Flash Team

-Esraa Abd-Elmoneam Abd-Elhamed

-Salma Osama Ibrahem Sayed

-Salma Mohamed Kishk

-Samah Bayoumi Abid

-Yasmeen Mohamed Shamakh

Shell Program in C

Dr/ Reda Elbasiony

Eng/ Marwa Badr

Objective:

- To make our own shell to process the commands and returns outputs

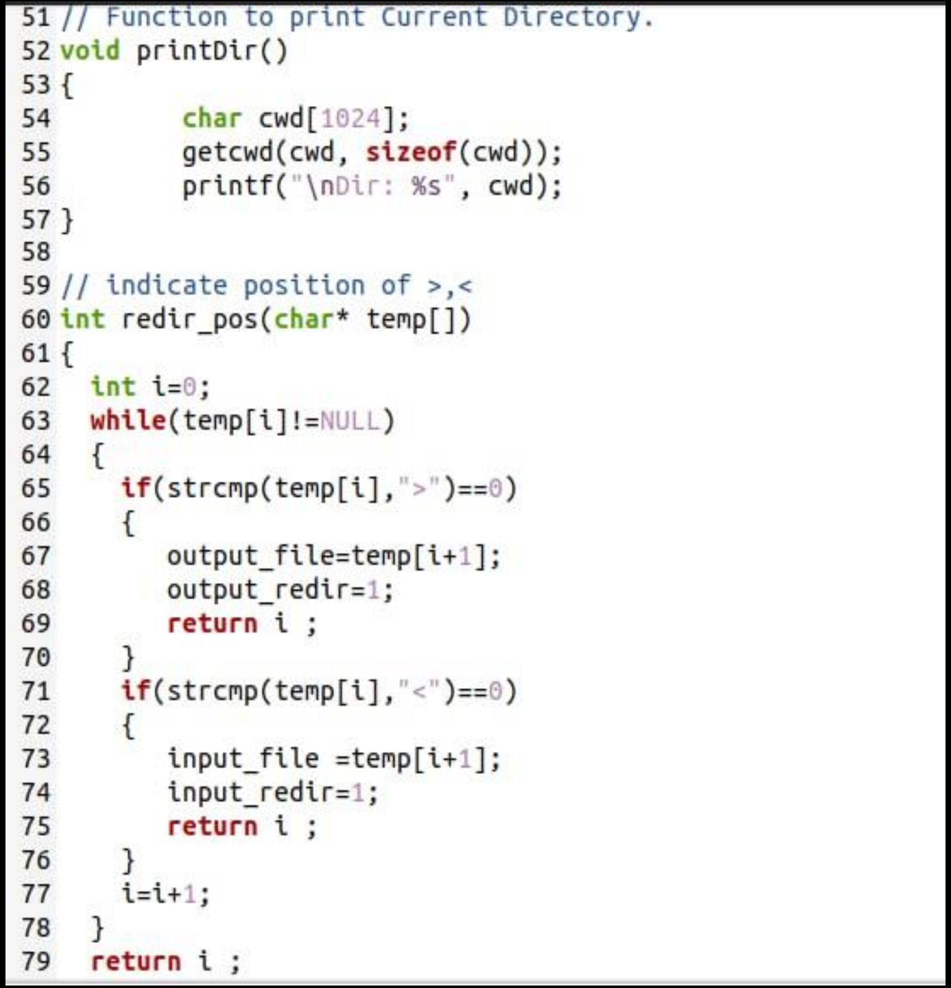
How can we make it ?!

- <u>We follow these steps:</u>
- *Take input.*
- *Parsing input into strings "command words".*
- *Check if the input has pipe or redirection.*
- *Check if the input is build in commands as clear,help,…*
- *Handle execution of the input in each cases.*

Printing directory

- *getcwd() function returns an absolute file name representing the current working directory, storing it in the character array cwd.*

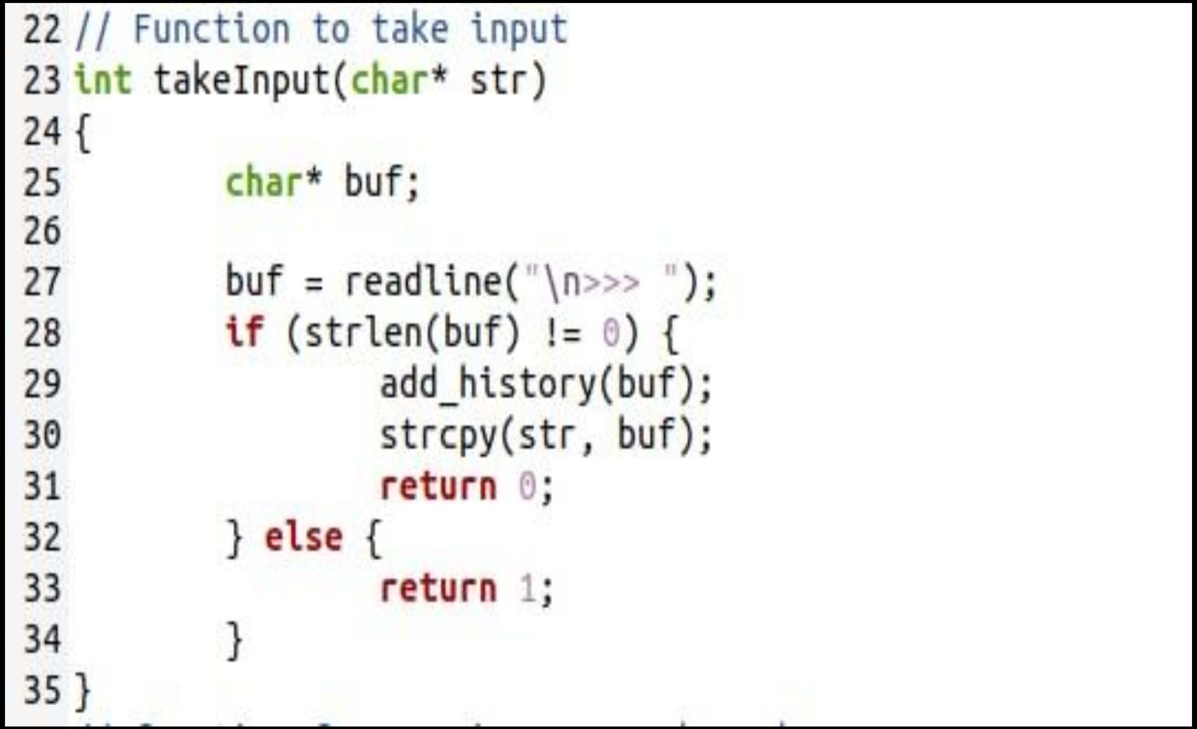
- *The size argument is how we tell the system the allocation size of buffer.*



```
// Function to print Current Directory.
void printDir()
{
        char cwd[1024];
        getcwd(cwd, sizeof(cwd));
        printf("\nDir: %s", cwd);
}

// indicate position of >,<
int redir_pos(char* temp[])
{
  int i=0;
  while(temp[i]!=NULL)
  {
    if(strcmp(temp[i],">")==0)
    {
       output_file=temp[i+1];
       output_redir=1;
       return i ;
    }
    if(strcmp(temp[i],"<")==0)
    {
       input_file =temp[i+1];
       input_redir=1;
       return i ;
    }
    i=i+1;
  }
  return i ;
```

Taking input

- *Command is entered and if length is non-null, keep it in history.*



```
// Function to take input
int takeInput(char* str)
{
        char* buf;

        buf = readline("\n>>> ");
        if (strlen(buf) != 0) {
                add_history(buf);
                strcpy(str, buf);
                return 0;
        } else {
                return 1;
        }
}
```

