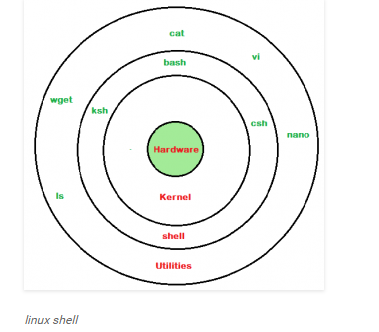
**Making OUR own Linux Shell in C**

**What is SHELL?**

A shell is special user program which provide an interface to user to use operating system services. Shell accept human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.



Shell is broadly classified into two categories –

* + Command Line Shell
  + Graphical shell

**Command Line Shell**

Shell can be accessed by user using a command line interface. A special program called **Terminal** in Linux/macOS or **Command Prompt** in Windows OS is provided to type in the human readable commands such as “cat”, “ls” etc. and then it is being execute. The result is then displayed on the terminal to the user.

It will list all the files in current working directory in long listing format.  
Working with command line shell is bit difficult for the beginners because it’s hard to memorize so many commands. It is very powerful, it allows user to store commands in a file and execute them together. This way any repetitive task can be easily automated. These files are usually called **batch files** in Windows and **Shell** Scripts in Linux/macOS systems.

**Graphical Shells**

Graphical shells provide means for manipulating programs based on graphical user interface (GUI), by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows. Window OS or Ubuntu OS can be considered as good example which provide GUI to user for interacting with program. User do not need to type in command for every action.

Summary Algorithm of our Project in Simple lines:

* Command is entered and if length is non-null, keep it in history.
* Parsing: Parsing is the breaking up of commands into individual words and strings
* Checking for special characters like pipes, etc. is done
* Checking if built-in commands are asked for.
* If pipes are present, handling pipes.
* Executing system commands and libraries by forking a child and calling execvp.
* Printing current directory name and asking for next input.

**Implementation**

First, we installed Readline Library

sudo apt-get install libreadline-dev

* Printing the directory can be done using **getcwd**.
* Getting user name can be done by **getenv(“USER”)**
* Parsing can be done by using **strsep(“”)**. It will separate words based on spaces. Always skip words with zero length to avoid storing of extra spaces.
* After parsing, check the list of built-in commands, and if present, execute it. If not, execute it as a system command. To check for built-in commands, store the commands in an array of character pointers, and compare all with **strcmp()**.  
  Note: “cd” does not work natively using execvp, so it is a built-in command, executed with **chdir()**.
* For executing a system command, a new child will be created and then by using the execvp, execute the command, and wait until it is finished.
* Detecting pipes can also be done by using **strsep(“|”)**.To handle pipes, first separate the first part of the command from the second part. Then after parsing each part, call both parts in two separate new children, using execvp. Piping means passing the output of first command as the input of second command.
  1. Declare an integer array of size 2 for storing file descriptors. File descriptor 0 is for reading and 1 is for writing.
  2. Open a pipe using the pipe() function.
  3. Create two children.
  4. In child 1->  
     Here the output has to be taken into the pipe.

Copy file descriptor 1 to stdout.

Close file descriptor 0.

Execute the first command using execvp()

* 1. In child 2->  
     Here the input has to be taken from the pipe.

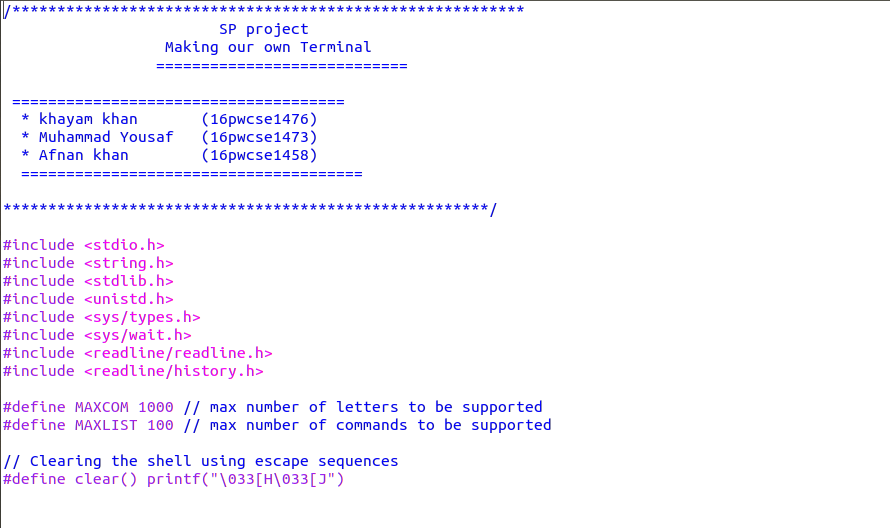
Copy file descriptor 0 to stdin.

