```
NAME        = minishell
B_NAME        = minishell_bonus

CC            = cc
LDFLAGS        = -L./libft -lft -lreadline -lncurses
CFLAGS        = -Wall -Wextra -Werror -g
CPPFLAGS    = -I./libft/include -I./include

SRCDIR = src
OBJDIR = build

ifeq ($(DEBUG), 1)
CFLAGS += -DDEBUG
endif

FILES = builtins/builtins            \
            builtins/cd                \
            builtins/echo            \
            builtins/env            \
            builtins/exit            \
            builtins/export            \
            builtins/pwd            \
            builtins/unset            \
            utils/dptr                \
            utils/ft_getenv            \
            utils/get_full_path        \
            utils/ft_list            \
            utils/print_error        \
            utils/tty                \
            signals                    \
            wait_children            \
            get_argv                \
             pipeline_control        \
            redirection_handler        \
            tokenizer                \
            heredoc                    \
            main                    \
            expand_env                \
            expand_str                \
#
M_FILES =    mandatory/execute_complex_command    \
            mandatory/execute_simple_command    \
            mandatory/check_syntax                \
            mandatory/expand_line                \
            mandatory/expand_tokens                \
            mandatory/cut_slice                    \
#
B_FILES =    bonus/execute_complex_command    \
            bonus/execute_simple_command    \
            bonus/check_syntax                \
            bonus/expand_line                \
            bonus/expand_tokens                \
            bonus/expand_wildcard            \
             bonus/flow_control                \
            bonus/subshell                    \
            bonus/cut_slice                    \

#

OBJECTS = $(FILES:%=$(OBJDIR)/%.o)
M_OBJECTS = $(M_FILES:%=$(OBJDIR)/%.o)
B_OBJECTS = $(B_FILES:%=$(OBJDIR)/%.o)

all: libft $(NAME)

bonus: libft $(B_NAME)

libft:
    @$(MAKE) -C libft

$(NAME): $(M_OBJECTS) $(OBJECTS) libft/libft.a
    $(CC) $(M_OBJECTS) $(OBJECTS) $(LDFLAGS) -o $@

$(B_NAME): $(B_OBJECTS) $(OBJECTS) libft/libft.a
    $(CC) $(B_OBJECTS) $(OBJECTS) $(LDFLAGS) -o $@

$(OBJDIR)/%.o: $(SRCDIR)/%.c
    @mkdir -p $(dir $@)
    $(CC) $(CFLAGS) $(CPPFLAGS) -c $< -o $@
```

```
clean:
#	$(MAKE) -C libft clean
	rm -f $(M_OBJECTS) $(OBJECTS) $(B_OBJECTS)

fclean: clean
	rm -f $(NAME) $(B_NAME)

re: fclean all

.PHONY: all libft clean fclean re
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   minishell.h                                        :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2024/12/17 09:15:08 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/03/21 12:35:02 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#ifndef MINISHELL_H
# define MINISHELL_H

# define __USE_XOPEN2K8
# define _GNU_SOURCE // for WUNTRACED

# include <libft.h>
# include <stddef.h>
# include <unistd.h>
# include <sys/wait.h>
# include <fcntl.h>
# include <errno.h>
# include <string.h>
# include <stdio.h>
# ifdef __linux__
#  include <linux/limits.h>
# else
#  include <limits.h>
# endif

# include <readline/readline.h>
# include <readline/history.h>
# include <sys/types.h>
# include <termios.h>
# include <signal.h>
# include <sys/ioctl.h>

# define PREFIX "minishell: "

enum	e_qouts
{
	SINGLE_QUOTE = 0x1,
	DOUBLE_QUOTE = 0x2
};

enum	e_is_pipe
{
	IS_NEXT_PIPE = 1,
	IS_PREV_PIPE = 2
};

typedef struct s_list
{
	char			*str;
	struct s_list	*next;
}	t_list;

typedef struct s_mini
{
	t_list	*tokens;
	t_list	*env;
	int		exit_status;
}	t_mini;

struct s_cmd
{
```

```
    char    **argv;
    char    **env;
    char    full_path[PATH_MAX];
    int        err;
};

extern volatile int    g_sig;

void    mini_clean(t_mini *mini);
void    exit_handler(t_mini *mini, int exit_status);

int        print_error(const char *file, int line);
int        print_file_error(const char *file, int line, const char *target);

/*  subshell  */
int        is_subshell(t_list *lst);
int        subshell_syntax(t_list *lst);
void    run_subshell(t_mini *mini);

/*  t_list  */
t_list    *lst_expand(t_list *lst, char **slices);
void    *lst_move2next(t_list **lst);
void    *lst_clean(t_list **lst);
char    **lst_2_argv(t_list **lst, int flcean);
char    **lst_2_dptr(t_list *lst);
void    lst_remove_one(t_list **lst, t_list *prev);

/*  tokenizer  */
t_list    *tokenizer(const char *s);
char    *cut_slice(char **s_r);

/*  expand  */
char    *expand_line(const char *s);
char    **expand_str(t_mini *mini, char *str);
char    *expand_env(t_mini *mini, char *str);
char    **expand_wildcard(char *pattern);
int        expand_tokens(t_mini *mini, t_list *lst);
t_list    *expand_tokens_2lst(t_mini *mini, const char *str);
char    *remove_qouts(char *str);

/*  execution  */
int        execute_line(t_mini *mini);
int        flow_control(t_mini *mini);
int        pipeline_control(t_mini *mini);
int        execute_complex_command(t_mini *mini, int in_fd,
            int pipefds[2], int pipe_mask);
int        execute_simple_command(t_mini *mini);
int        check_syntax(t_list *lst);

/*  Wait children  */
int        wait_children(pid_t victim);
int        wait_child_stop(pid_t victim);

/*  redirection handling  */
int        redirection_handler(t_mini *mini, int heredoc_fd);
int        heredoc_forever(t_mini *mini, t_list *lst);

/*  environment variables  */
t_list    *copy_env_variables(void);
char    *ft_getenv(t_list *env, const char *name);

/*  built-in commands  */
int        handle_builtin(t_mini *mini, char **argv, int _exit);
int        is_builtin(t_mini *mini, const char *cmd, int expand);
int        ft_cd(t_mini *mini, char **argv);
int        ft_echo(char **argv);
int        ft_pwd(char **argv);
int        ft_env(t_mini *mini, char **argv);
int        ft_exit(char **argv, int *_exit);
int        ft_test(t_mini *mini, char **argv);
int        ft_export(t_mini *mini, char **argv);
int        ft_unset(t_mini *mini, char **argv);

/* signal handling functions */
void    setup_signals(void);
void    setup_signals2(void);
void    reset_signals(void);

/*  utils functions  */
```

```
int        ft_ttyname_r(int fd, char *buf, size_t len);
int        restore_tty(char tty_path[PATH_MAX]);
char    **copy_dptr(char **dptr);
void    free_dptr(char **ptr);
int        get_full_path(t_list *env, char full_path[PATH_MAX], char *cmd);
char    *get_argv0(t_list *lst);
void    get_argv(t_list **lst);

#endif
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   wait_children.c                                    :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/08 17:47:06 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/21 17:28:32 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

#ifdef DEBUG

static int    get_process_status(int wstatus, pid_t pid)
{
    if (WIFEXITED(wstatus))
    {
        ft_fprintf(2, "%d: is exited by %d\n", pid, WEXITSTATUS(wstatus));
        return (WEXITSTATUS(wstatus));
    }
    else if (WIFSIGNALED(wstatus))
    {
        ft_fprintf(2, "%d: is signaled by %d\n", pid, WTERMSIG(wstatus));
        return (128 + WTERMSIG(wstatus));
    }
    else if (WIFSTOPPED(wstatus))
    {
        ft_fprintf(2, "%d: is stopped\n", pid);
        return (128 + SIGSTOP);
    }
    else if (WIFCONTINUED(wstatus))
    {
        ft_fprintf(2, "%d: is continued\n", pid);
        return (128 + SIGCONT);
    }
    return (1);
}

#else

static int    get_process_status(int wstatus, pid_t pid)
{
    (void)pid;
    if (WIFEXITED(wstatus))
        return (WEXITSTATUS(wstatus));
    else if (WIFSIGNALED(wstatus))
        return (128 + WTERMSIG(wstatus));
    else if (WIFSTOPPED(wstatus))
        return (128 + SIGSTOP);
    else if (WIFCONTINUED(wstatus))
        return (128 + SIGCONT);
    return (1);
}

#endif

int    wait_child_stop(pid_t victim)
{
    int    wstatus;

    if (waitpid(victim, &wstatus, WUNTRACED) == -1)
        return (-1);
    return (get_process_status(wstatus, victim));
}

int    wait_children(pid_t victim)
{
```

```
    int    wstatus;
    int    status;
    int    ret_status;
    int    child_pid;

    ret_status = 1;
    child_pid = waitpid(WAIT_ANY, &wstatus, WUNTRACED);
    while (child_pid != -1)
    {
        status = get_process_status(wstatus, child_pid);
        if (child_pid == victim)
            ret_status = status;
        child_pid = waitpid(WAIT_ANY, &wstatus, WUNTRACED);
    }
    return (ret_status);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   cut_slice.c                                        :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/27 20:36:57 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/27 21:18:02 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include <libft.h>
#include <stdlib.h>

char    *cut_slice(char **s_r)
{
    char    *start;
    int        nb_bracket;
    char    *s;

    s = *s_r;
    while (*s == ' ')
        s++;
    start = s;
    nb_bracket = 0;
    while (s && *s && (*s != ' ' || nb_bracket) && nb_bracket >= 0)
    {
        if (*s == '(')
            nb_bracket++;
        else if (*s == ')')
            nb_bracket--;
        else if (*s == '\'' || *s == '\"')
            s = ft_strchr(s + 1, *s);
        if (s)
            s++;
    }
    *s_r = s;
    if (s == NULL || nb_bracket != 0)
        return (NULL);
    return (start);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   execute_simple_command.c                           :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/03/21 21:59:26 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/27 20:52:52 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

int    execute_simple_command(t_mini *mini)
{
    struct s_cmd    cmd;

    if (is_subshell(mini->tokens))
        run_subshell(mini);
    if (expand_tokens(mini, mini->tokens) != 0)
```

```
        return (1);
    cmd.argv = lst_2_argv(&mini->tokens, 1);
    if (!cmd.argv)
        return (1);
    if (is_builtin(mini, cmd.argv[0], 0))
        handle_builtin(mini, cmd.argv, 1);
    cmd.err = get_full_path(mini->env, cmd.full_path, cmd.argv[0]);
    if (cmd.err == 0)
    {
        cmd.env = lst_2_dptr(mini->env);
        if (cmd.env)
        {
            execve(cmd.full_path, cmd.argv, cmd.env);
            print_file_error(__FILE__, __LINE__, cmd.full_path);
            free(cmd.env);
        }
        cmd.err = 1;
    }
    free_dptr(cmd.argv);
    return (cmd.err);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   expand_wildcard.c                                  :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/05 21:33:13 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/27 23:42:19 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"
#include <dirent.h>

static int    is_contain_wildcard(char *pattern)
{
    int    qout;

    qout = '\0';
    while (*pattern)
    {
        if ((*pattern == SINGLE_QUOTE || *pattern == DOUBLE_QUOTE)
            && (*pattern == qout || qout == '\0'))
        {
            if (qout)
                qout = '\0';
            else
                qout = *pattern;
        }
        else if (*pattern == '*' || *pattern == '?')