Parsing

- *we make Parsing To split the input line into list of arguments (or strings).*

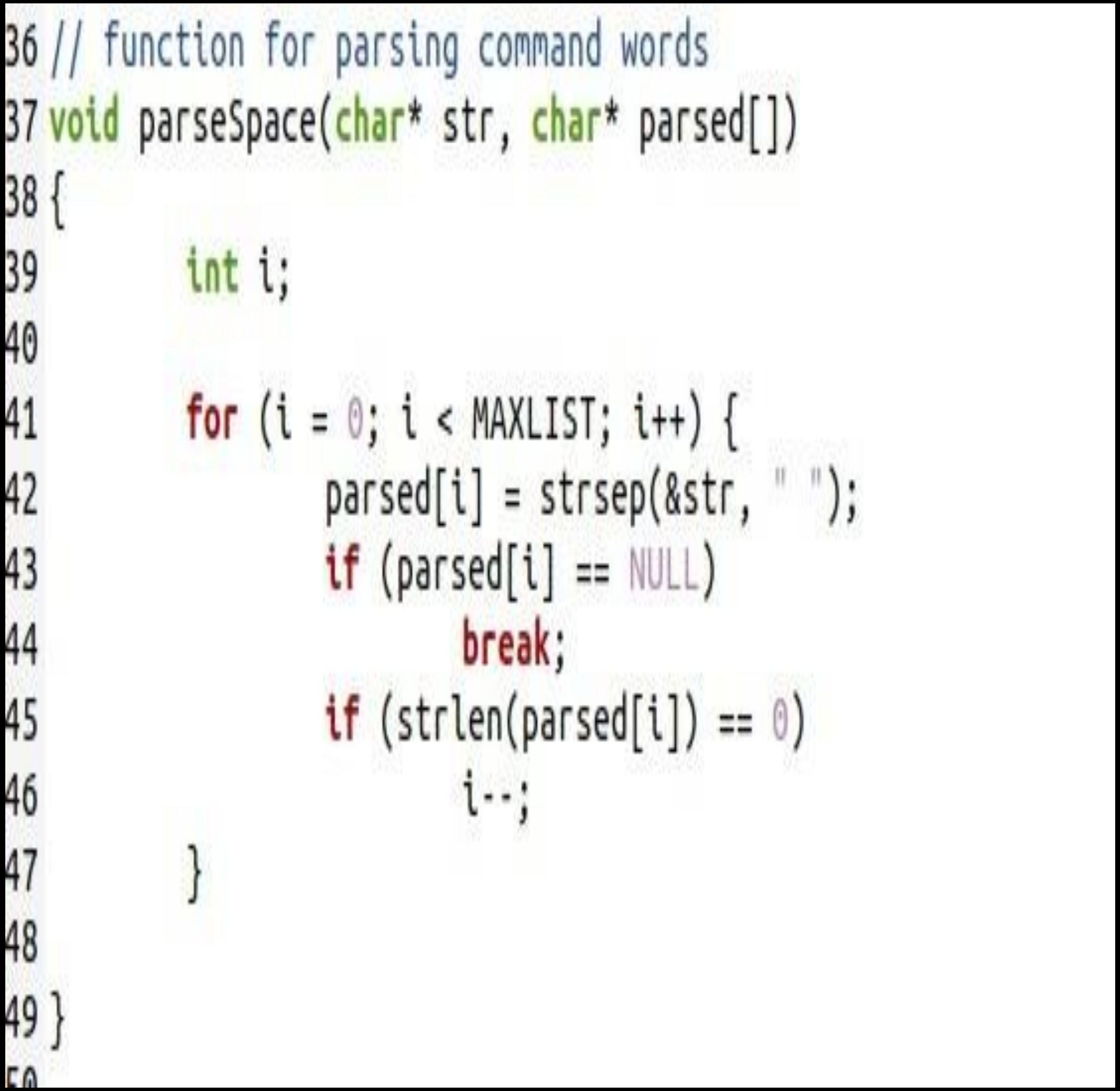
- *we use whitespace to separate line arguments.*

- *i is a variable which work like a counter for number of list (array).*

- *It means If we Enter "gcc project.c" It will parse to*

    pursed [0] = "gcc" ,

    parsed [1] = "project.c "

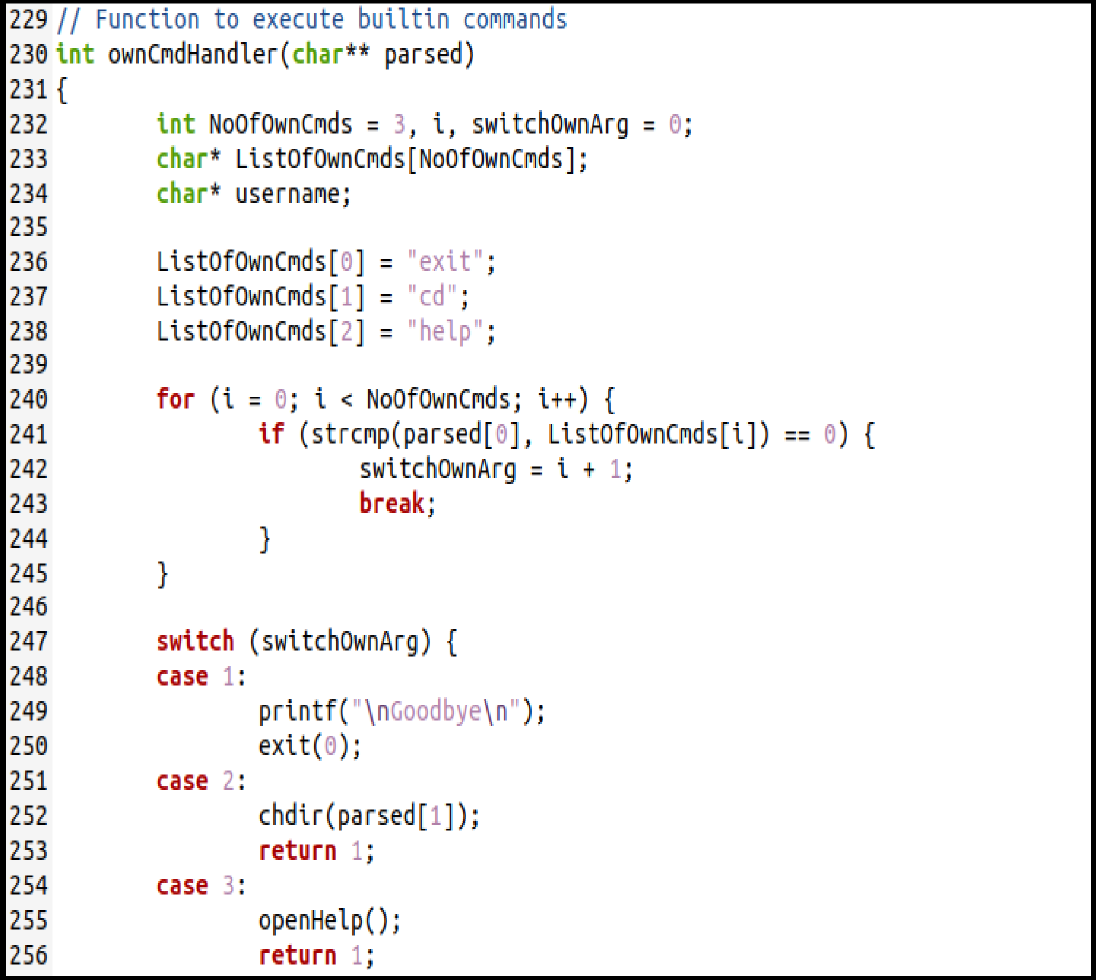


```
// function for parsing command words
void parseSpace(char* str, char* parsed[])
{
        int i;

        for (i = 0; i < MAXLIST; i++) {
                parsed[i] = strsep(&str, " ");
                if (parsed[i] == NULL)
                        break;
                if (strlen(parsed[i]) == 0)
                        i--;
        }

}
```

checking if the input has build-in command

- *Checking strings after parsing by storing them in array of character and compare with built-in commands using strcmp() function.*

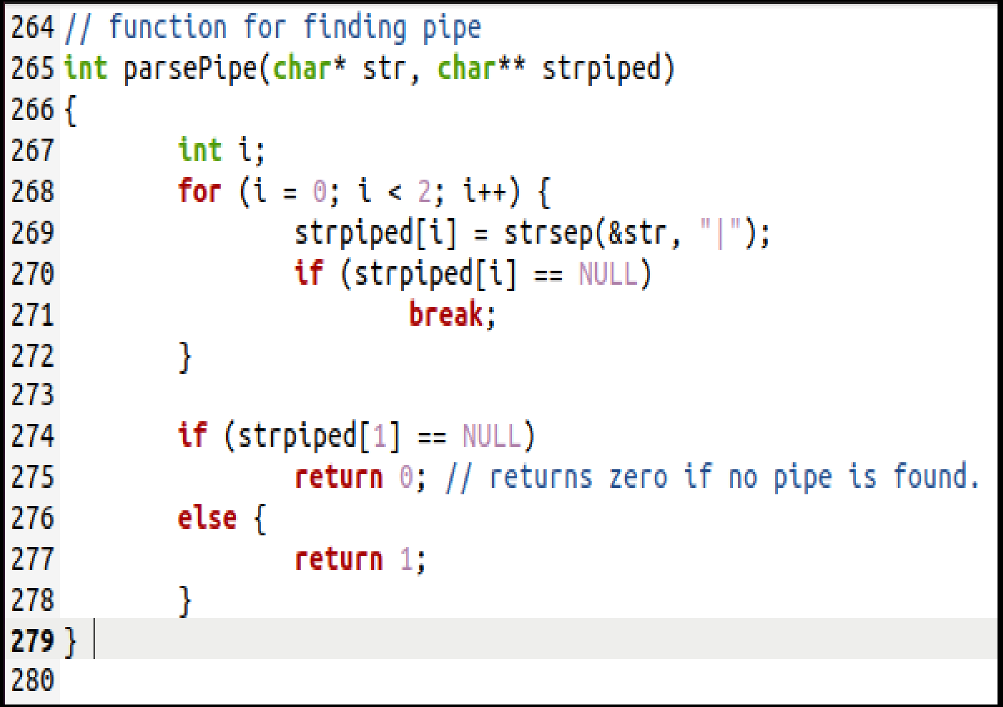
- *Strcmp() compare two strings lexicographically which mean compare character by character ,if return value equal '0' then two strings are equal.*



```
// Function to execute builtin commands
int ownCmdHandler(char** parsed)
{
        int NoOfOwnCmds = 3, i, switchOwnArg = 0;
        char* ListOfOwnCmds[NoOfOwnCmds];
        char* username;

        ListOfOwnCmds[0] = "exit";
        ListOfOwnCmds[1] = "cd";
        ListOfOwnCmds[2] = "help";

        for (i = 0; i < NoOfOwnCmds; i++) {
                if (strcmp(parsed[0], ListOfOwnCmds[i]) == 0) {
                        switchOwnArg = i + 1;
                        break;
                }
        }

        switch (switchOwnArg) {
        case 1:
                printf("\nGoodbye\n");
                exit(0);
        case 2:
                chdir(parsed[1]);
                return 1;
        case 3:
                openHelp();
                return 1;
```

