```

Removes an alias from the alias list from its name.

Parameters

name	The name of the alias to remove.

8.2 aliases.h

Go to the documentation of this file.

```
#ifndef ALIASES_H
#define ALIASES_H

#include "typedef.h"
#include <limits.h>

#define MAX_ALIASES 20

void handle_alias(char **args);

void handle_unalias(char **args);

void add_alias(Alias new_alias);

void remove_alias(const char *name);

int is_alias(const char *name);

void list_aliases();

void free_aliases();

char* get_alias_command(const char *name);

#endif // ALIASES_H
```

8.3 include/batch.h File Reference

```
#include "typedef.h"
#include "internal_commands.h"
#include "parser.h"
#include "executor.h"
```

Include dependency graph for batch.h: This graph shows which files directly or indirectly include this file:

Macros

- #define MAX_ARGUMENTS 3

Functions

- int handle_arguments (int argc, const char *argv[ ])

  Handles command-line arguments for batch mode. Possible options are:

8.3.1 Macro Definition Documentation

8.3.1.1 MAX_ARGUMENTS

```
#define MAX_ARGUMENTS 3
```

8.3.2 Function Documentation

8.3.2.1 handle_arguments()

```
int handle_arguments (
              int argc,
              const char * argv[ ])
```

Handles command-line arguments for batch mode. Possible options are:

- -c, –command <command>: Execute the specified command and exit.
- –help: Display help message and exit.

Parameters

argc	Argument count.
argv	Argument vector.

Returns

Returns EXIT_SUCCESS or EXIT_FAILURE.

8.4 batch.h

Go to the documentation of this file.

```
#ifndef BATCH_H
#define BATCH_H

#include "typedef.h"
#include "internal_commands.h"
#include "parser.h"
#include "executor.h"

#define MAX_ARGUMENTS 3

int handle_arguments(int argc, const char *argv[]);

#endif // BATCH_H
```

8.5 include/env.h File Reference

```
#include "typedef.h"
```

Include dependency graph for env.h: This graph shows which files directly or indirectly include this file:

Functions

- void set_env_var (const char ∗name, const char ∗value)
- void unset_env_var (const char ∗name)
- char ∗ get_env_var (const char ∗name)
- void print_env_vars ()

8.5.1 Function Documentation

8.5.1.1 get_env_var()

```
char * get_env_var (
            const char * name)
```

8.5.1.2 print_env_vars()

```
void print_env_vars ()
```

8.5.1.3 set_env_var()

```
void set_env_var (
            const char * name,
            const char * value)
```

8.5.1.4 unset_env_var()

```
void unset_env_var (
            const char * name)
```

8.6 env.h

Go to the documentation of this file.

```
#ifndef ENV_H
#define ENV_H

#include "typedef.h"

void set_env_var(const char* name, const char* value);
void unset_env_var(const char* name);

char* get_env_var(const char* name);
void print_env_vars();

#endif // ENV_H
```

8.7 include/executor.h File Reference

```
#include "typedef.h"
#include "internal_commands.h"
#include "aliases.h"
#include "history.h"
```

Include dependency graph for executor.h: This graph shows which files directly or indirectly include this file:

Functions

- int execute_command (Command ∗command, bool is_background)

    Executes a single command, checking for internal commands first.
- void execute_commands (Commands ∗commands, BackgroundProcess ∗bg_processes)

    Executes a series of commands with their associated operators.
- int create_child_process (Command ∗command, bool is_background)

    Creates a child process to execute a command.
- void check_bg_processes (BackgroundProcess ∗bg_processes)

    Checks the status of background processes and cleans up finished ones.
- int execute_pipes (Commands ∗commands, int start_index, int end_index)

    Executes a series of piped commands.
- int setup_redirections (Command ∗command)

    Sets up input and output redirections for a command.
- int handle_heredoc (Command ∗command)

    Handles here-document (heredoc) input for a command.

8.7.1 Function Documentation

8.7.1.1 check_bg_processes()

