LINUX SHELL

NAME: AMARTYA CHOUDHURY

ROLL: 001610501026

BCSE-III

JADAVPUR UNIVERSITY

Problem statement:

Develop a linux shell JUBCSEIII,that will display a prompt,accept user commands and execute them.

The overview of the functions of JUBCSEIII is as follows:

- At startup,the shell will display the welcome message:Hi!Good <...> according to the time of the day and then the prompt string :BCSE!! ,and ready to accept command.
- A command line has the following syntax:command[argument/s]
- When a valid command is entered by the user,the shell will execute the command.Commands may run in the foreground as well as in the background.
- The shell will return an appropriate error message when the command is invalid,or when there are problems with either the arguments or the execution of the command.

The commands to be executed by JUBCSEIII are given below:

1. **newdir** *directoryname* <action:creates new directory>
2. **editfile** [*filename*] ([ ] denotes argument is optional) <action:the file will be opened with the vi editor if filename is given otherwise the vi editor for a new file will be opened>
3. **content** *filename* <action:prints content of a file on the screen>
4. **info** [*filename*] ([ ] denotes argument is optional) <action: displays essential information about the file specified,it must list the following:
   - Full path of the file
   - Size of the file
   - Last modification date
   - Name of the creator
5. **exitbcse** <action: this will quit the shell>

Using *fork* and *exec*

The DOS and Windows API contains the spawn family of functions.These functions take as an argument the name of a program to run and create a new process instance of that program. Linux doesn't contain a single function that does all this in one step.Instead, Linux provides one function, fork, that makes a child process that is an exact copy of its parent process. Linux provides another set of functions, the exec family, that causes a particular process to cease being an instance of one program and to instead become an instance of another program.To spawn a new process, you first use fork to make a copy of the current process.Then you use exec to transform one of these processes into an instance of the program you want to spawn.

Calling *fork*

When a program calls fork, a duplicate process, called the *child process*, is created.The parent process continues executing the program from the point that fork was called. The child process, too, executes the same program from the same place. So how do the two processes differ? First, the child process is a new process and therefore has a new process ID, distinct from its parent's process ID.One way for a program to distinguish whether it's in the parent process or the child process is to call getpid. However, the fork function provides different return values to the parent and child processes—one process "goes in" to the fork call, and two processes "come out," with different return values.The return value in the parent process is the process ID of the child.The return value in the child process is zero. Because no process ever has a process ID of zero, this makes it easy for the program whether it is now running as the parent or the child process.

Using the *exec* Family

The exec functions replace the program running in a process with another program.When a program calls an exec function, that process immediately ceases executing that program and begins executing a new program from the beginning, assuming that the exec call doesn't encounter an error.

Using *fork* and *exec* Together

A common pattern to run a subprogram within a program is first to fork the process and then exec the subprogram.This allows the calling program to continue execution in the parent process while the calling program is replaced by the subprogram in the child process.