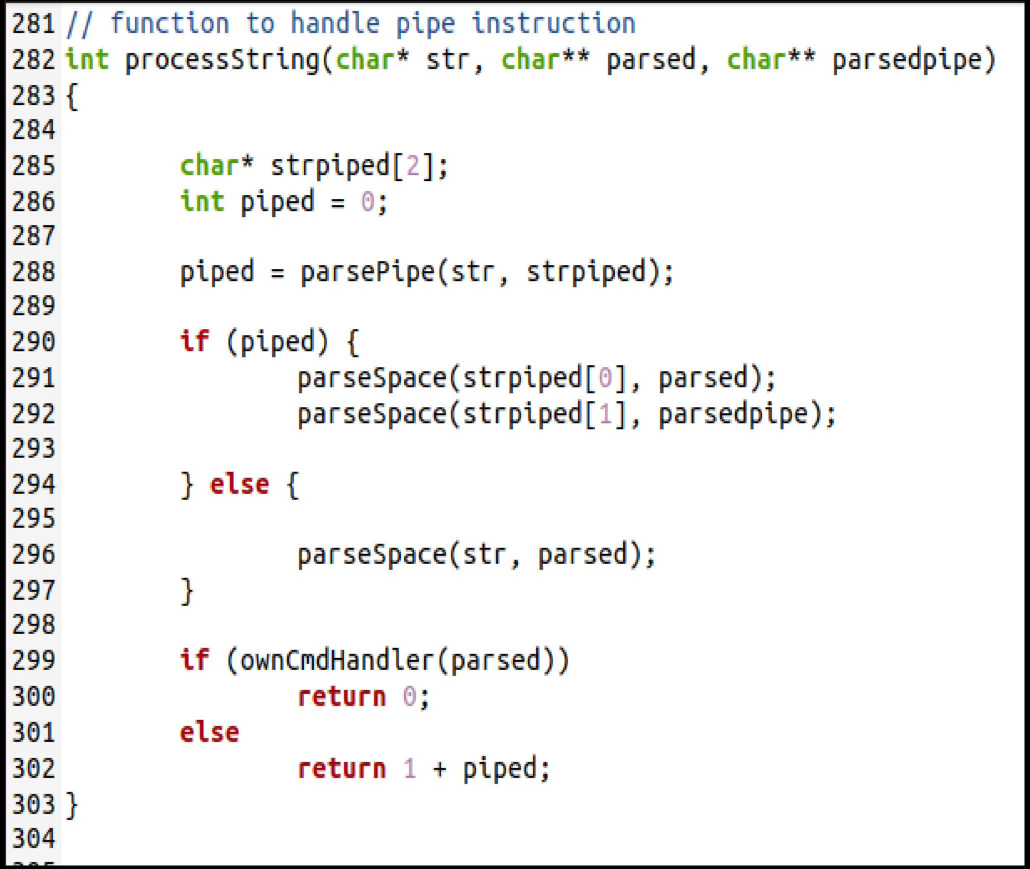
Checking if input has piping and splitting the input into two commands

- *By using parsepipe() function we can find the pipe and splits the input into two commands (before and after pipe mark '|')*


```
// function for finding pipe
int parsePipe(char* str, char** strpiped)
{
        int i;
        for (i = 0; i < 2; i++) {
                strpiped[i] = strsep(&str, "|");
                if (strpiped[i] == NULL)
                        break;
        }

        if (strpiped[1] == NULL)
                return 0; // returns zero if no pipe is found.
        else {
                return 1;
        }
}
```

Handle pipe instruction (parsed and parsedpiped) for excution

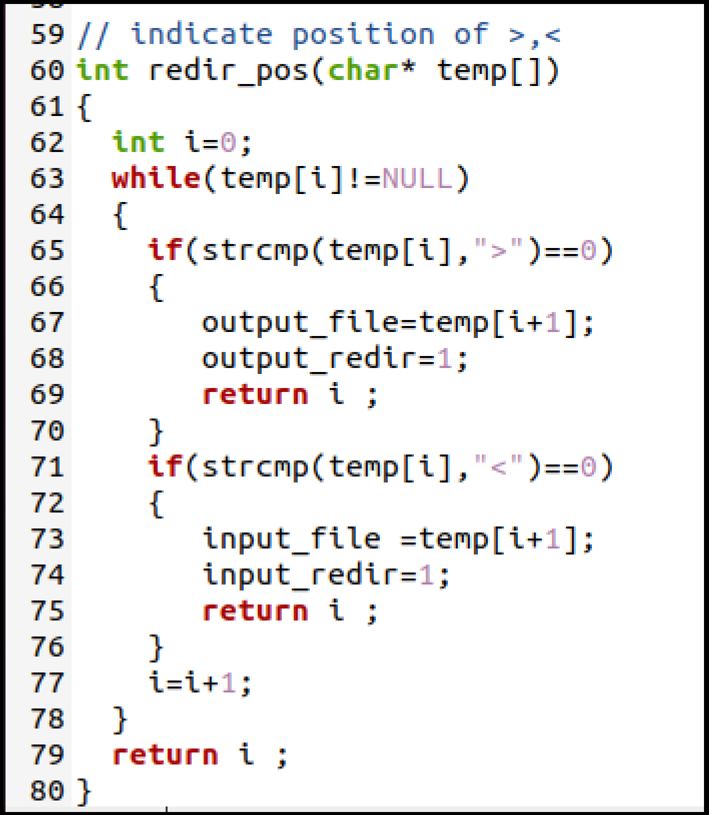
- *If there is pipe put command before '|' in parsed ,after parsedpiped.*
- *If not put all command in parsed.*
- *If there is build-in command pass parsed to ownCmdHandler()*



```
// function to handle pipe instruction
int processString(char* str, char** parsed, char** parsedpipe)
{

        char* strpiped[2];
        int piped = 0;

        piped = parsePipe(str, strpiped);

        if (piped) {
                parseSpace(strpiped[0], parsed);
                parseSpace(strpiped[1], parsedpipe);

        } else {

                parseSpace(str, parsed);
        }

        if (ownCmdHandler(parsed))
                return 0;
        else
                return 1 + piped;
}

```

Indicated position of input or output redirection

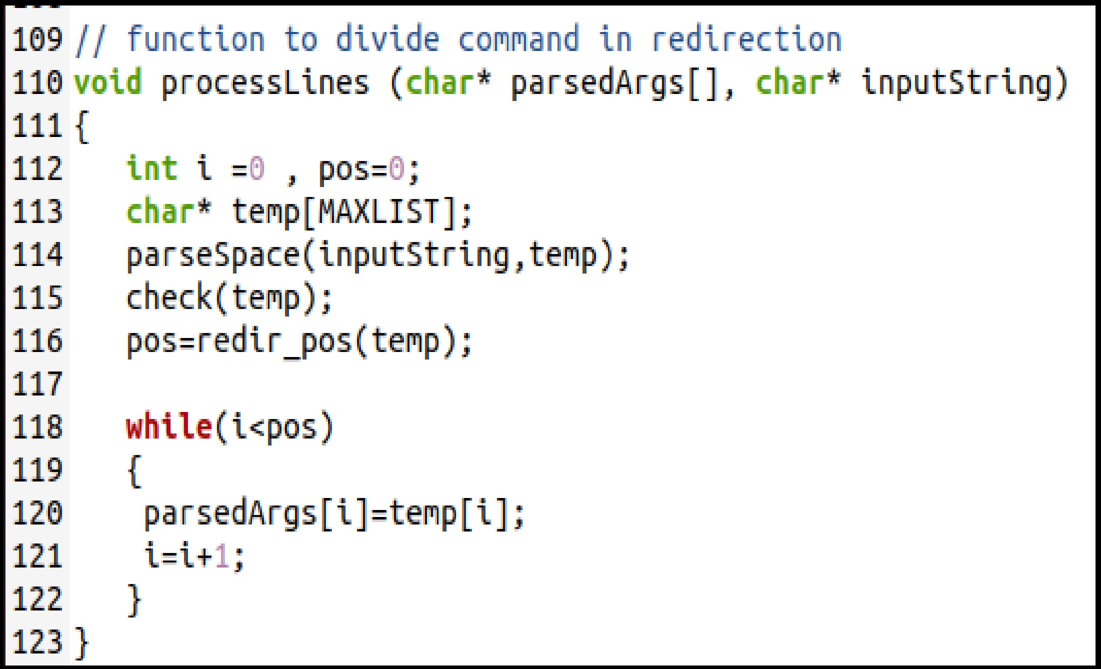
- *Indicate position of input or output redirection and return it.*
- *Store string after '<','>' as a file name for input or output.*
- *If the redirection carries out the flags change to 1.*



```
// indicate position of >,<
int redir_pos(char* temp[])
{
  int i=0;
  while(temp[i]!=NULL)
  {
    if(strcmp(temp[i],">")==0)
    {
      output_file=temp[i+1];
      output_redir=1;
      return i ;
    }
    if(strcmp(temp[i],"<")==0)
    {
      input_file =temp[i+1];
      input_redir=1;
      return i ;
    }
    i=i+1;
  }
  return i ;
}
```