            return (1);
        pattern++;
    }
    return (0);
}

/*
 * This function is like playing a matching game!
 * It checks if a word matches a special pattern.
 * For example:
 * Pattern: "cat*" will match: "cat", "cats", "catfood"
 * Pattern: "?at" will match: "cat", "rat", "hat"
 * The * is like a magic star that matches anything!
 * The ? is like a surprise box that matches any letter!
 * example:
 * Pattern:    h e * o ?
 * String:     h e l l o !
 *
 * Step 1:     h = h  (exact match)
 * Step 2:     e = e  (exact match)
 * Step 3:     * matches 'l l'  (asterisk can match multiple chars)
 * Step 4:     o = o  (exact match)
 * Step 5:     ? matches '!'  (question mark matches any single char)
 * Result: MATCH
 *
 * This function is like playing a matching game!
 * It checks if a word matches a special pattern.
```

```
 * For example:
 * Pattern: "cat*" will match: "cat", "cats", "catfood"
 * Pattern: "?at" will match: "cat", "rat", "hat"
 * The * is like a magic star that matches anything!
 * The ? is like a surprise box that matches any letter!
 */
static int    match_pattern(const char *pattern, const char *str, char qout)
{
    while (*pattern && *str)
    {
        if ((*pattern == SINGLE_QUOTE || *pattern == DOUBLE_QUOTE)
            && (*pattern == qout || qout == '\0'))
        {
            if (qout)
                qout = '\0';
            else
                qout = *pattern;
            pattern++;
            continue ;
        }
        else if (*pattern == '*' && qout == '\0')
        {
            while (*pattern == '*')
                pattern++;
            if (!*pattern)
                return (1);
            while (*str)
            {
                if (match_pattern(pattern, str, qout))
                    return (1);
                str++;
            }
            return (match_pattern(pattern, str, qout));
        }
        else if (*pattern == '?' || *pattern == *str)
        {
            pattern++;
            str++;
            continue ;
        }
        return (0);
    }
    while (*pattern == '*')
        pattern++;
    return (*pattern == '\0' && *str == '\0');
}

/*
* This function is like adding a new toy to your toy box!
* It takes your old toy box (array) and makes a bigger one
* to fit one more toy (string) inside.
* Then it carefully moves all your old toys to the new box
* and adds the new toy at the end!
*/
static char    **add_to_array(char **arr, char *str, int *size)
{
    char    **new_arr;
    int        i;

    new_arr = malloc(sizeof(char *) * (*size + 2));
    if (!new_arr)
        return (NULL);
    i = 0;
    while (i < *size)
    {
        new_arr[i] = arr[i];
        i++;
    }
    new_arr[i] = ft_strdup(str);
    new_arr[i + 1] = NULL;
    *size += 1;
    free(arr);
    return (new_arr);
}

/*
* Imagine arranging your toys in alphabetical order!
* This function is like organizing your toys from A to Z.
* Just like when you line up your stuffed animals:
```

```
* First comes Bear, then Cat, then Dog, then Elephant!
*/
static void    sort_strings(char **arr, int size)
{
    char    *temp;
    int        i;
    int        j;

    i = 0;
    while (i < size - 1)
    {
        j = 0;
        while (j < size - i - 1)
        {
            if (ft_strcmp(arr[j], arr[j + 1]) > 0)
            {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
            j++;
        }
        i++;
    }
}

static char    **original_argument(char *pattern)
{
    char    **ret;

    ret = ft_calloc(2, sizeof(char *));
    if (!ret)
        return (NULL);
    ret[0] = ft_strdup(pattern);
    if (!ret[0])
    {
        free(ret);
        return (NULL);
    }
    remove_qouts(ret[0]);
    return (ret);
}

/*
* This is like a treasure hunt in your room!
* When you give it a pattern (like *.txt),
* it looks through all files in the folder
* and finds the ones that match your pattern!
* Just like finding all blue toys in your room!
*/
char    **expand_wildcard(char *pattern)
{
    DIR               *dir;
    struct dirent    *entry;
    char            **files;
    int               size;

    if (is_contain_wildcard(pattern) == 0)
        return (original_argument(pattern));
    dir = opendir(".");
    if (!dir)
        return (NULL);
    files = ft_calloc(1, sizeof(char *));
    if (!files)
    {
        closedir(dir);
        return (NULL);
    }
    size = 0;
    entry = readdir(dir);
    while (entry)
    {
        if (entry->d_name[0] == '.' && pattern[0] != '.')