```
void check_bg_processes (
             BackgroundProcess * bg_processes)
```

Checks the status of background processes and cleans up finished ones.

Parameters

bg_processes	Pointer to the BackgroundProcess struct managing background jobs.

8.7.1.2 create_child_process()

```
int create_child_process (
             Command * command,
             bool is_background)
```

Creates a child process to execute a command.

Parameters

command	Pointer to the Command struct to execute.
is_background	Boolean indicating if the command should run in the background.

Returns

Exit status of the command execution.

8.7.1.3 execute_command()

```
int execute_command (
            Command * command,
            bool is_background)
```

Executes a single command, checking for internal commands first.

Parameters

command	Pointer to the Command struct to execute.
is_background	Boolean indicating if the command should run in the background.

Returns

Exit status of the command execution.

8.7.1.4 execute_commands()

```
void execute_commands (
            Commands * commands,
            BackgroundProcess * bg_processes)
```

Executes a series of commands with their associated operators.

Parameters

commands	Pointer to the Commands struct containing commands and operators.

8.7.1.5 execute_pipes()

```
int execute_pipes (
            Commands * commands,
            int start_index,
            int end_index)
```

Executes a series of piped commands.

Parameters

commands	Pointer to the Commands struct containing commands and operators.
start_index	Index of the first command in the pipe sequence.
end_index	Index of the last command in the pipe sequence.

Returns

Exit status of the last command in the pipe sequence.

8.7.1.6 handle_heredoc()

```
int handle_heredoc (
            Command * command)
```

Handles here-document (heredoc) input for a command.

Parameters

command	Pointer to the Command struct containing the heredoc delimiter.

Returns

0 on success, -1 on failure.

8.7.1.7 setup_redirections()

```
int setup_redirections (
            Command * command)
```

Sets up input and output redirections for a command.

Parameters

command	Pointer to the Command struct to set up redirections for.

Returns

0 on success, -1 on failure.

8.8 executor.h

Go to the documentation of this file.

```
#ifndef EXECUTOR_H
#define EXECUTOR_H

#include "typedef.h"
#include "internal_commands.h"
#include "aliases.h"
#include "history.h"

int execute_command(Command* command, bool is_background);

void execute_commands(Commands* commands, BackgroundProcess* bg_processes);

int create_child_process(Command* command, bool is_background);

void check_bg_processes(BackgroundProcess* bg_processes);

int execute_pipes(Commands* commands, int start_index, int end_index);

int setup_redirections(Command* command);

int handle_heredoc(Command* command);

#endif // EXECUTOR_H
```

8.9 include/history.h File Reference

`#include "typedef.h"`

Include dependency graph for history.h: This graph shows which files directly or indirectly include this file:

Functions

- void handle_history (char ∗∗args)

  Built-in history command handler. Handles displaying the command history to the user.
- void add_to_history (char ∗command_line)

  Handles writing inputted commands in history file (∼/minishell_logs/command_history).

8.9.1 Function Documentation

8.9.1.1 add_to_history()

```
void add_to_history (
            char * command_line)
```

Handles writing inputted commands in history file (∼/minishell_logs/command_history).

Parameters

command_line	The entire command line that needs to be added in the history file.

8.9.1.2 handle_history()

```
void handle_history (
            char ** args)
```

Built-in history command handler. Handles displaying the command history to the user.

Parameters

args	voided

8.10 history.h

Go to the documentation of this file.

```
#ifndef HISTORY_H
#define HISTORY_H

#include "typedef.h"

void handle_history(char** args);

void add_to_history(char* command_line);

#endif // HISTORY_H
```

8.11 include/internal_commands.h File Reference

```
#include "typedef.h"
#include "aliases.h"
```

Include dependency graph for internal_commands.h: This graph shows which files directly or indirectly include this file:

Functions

- void handle_cd (char **args)

  Built-in cd command handler.
- void handle_pwd (char **args)

  Built-in pwd command handler.
- void handle_echo (char **args)

  Built-in echo command handler.
- void handle_exit (Commands *commands)

  Built-in exit command handler. Exits the shell after freeing necessary resources.
- void free_if_needed (void *to_free)

  Frees memory if the pointer is not NULL.
- void free_commands (Commands *commands)

  Frees all allocated memory in the Commands struct.

Variables

- char * work_dir

8.11.1 Function Documentation

8.11.1.1 free_commands()