Handel redirection instruction for execution(processlines( ))

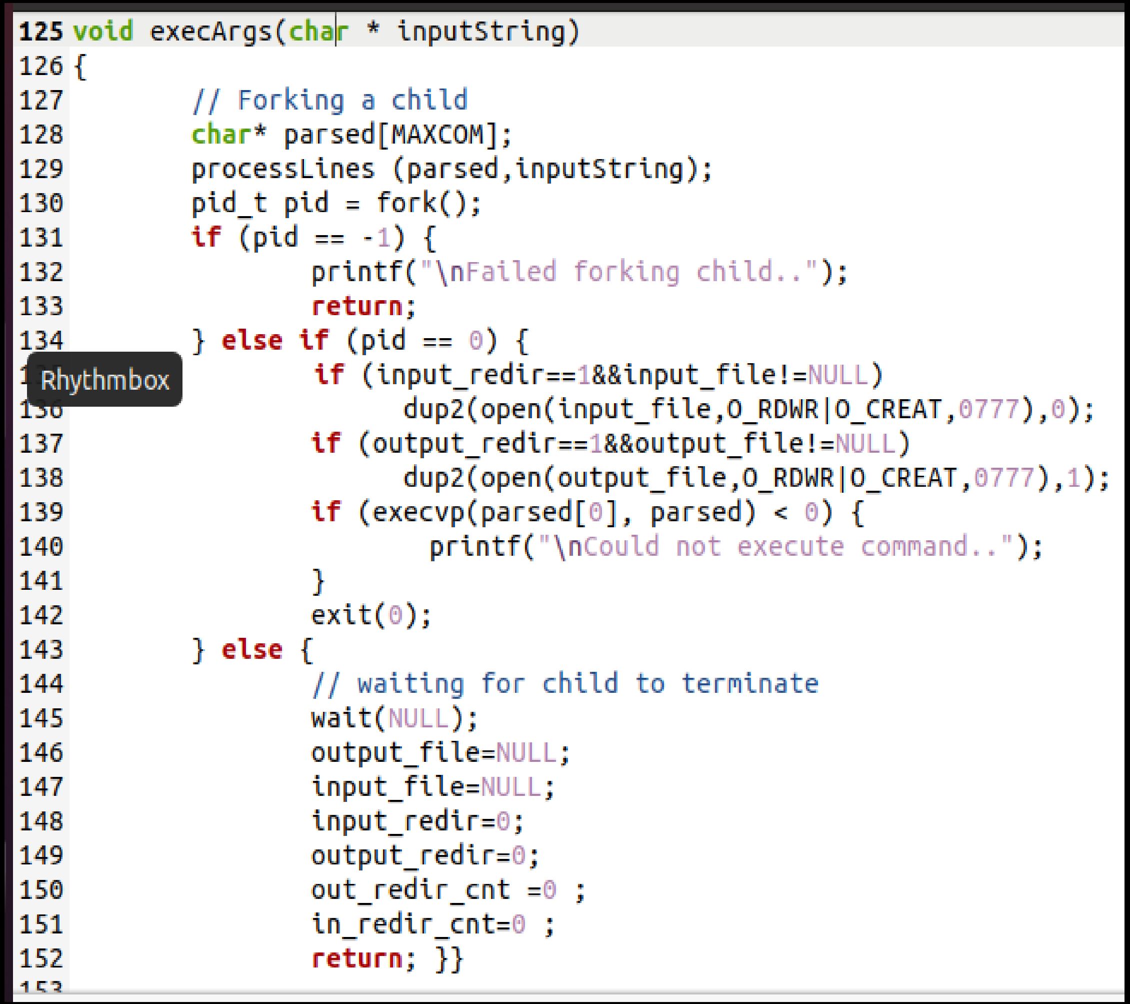
- *If there is redirection put command before '<,>' in parsedArgs to execute.*
- *Pass all commands in check() to certain that there isn't multi redirection or pipe.*



```
// function to divide command in redirection
void processLines (char* parsedArgs[], char* inputString)
{
   int i =0 , pos=0;
   char* temp[MAXLIST];
   parseSpace(inputString,temp);
   check(temp);
   pos=redir_pos(temp);

   while(i<pos)
   {
    parsedArgs[i]=temp[i];
    i=i+1;
   }
}
```

Execution for normal or redirection system commands

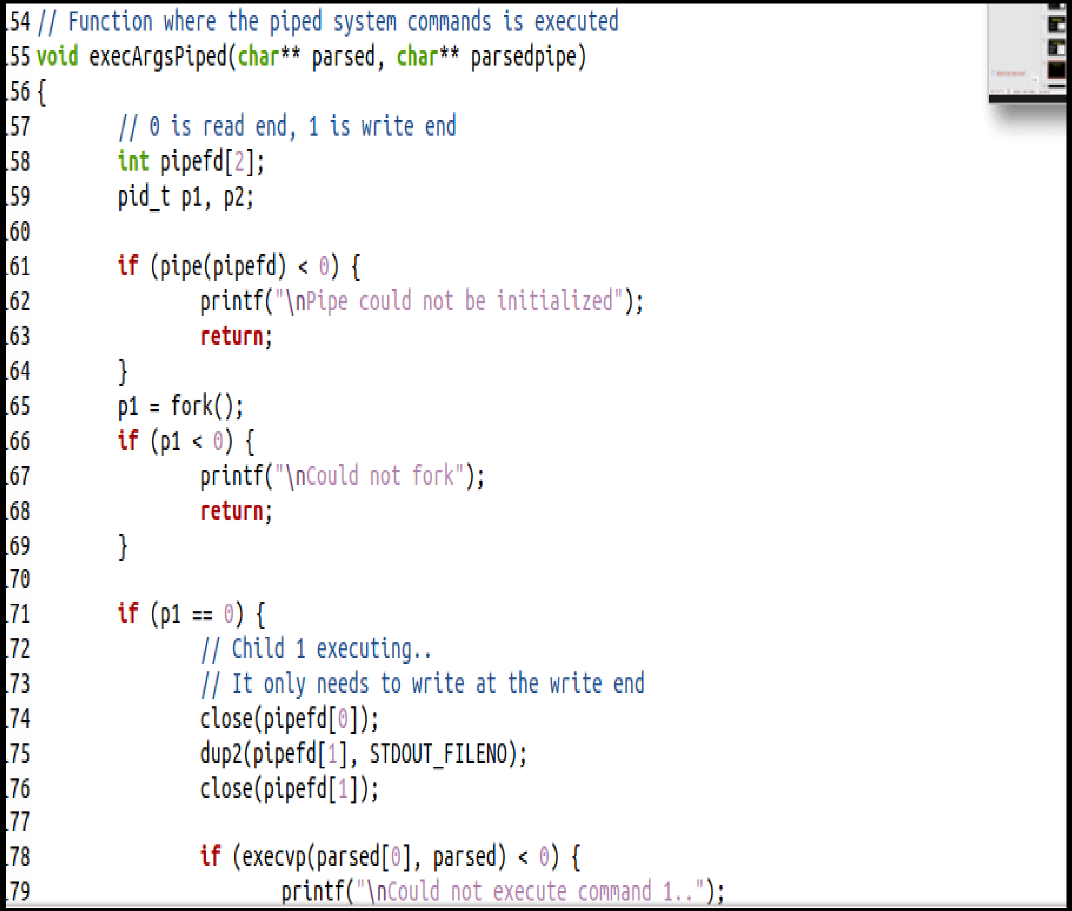
- *Command (parent) creates child using fork().*
- *In child ,if there is input/output redirection the distention is changed to the file in the command to read / write*
- *Execute the command by execvp()"call system"*
- *Parent wait child to finish its execution*



```c
void execArgs(char * inputString)
{
        // Forking a child
        char* parsed[MAXCOM];
        processLines (parsed,inputString);
        pid_t pid = fork();
        if (pid == -1) {
                printf("\nFailed forking child..");
                return;
        } else if (pid == 0) {
                if (input_redir==1&&input_file!=NULL)
                        dup2(open(input_file,O_RDWR|O_CREAT,0777),0);
                if (output_redir==1&&output_file!=NULL)
                        dup2(open(output_file,O_RDWR|O_CREAT,0777),1);
                if (execvp(parsed[0], parsed) < 0) {
                        printf("\nCould not execute command..");
                }
                exit(0);
        } else {
                // waiting for child to terminate
                wait(NULL);
                output_file=NULL;
                input_file=NULL;
                input_redir=0;
                output_redir=0;
                out_redir_cnt =0 ;
                in_redir_cnt=0 ;
                return; }}
```