            ;
        else if (match_pattern(pattern, entry->d_name, '\0'))
        {
            files = add_to_array(files, entry->d_name, &size);
            if (!files)
            {
```

```
                closedir(dir);
                return (NULL);
            }
        }
        entry = readdir(dir);
    }
    closedir(dir);
    if (size > 0)
        sort_strings(files, size);
    if (size == 0)
    {
        free(files);
        return (original_argument(pattern));
    }
    return (files);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   check_syntax.c                                     :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/03 20:44:58 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/24 12:11:09 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static int    operation_type(char *str)
{
    if (ft_strcmp(str, ">>") == 0
        || ft_strcmp(str, ">") == 0
        || ft_strcmp(str, "<<") == 0
        || ft_strcmp(str, "<") == 0)
        return (1);
    else if (ft_strcmp(str, "|") == 0
        || ft_strcmp(str, "||") == 0
        || ft_strcmp(str, "&&") == 0)
        return (2);
    else
        return (0);
}

static int    check_redirection_syntax(t_list *lst)
{
    if (!lst->next || !lst->next->str)
    {
        ft_fprintf(2, PREFIX"syntax error near unexpected token `newline'\n");
        return (0);
    }
    if (operation_type(lst->next->str) != 0)
    {
        ft_fprintf(2, PREFIX"syntax error near unexpected token `%s'\n",
            lst->next->str);
        return (0);
    }
    return (1);
}

static int    check_pipe_or_logical_syntax(t_list *lst, t_list *prev_tokens)
{
    if (!lst->next || !lst->next->str || !prev_tokens || !prev_tokens->str)
    {
        ft_fprintf(2, PREFIX"syntax error near unexpected token `%s'\n",
            lst->str);
        return (0);
    }
    if (ft_strcmp(lst->next->str, "|") == 0
        || ft_strcmp(lst->next->str, "||") == 0
        || ft_strcmp(lst->next->str, "&&") == 0)
    {
        ft_fprintf(2, PREFIX"syntax error near unexpected token `%s'\n",
            lst->next->str);
        return (0);
    }
    return (1);
}
```

```
int    check_syntax(t_list *lst)
{
    t_list    *prev_tokens;
    int        op_type;

    prev_tokens = NULL;
    while (lst && lst->str)
    {
        op_type = operation_type(lst->str);
        if (op_type == 1)
        {
            if (!check_redirection_syntax(lst))
                return (0);
        }
        else if (op_type == 2)
        {
            if (!check_pipe_or_logical_syntax(lst, prev_tokens))
                return (0);
        }
        prev_tokens = lst;
        lst = lst->next;
    }
    return (1);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   execute_complex_command.c                          :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/09 23:37:40 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/24 12:12:16 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static int    pipex_handler(int pipe_mask, int in_fd, int pipefds[2])
{
    if (pipe_mask & IS_PREV_PIPE)
    {
        if (dup2(in_fd, STDIN_FILENO) == -1)
        {
            close(in_fd);
            perror(PREFIX"pipex dup2 to STDIN");
            return (-1);
        }
        close(in_fd);
    }
    if (pipe_mask & IS_NEXT_PIPE)
    {
        if (dup2(pipefds[1], STDOUT_FILENO) == -1)
        {
            close(pipefds[1]);
            perror(PREFIX"pipex dup2 to STDOUT");
            return (-1);
        }
        close(pipefds[1]);
    }
    return (0);
}

static int    stop_process(void)
{
    if (g_sig != 0)
        return (-1);
    if (kill(getpid(), SIGSTOP) == -1)
        return (-1);
    if (g_sig != 0)
        return (-1);
    reset_signals();
    return (0);
}

static int    handle_file_descriptor(t_mini *mini, int in_fd,
                                     int pipefds[2], int pipe_mask)
{
```

```
    int    heredoc_fd;
    int    err;

    heredoc_fd = heredoc_forever(mini, mini->tokens);
    if (heredoc_fd < 0)
    {
        if (pipe_mask & IS_PREV_PIPE)
            close(in_fd);
        if (pipe_mask & IS_NEXT_PIPE)
            close(pipefds[1]);
        return (-1);
    }
    if (pipex_handler(pipe_mask, in_fd, pipefds) != 0)
        return (-1);
    if (is_subshell(mini->tokens) && subshell_syntax(mini->tokens) == 0)
    {
        ft_fprintf(2, PREFIX"syntax error near unexpected token `('\n");
        exit_handler(mini, 2);
    }
    if (stop_process() != 0)
        return (-1);
    err = redirection_handler(mini, heredoc_fd);
    if (heredoc_fd > 0)
        close(heredoc_fd);
    if (err != 0)
        return (err);
    return (0);
}

int    execute_complex_command(t_mini *mini, int in_fd,
                            int pipefds[2], int pipe_mask)
{
    int    pid;

    pid = fork();
    if (pid == 0)
    {
        if (pipe_mask & IS_NEXT_PIPE)
            close(pipefds[0]);
        g_sig = 0;
        if (handle_file_descriptor(mini, in_fd, pipefds, pipe_mask) != 0)
            exit_handler(mini, EXIT_FAILURE);
        get_argv(&mini->tokens);
        if (!mini->tokens)
            exit_handler(mini, EXIT_FAILURE);
        execute_simple_command(mini);
        exit_handler(mini, EXIT_FAILURE);
    }
    if (pipe_mask & IS_NEXT_PIPE)
        close(pipefds[1]);
    return (pid);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   expand_tokens.c                                    :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/21 14:16:31 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/25 01:44:30 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static void    tok_remove_qouts(t_list *lst)
{
    while (lst && lst->str)
    {
        remove_qouts(lst->str);
        lst = lst->next;
    }
}

int    expand_tokens(t_mini *mini, t_list *lst)
{
    char    **slices;
    t_list    *end;
```

```
    t_list    *cur;

    cur = lst;
    while (cur && cur->str)
    {
        slices = expand_str(mini, cur->str);
        if (!slices)
            return (-1);
        end = lst_expand(cur, slices);
        free(slices);
        if (!end)
            return (-1);
        end = end->next;
        while (cur && cur->str && cur != end)
        {
            slices = expand_wildcard(cur->str);
            if (!slices)
                return (-1);
            cur = lst_expand(cur, slices);
            free(slices);
            if (!cur)
                return (-1);
            cur = cur->next;
        }
    }
    tok_remove_qouts(lst);
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   subshell.c                                         :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/24 12:12:22 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/24 12:12:34 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

void    run_subshell(t_mini *mini)
{
    char    *argv0;
    size_t    line_len;
    char    *line;

    argv0 = get_argv0(mini->tokens);
    ++argv0;
    line_len = ft_strlen(argv0);
    if (argv0[line_len - 1] != ')')
        exit_handler(mini, 1);
    argv0[line_len - 1] = '\0';
    line = ft_strdup(argv0);
    if (!line)
        exit_handler(mini, 1);
    lst_clean(&mini->tokens);
    mini->tokens = tokenizer(line);
    free(line);
    if (!mini->tokens)
        exit_handler(mini, 1);
    if (flow_control(mini) != 0)
    {
        perror("flow_control");
        exit_handler(mini, 1);
    }
    exit_handler(mini, 0);
}

int    is_subshell(t_list *lst)
{
    char    *argv0;

    argv0 = get_argv0(lst);
    if (!argv0)
        return (0);
    if (*argv0 == '(')
        return (1);
```

```
    return (0);
}

int    subshell_syntax(t_list *lst)
{
    if (lst->next || lst->next->str)
        return (0);
    return (1);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   flow_control.c                                     :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/13 02:19:13 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/24 13:51:02 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static void    set_null_token(t_list *lst, int *op)
{
    *op = 0;
    while (lst && lst->str)
    {
        if (ft_strcmp(lst->str, "&&") == 0
            || ft_strcmp(lst->str, "||") == 0)
        {
            if (ft_strcmp(lst->str, "&&") == 0)
                *op = 1;
            else if (ft_strcmp(lst->str, "||") == 0)
                *op = 2;
            free(lst->str);
            lst->str = NULL;
            return ;
        }
        lst = lst->next;
    }
}

int    flow_control(t_mini *mini)
{
    int            op;
    int            test;

    test = 1;
    while (mini->tokens && mini->tokens->str)
    {
        set_null_token(mini->tokens, &op);
        if (test)
        {
            if (pipeline_control(mini) == -1)
                return (-1);
            if (op == 1)
                test = !mini->exit_status;
            else if (op == 2)
                test = mini->exit_status;
        }
        if (op == 0)
            break ;
        if (op == 2)
            test = !test;
        lst_move2next(&mini->tokens);
    }
    return (0);
}

int    execute_line(t_mini *mini)
{
    return (flow_control(mini));
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   expand_line.c                                      :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
```

```
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/24 12:13:24 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/28 05:54:41 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static size_t    operation_len(const char *s)