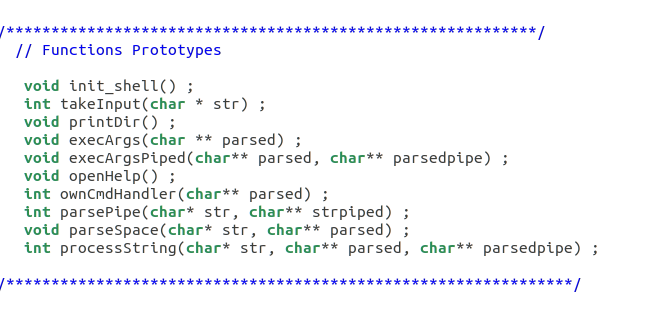
Close file descriptor 1.

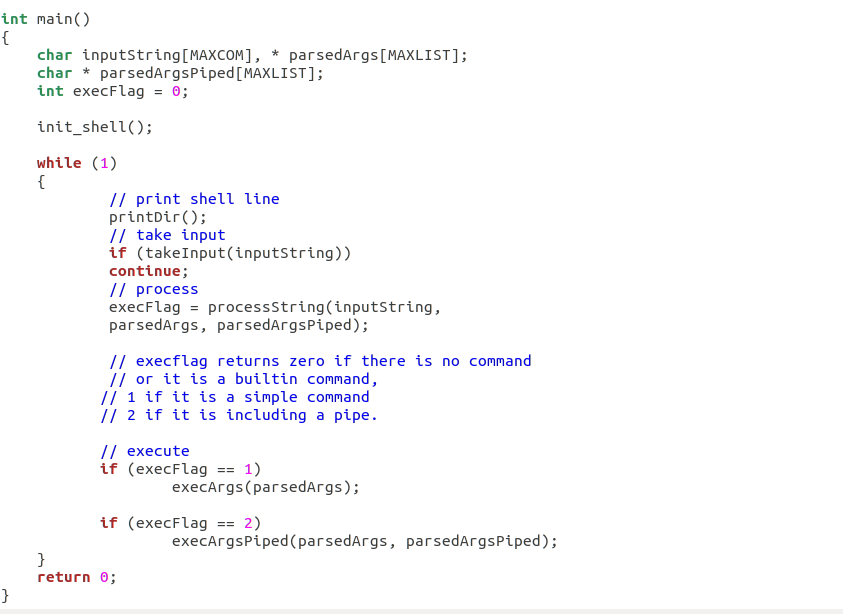
Execute the second command using execvp()

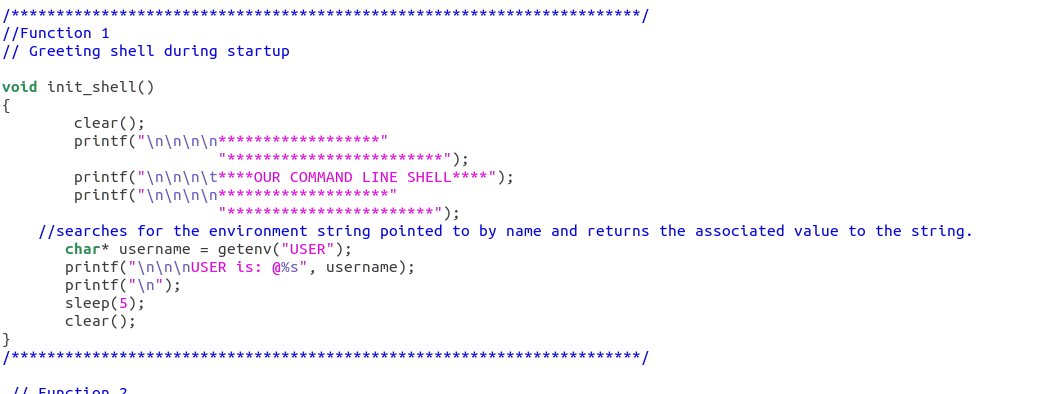
* 1. Wait for the two children to finish in the parent.