Execution for piped

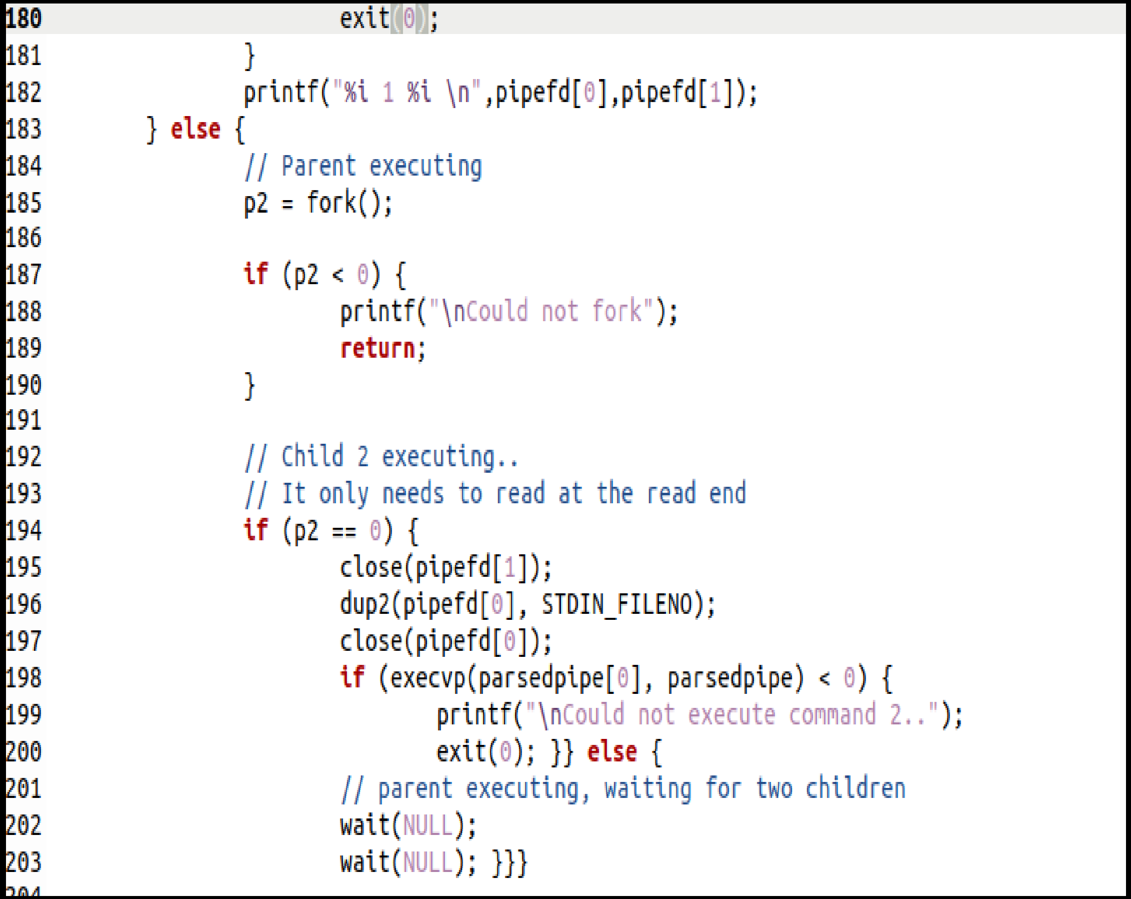
- *main command create (grand parent) child1 (parent).*
- *Right command (parent) creates child2(grand child) using fork().*
- *Left command (grand child) executes by execvp()"call system"*



```
// Function where the piped system commands is executed
void execArgsPiped(char** parsed, char** parsedpipe)
{
	// 0 is read end, 1 is write end
	int pipefd[2];
	pid_t p1, p2;

	if (pipe(pipefd) < 0) {
		printf("\nPipe could not be initialized");
		return;
	}
	p1 = fork();
	if (p1 < 0) {
		printf("\nCould not fork");
		return;
	}

	if (p1 == 0) {
		// Child 1 executing..
		// It only needs to write at the write end
		close(pipefd[0]);
		dup2(pipefd[1], STDOUT_FILENO);
		close(pipefd[1]);

		if (execvp(parsed[0], parsed) < 0) {
			printf("\nCould not execute command 1..");
```

Execution for piped

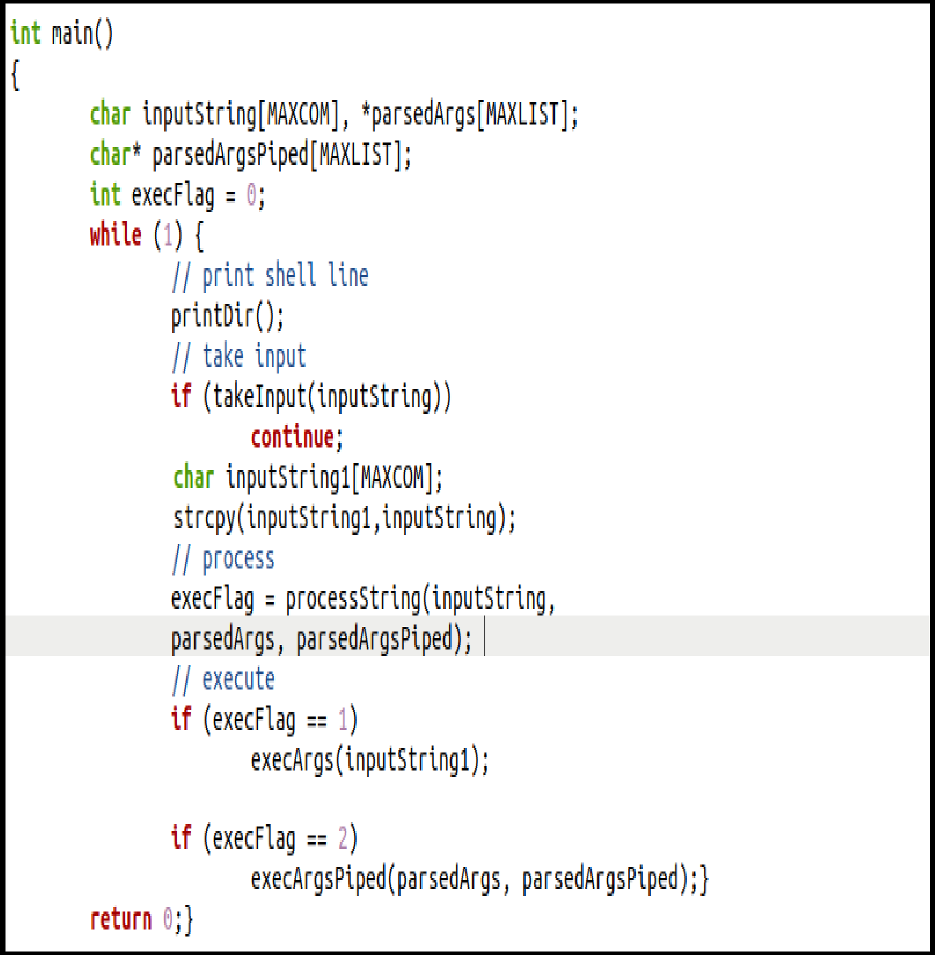
- *Then right command (parent) waits grand child to finish execution then starts to carry out by execvp().*
- *Main command (grand parent) waits two children to finish.*
- *Note that we using pipe() to create shared file between two process for input and output.*



```
                exit(0);
            }
            printf("%i 1 %i \n",pipefd[0],pipefd[1]);
    } else {
            // Parent executing
            p2 = fork();

            if (p2 < 0) {
                printf("\nCould not fork");
                return;
            }

            // Child 2 executing..
            // It only needs to read at the read end
            if (p2 == 0) {
                close(pipefd[1]);
                dup2(pipefd[0], STDIN_FILENO);
                close(pipefd[0]);
                if (execvp(parsedpipe[0], parsedpipe) < 0) {
                    printf("\nCould not execute command 2..");
                    exit(0); }} else {
                // parent executing, waiting for two children
                wait(NULL);
                wait(NULL); }}}
```

Main function

- *Print directory.*
- *allows the user to enter command forever.*
- *If there is pipe call execArgs().*
- *If not call execArgsPiped().*