{
    if (ft_strncmp(s, "&&", 2) == 0 || ft_strncmp(s, "||", 2) == 0
        || ft_strncmp(s, ">>", 2) == 0 || ft_strncmp(s, "<<", 2) == 0)
        return (2);
    else if (*s == '|' || *s == '<' || *s == '>')
        return (1);
    return (0);
}

static char    *get_line(int fd)
{
    size_t    nbytes;
    char    *line;
    ssize_t    lines_read;

    nbytes = 0;
    if (ioctl(fd, FIONREAD, &nbytes) == -1)
    {
        print_error(__FILE__, __LINE__);
        return (NULL);
    }
    line = malloc(sizeof(char) * (nbytes + 1));
    if (line == NULL)
    {
        print_error(__FILE__, __LINE__);
        return (NULL);
    }
    lines_read = read(fd, line, nbytes);
    if (lines_read == -1)
    {
        free(line);
        print_error(__FILE__, __LINE__);
        return (NULL);
    }
    line[lines_read] = 0;
    return (line);
}

char    *expand_line(const char *s)
{
    int        pipe_fds[2];
    size_t    op_len;
    char    *line;

    if (pipe(pipe_fds) == -1)
    {
        print_error(__FILE__, __LINE__);
        return (NULL);
    }
    while (*s)
    {
        op_len = operation_len(s);
        if (op_len)
        {
            write(pipe_fds[1], " ", 1);
            write(pipe_fds[1], s, op_len);
            write(pipe_fds[1], " ", 1);
            s += op_len;
            continue ;
        }
        else
            write(pipe_fds[1], s, 1);
        s++;
    }
    close(pipe_fds[1]);
    line = get_line(pipe_fds[0]);
    close(pipe_fds[0]);
    return (line);
}
```

```
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   pipeline_control.c                                 :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/04 23:32:02 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/26 18:54:41 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static void    set_null_token(t_list *lst, int *pipe_mask)
{
    *pipe_mask <<= 1;
    while (lst && lst->str)
    {
        if (ft_strcmp(lst->str, "|") == 0)
        {
            *pipe_mask |= 1;
            free(lst->str);
            lst->str = NULL;
            return ;
        }
        lst = lst->next;
    }
}

static int    run_builtin_command(t_mini *mini)
{
    int        heredoc_fd;
    char    **argv;

    heredoc_fd = heredoc_forever(mini, mini->tokens);
    if (heredoc_fd < 0)
        return (print_error(__FILE__, __LINE__));
    if (redirection_handler(mini, heredoc_fd) != 0)
    {
        if (heredoc_fd > 0)
            close(heredoc_fd);
        return (print_error(__FILE__, __LINE__));
    }
    if (heredoc_fd > 0)
        close(heredoc_fd);
    get_argv(&mini->tokens);
    if (expand_tokens(mini, mini->tokens) != 0)
        return (print_error(__FILE__, __LINE__));
    argv = lst_2_argv(&mini->tokens, 0);
    if (!argv)
        return (print_error(__FILE__, __LINE__));
    mini->exit_status = handle_builtin(mini, argv, 0);
    return (0);
}

static pid_t    execute_command(t_mini *mini, int in_fd,    \
                                int pipefds[2], int pipe_mask)
{
    pid_t    victim;

    victim = execute_complex_command(mini, in_fd, pipefds, pipe_mask);
    if (victim == -1)
    {
        if (pipe_mask & IS_NEXT_PIPE)
            close(pipefds[0]);
        return (-1);
    }
    mini->exit_status = wait_child_stop(victim);
    if (mini->exit_status != 128 + SIGSTOP)
    {
        if (pipe_mask & IS_NEXT_PIPE)
            close(pipefds[0]);
        return (-2);
    }
    return (victim);
}

/*
```

```
    if builtin run return 0 and stored exit status in mini.exit_status
    if syscall fail return -1
    return child_pid
    valgrind --leak-check=full --show-leak-kinds=all
            --trace-children=yes --track-fds=yes
            --suppressions=readline_curses.supp ./minishell
    << 1 cat > 1 | << 2 cat > 2| << 3 cat > 3
*/
static int    pipeline_control_iter(t_mini *mini, int in_fd, int pipe_mask)
{
    pid_t    victim[2];
    int        pipefds[2];

    set_null_token(mini->tokens, &pipe_mask);
    if (pipe_mask == 0 && is_builtin(mini, get_argv0(mini->tokens), 1))
        return (run_builtin_command(mini));
    if ((pipe_mask & IS_NEXT_PIPE) && pipe(pipefds) == -1)
        return (-1);
    victim[0] = execute_command(mini, in_fd, pipefds, pipe_mask);
    if (victim[0] < 0)
        return (victim[0]);
    if ((pipe_mask & IS_NEXT_PIPE) && mini->tokens && mini->tokens->str)
    {
        lst_move2next(&mini->tokens);
        victim[1] = pipeline_control_iter(mini, pipefds[0], pipe_mask);
        if (victim[1] == -1)
            kill(victim[0], SIGKILL);
        else
            kill(victim[0], SIGCONT);
        close(pipefds[0]);
        return (victim[1]);
    }
    kill(victim[0], SIGCONT);
    return (victim[0]);
}

int    pipeline_control(t_mini *mini)
{
    pid_t    victim;

    victim = pipeline_control_iter(mini, 0, 0);
    if (victim < 0)
    {
        wait_children(victim);
        if (victim == -1)
        {
            ft_fprintf(2, PREFIX"%s:%d: %s\n", __FILE__,
                __LINE__, strerror(errno));
            return (-1);
        }
        return (0);
    }
    if (victim > 0)
        mini->exit_status = wait_children(victim);
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   expand_str.c                                       :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/23 21:28:44 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/26 18:50:55 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static char    **ft_split(char *s, int i)
{
    char    *start;
    char    **tokens;
    char    *token;

    while (*s == ' ')
        s++;
    start = s;
```

```
    while (*s && *s != ' ')
    {
        if (*s == SINGLE_QUOTE || *s == DOUBLE_QUOTE)
            s = ft_strchr(s + 1, *s);
        s++;
    }
    if (start == s && !*s)
        return (ft_calloc(i + 1, sizeof(char *)));
    token = ft_substr(start, 0, s - start);
    if (!token)
        return (NULL);
    tokens = ft_split(s + !!*s, i + 1);
    if (tokens)
        tokens[i] = token;
    else
        free(token);
    return (tokens);
}

void    replace_qouts(char *s)
{
    char    *next;

    while (*s)
    {
        if (*s == '\'' || *s == '\"')
        {
            next = ft_strchr(s + 1, *s);
            if (!next)
                break ;
            if (*s == '\'')
            {
                *s = SINGLE_QUOTE;
                *next = SINGLE_QUOTE;
            }
            else if (*s == '\"')
            {
                *s = DOUBLE_QUOTE;
                *next = DOUBLE_QUOTE;
            }
            s = next;
        }
        s++;
    }
}

char    *remove_qouts(char *str)
{
    char    *src;
    char    *dst;
    char    qout;

    dst = str;
    src = str;
    qout = '\0';
    while (*src)
    {
        if ((*src == SINGLE_QUOTE || *src == DOUBLE_QUOTE)
            && (qout == *src || qout == '\0'))
        {
            if (!qout)
                qout = *src;
            else
                qout = '\0';
            src++;
        }
        else
            *dst++ = *src++;
    }
    *dst = '\0';
    return (str);
}

char    **expand_str(t_mini *mini, char *str)
{
    char    *expanded_str;
    char    **slices;

    replace_qouts(str);
```

```
    expanded_str = expand_env(mini, str);
    if (!expanded_str)
        return (NULL);
    slices = ft_split(expanded_str, 0);
    free(expanded_str);
    return (slices);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   redirection_handler.c                              :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/07 08:12:35 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/26 23:53:24 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static int    is_ambiguous(t_mini *mini, char **filename_r)
{
    t_list    *lst;

    lst = expand_tokens_2lst(mini, *filename_r);
    if (!lst)
        return (print_error(__FILE__, __LINE__));
    if (!lst->str || (lst->next && lst->next->str))
    {
        lst_clean(&lst);
        ft_fprintf(2, PREFIX"%s: ambiguous redirect\n", *filename_r);
        return (1);
    }
    *filename_r = lst->str;
    lst->str = NULL;
    lst_clean(&lst);
    return (0);
}

static int    in_redirection(t_mini *mini, char *token)
{
    int        fd;
    char    *filename;

    filename = token;
    if (is_ambiguous(mini, &filename) != 0)
        return (-1);
    fd = open(filename, O_RDONLY);
    if (fd == -1)
    {
        print_file_error(__FILE__, __LINE__, filename);
        free(filename);
        return (-1);
    }
    free(filename);
    if (dup2(fd, STDIN_FILENO))
    {
        print_error(__FILE__, __LINE__);
        close(fd);
        return (-1);
    }
    close(fd);
    return (0);
}

static int    out_append(t_mini *mini, char *token)
{
    int        fd;
    char    *filename;

    filename = token;
    if (is_ambiguous(mini, &filename) != 0)
        return (-1);
    fd = open(filename, O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (fd == -1)
    {
        print_file_error(__FILE__, __LINE__, filename);
        free(filename);
```

```
        return (-1);
    }
    free(filename);
    if (dup2(fd, STDOUT_FILENO) == -1)
    {
        print_error(__FILE__, __LINE__);
        close(fd);
        return (-1);
    }
    close(fd);
    return (0);
}

static int    out_redirection(t_mini *mini, char *token)
{
    int        fd;
    char    *filename;

    filename = token;
    if (is_ambiguous(mini, &filename) != 0)
        return (-1);
    fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1)
    {
