Simple Shell program

OS – LAB 1

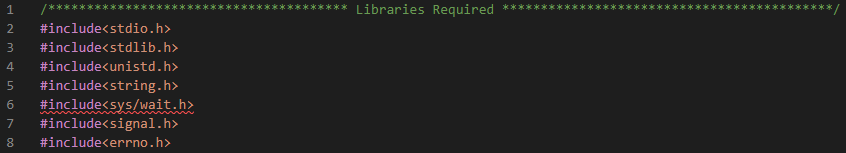
Abdullah Mohamed AbdelHakim Gomaa

19015953

**Code Summary:**

This is a C program that implements a basic Unix shell. The main function registers a signal handler for SIGCHILD signal and sets the working environment to the current directory then it enters an infinite loop and takes input from the user using the parseInput() function. The program then accepts user input, parses it, and determines whether the command is a shell built-in command or an executable command using the inputType() function. If it is a shell built-in command, the program executes it. Otherwise, the program executes the command by calling execute\_command() function which uses the execvp system call. The program continues the loop until the user enters the "exit" command to exit the shell. The code includes a section that creates a log file called "log.txt" to record when child processes are terminated.

**Librares:**

****

**Fig.1**

**Functions:**

**Text

Description automatically generated**

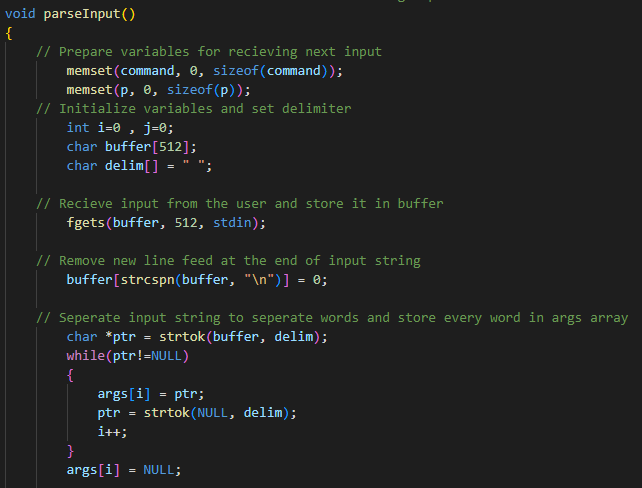
**Fig.2**

1. **parseInput():**

Step1:

* Taking input from the user and store every word in a separate element in array called args as shown in fig.3.

- The memset function is used to clear the arrays for future re-use of the arrays when the function is called again from the main program.

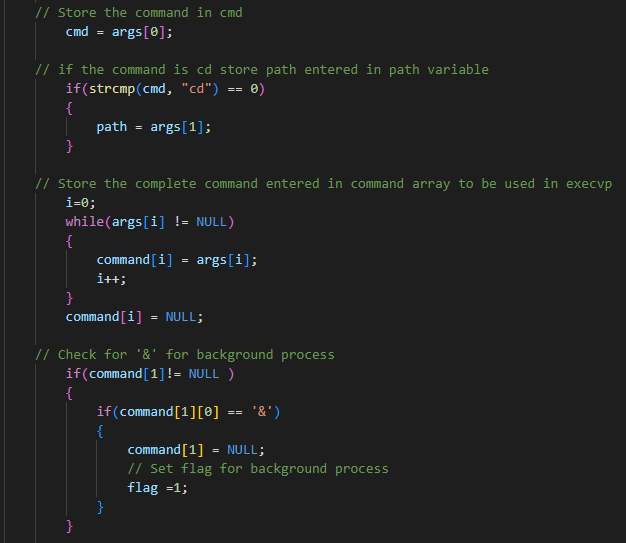


**Fig.3**

|  |  |
| --- | --- |
|  |  |

Step2:

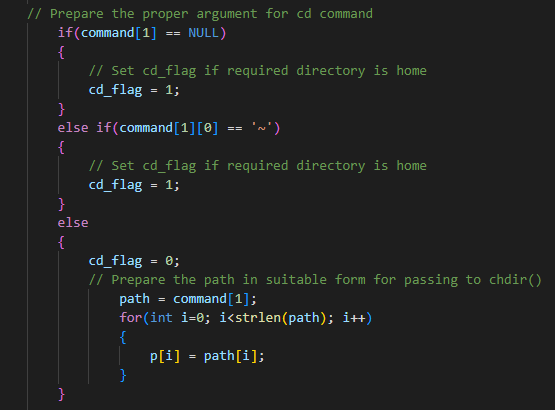
* Copy the command which is the first element in args array to cmd variable.
* If the command is “cd”, store the second element in args array to the path variable that clearly stores the path of the destination directory.
* Prepare the command array that is fed as input to the execvp function which will be illustrated later.
* Check for ‘&” and set the flag for background process.



**Fig.4**

Step3:

* Set cd\_flag to 1 if the destination path is home directory.
* The chdir function requires a char array as an argument, so in this part the path received is stored in a character array p.



**Fig.5**

Step4:

* If there is an option or an argument to the command, it is going to be stored in character array called arr element by element including spaces which will help in future use of these options or arguments in other functions.
* Remove all quotes from arr.
* This was the last step in parsing input from the user and preparing data entered.

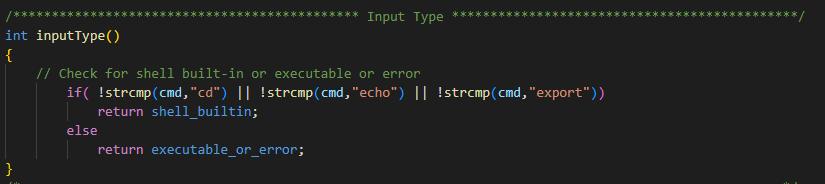
Text

Description automatically generated

**Fig.6**

1. **inputType():**

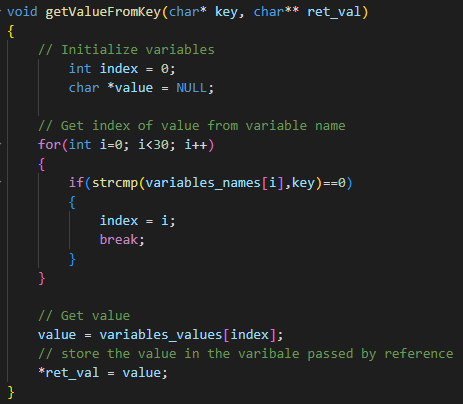
A helping function that checks if the entered command is shell built-in or if it is executable or error.



**Fig.7**

1. **getValueFromKey():**

Another helping function that facilitates getting the value of an environment defined variable from the variables array that is defined by the values entered by the user by export command.

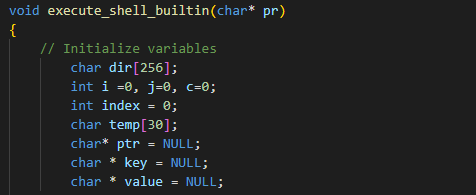


**Fig.8**

1. **execute\_shell\_builtin(char\* pr):**

Step1:

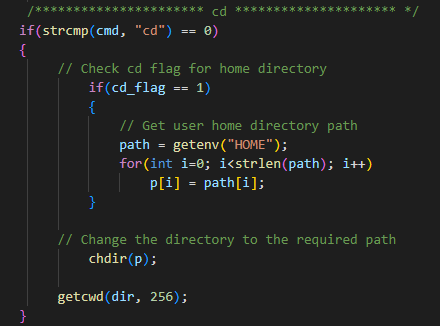
* Initialize the variables.