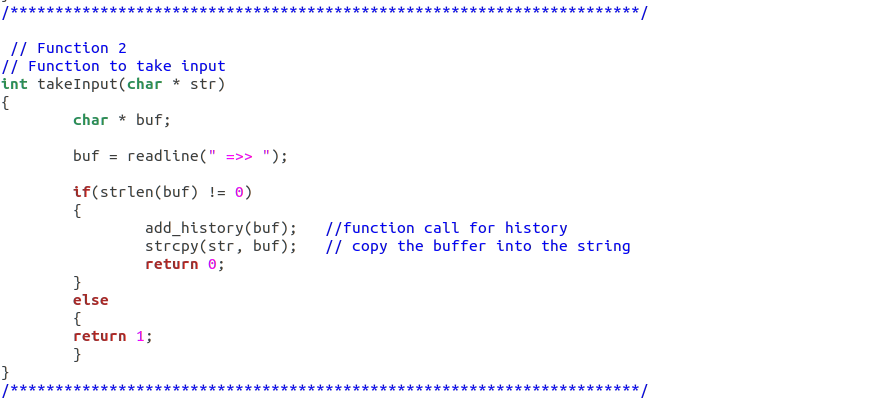
**SCREENSHOTS OF OUR CODE**

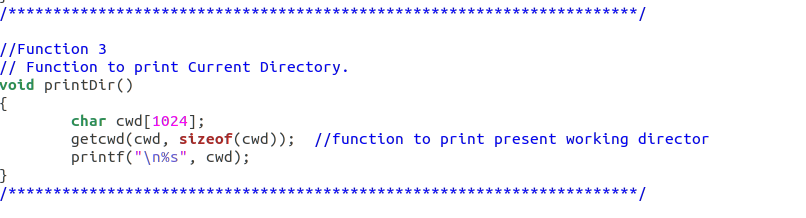


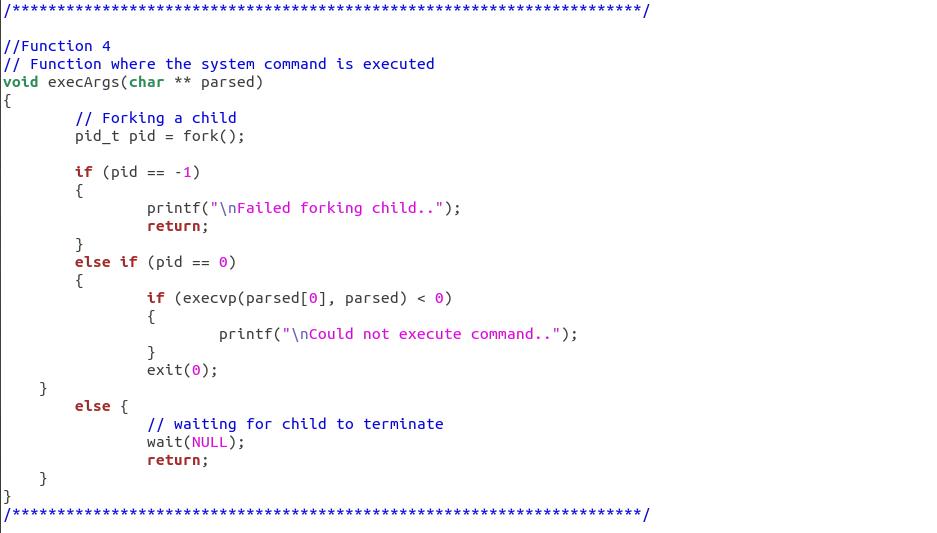