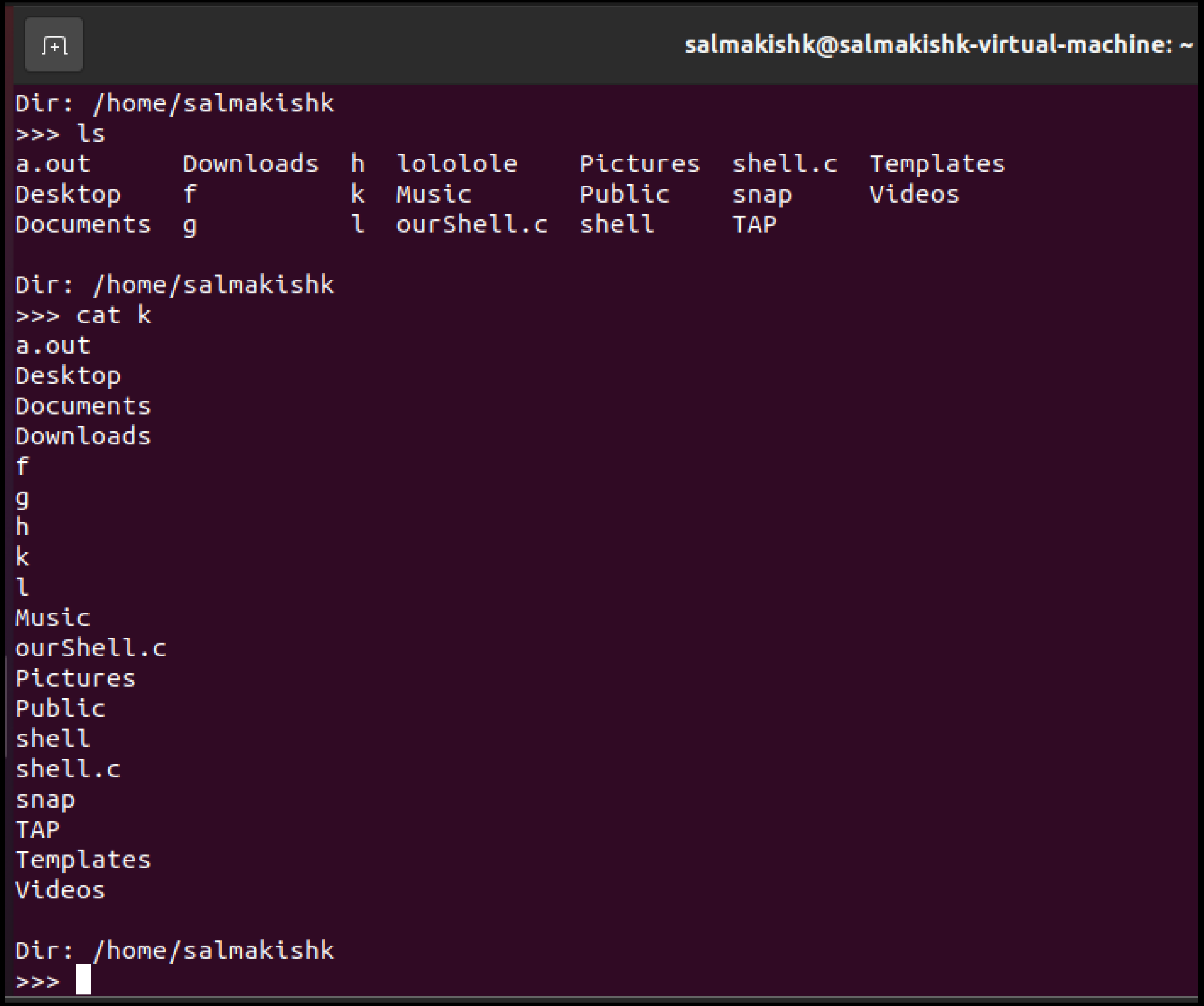


```
int main()
{
        char inputString[MAXCOM], *parsedArgs[MAXLIST];
        char* parsedArgsPiped[MAXLIST];
        int execFlag = 0;
        while (1) {
                // print shell line
                printDir();
                // take input
                if (takeInput(inputString))
                        continue;
                char inputString1[MAXCOM];
                strcpy(inputString1,inputString);
                // process
                execFlag = processString(inputString,
                parsedArgs, parsedArgsPiped);
                // execute
                if (execFlag == 1)
                        execArgs(inputString1);

                if (execFlag == 2)
                        execArgsPiped(parsedArgs, parsedArgsPiped);}
        return 0;}
```

Code running

Ls ---→ list items in current work directory

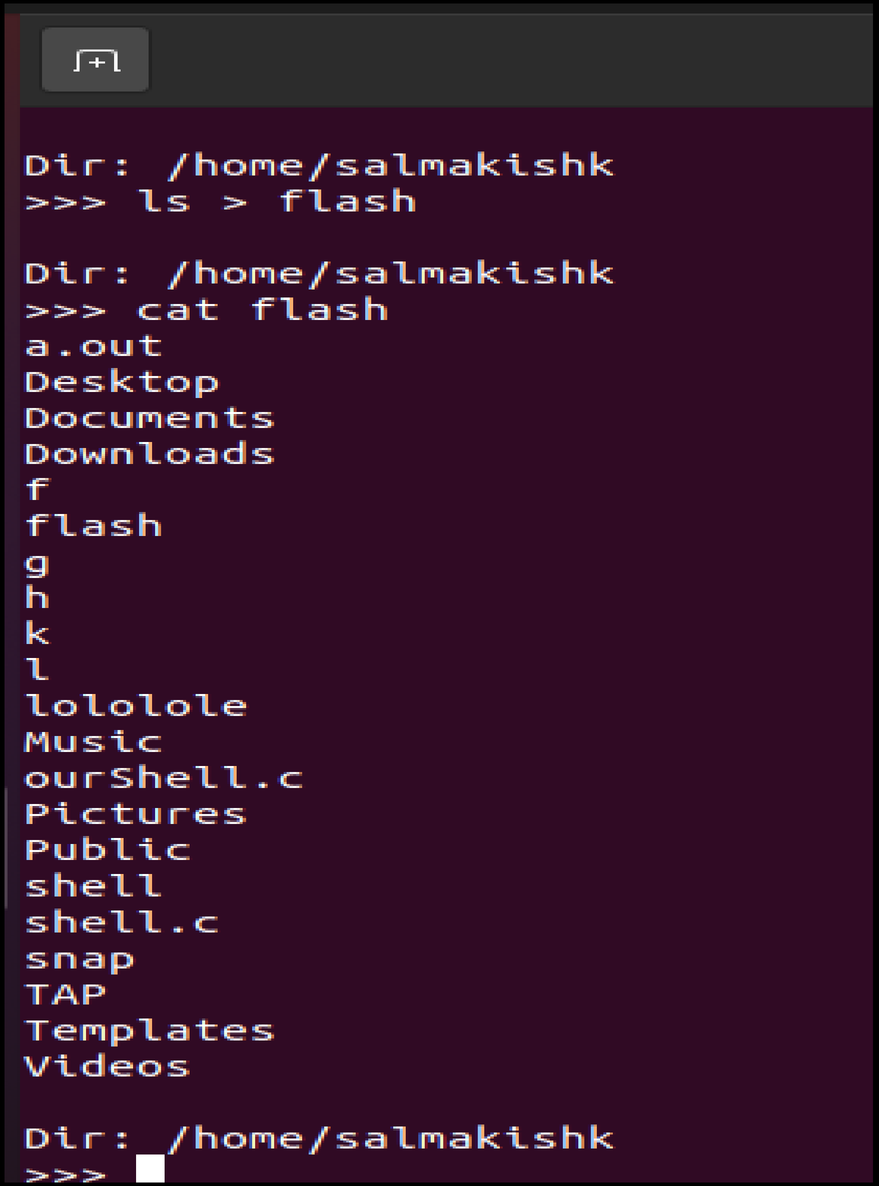
Cat --→ display content of the file



```
Dir: /home/salmakishk
>>> ls
a.out      Downloads  h  lololole    Pictures  shell.c  Templates
Desktop    f          k  Music       Public    snap     Videos
Documents  g          l  ourShell.c  shell     TAP

Dir: /home/salmakishk
>>> cat k
a.out
Desktop
Documents
Downloads
f
g
h
k
l
Music
ourShell.c
Pictures
Public
shell
shell.c
snap
TAP
Templates
Videos

Dir: /home/salmakishk
>>> 
```

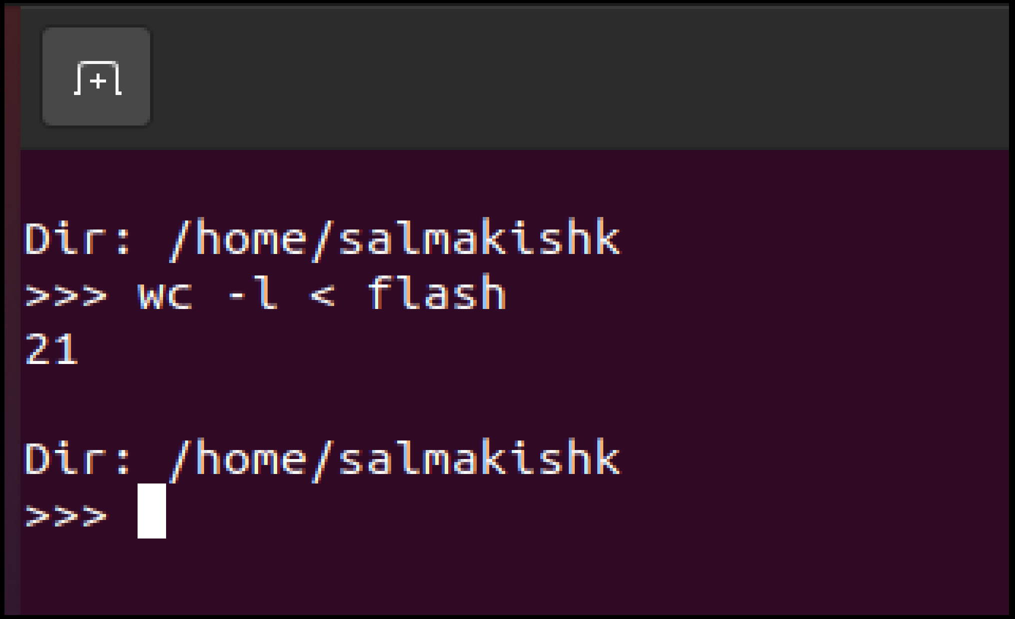
Example on output redirection:

Ls> flash --→ create file its name is flash and write items in current work directory



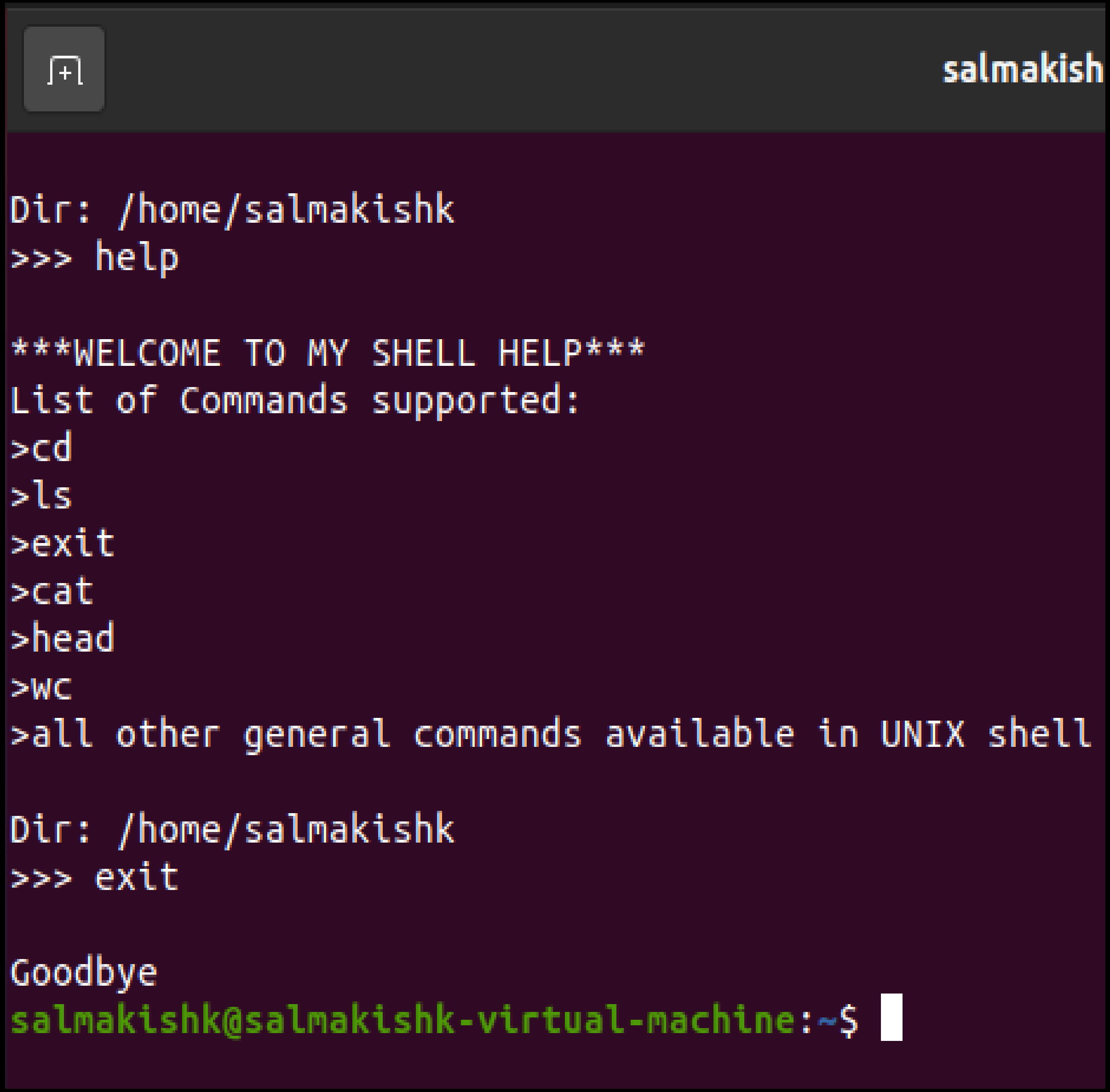
```
Dir: /home/salmakishk
>>> ls > flash

Dir: /home/salmakishk
>>> cat flash
a.out
Desktop
Documents
Downloads
f
flash
g
h
k
l
lololole
Music
ourShell.c
Pictures
Public
shell
shell.c
snap
TAP
Templates
Videos

Dir: /home/salmakishk
>>> 
```

WC – L < flash --→ count number of line in file flash



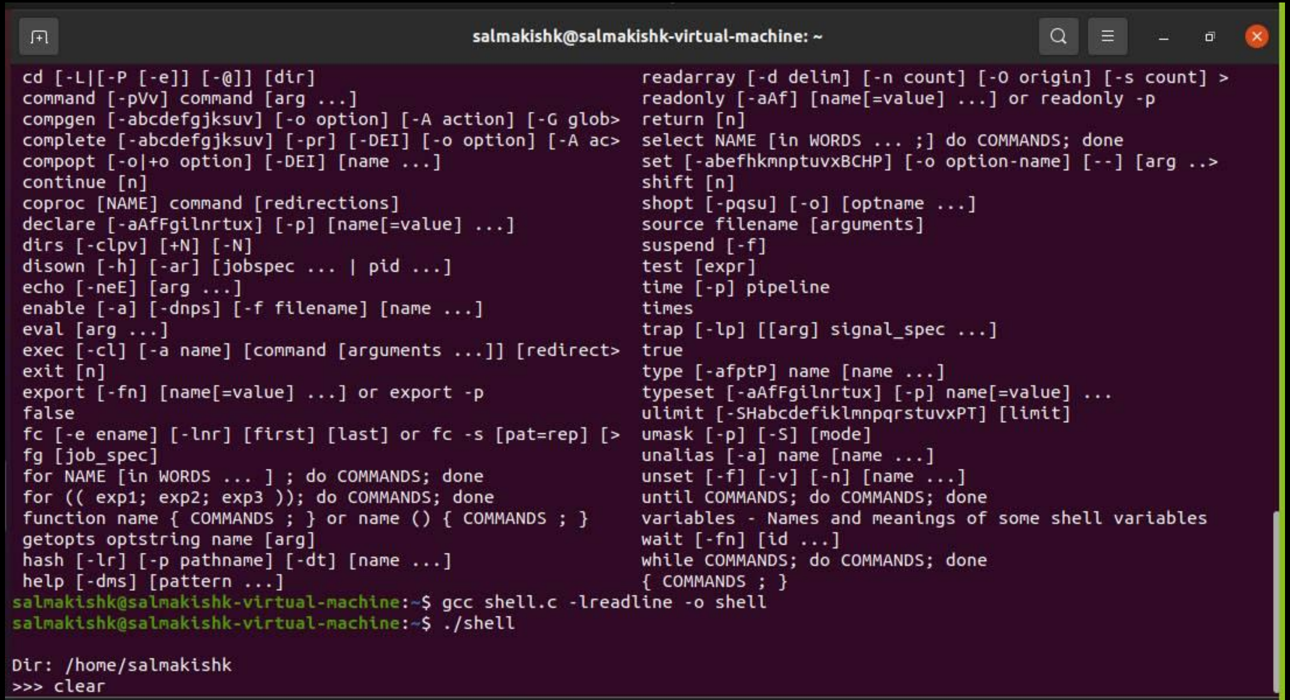
```
Dir: /home/salmakishk
>>> wc -l < flash
21

Dir: /home/salmakishk
>>> 
```