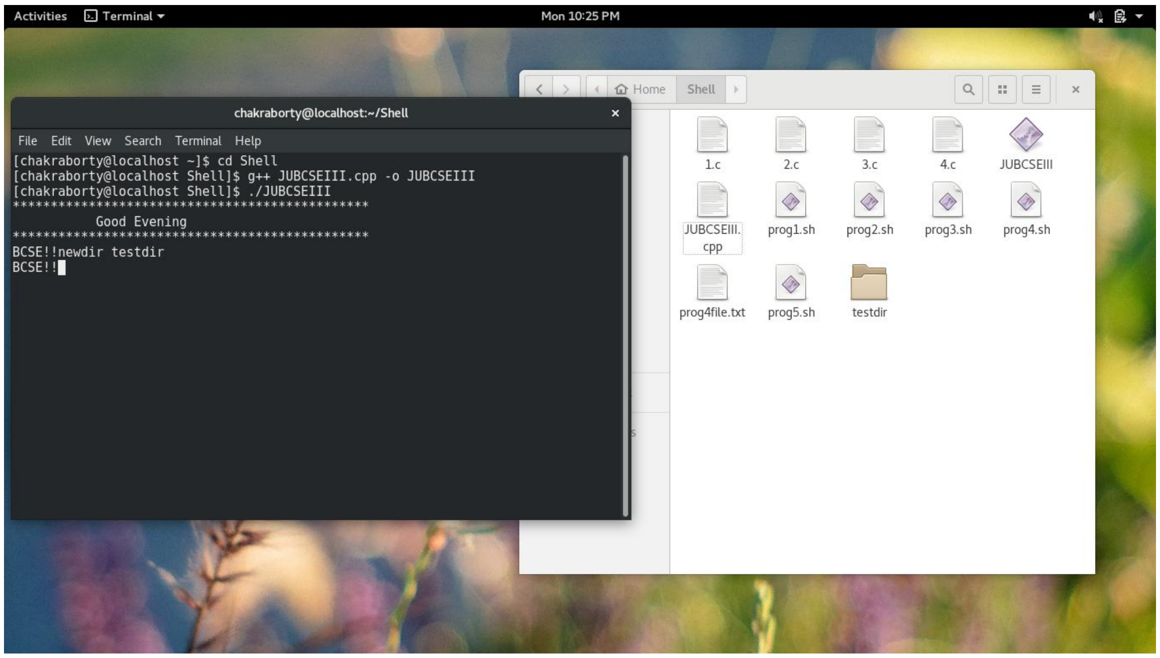
C++ Program :

```
#include<iostream>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<bits/stdc++.h>
using namespace std;

/*flag is_BG is set true if input command needs to be run in background*/
bool  is_BG=false;



/* Greet the user with "Good Morning","Good Afternoon" or "Good Evening" ,according to
the local time */

void init_shell(){
            time_t curTime=time(NULL);
            struct tm *cTime=localtime(&curTime);
            int hr=cTime->tm_hour;
            string tmofDay;
            if(hr<12)tmofDay="Morning";
            else if(hr>=12 && hr<19) tmofDay="Afternoon";
            else tmofDay="Evening";
            cout<<"**************************************************\n";
            cout<<"          "<<"Good "<<tmofDay<<endl;
            cout<<"**************************************************\n";
            }

/* INPUT  is the space separated input command(1st argument);
   ARGS is array of char* pointers(2nd argument) to store program name and arguments.
   The input command is parsed into its components (program name and arguments) and
stored in the 2-D char array .
   If the command is to be run in background(ends with ampersand("&") ,is_BG flag is set.
*/

void processString(char* input,char **args){
            int i=0;
            args[i]=new char[100];

            /*break the input command into words(space acts as delimiter) and store it in 2-
D character array*/

            args[i]=strtok(input," ");
            while(args[i]!=NULL){
              i++;
              args[i]=strtok(NULL," ");
            }

            //replace the program name(command) with equivalent linux shell command
```

```
                if(strcmp(args[0],"newdir")==0){
                                string s("mkdir");
                                strcpy(args[0],s.c_str());
                                }
                else if(strcmp(args[0],"editfile")==0){
                                string s("vi");
                                strcpy(args[0],s.c_str());
                }
                else if(strcmp(args[0],"content")==0){
                                string s("cat");
                                strcpy(args[0],s.c_str());
                }
                else if(strcmp(args[0],"info")==0){
                                string s("stat");
                                strcpy(args[0],s.c_str());
                }

                /*check if command asks to be run in background,If true,set background flag
                and trim the command*/

                if(strcmp(args[i-1],"&")==0){
                        is_BG=1;
                    args[i-1]=NULL;
                }
}

 /*Spawn a child process running a new program.
 "program" is the name of program to run;
 "arg_list" is null-terminated list of character strings to be passed as program's argument
list.
 returns process Id of spawned process. */


int spawn(char *program,char **arg_list){

                pid_t child_pid;
                /* duplicate this  process*/
                child_pid=fork();

                if(child_pid!=0){
                        /*this is parent process*/

                        /*if the program is to be run on the foreground ,
                        wait till the child process completes its execution.
                        Otherwise,if it is a background process,do not wait.
                        Set background flag as false for next command */

                        if(!is_BG){
                                waitpid(child_pid,nullptr,0);
                                }
                        is_BG=0;
                        return child_pid;
                        }
                else{
                                /*child process*/

                       /*Execute program searching for it in the path*/
                       execvp(program,arg_list);

                       /*execvp returns only if error occured*/

                                fprintf(stderr,"invalid command\n");
                                abort();
                   }

                }
int main(){

                init_shell();
        string input;
                char  inputString[1000];
                while(1){

      /*prompt "BCSE!!" to take commands*/
                  cout<<"BCSE!!";

                  /*get user input from stdin*/
                  getline(cin,input);

                  /*convert input string to char array*/
                  char inputString[input.length()+1];
                  strcpy(inputString,input.c_str());
                  char *parsedArgs[100];

                  /*parse the command*/
                  processString(inputString,parsedArgs);

                  /*if command is "exitbcse" , quit shell ;otherwise execute the command using
fork() and exec() */
                  if(strcmp(parsedArgs[0],"exitbcse")==0) break;
                  spawn(parsedArgs[0],parsedArgs);
                }
                return 0;
}
```

Output:

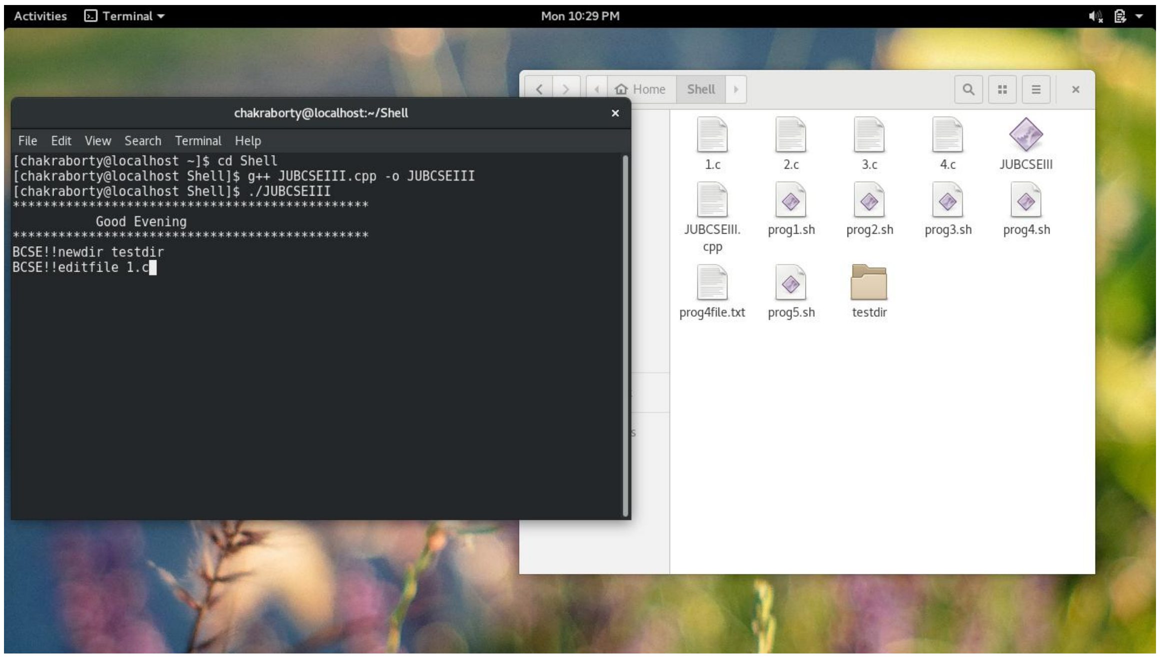
1.



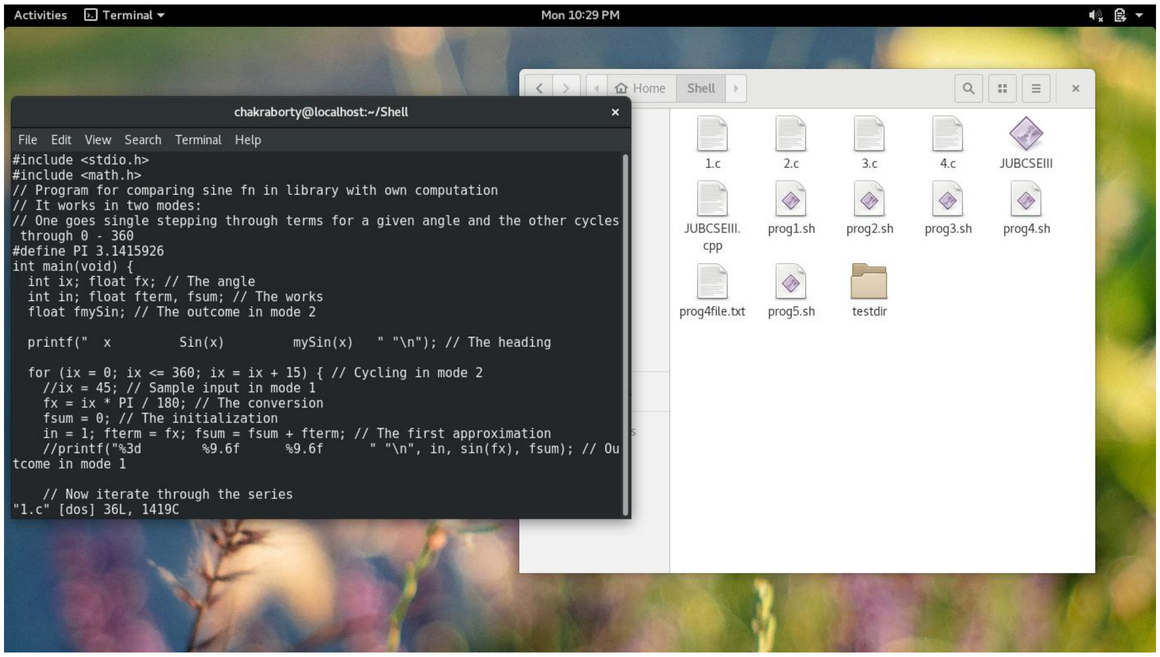
```
[chakraborty@localhost ~]$ cd Shell
[chakraborty@localhost Shell]$ g++ JUBCSEIII.cpp -o JUBCSEIII
[chakraborty@localhost Shell]$ ./JUBCSEIII
***********************************************
           Good Evening
***********************************************
BCSE!!newdir testdir
BCSE!!
```