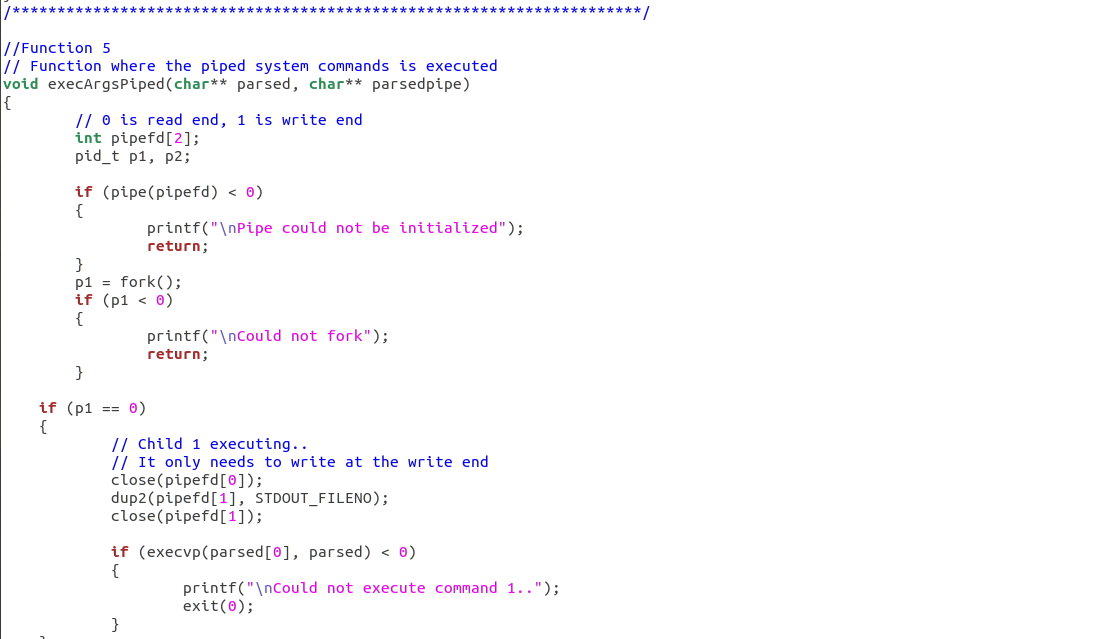


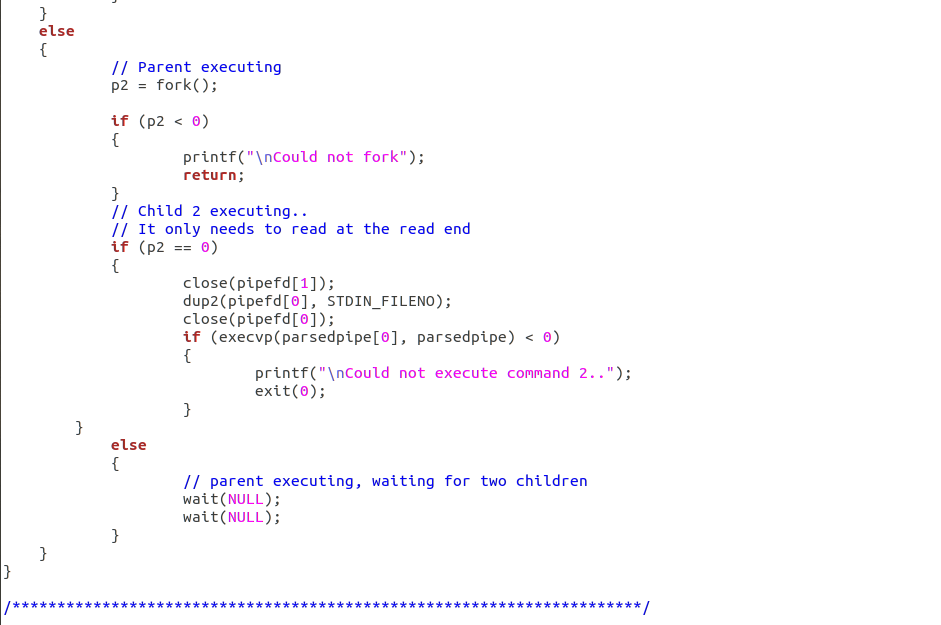




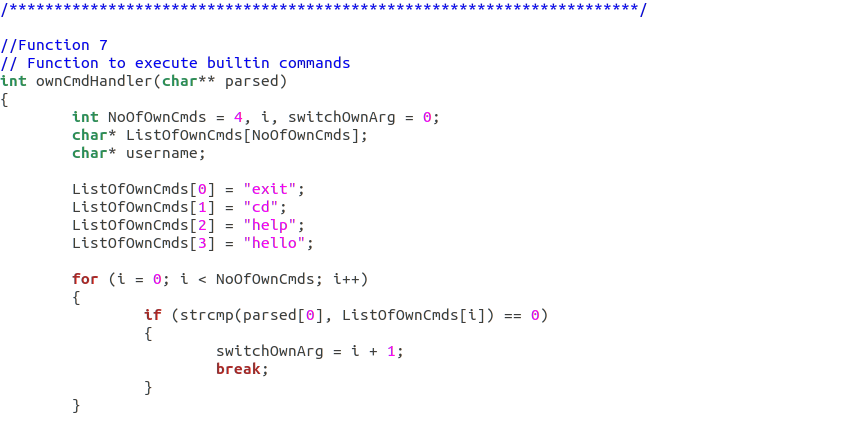