```
void free_commands (
            Commands * commands)
```

Frees all allocated memory in the Commands struct.

Parameters

commands	Pointer to the Commands struct to free.

8.11.1.2 free_if_needed()

```
void free_if_needed (
             void * to_free)
```

Frees memory if the pointer is not NULL.

Parameters

to_free	Pointer to the memory to free.

8.11.1.3 handle_cd()

```
void handle_cd (
             char ** args)
```

Built-in cd command handler.

Parameters

args	Command arguments, handles correctly changing directories, and specific cases like "cd ∼".

8.11.1.4 handle_echo()

```
void handle_echo (
             char ** args)
```

Built-in echo command handler.

Parameters

args	Command arguments to echo back to the user.

8.11.1.5 handle_exit()

```
void handle_exit (
             Commands * commands)
```

Built-in exit command handler. Exits the shell after freeing necessary resources.

Parameters

commands	Pointer to the Commands struct containing commands and operators.

8.11.1.6 handle_pwd()

```
void handle_pwd (
             char ** args)
```

Built-in pwd command handler.

Parameters

args	voided

8.11.2 Variable Documentation

8.11.2.1 work_dir

```
char* work_dir  [extern]
```

The path to the current workdir

8.12 internal_commands.h

Go to the documentation of this file.

```
#ifndef INTERNAL_COMMANDS_H
#define INTERNAL_COMMANDS_H

#include "typedef.h"
#include "aliases.h"

extern char *work_dir;

void handle_cd(char** args);

void handle_pwd(char** args);

void handle_echo(char** args);

void handle_exit(Commands* commands);

void free_if_needed(void* to_free);

void free_commands(Commands* commands);

#endif // INTERNAL_COMMANDS_H
```

8.13 include/parser.h File Reference

```
#include "typedef.h"
#include "internal_commands.h"
```

Include dependency graph for parser.h: This graph shows which files directly or indirectly include this file:

Functions

- int detect_operator (const char ∗p, const char ∗∗op_found)

  Detects if the current position in the input line matches any known operator.
- char ∗ trim_whitespace (const char ∗line)

  Trims leading and trailing whitespace from a given line.
- void split_line (const char ∗line, Commands ∗commands)

  Splits the input line into commands and operators, populating the Commands struct.
- void parse_command (Command ∗command)

  Parses a command string into its arguments and populates the Command struct.

8.13.1 Function Documentation

8.13.1.1 detect_operator()

```
int detect_operator (
             const char * p,
             const char ** op_found)
```

Detects if the current position in the input line matches any known operator.

Parameters

p	Pointer to the current position in the input line.
op_found	Pointer to store the found operator string.

Returns

Length of the detected operator, or 0 if none found.

8.13.1.2 parse_command()

```
void parse_command (
             Command * command)
```

Parses a command string into its arguments and populates the Command struct.

Parameters

command	Pointer to the Command struct to populate.

8.13.1.3 split_line()

```
void split_line (
            const char * line,
            Commands * commands)
```

Splits the input line into commands and operators, populating the Commands struct.

Parameters

line	Pointer to the input line.
commands	Pointer to the Commands struct to populate.

8.13.1.4 trim_whitespace()

```
char * trim_whitespace (
            const char * line)
```

Trims leading and trailing whitespace from a given line.

Parameters

line	Pointer to the input line.

Returns

Newly allocated string with trimmed whitespace.

8.14 parser.h

Go to the documentation of this file.