Help command



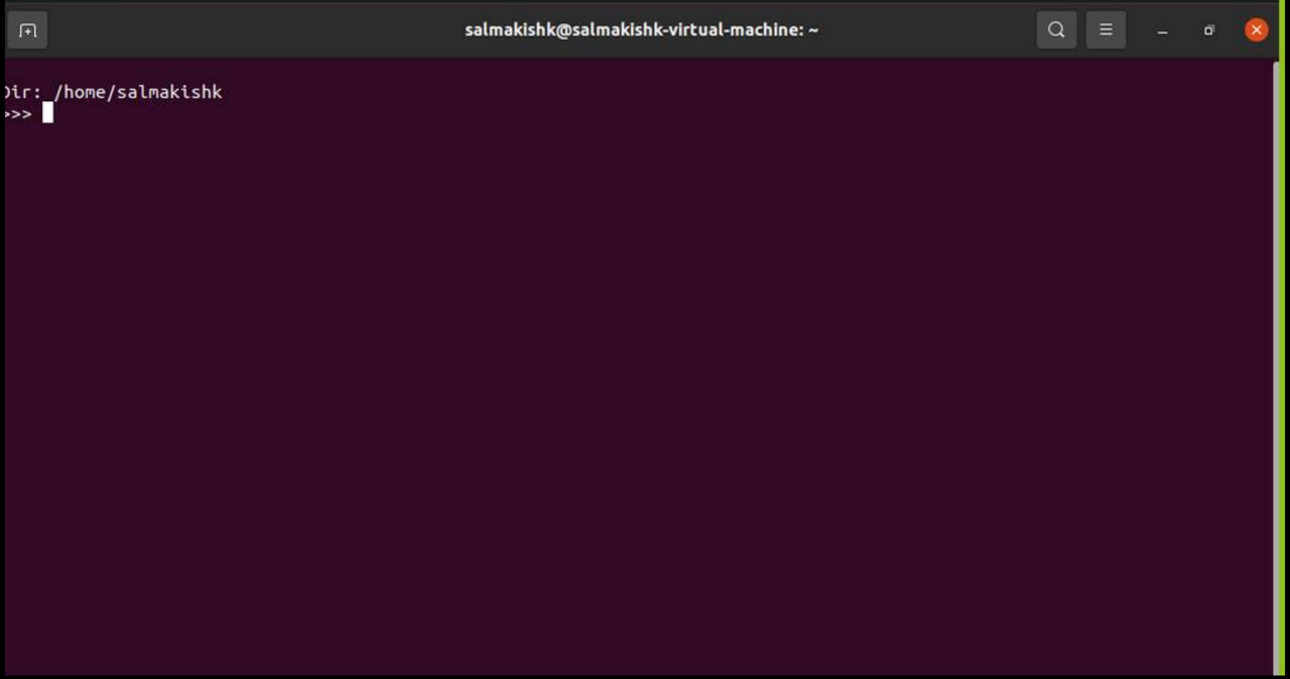
```
Dir: /home/salmakishk
>>> help

***WELCOME TO MY SHELL HELP***
List of Commands supported:
>cd
>ls
>exit
>cat
>head
>wc
>all other general commands available in UNIX shell

Dir: /home/salmakishk
>>> exit

Goodbye
salmakishk@salmakishk-virtual-machine:~$
```

Clear command



salmakishk@salmakishk-virtual-machine: ~

```
cd [-L|[-P [-e]] [-@]] [dir]                                readarray [-d delim] [-n count] [-O origin] [-s count] >
command [-pVv] command [arg ...]                            readonly [-aAf] [name[=value] ...] or readonly -p
compgen [-abcdefgjksuv] [-o option] [-A action] [-G glob>   return [n]
complete [-abcdefgjksuv] [-pr] [-DEI] [-o option] [-A ac>   select NAME [in WORDS ... ;] do COMMANDS; done
compopt [-o|+o option] [-DEI] [name ...]                    set [-abefhkmnptuvxBCHP] [-o option-name] [--] [arg ..>
continue [n]                                                shift [n]
coproc [NAME] command [redirections]                        shopt [-pqsu] [-o] [optname ...]
declare [-aAfFgilnrtux] [-p] [name[=value] ...]             source filename [arguments]
dirs [-clpv] [+N] [-N]                                      suspend [-f]
disown [-h] [-ar] [jobspec ... | pid ...]                   test [expr]
echo [-neE] [arg ...]                                       time [-p] pipeline
enable [-a] [-dnps] [-f filename] [name ...]                times
eval [arg ...]                                              trap [-lp] [[arg] signal_spec ...]
exec [-cl] [-a name] [command [arguments ...]] [redirect>   true
exit [n]                                                    type [-afptP] name [name ...]
export [-fn] [name[=value] ...] or export -p                typeset [-aAfFgilnrtux] [-p] name[=value] ...
false                                                       ulimit [-SHabcdefiklmnpqrstuvxPT] [limit]
fc [-e ename] [-lnr] [first] [last] or fc -s [pat=rep] [>   umask [-p] [-S] [mode]
fg [job_spec]                                               unalias [-a] name [name ...]
for NAME [in WORDS ... ] ; do COMMANDS; done                unset [-f] [-v] [-n] [name ...]
for (( exp1; exp2; exp3 )); do COMMANDS; done               until COMMANDS; do COMMANDS; done
function name { COMMANDS ; } or name () { COMMANDS ; }      variables - Names and meanings of some shell variables
getopts optstring name [arg]                                wait [-fn] [id ...]
hash [-lr] [-p pathname] [-dt] [name ...]                   while COMMANDS; do COMMANDS; done
help [-dms] [pattern ...]                                   { COMMANDS ; }
salmakishk@salmakishk-virtual-machine:~$ gcc shell.c -lreadline -o shell
salmakishk@salmakishk-virtual-machine:~$ ./shell

Dir: /home/salmakishk
>>> clear
```

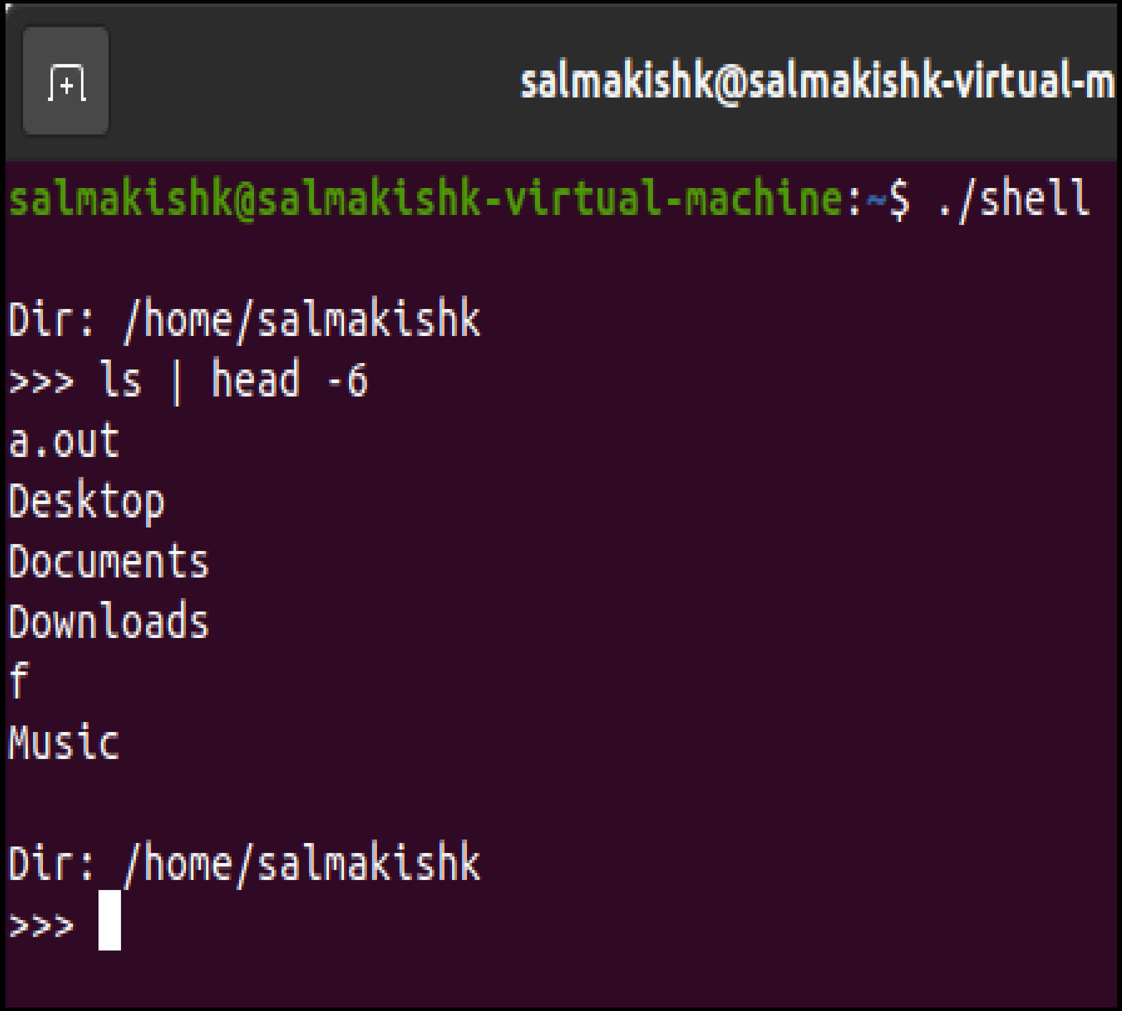


salmakishk@salmakishk-virtual-machine: ~

```
Dir: /home/salmakishk
>>> 
```

Example on piped command:

ls | head -6 --→ ls list items in current work directory in file then head command read first 6 items in this file



```
salmakishk@salmakishk-virtual-machine:~$ ./shell

Dir: /home/salmakishk
>>> ls | head -6
a.out
Desktop
Documents
Downloads
f
Music

Dir: /home/salmakishk
>>> 
```

Thank you