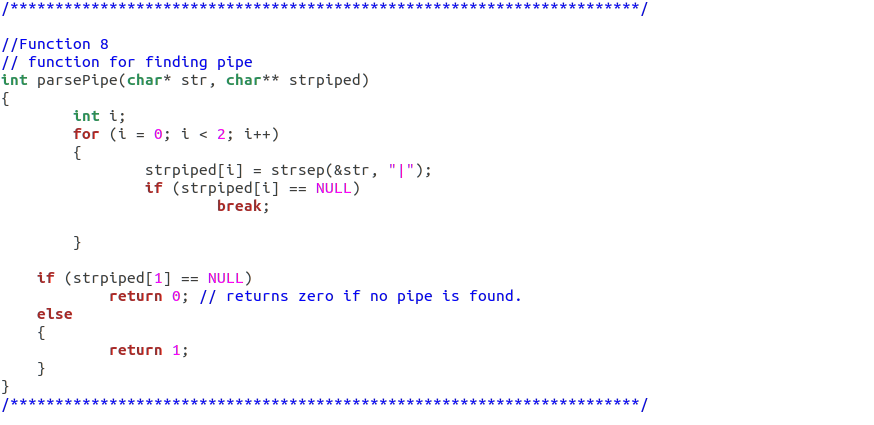


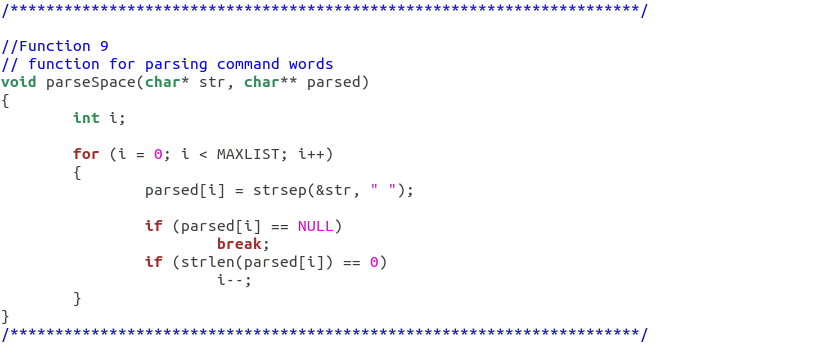


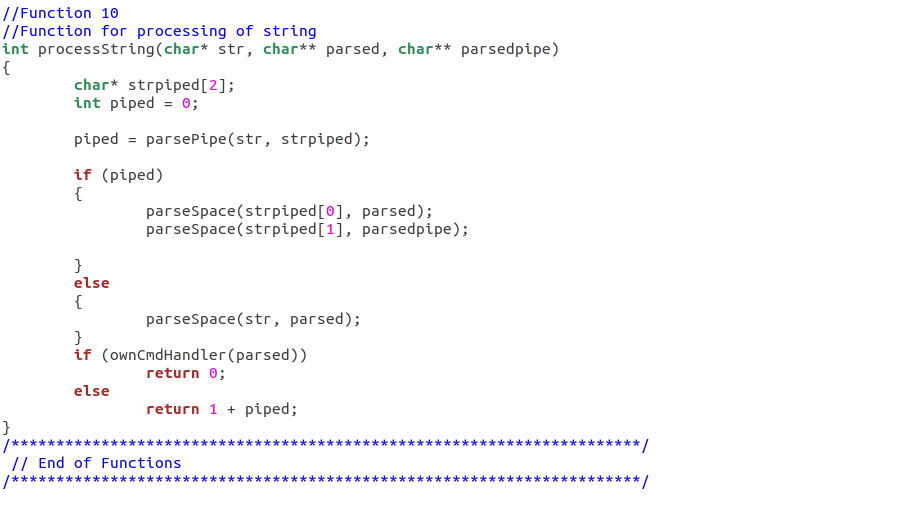




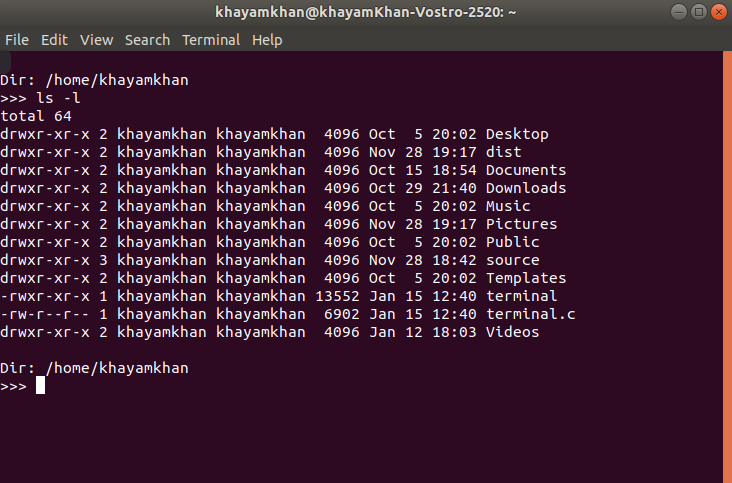