```
#ifndef PARSER_H
#define PARSER_H

#include "typedef.h"
#include "internal_commands.h"

int detect_operator(const char* p, const char** op_found);

char* trim_whitespace(const char* line);

void split_line(const char* line, Commands *commands);

void parse_command(Command* command);

#endif // PARSER_H
```

8.15 include/typedef.h File Reference

```
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
```

Include dependency graph for typedef.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct Command
- struct Commands
- struct Jobs
- struct BackgroundProcess
- struct Alias
- struct EnvVar

Macros

- #define MAX_COMMANDS 3
- #define MAX_BG_PROCESSES 10
- #define GREEN "\033[1;32m"
- #define BLUE "\033[1;34m"
- #define COLOR_RESET "\033[0m"

Typedefs

- typedef struct Command Command
- typedef struct Commands Commands
- typedef struct Jobs Jobs
- typedef struct BackgroundProcess BackgroundProcess
- typedef struct Alias Alias
- typedef struct EnvVar EnvVar

8.15.1 Macro Definition Documentation

8.15.1.1 BLUE

```
#define BLUE "\033[1;34m"
```

8.15.1.2 COLOR_RESET

```
#define COLOR_RESET "\033[0m"
```

8.15.1.3 GREEN

```
#define GREEN "\033[1;32m"
```

8.15.1.4 MAX_BG_PROCESSES

```
#define MAX_BG_PROCESSES 10
```

8.15.1.5 MAX_COMMANDS

```
#define MAX_COMMANDS 3
```

8.15.2 Typedef Documentation

8.15.2.1 Alias

```
typedef struct Alias Alias
```

Structure for alias management

8.15.2.2 BackgroundProcess

```
typedef struct BackgroundProcess BackgroundProcess
```

Structure to manage background processes

8.15.2.3 Command

```
typedef struct Command Command
```

Structure to hold a single command and its details

8.15.2.4 Commands

```
typedef struct Commands Commands
```

Structure to hold multiple commands and their operators

8.15.2.5 EnvVar

```
typedef struct EnvVar EnvVar
```

8.15.2.6 Jobs

```
typedef struct Jobs Jobs
```

Structure to hold background job details

8.16 typedef.h

Go to the documentation of this file.

```
#ifndef TYPEDEF_H
#define TYPEDEF_H


#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>

#define MAX_COMMANDS      3
#define MAX_BG_PROCESSES  10

#define GREEN         "\033[1;32m"
#define BLUE          "\033[1;34m"
#define COLOR_RESET   "\033[0m"

typedef struct Command {
  char*   command;                    // Original command string
  size_t  length;                     // Length of the command string
  char**  args;                       // Argument list
  int     arg_count;                  // Number of arguments related to the command
  char*   input_redirect;             // file for <
  char*   output_redirect;            // file for > or »
  char*   heredoc_delimiter;          // delimiter for «
  bool    append_output;              // true for », false for >
} Command;

typedef struct Commands {
  Command commands[MAX_COMMANDS];     // Array of parsed commands
  char*   operators[MAX_COMMANDS];    // Operators between commands
  int     command_count;              // Number of commands parsed
} Commands;

typedef struct Jobs {
  pid_t   pid;                        // Process ID
  char*   command;                    // Command string
} Jobs;

typedef struct BackgroundProcess {
  int     count;                      // Number of background processes
  Jobs    processes[MAX_BG_PROCESSES];// Array of background processes
} BackgroundProcess;

typedef struct Alias {
  char*   name;                       // Alias name
  char*   command;                    // Command string
} Alias;

typedef struct EnvVar {
    char* name;                       // Environment variable name
    char* value;                      // Environment variable value
    struct EnvVar* next;              // Pointer to the next variable in the list
} EnvVar;

#endif // TYPEDEF_H
```

8.17 README.md File Reference

8.18 src/aliases.c File Reference

Implementation of alias handling for the minishell.

```
#include "../include/aliases.h"
```

Include dependency graph for aliases.c:

8.19 src/batch.c File Reference

Handles batch mode execution for the minishell.