****

**Fig.9**

Step2:

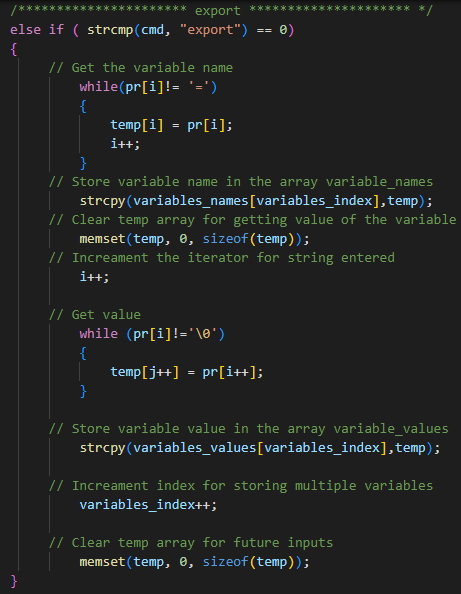
* If the command is cd, check for home directory flag (cd\_flag), then change directory to the required path.



**Fig.10**

Step3:

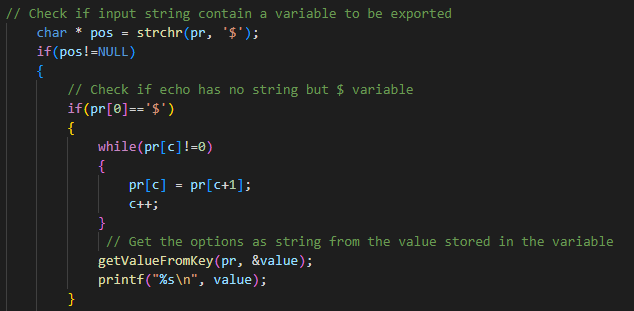
* Export command.
* There is an array called variable\_names to store variable names entered by user and another called variable\_values to store value of the variable such that by the same index both variable name and it’s value could be accessed easily.



**Fig.11**

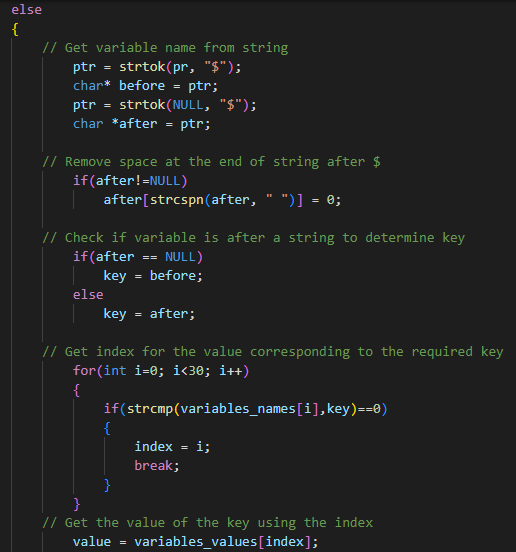
Step4: echo command.

1. Check for $ in string then check if it is the first occurrence.
2. If it is the first occurrence get the value using the helping function getValueFromKey of the variable after $ and print the output.



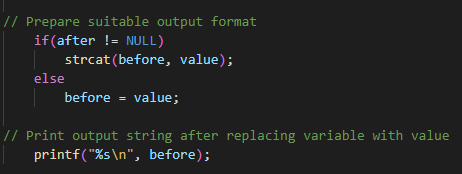
**Fig.12**

1. If it is not first occurrence get variable name after $ to get required value.



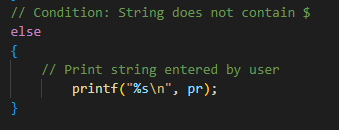
**Fig.13**

1. Prepare output in suitable format for printing.



**Fig.14**

1. If the string does not contain $, print the output directly.

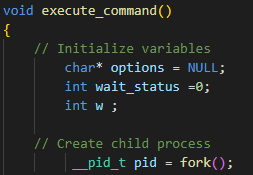


**Fig.15**

1. **execute\_command():**

step1:

* Initialize the variables.
* Create child process using fork.