        print_file_error(__FILE__, __LINE__, filename);
        free(filename);
        return (-1);
    }
    free(filename);
    if (dup2(fd, STDOUT_FILENO) == -1)
    {
        print_error(__FILE__, __LINE__);
        close(fd);
        return (-1);
    }
    close(fd);
    return (0);
}

int    redirection_handler(t_mini *mini, int heredoc_fd)
{
    int        err;
    t_list    *lst;

    err = 0;
    lst = mini->tokens;
    while (lst && lst->str)
    {
        if (ft_strcmp(lst->str, "<<") == 0)
            err = dup2(heredoc_fd, STDIN_FILENO);
        else if (ft_strcmp(lst->str, "<") == 0)
            err = in_redirection(mini, lst->next->str);
        else if (ft_strcmp(lst->str, ">>") == 0)
            err = out_append(mini, lst->next->str);
        else if (ft_strcmp(lst->str, ">") == 0)
            err = out_redirection(mini, lst->next->str);
        else
        {
            lst = lst->next;
            continue ;
        }
        if (err != 0)
            return (err);
        lst = lst->next->next;
    }
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   cut_slice.c                                        :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/27 20:36:57 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/27 21:17:51 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */
```

```c
#include <libft.h>
#include <stdlib.h>

char    *cut_slice(char **s_r)
{
    char    *start;
    char    *s;

    s = *s_r;
    while (*s == ' ')
        s++;
    start = s;
    while (s && *s && *s != ' ')
    {
        if (*s == '\'' || *s == '\"')
            s = ft_strchr(s + 1, *s);
        if (s)
            s++;
    }
    *s_r = s;
    if (s == NULL)
        return (NULL);
    return (start);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   execute_simple_command.c                           :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/03/21 21:59:26 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/27 21:17:17 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

int    execute_simple_command(t_mini *mini)
{
    struct s_cmd    cmd;

    if (expand_tokens(mini, mini->tokens) != 0)
        return (1);
    cmd.argv = lst_2_argv(&mini->tokens, 1);
    if (!cmd.argv)
        return (1);
    if (is_builtin(mini, cmd.argv[0], 0))
        handle_builtin(mini, cmd.argv, 1);
    cmd.err = get_full_path(mini->env, cmd.full_path, cmd.argv[0]);
    if (cmd.err == 0)
    {
        cmd.env = lst_2_dptr(mini->env);
        if (cmd.env)
        {
            execve(cmd.full_path, cmd.argv, cmd.env);
            print_file_error(__FILE__, __LINE__, cmd.full_path);
            free(cmd.env);
        }
        cmd.err = 1;
    }
    free_dptr(cmd.argv);
    return (cmd.err);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   check_syntax.c                                     :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/03 20:44:58 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/24 12:16:02 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static int    operation_type(char *str)
```

```
{
    if (ft_strcmp(str, ">>") == 0
        || ft_strcmp(str, ">") == 0
        || ft_strcmp(str, "<<") == 0
        || ft_strcmp(str, "<") == 0)
        return (1);
    else if (ft_strcmp(str, "|") == 0)
        return (2);
    else
        return (0);
}

static int    check_redirection_syntax(t_list *lst)
{
    if (!lst->next || !lst->next->str)
    {
        ft_fprintf(2, PREFIX"syntax error near unexpected token `newline'\n");
        return (0);
    }
    if (operation_type(lst->next->str) != 0)
    {
        ft_fprintf(2, PREFIX"syntax error near unexpected token `%s'\n",
            lst->next->str);
        return (0);
    }
    return (1);
}

static int    check_pipe_or_logical_syntax(t_list *lst, t_list *prev_tokens)
{
    if (!lst->next || !lst->next->str || !prev_tokens || !prev_tokens->str)
    {
        ft_fprintf(2, PREFIX"syntax error near unexpected token `%s'\n",
            lst->str);
        return (0);
    }
    if (ft_strcmp(lst->next->str, "|") == 0)
    {
        ft_fprintf(2, PREFIX"syntax error near unexpected token `%s'\n",
            lst->next->str);
        return (0);
    }
    return (1);
}

int    check_syntax(t_list *lst)
{
    t_list    *prev_tokens;
    int        op_type;

    prev_tokens = NULL;
    while (lst && lst->str)
    {
        op_type = operation_type(lst->str);
        if (op_type == 1)
        {
            if (!check_redirection_syntax(lst))
                return (0);
        }
        else if (op_type == 2)
        {
            if (!check_pipe_or_logical_syntax(lst, prev_tokens))
                return (0);
        }
        prev_tokens = lst;
        lst = lst->next;
    }
    return (1);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   execute_complex_command.c                          :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/09 23:37:40 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/27 20:48:34 by yaltayeh         ###   ########.fr       */
/*                                                                            */
```

```
/* ************************************************************************** */

#include "minishell.h"

static int	pipex_handler(int pipe_mask, int in_fd, int pipefds[2])
{
	if (pipe_mask & IS_PREV_PIPE)
	{
		if (dup2(in_fd, STDIN_FILENO) == -1)
		{
			close(in_fd);
			perror(PREFIX"pipex dup2 to STDIN");
			return (-1);
		}
		close(in_fd);
	}
	if (pipe_mask & IS_NEXT_PIPE)
	{
		if (dup2(pipefds[1], STDOUT_FILENO) == -1)
		{
			close(pipefds[1]);
			perror(PREFIX"pipex dup2 to STDOUT");
			return (-1);
		}
		close(pipefds[1]);
	}
	return (0);
}

static int	stop_process(void)
{
	if (g_sig != 0)
		return (-1);
	if (kill(getpid(), SIGSTOP) == -1)
		return (-1);
	if (g_sig != 0)
		return (-1);
	reset_signals();
	return (0);
}

static int	handle_file_descriptor(t_mini *mini, int in_fd,
					int pipefds[2], int pipe_mask)
{
	int	heredoc_fd;
	int	err;

	heredoc_fd = heredoc_forever(mini, mini->tokens);
	if (heredoc_fd < 0)
	{
		print_error(__FILE__, __LINE__);
		if (pipe_mask & IS_PREV_PIPE)
			close(in_fd);
		if (pipe_mask & IS_NEXT_PIPE)
			close(pipefds[1]);
		return (-1);
	}
	if (pipex_handler(pipe_mask, in_fd, pipefds) != 0)
		return (-1);
	if (stop_process() != 0)
		return (-1);
	err = redirection_handler(mini, heredoc_fd);
	if (heredoc_fd > 0)
		close(heredoc_fd);
	if (err != 0)
		return (err);
	return (0);
}

int	execute_complex_command(t_mini *mini, int in_fd,
				int pipefds[2], int pipe_mask)
{
	int	pid;
	int	err;

	pid = fork();
	if (pid == 0)
	{
		if (pipe_mask & IS_NEXT_PIPE)
```

```
            close(pipefds[0]);
        g_sig = 0;
        if (handle_file_descriptor(mini, in_fd, pipefds, pipe_mask) != 0)
            exit_handler(mini, EXIT_FAILURE);
        get_argv(&mini->tokens);
        if (!mini->tokens)
            exit_handler(mini, EXIT_FAILURE);
        err = execute_simple_command(mini);
        print_error(__FILE__, __LINE__);
        exit_handler(mini, err);
    }
    if (pipe_mask & IS_NEXT_PIPE)
        close(pipefds[1]);
    if (pipe_mask & IS_PREV_PIPE)
        close(in_fd);
    return (pid);
}

int    execute_line(t_mini *mini)
{
    return (pipeline_control(mini));
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   expand_tokens.c                                    :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/04 21:29:22 by mkurkar          #+#    #+#             */
/*   Updated: 2025/04/25 01:44:05 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

int    expand_tokens(t_mini *mini, t_list *lst)
{
    char    **slices;
    t_list    *cur;

    cur = lst;
    while (cur && cur->str)
    {
        slices = expand_str(mini, cur->str);
        if (!slices)
            return (-1);
        cur = lst_expand(cur, slices);
        free(slices);
        if (!cur)
            return (-1);
        cur = cur->next;
    }
    cur = lst;
    while (cur && cur->str)
    {
        remove_qouts(cur->str);
        cur = cur->next;
    }
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   expand_line.c                                      :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/23 19:16:58 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/23 19:17:42 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static size_t    operation_len(const char *s)
{
    if (ft_strncmp(s, ">>", 2) == 0 || ft_strncmp(s, "<<", 2) == 0)
        return (2);
```

```
    else if (*s == '|' || *s == '<' || *s == '>')
        return (1);
    return (0);
}

char    *get_line(int fd)
{
    size_t    nbytes;
    char    *line;
    ssize_t    lines_read;

    nbytes = 0;
    if (ioctl(fd, FIONREAD, &nbytes) == -1)
        return (NULL);
    line = malloc(sizeof(char) * (nbytes + 1));
    if (!line)
        return (NULL);
    lines_read = read(fd, line, nbytes);
    if (lines_read == -1)
        return (free(line), NULL);
    line[lines_read] = 0;
    return (line);
}

char    *expand_line(const char *s)
{
    int        pipe_fds[2];
    size_t    op_len;
    char    *line;

    if (pipe(pipe_fds) == -1)
        return (NULL);
    while (*s)
    {
        op_len = operation_len(s);
        if (op_len)
        {
            write(pipe_fds[1], " ", 1);
            write(pipe_fds[1], s, op_len);
            write(pipe_fds[1], " ", 1);
            s += op_len;
            continue ;
        }
        else
            write(pipe_fds[1], s, 1);
        s++;
    }
    close(pipe_fds[1]);