```
#include "../include/batch.h"
```

Include dependency graph for batch.c:

Functions

- int handle_arguments (int argc, const char ∗argv[ ])

  Handles command-line arguments for batch mode. Possible options are:

Variables

- const char ∗ minishell_options [ ] = {"-c", "--command", "--help", NULL}
- Commands batch_command
- BackgroundProcess bg_processes

8.19.1 Detailed Description

Handles batch mode execution for the minishell.

Author

BRENNER Quentin, NEAGELY Jeannot

Date

2025-2026

8.19.2 Function Documentation

8.19.2.1 handle_arguments()

```
int handle_arguments (
            int argc,
            const char * argv[])
```

Handles command-line arguments for batch mode. Possible options are:

- -c, –command <command>: Execute the specified command and exit.
- –help: Display help message and exit.

Parameters

argc	Argument count.
argv	Argument vector.

Returns

Returns EXIT_SUCCESS or EXIT_FAILURE.

8.19.3 Variable Documentation

8.19.3.1 batch_command

```
Commands batch_command
```

The command to be run

8.19.3.2 bg_processes

```
BackgroundProcess bg_processes  [extern]
```

8.19.3.3 minishell_options

```
const char* minishell_options[] = {"-c", "--command", "--help", NULL}
```

List of possible options to use for batch mode

8.20 src/env.c File Reference

Implementation of environment variable management for the minishell.

```
#include "../include/env.h"
```

Include dependency graph for env.c:

Functions

- void set_env_var (const char ∗name, const char ∗value)
- void unset_env_var (const char ∗name)
- char ∗ get_env_var (const char ∗name)
- void print_env_vars ()

Variables

- EnvVar ∗ env_vars_head = NULL
- EnvVar ∗ env_vars_tail = NULL

8.20.1 Detailed Description

Implementation of environment variable management for the minishell.

Author

BRENNER Quentin, NEAGELY Jeannot

Date

2025-2026

8.20.2 Function Documentation

8.20.2.1 get_env_var()

```
char * get_env_var (
             const char * name)
```

8.20.2.2 print_env_vars()

```
void print_env_vars ()
```

8.20.2.3 set_env_var()

```
void set_env_var (
             const char * name,
             const char * value)
```

8.20.2.4 unset_env_var()

```
void unset_env_var (
             const char * name)
```

8.20.3 Variable Documentation

8.20.3.1 env_vars_head

```
EnvVar* env_vars_head = NULL
```

8.20.3.2 env_vars_tail

```
EnvVar* env_vars_tail = NULL
```

8.21 src/executor.c File Reference

Implementation of command execution for the minishell.

`#include "../include/executor.h"`
Include dependency graph for executor.c:

Functions

- int create_child_process (Command ∗command, bool is_background)

    Creates a child process to execute a command.
- void execute_commands (Commands ∗commands, BackgroundProcess ∗bg_processes)

    Executes a series of commands with their associated operators.
- int execute_command (Command ∗command, bool is_background)

    Executes a single command, checking for internal commands first.
- void check_bg_processes (BackgroundProcess ∗bg_processes)

    Checks the status of background processes and cleans up finished ones.
- int execute_pipes (Commands ∗commands, int start_index, int end_index)

    Executes a series of piped commands.
- int setup_redirections (Command ∗command)

    Sets up input and output redirections for a command.
- int handle_heredoc (Command ∗command)

    Handles here-document (heredoc) input for a command.

Variables

- const char ∗ internal_cmds_list [ ] = {"cd", "pwd", "echo", "history", "alias", "unalias", NULL}
- void(∗ internal_cmds [ ])(char ∗∗) = {&handle_cd, &handle_pwd, &handle_echo, &handle_history, &handle_alias, &handle_unalias, NULL}

8.21.1 Detailed Description

Implementation of command execution for the minishell.

Author

BRENNER Quentin, NEAGELY Jeannot

Date

2025-2026

8.21.2 Function Documentation

8.21.2.1 check_bg_processes()

```
void check_bg_processes (
            BackgroundProcess * bg_processes)
```

Checks the status of background processes and cleans up finished ones.

Parameters

bg_processes	Pointer to the BackgroundProcess struct managing background jobs.

8.21.2.2 create_child_process()

```
int create_child_process (
            Command * command,
            bool is_background)
```

Creates a child process to execute a command.

Parameters

command	Pointer to the Command struct to execute.
is_background	Boolean indicating if the command should run in the background.

Returns

Exit status of the command execution.

8.21.2.3 execute_command()

