SHELL

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>

#define MAX_COMMAND_LENGTH 1024
#define MAX_ARGUMENTS 64
#define MAX_TOKENS 64

void parse_command(char *input, char **args, int *background);
void execute_command(char **args, int background);

int main() {
    char input[MAX_COMMAND_LENGTH];
    char *args[MAX_ARGUMENTS];
    int background;

    while (1) {
        background = 0;
        printf("mysh> ");
        fgets(input, MAX_COMMAND_LENGTH, stdin);

        // Remove newline character
        input[strcspn(input, "\n")] = '\0';

        if (strcmp(input, "exit") == 0)
            break;
```

```
        parse_command(input, args, &background);
        execute_command(args, background);
    }

    return 0;
}

void parse_command(char *input, char **args, int *background) {
    char *token;
    int token_count = 0;

    // Tokenize input based on whitespace
    token = strtok(input, " ");
    while (token != NULL && token_count < MAX_TOKENS - 1) {
        args[token_count++] = token;
        token = strtok(NULL, " ");
    }
    args[token_count] = NULL;

    // Check if the last argument is '&', indicating background process
    if (token_count > 0 && strcmp(args[token_count - 1], "&") == 0) {
        *background = 1;
        args[token_count - 1] = NULL;
    }
}

void execute_command(char **args, int background) {
    pid_t pid;
    int status;

    pid = fork();
    if (pid < 0) {
```

```
        fprintf(stderr, "Fork failed\n");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        if (execvp(args[0], args) == -1) {
            perror("Error executing command");
            exit(EXIT_FAILURE);
        }
    } else {
        // Parent process
        if (!background)
            waitpid(pid, &status, 0);
    }
}
```