    line = get_line(pipe_fds[0]);
    close(pipe_fds[0]);
    return (line);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   print_error.c                                      :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+#+   +#+           */
/*   Created: 2025/04/26 17:23:00 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/26 23:51:49 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

int    print_error(const char *file, int line)
{
    ft_fprintf(2, PREFIX"%s:%d: %s\n", file, line, strerror(errno));
    return (-1);
}

int    print_file_error(const char *file, int line, const char *target)
{
    ft_fprintf(2, PREFIX"%s:%d: %s: %s\n", file, line, target, strerror(errno));
    return (-1);
}
/* ************************************************************************** */
/*                                                                            */
```

```
/*                                                        :::      ::::::::   */
/*   dptr.c                                             :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/03/24 13:05:17 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/28 00:25:37 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

void    free_dptr(char **ptr)
{
    char    **_ptr;

    if (!ptr)
        return ;
    _ptr = ptr;
    while (*_ptr)
        free(*_ptr++);
    free(ptr);
}

char    **copy_dptr(char **dptr)
{
    char    **ptr;
    char    **dst;

    ptr = dptr;
    while (*ptr)
        ptr++;
    dst = ft_calloc(ptr - dptr + 1, sizeof(char *));
    if (!dst)
        return (NULL);
    ptr = dst;
    while (*dptr)
    {
        *ptr = ft_strdup(*dptr++);
        if (*ptr == NULL)
        {
            free_dptr(dst);
            return (NULL);
        }
        ptr++;
    }
    return (dst);
}

char    **lst_2_dptr(t_list *lst)
{
    char        **dptr;
    static int    i;
    int            _i;

    _i = i++;
    if (!lst || !lst->str)
    {
        i = 0;
        return (ft_calloc(_i + 1, sizeof(char *)));
    }
    dptr = lst_2_dptr(lst->next);
    if (dptr)
        dptr[_i] = lst->str;
    return (dptr);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   ft_getenv.c                                        :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2024/11/16 07:25:20 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/24 12:28:58 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"
```

```
char    *ft_getenv(t_list *env, const char *name)
{
    size_t    name_len;

    if (!name)
        return (NULL);
    name_len = ft_strlen(name);
    while (env && env->str)
    {
        if (ft_strncmp(env->str, name, name_len) == 0
            && env->str[name_len] == '=')
            return (ft_strdup(env->str + name_len + 1));
        env = env->next;
    }
    return (NULL);
}

t_list    *copy_env_variables(void)
{
    t_list        *lst;
    extern char    **environ;
    static int    i;
    int            _i;

    _i = i++;
    if (!environ[_i])
    {
        i = 0;
        return (ft_calloc(1, sizeof(t_list)));
    }
    lst = malloc(sizeof(t_list));
    if (!lst)
        return (NULL);
    lst->str = ft_strdup(environ[_i]);
    if (lst->str)
    {
        lst->next = copy_env_variables();
        if (lst->next)
            return (lst);
        free(lst->str);
    }
    free(lst);
    return (NULL);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   get_full_path.c                                    :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2024/11/17 21:33:51 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/27 21:25:22 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static int    search_command_path(t_list *env,
                            char full_path[PATH_MAX], char *cmd)
{
    char    *path_env;
    char    *path;

    path_env = ft_getenv(env, "PATH");
    if (!path_env && errno == ENOMEM)
        return (-1);
    if (!path_env)
        return (1);
    path = ft_strtok(path_env, ":");
    while (path)
    {
        if (ft_snprintf(full_path, PATH_MAX, "%s/%s", path, cmd) < PATH_MAX
            && access(full_path, X_OK) == 0)
        {
            free(path_env);
            return (0);
        }
```

```
        path = ft_strtok(NULL, ":");
    }
    free(path_env);
    return (1);
}

int    get_full_path(t_list *env, char full_path[PATH_MAX], char *cmd)
{
    int    err;

    if (ft_strlcpy(full_path, cmd, PATH_MAX) >= PATH_MAX)
    {
        errno = ENAMETOOLONG;
        return (1);
    }
    if (ft_strncmp(cmd, "/", 1) == 0
        || ft_strncmp(cmd, "./", 2) == 0
        || ft_strncmp(cmd, "../", 3) == 0)
    {
        if (access(full_path, X_OK) == 0)
            return (0);
        return (1);
    }
    err = search_command_path(env, full_path, cmd);
    if (err == -1)
        return (1);
    if (err == 1)
    {
        ft_fprintf(2, PREFIX"%s: command not found\n", cmd);
        return (127);
    }
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   tty.c                                              :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+#+   +#+           */
/*   Created: 2025/04/28 00:16:38 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/28 05:54:14 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include <unistd.h>
#include <fcntl.h>
#include <libft.h>

int    ft_ttyname_r(int fd, char *buf, size_t len)
{
    char    *tty_path;

    tty_path = ttyname(fd);
    if (!tty_path)
        return (-1);
    if (ft_strlcpy(buf, tty_path, len) >= len)
    {
        free(tty_path);
        return (-1);
    }
    free(tty_path);
    return (0);
}

int    restore_tty(char tty_path[PATH_MAX])
{
    int    fd;
    int    err;

    err = 0;
    if (!isatty(STDERR_FILENO))
    {
        fd = open(tty_path, O_RDONLY);
        if (fd == -1)
            return (-1);
        if (fd != STDERR_FILENO)
            err = dup2(fd, STDERR_FILENO);
        close(fd);
```

```
    }
    if (err == 0 && !isatty(STDOUT_FILENO))
    {
        fd = open(tty_path, O_WRONLY);
        if (fd == -1)
            return (-1);
        if (fd != STDOUT_FILENO)
            err = dup2(fd, STDOUT_FILENO);
        close(fd);
    }
    return (err);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   ft_list.c                                          :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/07 17:59:57 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/28 05:50:40 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

void    lst_remove_one(t_list **lst, t_list *prev)
{
    t_list    *cur;

    if (!*lst)
        return ;
    cur = *lst;
    if (prev)
        prev->next = cur->next;
    *lst = cur->next;
    if (cur->str)
        free(cur->str);
    free(cur);
}

void    *lst_clean(t_list **lst)
{
    t_list    *next;

    while (*lst)
    {
        free((*lst)->str);
        next = (*lst)->next;
        free(*lst);
        *lst = next;
    }
    return (NULL);
}

void    *lst_move2next(t_list **lst)
{
    t_list    *next;

    while (*lst && (*lst)->str)
    {
        free((*lst)->str);
        next = (*lst)->next;
        free(*lst);
        *lst = next;
    }
    if (*lst)
    {
        next = (*lst)->next;
        free(*lst);
        *lst = next;
    }
    return (*lst);
}

t_list    *lst_expand(t_list *lst, char **slices)
{
    t_list    *next;
```

```
    if (*slices)
        free(lst->str);
    next = lst->next;
    lst->next = NULL;
    while (*slices)
    {
        lst->str = *slices;
        if (!*++slices)
            break ;
        if (!lst->next)
            lst->next = ft_calloc(1, sizeof(t_list));
        if (!lst->next)
        {
            lst->next = next;
            while (*slices)
                free(*slices++);
            return (NULL);
        }
        lst = lst->next;
    }
    lst->next = next;
    return (lst);
}

t_list    *expand_tokens_2lst(t_mini *mini, const char *str)
{
    t_list    *lst;

    lst = ft_calloc(1, sizeof(*lst));
    if (!lst)
        return (NULL);
    lst->str = ft_strdup(str);
    if (!lst->str)
    {
        lst_clean(&lst);
        return (NULL);
    }
    if (expand_tokens(mini, lst) != 0)
    {
        lst_clean(&lst);
        return (NULL);
    }
    return (lst);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   builtins.c                                         :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/04 20:57:28 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/26 18:59:03 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static int    check_builtin(const char *cmd)
{
    if (!cmd)
        return (0);
    if (ft_strcmp(cmd, "cd") == 0
        || ft_strcmp(cmd, "exit") == 0
        || ft_strcmp(cmd, "export") == 0
        || ft_strcmp(cmd, "unset") == 0
        || ft_strcmp(cmd, "echo") == 0
        || ft_strcmp(cmd, "pwd") == 0
        || ft_strcmp(cmd, "env") == 0)
    {
        return (1);
    }
    return (0);
}

/*
* Regular built-ins can run in child process (output only)
* Shell built-ins must run in parent process (modify shell state)
*/
```

```
int    is_builtin(t_mini *mini, const char *cmd, int expand)
{
    int        test;
    t_list    *lst;

    if (!cmd)
        return (0);
    if (expand)
    {
        lst = expand_tokens_2lst(mini, cmd);
        if (!lst)
            return (-1);
        if (!lst->str)
        {
            lst_clean(&lst);
            return (0);
        }
        test = check_builtin(lst->str);
        lst_clean(&lst);
    }
    else
        test = check_builtin(cmd);