**Fig.16**

Step2:

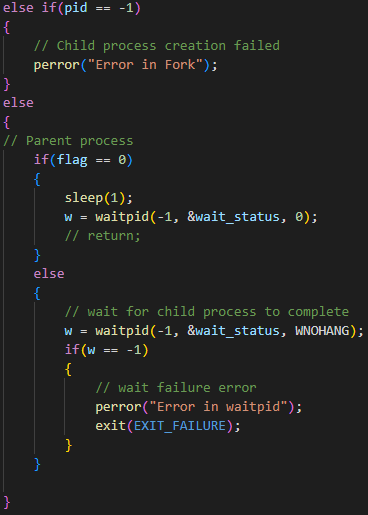
* Child Process
* Check if command is executable else raise an error.
* Check if command option start with $ to get value of the variable after $.
* Prepare command array to be passed to the execvp system call that executes the command.



**Fig.17**

Step3:

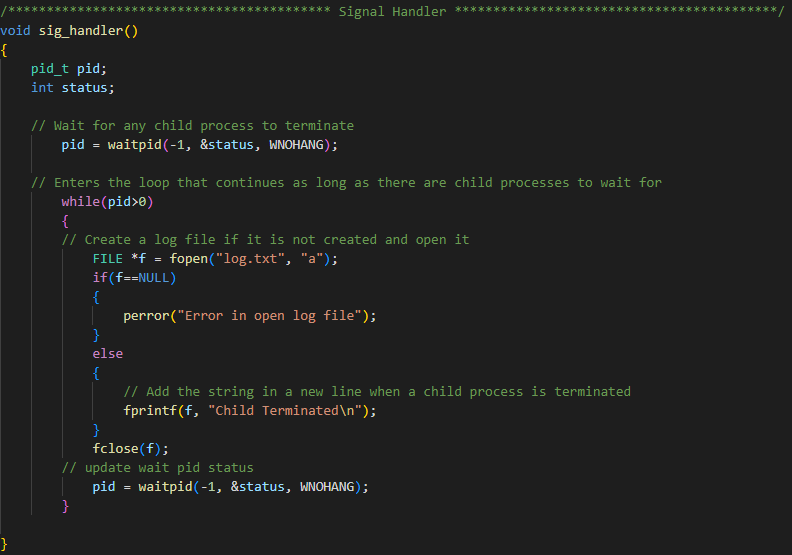
* Parent or Error
* Check if error in child creation raise a fork error.
* In parent check for flag for background process and configure waitpid option to 0 to make the process run in background.
* For foreground process waitpid option should be WNOHANG to wait for child process to complete.



**Fig.18**

1. **sig\_handler():**

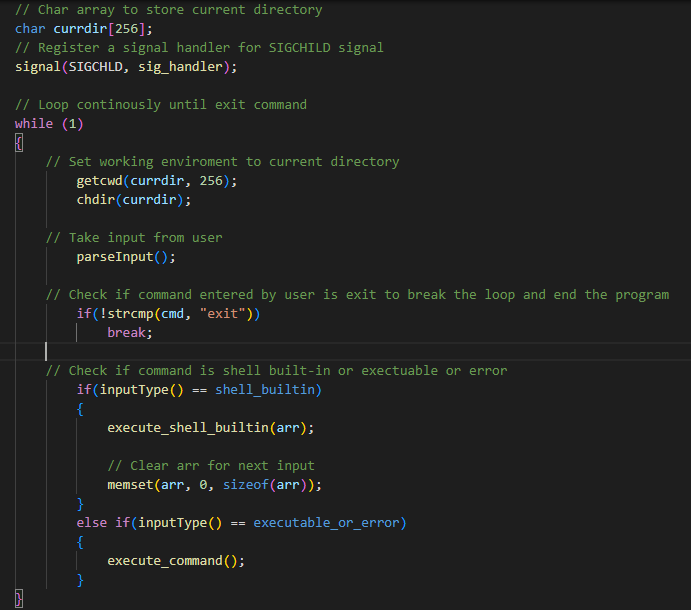
* Waits for child process to terminate to reap zombie processes.
* Create a log file which is appended by the statement “Child Terminated” every time a child process is terminated.

****

**Fig.19**

1. **main():**

* Register a signal handler for signal SIGCHILD.
* Check for exit command to break the super loop and ends the program.
* Calls suitable function based on the command input type.

****

**Fig.20**