# Shell lab

- **Problem statement:**

  **Ω implementing a simple shell using c programming language.**

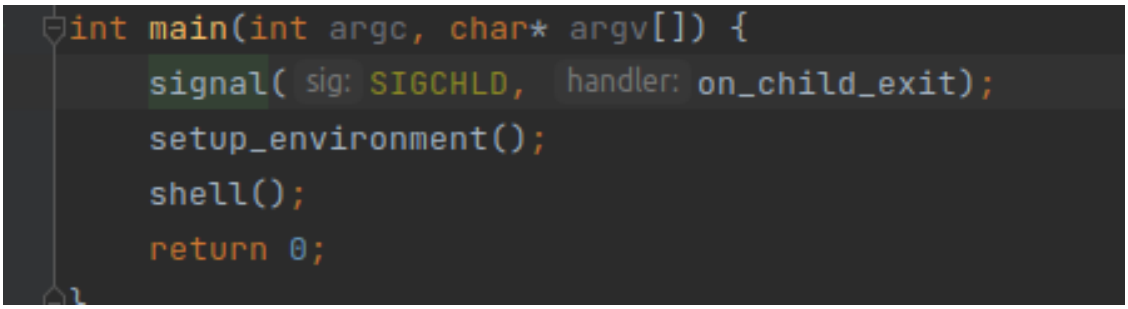
  **Ω shell supports the commands with and without arguments.**

  **Ω shell supports the commands executed in background and foreground.**

---

- **Code Description:**

  **Ω main function: uses the signal function to call handler function when children terminated then set up the environment to be the current directory. Finally, call shell implementation function.**



**Ω shell function: a loop is generated until the user inputs an exit command. Firstly, in the loop, the user enters the wanted command. secondly, the parsing function called to separate the command to the actual command and options or command arguments in an array of strings. Finally, call one of execution functions: built in and non-built in because each kind has a different implementation.**

```
void shell() {
    do {
        // print for show
        printCurrentPath();

        // read and parse
        read_input();
        int in_background = parse_input();

        // substitute variable by its value in non echo command only at the end of command.
        if(strcmp(cmd.args[0], "echo") != 0) {
            substitute();
        }

        // directing by type created in parsing.
        if (strcmp(cmd.input_type, "execute_shell_bultin") == 0) {
            execute_shell_bultin();
        } else if (strcmp(cmd.input_type, "executable_or_error") == 0){
            execute_command(in_background);
        }

    }while(command_is_not_exit());
}
```

**Ω each command entered by the user after parsing converted into array of strings of known length and type of execution function will be used. All of this information is stored in struct.**

```
struct Command {
    char* input_type;   // built-in and non-built-in.
    int length;         // top pointer index.
    char* args[10];     // max arguments for a command line.
}cmd;                   // create global struct (obj) instance.
```

**Ω in the first type of execution functions (built in functions), there is an implementation of three commands (cd, export and echo). Firstly, cd command implemented by using built in function (chdir) with all cases of optional arguments. Secondly, the echo command is implemented by deleting the double quotes and substituting each variable with its equivalent value using function (evaluate_expression). Finally, the export command to create a new variable implemented in (parse_export) function to get the name of variable and its value and then create a new variable if there is no old variable has the same name, else, give the existing variable the value entered by user.**

```
void execute_shell_bultin() {
    if(strcmp(cmd.args[0], "cd") == 0) {

        if( cmd.args[1] == NULL || isEqual( s1: cmd.args[1], s2: "~" ) ){...}
        else {...}

    }
    else if (strcmp(cmd.args[0], "echo") == 0) {

        // echo command
        evaluate_expression( exp: cmd.args[1]);

    }
    else if (strcmp(cmd.args[0], "export") == 0) {

        // export command
        parse_export();

    }
}
```

**Ω the variables created is stored in linked structs each one is consist of identifier and value.**

```
struct Var {
    char identifier[15];
    char value[15];
    Var* nxt;
};
```

Ω in the second type of execution functions(non-built in), we create a child process to execute the user command by function fork and the parent process choose the way to deal with the executed functions either wait until the child terminated or continue to execute another commands without waiting the child to terminate depending on char “&”.

```
int execute_command(int in_background)
{
    int child_id = fork();
    int isParent = child_id;
    if ( !isParent ) {

        if( cmd.args[1] != NULL && isEqual( s1: cmd.args[1], s2: "&") ){
            cmd.args[1] = NULL;
        }
        int res = execvp( file: cmd.args[0], argv: cmd.args);
        if(res < 0 && strcmp(cmd.args[0], "exit") != 0){
            printf( format: "Error Message not found\n");
        }
        exit( status: 0);

    }
    // wait until the user exit the opened app.
    else if ( !in_background ) {
        int status;
        waitpid( pid: child_id, stat_loc: &status, options: 0);
    }
    return 0;
}
```

Ω when the child ends its task the handler function called in the parent process and waitpid function is called to delete the data of child stored.

```
void on_child_exit() {
    write( fd: log_file, buf: "Child terminated\n", n: strlen( s: "Child terminated\n"));
    int status;
    waitpid( pid: -1, stat_loc: &status, options: WNOHANG);
}
```

- **Sample runs:**

[1] test case:

```
shell: /home/abdelmotlb/Public/OS_labs/others >> ls
cd  cmake-build-debug  CMakeLists.txt  folder1  log_file.txt  main.c  test
```

[2] test case:

```
shell: /home/abdelmotlb/Public/OS_labs/others >> cd ~
shell: /home/abdelmotlb >> cd ..
shell: /home >> cd abdelmotlb/Public/OS_labs
shell: /home/abdelmotlb/Public/OS_labs >>
```

[3] test case:

```
shell: /home/abdelmotlb/Public/OS_labs/others >> export x="hello world"
shell: /home/abdelmotlb/Public/OS_labs/others >> echo "variable's value is $x"
variable's value is hello world
```

[4] test case:

```
shell: /home/abdelmotlb/Public/OS_labs/others >> hello world
Error Message not found
```

[5] test case:

```
shell: /home/abdelmotlb/Public/OS_labs/others >> ls -l
total 80
-rwxrwxr-x 1 abdelmotlb abdelmotlb 21520 21:07 6  مار cd
-rwxrwxr-x 1 abdelmotlb abdelmotlb 21520 04:01 9  مار cd2
drwxrwxr-x 5 abdelmotlb abdelmotlb  4096 19:17 6  مار cmake-build-debug
-rw-rw-r-- 1 abdelmotlb abdelmotlb   112 01:33 5  مار CMakeLists.txt
drwxrwxr-x 2 abdelmotlb abdelmotlb  4096 21:32 6  مار folder1
-rw-rw-r-- 1 abdelmotlb abdelmotlb    17 04:05 9  مار log_file.txt
-rw-rw-r-- 1 abdelmotlb abdelmotlb  8626 04:01 9  مار main.c
drwxrwxr-x 2 abdelmotlb abdelmotlb  4096 19:57 6  مار test
```

- **Processes hierarchy screenshots:**