    return (test);
}

int    handle_builtin(t_mini *mini, char **argv, int _exit)
{
    int        err;

    err = 1;
    if (ft_strcmp(*argv, "cd") == 0)
        err = ft_cd(mini, argv);
    else if (ft_strcmp(*argv, "exit") == 0)
        err = ft_exit(argv, &_exit);
    else if (ft_strcmp(*argv, "export") == 0)
        err = ft_export(mini, argv);
    else if (ft_strcmp(*argv, "unset") == 0)
        err = ft_unset(mini, argv);
    else if (ft_strcmp(*argv, "echo") == 0)
        err = ft_echo(argv);
    else if (ft_strcmp(*argv, "pwd") == 0)
        err = ft_pwd(argv);
    else if (ft_strcmp(*argv, "env") == 0)
        err = ft_env(mini, argv);
    free_dptr(argv);
    if (_exit)
    {
        mini_clean(mini);
        exit(err);
    }
    return (err);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   exit.c                                             :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+#+   +#+           */
/*   Created: 2025/01/04 21:30:57 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/22 15:29:20 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

int    ft_exit(char **argv, int *_exit)
{
    int    status;

    if (isatty(STDIN_FILENO) && isatty(STDOUT_FILENO))
        ft_printf("exit\n");
    if (!argv[1])
    {
        *_exit = 1;
        if (g_sig != 0)
            return (128 + g_sig);
        return (0);
    }
```

```
    status = ft_atoi(argv[1]);
    if (argv[2])
    {
        ft_fprintf(2, PREFIX"exit: too many arguments\n");
        return (1);
    }
    else if (ft_str_is_numeric(argv[1]) == 0)
    {
        ft_fprintf(2, PREFIX"exit: %s: numeric argument required\n", argv[1]);
        status = 255;
    }
    *_exit = 1;
    return (status);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   unset.c                                            :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/11 12:00:00 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/26 18:59:18 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static void    remove_env_var(t_list **env, char *var_name)
{
    size_t    name_len;
    t_list    *cur;
    t_list    *prev;

    name_len = ft_strlen(var_name);
    cur = *env;
    prev = NULL;
    while (cur && cur->str)
    {
        if (ft_strncmp(cur->str, var_name, name_len) == 0
            && cur->str[name_len] == '=')
        {
            if (!prev)
                lst_remove_one(env, prev);
            else
                lst_remove_one(&cur, prev);
            return ;
        }
        prev = cur;
        cur = cur->next;
    }
}

int    ft_unset(t_mini *mini, char **argv)
{
    int    i;

    if (!argv[1])
        return (0);
    i = 1;
    while (argv[i])
    {
        if (ft_strchr(argv[i], '='))
        {
            ft_fprintf(2, PREFIX"unset: not a valid identifier\n");
            return (1);
        }
        remove_env_var(&mini->env, argv[i]);
        i++;
    }
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   env.c                                              :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
```

```
/*   Created: 2025/01/04 21:20:57 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/13 23:39:14 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

int    ft_env(t_mini *mini, char **argv)
{
    t_list    *cur;

    if (argv && argv[1])
    {
        ft_fprintf(2, PREFIX"'%s': No such file or directory\n", argv[0]);
        return (127);
    }
    cur = mini->env;
    while (cur && cur->str)
    {
        ft_printf("%s\n", cur->str);
        cur = cur->next;
    }
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   pwd.c                                              :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/04 21:05:57 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/19 12:16:57 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

int    ft_pwd(char **argv)
{
    char    cwd[PATH_MAX];

    if (argv[0] && argv[1])
    {
        ft_fprintf(2, "pwd: too many arguments\n");
        return (1);
    }
    if (getcwd(cwd, sizeof(cwd)) == NULL)
    {
        ft_fprintf(2, "minishell: pwd: %s\n", strerror(errno));
        return (1);
    }
    ft_printf("%s\n", cwd);
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   export.c                                           :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/11 12:00:00 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/27 05:29:11 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static int    is_valid_identifier(char *s)
{
    if (!*s || (*s >= '0' && *s <= '9'))
        return (0);
    while (*s && *s != '=')
    {
        if (!(((*s >= 'a') && (*s <= 'z'))
                || ((*s >= 'A') && (*s <= 'Z'))
                || ((*s >= '0') && (*s <= '9'))
                || *s == '_'))
```

```
            return (0);
        s++;
    }
    return (1);
}

static char    *make_env_variable(char *name, char *value)
{
    size_t    len;
    char    *new_env;

    len = ft_strlen(name);
    len++;
    len += ft_strlen(value);
    new_env = malloc(++len);
    if (!new_env)
        return (NULL);
    ft_snprintf(new_env, len, "%s=%s", name, value);
    return (new_env);
}

static int    add_env_var(t_list **env, char *name, char *value)
{
    t_list    *cur;
    size_t    name_len;

    name_len = ft_strlen(name);
    if (!*env)
        *env = ft_calloc(1, sizeof(t_list));
    cur = *env;
    while (cur && cur->str)
    {
        if (ft_strncmp(cur->str, name, name_len) == 0
            && cur->str[name_len] == '=')
            break ;
        if (!cur->next)
            cur->next = ft_calloc(1, sizeof(t_list));
        cur = cur->next;
    }
    if (!cur)
        return (1);
    free(cur->str);
    cur->str = make_env_variable(name, value);
    if (!cur->str)
        return (1);
    return (0);
}

static int    update_env(t_list **env, char *identify)
{
    char    *equals;

    equals = ft_strchr(identify, '=');
    if (equals)
    {
        *equals++ = '\0';
        if (!*equals)
            return (0);
        if (!is_valid_identifier(identify))
        {
            ft_fprintf(2, PREFIX"export: not a valid identifier\n");
            return (1);
        }
        if (add_env_var(env, identify, equals))
        {
            print_error(__FILE__, __LINE__);
            return (1);
        }
    }
    return (0);
}

int    ft_export(t_mini *mini, char **argv)
{
    int        i;
    t_list    *cur;

    if (!argv[1])
    {
```

```
        cur = mini->env;
        while (cur && cur->str)
        {
            ft_printf("declare -x %s\n", cur->str);
            cur = cur->next;
        }
        return (0);
    }
    i = 1;
    while (argv[i])
    {
        if (update_env(&mini->env, argv[i]) != 0)
            return (1);
        i++;
    }
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   cd.c                                               :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/04 20:57:32 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/14 06:15:15 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

int    ft_cd(t_mini *mini, char **argv)
{
    char    *path;
    char    cwd[PATH_MAX];

    if (!argv[1])
    {
        path = ft_getenv(mini->env, "HOME");
        if (!path)
        {
            ft_fprintf(2, PREFIX"cd: HOME not set\n");
            return (1);
        }
    }
    else
        path = argv[1];
    if (chdir(path) == -1)
    {
        ft_fprintf(2, PREFIX"cd: %s: %s\n", path, strerror(errno));
        return (1);
    }
    if (getcwd(cwd, sizeof(cwd)) == NULL)
    {
        ft_fprintf(2, PREFIX"cd: %s\n", strerror(errno));
        return (1);
    }
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   echo.c                                             :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/04 21:01:46 by mkurkar           #+#    #+#             */
/*   Updated: 2025/03/21 12:39:39 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

int    ft_echo(char **argv)
{
    int    i;
    int    newline;

    i = 1;
```

```
    newline = 1;
    if (argv[1] && ft_strcmp(argv[1], "-n") == 0)
    {
        newline = 0;
        i++;
    }
    while (argv[i])
    {
        ft_printf("%s", argv[i]);
        if (argv[i + 1])
            ft_printf(" ");
        i++;
    }
    if (newline)
        ft_printf("\n");
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   expand_env.c                                       :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/04/23 21:27:16 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/27 20:08:13 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static char    *join_and_free(char *s1, char *s2)
{
    char    *result;

    if (!s1 || !s2)
    {
        free(s1);
        free(s2);
        return (NULL);
    }
    result = ft_strjoin(2, s1, s2);
    free(s1);
    free(s2);
    return (result);
}

static char    *get_env_value(t_mini *mini, char *str, int *i)
{
    int        start;
    char    *var_name;
    char    *var_value;

    start = *i;
    while (str[*i] && (ft_isalnum(str[*i]) || str[*i] == '_'))
        (*i)++;
    if (*i == start)
        return (ft_strdup("$"));
    var_name = ft_substr(str, start, *i - start);
    if (!var_name)
        return (NULL);
    var_value = ft_getenv(mini->env, var_name);
    free(var_name);
    if (!var_value && errno != ENOMEM)
        return (ft_strdup(""));
    return (var_value);
}

static char    *expand_env_var(t_mini *mini, char *str, int *i)
{
    (*i)++;
    if (str[*i] == '?')