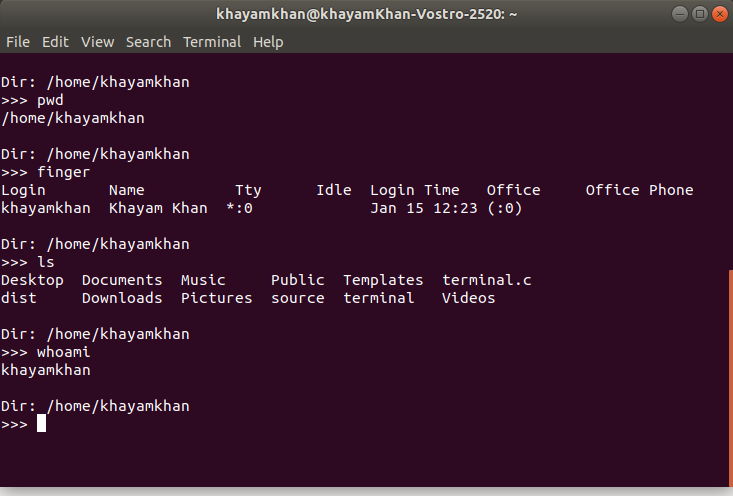






**OUTPUT**





Reference:

<https://www.geeksforgeeks.org/making-linux-shell-c/>