```
int execute_command (
            Command * command,
            bool is_background)
```

Executes a single command, checking for internal commands first.

Parameters

command	Pointer to the Command struct to execute.
is_background	Boolean indicating if the command should run in the background.

Returns

Exit status of the command execution.

8.21.2.4 execute_commands()

```
void execute_commands (
            Commands * commands,
            BackgroundProcess * bg_processes)
```

Executes a series of commands with their associated operators.

Parameters

commands	Pointer to the Commands struct containing commands and operators.

8.21.2.5 execute_pipes()

```
int execute_pipes (
            Commands * commands,
            int start_index,
            int end_index)
```

Executes a series of piped commands.

Parameters

commands	Pointer to the Commands struct containing commands and operators.
start_index	Index of the first command in the pipe sequence.
end_index	Index of the last command in the pipe sequence.

Returns

Exit status of the last command in the pipe sequence.

8.21.2.6 handle_heredoc()

```
int handle_heredoc (
            Command * command)
```

Handles here-document (heredoc) input for a command.

Parameters

command	Pointer to the Command struct containing the heredoc delimiter.

Returns

0 on success, -1 on failure.

8.21.2.7 setup_redirections()

```
int setup_redirections (
            Command * command)
```

Sets up input and output redirections for a command.

Parameters

command	Pointer to the Command struct to set up redirections for.

Returns

0 on success, -1 on failure.

8.21.3 Variable Documentation

8.21.3.1 internal_cmds

```
void(* internal_cmds[])(char **) (
            char ** ) = {&handle_cd, &handle_pwd, &handle_echo, &handle_history, &handle_alias,
&handle_unalias, NULL}
```

List of pointers to internal command functions

8.21.3.2 internal_cmds_list

```
const char* internal_cmds_list[] = {"cd", "pwd", "echo", "history", "alias", "unalias", NULL}
```

List of internal commands

8.22 src/history.c File Reference

Implementation of command history handling for the minishell.

```
#include "../include/history.h"
```

Include dependency graph for history.c:

Functions

- void handle_history (char **args)

  Built-in history command handler. Handles displaying the command history to the user.
- void add_to_history (char *command_line)

  Handles writing inputted commands in history file (~/minishell_logs/command_history).

8.22.1 Detailed Description

Implementation of command history handling for the minishell.

Author

BRENNER Quentin, NEAGELY Jeannot

Date

2025-2026

8.22.2 Function Documentation

8.22.2.1 add_to_history()

```
void add_to_history (
             char * command_line)
```

Handles writing inputted commands in history file (∼/minishell_logs/command_history).

Parameters

command_line	The entire command line that needs to be added in the history file.

8.22.2.2 handle_history()

```
void handle_history (
             char ** args)
```

Built-in history command handler. Handles displaying the command history to the user.

Parameters

args	voided

8.23 src/internal_commands.c File Reference

Implementation of internal commands for the minishell.

```
#include "../include/internal_commands.h"
```

Include dependency graph for internal_commands.c:

Functions

- void handle_exit (Commands *commands)
  Built-in exit command handler. Exits the shell after freeing necessary resources.
- void handle_pwd (char **args)
  Built-in pwd command handler.
- void handle_cd (char **args)
  Built-in cd command handler.
- void handle_echo (char **args)
  Built-in echo command handler.
- void free_if_needed (void *to_free)
  Frees memory if the pointer is not NULL.
- void free_commands (Commands *commands)
  Frees all allocated memory in the Commands struct.

Variables

- char * work_dir

8.23.1 Detailed Description

Implementation of internal commands for the minishell.

Author

BRENNER Quentin, NEAGELY Jeannot

Date

2025-2026

8.23.2 Function Documentation

8.23.2.1 free_commands()

```
void free_commands (
            Commands * commands)
```

Frees all allocated memory in the Commands struct.

Parameters

commands	Pointer to the Commands struct to free.

8.23.2.2 free_if_needed()

```
void free_if_needed (
             void * to_free)
```

Frees memory if the pointer is not NULL.

Parameters

to_free	Pointer to the memory to free.

8.23.2.3 handle_cd()