    {
        (*i)++;
        return (ft_itoa(mini->exit_status, 0));
    }
    else if (str[*i] == '\0' || str[*i] == ' ')
        return (ft_strdup("$"));
    else if (str[*i] == SINGLE_QUOTE || str[*i] == DOUBLE_QUOTE)
```

```
        return (ft_strdup(""));
    else if (ft_isdigit(str[*i]))
    {
        (*i)++;
        return (ft_strdup(""));
    }
    else
        return (get_env_value(mini, str, i));
}

char    *expand_env(t_mini *mini, char *str)
{
    char    *result;
    char    *temp;
    char    quote_char;
    int        i;

    result = ft_strdup("");
    quote_char = 0;
    i = 0;
    while (str[i] && result)
    {
        if ((str[i] == SINGLE_QUOTE || str[i] == DOUBLE_QUOTE) && !quote_char)
            quote_char = str[i];
        else if (str[i] == quote_char)
            quote_char = 0;
        else if (str[i] == '$' && quote_char != SINGLE_QUOTE)
        {
            temp = expand_env_var(mini, str, &i);
            result = join_and_free(result, temp);
            continue ;
        }
        temp = ft_substr(str, i, 1);
        result = join_and_free(result, temp);
        i++;
    }
    return (result);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   main.c                                             :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/07 13:09:28 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/28 05:53:38 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"
#include <unistd.h>
#include <stdio.h>
#include <strings.h>
#include <stdlib.h>

void    mini_clean(t_mini *mini)
{
    if (mini->tokens)
        lst_clean(&mini->tokens);
    if (mini->env)
        lst_clean(&mini->env);
}

void    exit_handler(t_mini *mini, int exit_status)
{
    mini_clean(mini);
    if (g_sig != 0)
        exit(128 + g_sig);
    exit(exit_status);
}

char    *read_prompt(void)
{
    char    prompt[PATH_MAX + 3];
    char    cwd[PATH_MAX];

    if (getcwd(cwd, sizeof(cwd)) == NULL)
        strcpy(cwd, "~");
```

```
    cwd[PATH_MAX - 1] = '\0';
    ft_snprintf(prompt, PATH_MAX + 3, "%s$ ", cwd);
    return (readline(prompt));
}

int    start(t_mini *mini, char tty_path[PATH_MAX])
{
    char    *line;

    if (restore_tty(tty_path) == -1)
        return (1);
    setup_signals();
    line = read_prompt();
    setup_signals2();
    if (!line)
    {
        ft_printf("\nexit\n");
        return (0);
    }
    if (!*line)
        return (1);
    add_history(line);
    mini->tokens = tokenizer(line);
    free(line);
    if (!mini->tokens)
        return (0);
    if (mini->tokens == (void *)0x1)
        return (1);
    if (check_syntax(mini->tokens))
        execute_line(mini);
    lst_clean(&mini->tokens);
    return (1);
}

int    main(void)
{
    t_mini        mini;
    char        tty_path[PATH_MAX];

    if (!isatty(0) || !isatty(1) || !isatty(2))
    {
        ft_fprintf(2, PREFIX"not a tty\n");
        return (1);
    }
    if (ft_ttyname_r(0, tty_path, sizeof(tty_path)) != 0)
        return (1);
    ft_bzero(&mini, sizeof(t_mini));
    mini.env = copy_env_variables();
    if (!mini.env)
        return (1);
    g_sig = 0;
    while (start(&mini, tty_path))
        mini.tokens = NULL;
    mini_clean(&mini);
    return (0);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   get_argv.c                                         :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/09 19:32:20 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/22 15:13:02 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

char    **lst_2_argv(t_list **lst, int flcean)
{
    char        **argv;
    t_list        *current;
    static int    i;
    int            _i;

    _i = i++;
    if (!*lst || !(*lst)->str)
```

```
    {
        if (flcean)
            lst_clean(lst);
        i = 0;
        return (ft_calloc(_i + 1, sizeof(char *)));
    }
    current = *lst;
    *lst = (*lst)->next;
    argv = lst_2_argv(lst, flcean);
    if (!argv)
        free(current->str);
    else
        argv[_i] = current->str;
    free(current);
    return (argv);
}

char    *get_argv0(t_list *lst)
{
    while (lst && lst->str)
    {
        if (ft_strcmp(lst->str, "<<") == 0
            || ft_strcmp(lst->str, ">>") == 0
            || ft_strcmp(lst->str, "<") == 0
            || ft_strcmp(lst->str, ">") == 0)
            lst = lst->next;
        else
            return (lst->str);
        lst = lst->next;
    }
    return (NULL);
}

void    get_argv(t_list **lst)
{
    t_list    *prev;
    t_list    *cur;
    t_list    *start;

    cur = *lst;
    prev = NULL;
    start = NULL;
    while (cur && cur->str)
    {
        if (ft_strcmp(cur->str, "<<") == 0
            || ft_strcmp(cur->str, "<") == 0
            || ft_strcmp(cur->str, ">>") == 0
            || ft_strcmp(cur->str, ">") == 0)
        {
            lst_remove_one(&cur, prev);
            lst_remove_one(&cur, prev);
            continue ;
        }
        if (!prev)
            start = cur;
        prev = cur;
        cur = cur->next;
    }
    *lst = start;
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   heredoc.c                                          :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2024/12/17 21:42:59 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/27 20:40:37 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"
#include "get_next_line.h"
#include <sys/ioctl.h>

/*
** Processes a line read during heredoc input
** Returns 1 if limiter is matched, 0 to continue, -1 on error
```

```
*/
static int    line_cmp(char *line, char *limiter)
{
    size_t    limiter_len;

    if (!*line)
    {
        if (isatty(STDIN_FILENO) && isatty(STDOUT_FILENO))
            ft_fprintf(2, "\n" PREFIX ": warning: here-document delimited by "
                "end-of-file (wanted `%s`)\n", limiter);
        free(line);
        return (0);
    }
    limiter_len = ft_strlen(limiter);
    if (ft_strncmp(line, limiter, limiter_len) == 0
        && (line[limiter_len] == '\n' || line[limiter_len] == '\0'))
    {
        free(line);
        return (0);
    }
    return (1);
}

static int    handle_chunk(t_mini *mini, char *limiter,
                        ssize_t nbytes, int out_fd)
{
    char    *line;
    char    *line_expanded;
    ssize_t    line_len;

    line = malloc(nbytes + 1);
    if (!line)
        return (-1);
    nbytes = read(STDIN_FILENO, line, nbytes);
    if (nbytes == -1)
    {
        free(line);
        return (-1);
    }
    line[nbytes] = '\0';
    if (line_cmp(line, limiter) == 0)
        return (0);
    line_expanded = expand_env(mini, line);
    free(line);
    if (!line_expanded)
        return (-1);
    line_len = ft_strlen(line_expanded);
    if (write(out_fd, line_expanded, line_len) != line_len)
        return (-1);
    free(line_expanded);
    return (1);
}

/*
** Main heredoc reading function
** Handles input for a heredoc until delimiter is reached
*/
static int    heredoc_start_read(t_mini *mini, char *limiter, int out_fd)
{
    int        err;
    int        nbytes;

    if (isatty(STDIN_FILENO) && isatty(STDOUT_FILENO))
        write(STDOUT_FILENO, "> ", 2);
    remove_qouts(limiter);
    while (1)
    {
        if (ioctl(STDIN_FILENO, FIONREAD, &nbytes) == -1)
            return (-1);
        if (g_sig != 0)
            return (1);
        if (nbytes > 0)
        {
            err = handle_chunk(mini, limiter, nbytes, out_fd);
            if (err <= 0)
                return (err);
            if (isatty(STDIN_FILENO) && isatty(STDOUT_FILENO))
                write(STDOUT_FILENO, "> ", 2);
        }
```

```
    }
    return (0);
}

/*
** Sets up and processes multiple heredocs in a command
*/
int    heredoc_forever(t_mini *mini, t_list *lst)
{
    int    fd;
    int    pipefd[2];

    fd = 0;
    while (lst && lst->str)
    {
        if (ft_strcmp(lst->str, "<<") == 0)
        {
            lst = lst->next;
            if (fd > 0)
                close(fd);
            if (pipe(pipefd) == -1)
                return (-1);
            if (heredoc_start_read(mini, lst->str, pipefd[1]) != 0)
            {
                close(pipefd[0]);
                close(pipefd[1]);
                return (-1);
            }
            close(pipefd[1]);
            fd = pipefd[0];
        }
        lst = lst->next;
    }
    return (fd);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   tokenizer.c                                        :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2024/12/17 12:19:29 by yaltayeh          #+#    #+#             */
/*   Updated: 2025/04/27 23:51:42 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"

static t_list    *add_token(t_list **lst, char *token)
{
    t_list    *new;

    if (!token)
    {
        lst_clean(lst);
        return (NULL);
    }
    new = malloc(sizeof(t_list));
    if (!new)
    {
        free(token);
        lst_clean(lst);
        return (NULL);
    }
    new->next = *lst;
    new->str = token;
    *lst = new;
    return (new);
}

static t_list    *tokenizer_iter(char *s, int i)
{
    char    *start;
    t_list    *lst;

    start = cut_slice(&s);
    if (!start || !s)
    {
```

```
        write(2, PREFIX"syntax error\n", sizeof(PREFIX"syntax error\n") - 1);
        return ((void *)0x1);
    }
    if (start == s && !*s)
        return (ft_calloc(1, sizeof(t_list)));
    lst = tokenizer_iter(s + !!*s, i + 1);
    if (!lst || lst == (void *)0x1)
        return (lst);
    add_token(&lst, ft_substr(start, 0, s - start));
    if (!lst)
        return (NULL);
    return (lst);
}

t_list    *tokenizer(const char *s)
{
    char    *expand_str;
    t_list    *tokens;

    expand_str = expand_line(s);
    if (!expand_str)
    {
        print_error(__FILE__, __LINE__);
        return (NULL);
    }
    tokens = tokenizer_iter(expand_str, 0);
    free(expand_str);
    return (tokens);
}
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   signals.c                                          :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: yaltayeh <yaltayeh@student.42amman.com>    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2025/01/07 13:07:47 by mkurkar           #+#    #+#             */
/*   Updated: 2025/04/21 01:20:11 by yaltayeh         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#include "minishell.h"
#include <signal.h>
#include <termios.h>

volatile int    g_sig;

// rl_replace_line("", 1);
static void    restore_prompt(int sig)
{
    g_sig = sig;
    write(STDOUT_FILENO, "\n", 1);
    rl_on_new_line();
    rl_redisplay();
}

static void    signal_handler(int sig)
{
    g_sig = sig;
}

void    setup_signals(void)
{
    struct sigaction    sa;

    sa.sa_handler = restore_prompt;
    sa.sa_flags = SA_RESTART;
    sigemptyset(&sa.sa_mask);
    sigaction(SIGINT, &sa, NULL);
    signal(SIGQUIT, SIG_IGN);
    signal(SIGTSTP, SIG_IGN);
}

void    setup_signals2(void)
{
    struct sigaction    sa;

    sa.sa_handler = signal_handler;
    sa.sa_flags = SA_RESTART;
```

```
    sigemptyset(&sa.sa_mask);
    sigaction(SIGINT, &sa, NULL);
    signal(SIGQUIT, SIG_IGN);
    signal(SIGTSTP, SIG_IGN);
}

void    reset_signals(void)
{
    signal(SIGINT, SIG_DFL);
    signal(SIGQUIT, SIG_DFL);
    signal(SIGTSTP, SIG_DFL);
}
```