The code is organized as follows:

1-we included the libraries that we would use

2-Initialize our common variables

3-defining the parse function that gets takes raw strings from user and then translates it into code lines and we have in it two cases for handling (“&”)in order to understand that if the process should run in foreground or background

4-defining the execution function with two cases ,one for foreground and the second for background and in both if there is error in command (execution error is printed)

5-defining the function readcmd that compares the number of letters entered by the user with 512 if above it prints (error exceeding max size).

6-defining the main function that reads the input first then compare with max number and finally handle special cases.

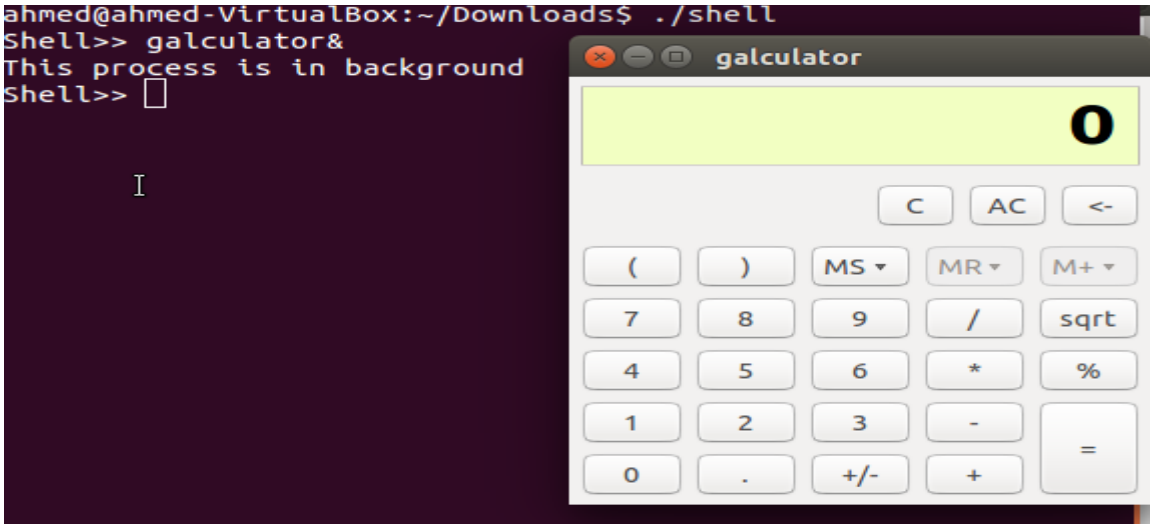
parse	execute	readcmd	main
Containing the part of analyzing user’s data	Containing the part that creates the processes in both foreground and background	Containing the part of comparing the input data with 512 letters number so as not to exceed	Calling all the previous functions and performing the exit part

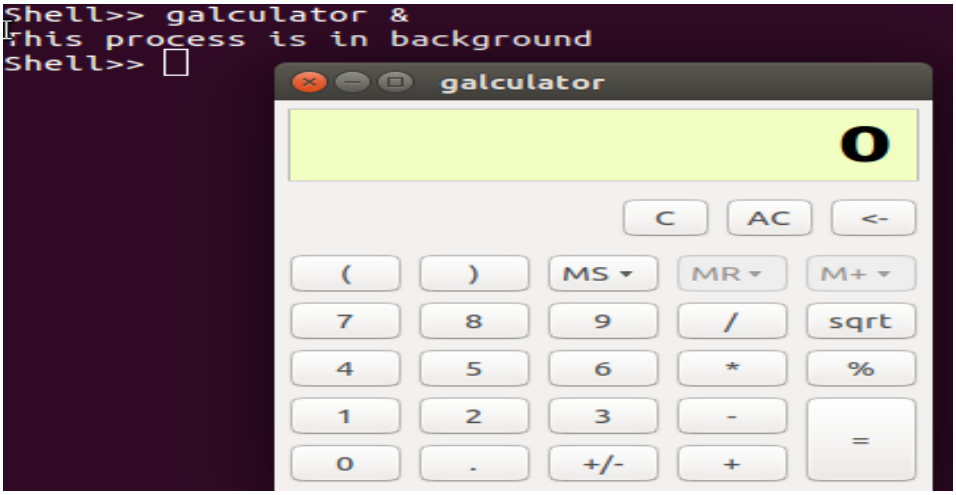
Code run and compilation:

1-Type in terminal (gcc shell.c –o shell) in order to compile

2-Type(./shell) in order to start running

Some sample runs:






```
Shell>> ls -l -a
total 409328
drwxr-xr-x  3 ahmed ahmed      4096 أبريل 21 23:05 .
drwxr-xr-x 25 ahmed ahmed      4096 أبريل 21 22:42 ..
-rw-r--r--  1 ahmed ahmed      1329 أبريل 12 03:04 001- shell_fil
es.zip
-rw-r--r--  1 ahmed ahmed 419038579 يوليو 30  2016 Anaconda2-4.1.
1-Linux-x86_64.sh
-rw-rw-r--  1 ahmed ahmed         0 أبريل 21 00:35 a.out
-rwxrwxr-x  1 ahmed ahmed      9131 أبريل 20 22:21 err
-rwxrwxr-x  1 ahmed ahmed      9105 أبريل 20 22:47 err2
-rw-r--r--  1 ahmed ahmed      2494 أبريل 20 21:59 error1.c
-rw-r--r--  1 ahmed ahmed      2492 أبريل 20 21:57 error1.c~
-rw-r--r--  1 ahmed ahmed      2494 أبريل 20 22:31 error2.c
-rw-r--r--  1 ahmed ahmed      2492 أبريل 20 22:29 error2.c~
-rw-r--r--  1 ahmed ahmed      2352 أبريل 20 21:53 error.c
-rw-r--r--  1 ahmed ahmed      2352 أبريل 14 21:42 error.c~
-rwxrwxr-x  1 ahmed ahmed     13260 أبريل 21 23:05 shell
-rw-r--r--  1 ahmed ahmed      3798 أبريل 21 20:20 shell1.c
drwxrwxr-x  2 ahmed ahmed      4096 أبريل  9 18:58 shell2
```

```
Shell>> cp shell.c program.c
Shell>> cp fvdsdvcs dfscsdddc
cp: cannot stat ‘fvdsdvcs’: No such file or directory
Shell>>
```

```
Shell>> gedit &.txt
Shell>> gedit &.txt &
This process is in background
Shell>>
```

```
Shell>> ls -a -a -a -a -a -a -a -a -a -a -a -a
.                                   error.c
..                                  error.c~
001- shell_files.zip                program.c
Anaconda2-4.1.1-Linux-x86_64.sh     shell
a.out                               shell1.c
err                                 shell2
err2                                shell.c
error1.c                            shell.c~
error1.c~                           shell_copy.c
error2.c                            shell_with_512_error.c
error2.c~
Shell>>
```

```
Shell>> cd ..
Shell>> ls
ahmed
Shell>> cd /home/ahmed/Desktop/shell2
Shell>> ls
a.out  shell2  shell.c  shell.c~  simpleShell_tests_2017.txt
Shell>>
```

```
Shell>> ls
001- shell_files.zip                error.c~
Anaconda2-4.1.1-Linux-x86_64.sh     program.c
a.out                               shell
err                                 shell1.c
err2                                shell2
error1.c                            shell.c
error1.c~                           shell.c~
error2.c                            shell_copy.c
error2.c~                           shell_with_512_error.c
error.c
Shell>> exit
Terminated
ahmed@ahmed-VirtualBox:~/Downloads$
```

```
#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdbool.h>

#define BUFFER_SIZE        512 //decalre buffer used for max size easily to change for entire code
```

```
#define MAX_INPUT_LENGTH      512 //max input too

int T;                //initialize T a checking variable of not true commands

unsigned char bgflag=0;  //initializing background flag status with 0


void parse(char *line, char **argv)  // Parses the line array into a vector of arguments

{

        while(*line != '\0')           //as long as the line hasn't end keep going

        {

                while(*line == ' ' || *line == '\t' || *line == '&')  // Replace separators by terminators

                {           if((*line=='&' && *(line+1)=='\n')||(*line=='&' && *(line+1)==' ' &&
*(line+2)=='\n')||(*line=='&' && *(line+1)=='\t' && *(line+2)=='\n'))

                                        //scanning for special separators (like space ,&.|,and &)

                                        {

                                                bgflag=1;  //set the flag to indicate running in background

                                                return;    //end function

                                        }

                                else if(*line=='&')

                                        {

                                                break;   //break from if statement .. & is considered just an
argument not a command

                                        }


                                else if((*line==' ' && *(line+1)=='\n')||(*line=='\t' &&
*(line+1)=='\n'))//check for space and tap

                                        {

                                                *line='\0';    //replace by end array

            return;       //end function