```
void handle_cd (
             char ** args)
```

Built-in cd command handler.

Parameters

args	Command arguments, handles correctly changing directories, and specific cases like "cd ∼".

8.23.2.4 handle_echo()

```
void handle_echo (
             char ** args)
```

Built-in echo command handler.

Parameters

args	Command arguments to echo back to the user.

8.23.2.5 handle_exit()

```
void handle_exit (
             Commands * commands)
```

Built-in exit command handler. Exits the shell after freeing necessary resources.

Parameters

commands	Pointer to the Commands struct containing commands and operators.

8.23.2.6 handle_pwd()

```
void handle_pwd (
             char ** args)
```

Built-in pwd command handler.

Parameters

args	voided

8.23.3 Variable Documentation

8.23.3.1 work_dir

```
char* work_dir
```

The path to the current workdir

8.24 src/main.c File Reference

Main entry point for the minishell application.

```
#include "../include/typedef.h"
#include "../include/parser.h"
#include "../include/internal_commands.h"
#include "../include/executor.h"
#include "../include/batch.h"
```

Include dependency graph for main.c:

Functions

- void write_prompt (bool is_compact)
- int main (int argc, const char ∗argv[ ])

Variables

- Commands parsed_commands
- BackgroundProcess bg_processes = {.count = 0}

8.24.1 Detailed Description

Main entry point for the minishell application.

Author

BRENNER Quentin, NEAGELY Jeannot

Date

2025-2026

8.24.2 Function Documentation

8.24.2.1 main()

```
int main (
            int argc,
            const char * argv[ ])
```

8.24.2.2 write_prompt()

```
void write_prompt (
            bool is_compact)
```

8.24.3 Variable Documentation

8.24.3.1 bg_processes

```
BackgroundProcess bg_processes = {.count = 0}
```

8.24.3.2 parsed_commands

```
Commands parsed_commands
```

8.25 src/parser.c File Reference

Implementation of the command line parser for the minishell.

```
#include "../include/parser.h"
```

Include dependency graph for parser.c:

Functions

- int detect_operator (const char ∗p, const char ∗∗op_found)

  Detects if the current position in the input line matches any known operator.
- char ∗ trim_whitespace (const char ∗line)

  Trims leading and trailing whitespace from a given line.
- void split_line (const char ∗line, Commands ∗commands)

  Splits the input line into commands and operators, populating the Commands struct.
- void parse_command (Command ∗command)

  Parses a command string into its arguments and populates the Command struct.

Variables

- const char ∗ OPERATORS [ ] = {"&&", "||", "&", "|", ";", NULL}
- const char ∗ REDIRECTORS [ ] = {"<<", ">>", "<", ">", NULL}

8.25.1 Detailed Description

Implementation of the command line parser for the minishell.

Author

BRENNER Quentin, NEAGELY Jeannot

Date

2025-2026

8.25.2 Function Documentation

8.25.2.1 detect_operator()

```
int detect_operator (
             const char * p,
             const char ** op_found)
```

Detects if the current position in the input line matches any known operator.

Parameters

p	Pointer to the current position in the input line.
op_found	Pointer to store the found operator string.

Returns

Length of the detected operator, or 0 if none found.

8.25.2.2 parse_command()

```
void parse_command (
             Command * command)
```

Parses a command string into its arguments and populates the Command struct.

Parameters

command	Pointer to the Command struct to populate.

8.25.2.3 split_line()

```
void split_line (
            const char * line,
            Commands * commands)
```

Splits the input line into commands and operators, populating the Commands struct.

Parameters

line	Pointer to the input line.
commands	Pointer to the Commands struct to populate.

8.25.2.4 trim_whitespace()

```
char * trim_whitespace (
            const char * line)
```

Trims leading and trailing whitespace from a given line.

Parameters

line	Pointer to the input line.

Returns

Newly allocated string with trimmed whitespace.

8.25.3 Variable Documentation

8.25.3.1 OPERATORS

```
const char* OPERATORS[] = {"&&", "||", "&", "|", ";", NULL}
```

8.25.3.2 REDIRECTORS

```
const char* REDIRECTORS[] = {"<<", ">>", "<", ">", NULL}
```

Index