                                        }
```

```
                else
                        {
                                *line = '\0'; //end the array line to indicate end of single argument
                                line++; //
                        }

        }
        // Save the pointer to the argument
        *argv = line;
        argv++;
        while(*line != ' ' && *line != '\t' && *line != '\0')   // Skip over the rest of the characters of the argument
        {
                if((*line=='&' && *(line+1)=='\n')||(*line=='&' && *(line+1)==' ' && *(line+2)=='\n')||(*line=='&' && *(line+1)=='\t' && *(line+2)=='\n'))
                        {
                                bgflag=1;  //set the flag to run in background
                                *line='\0';
                        }
                if (*line =='\n')
                        {
                                *line = '\0';
                        }
                line++;
        }
    }
```

```
        *argv = '\0';
}

void execute(char **argv)  // Forks a child process to run execvp
{
        pid_t pid;
        pid = fork();
        if(pid < 0)  // Fork failed
                {
                        printf("Error, no fork\n");
                }
        else if (pid == 0)  // In child process
        {
                execvp(argv[0], argv); //execute the command
    T=execvp(argv[0], argv);//check for error
         if (T<0 && argv[0]!='\0')
         printf("command is not exist or cannot be executed\n");
        }
        else  // In parent process
        {       if(bgflag==0)
                        {
                                wait(NULL); //in foreground
                        }
                else
                        {
                                printf("This process is in background\n");  //in background
                                bgflag=0; //set flag to 0 again for the next user commands
```

```
                        }
        }

}
//-----------------------------------------------------------------------------------------

//-------------------------------------------------------------------------------------------------
/*
function name     : readcmd
description       : a function which prevents user of entering more than 512 characters in command line

inputs:
cmd               : char pointer : pointer which will carry the output of fgets function
maxInputsize      : integer which holds maximum length of characters to be entered
buffersize        : integer which holds maximum length that buffer can carry
*/
bool readcmd(char *cmd , int maxInputsize, int bufferSize)
{

        int counter = 0;
        fgets(cmd, bufferSize, stdin); //geting line command
        while(cmd[counter] != '\0')
        {
                if(counter > maxInputsize)
                        {
                                printf("Input exceeds the max input size = %d \n", maxInputsize);
                                return false;
```

```
                        }
                        counter ++;
        }
        return true;
}

int main()
{
        pid_t pid;
        char line[BUFFER_SIZE];//previously delcared
        char *argv[128];
        while(1)  // Main loop
        {
                printf("Shell>> ");
                if(readcmd(line, MAX_INPUT_LENGTH, BUFFER_SIZE) == true)
                {
                        parse(line, argv);

                        //find(argv);

                        if(strcmp(argv[0], "cd") == 0)   // An example of a shell built-in .. change
directory
                                {/*if (argv[1]=="--"){
                  argv[1]="..";
                 }*/
                  if( chdir(argv[1])!=0){
                 printf("no such directory\n");}
                  else{
```

```
                    chdir(argv[1]);}
                }
            else if(strcmp(argv[0], "\0") == 0) // An example of empty command
                {
                    continue;
                }
            else if(strcmp(argv[0], "exit") == 0) // An example of exit command
                {
                     kill(pid, SIGTERM);//kill child
    sleep(2);
    kill(pid, SIGKILL);//kill parent
                }
            else
                execute(argv);
        }
}

return 0;
                                    }
```