"newdir" creates new folder in present working directory

2.



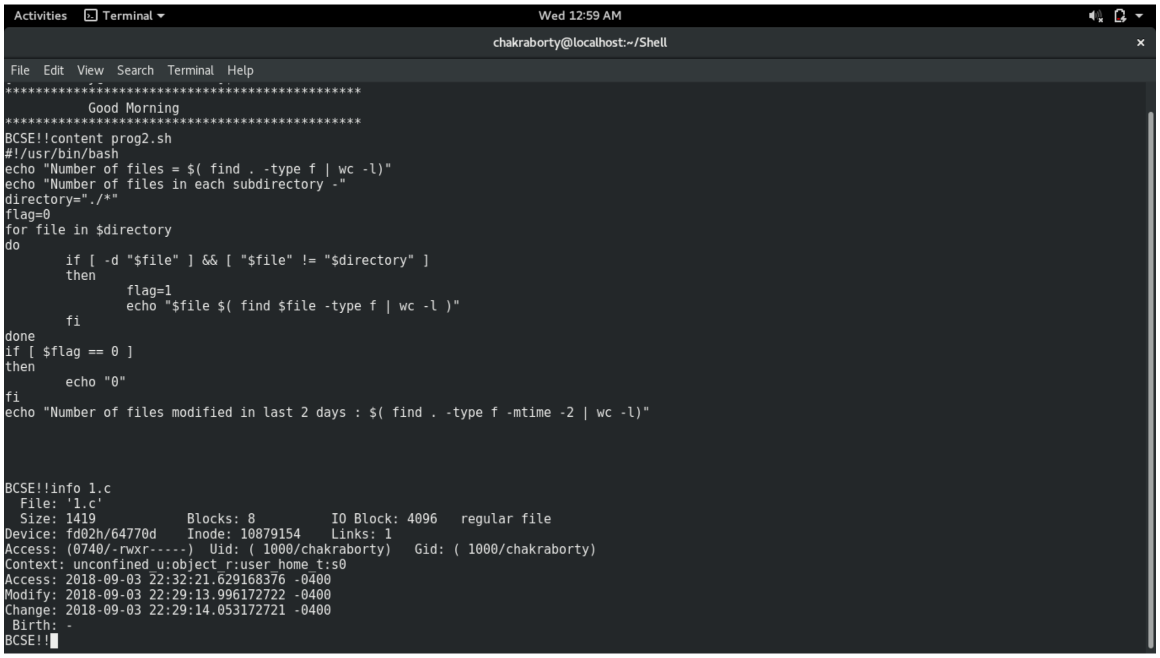
```
[chakraborty@localhost ~]$ cd Shell
[chakraborty@localhost Shell]$ g++ JUBCSEIII.cpp -o JUBCSEIII
[chakraborty@localhost Shell]$ ./JUBCSEIII
***********************************************
           Good Evening
***********************************************
BCSE!!newdir testdir
BCSE!!editfile 1.c
```



```
#include <stdio.h>
#include <math.h>
// Program for comparing sine fn in library with own computation
// It works in two modes:
// One goes single stepping through terms for a given angle and the other cycles
 through 0 - 360
#define PI 3.1415926
int main(void) {
  int ix; float fx; // The angle
  int in; float fterm, fsum; // The works
  float fmySin; // The outcome in mode 2

  printf("  x         Sin(x)         mySin(x)   " "\n"); // The heading

  for (ix = 0; ix <= 360; ix = ix + 15) { // Cycling in mode 2
    //ix = 45; // Sample input in mode 1
    fx = ix * PI / 180; // The conversion
    fsum = 0; // The initialization
    in = 1; fterm = fx; fsum = fsum + fterm; // The first approximation
    //printf("%3d        %9.6f      %9.6f      " "\n", in, sin(fx), fsum); // Ou
tcome in mode 1

    // Now iterate through the series
"1.c" [dos] 36L, 1419C
```

"editfile" opens file in vi editor

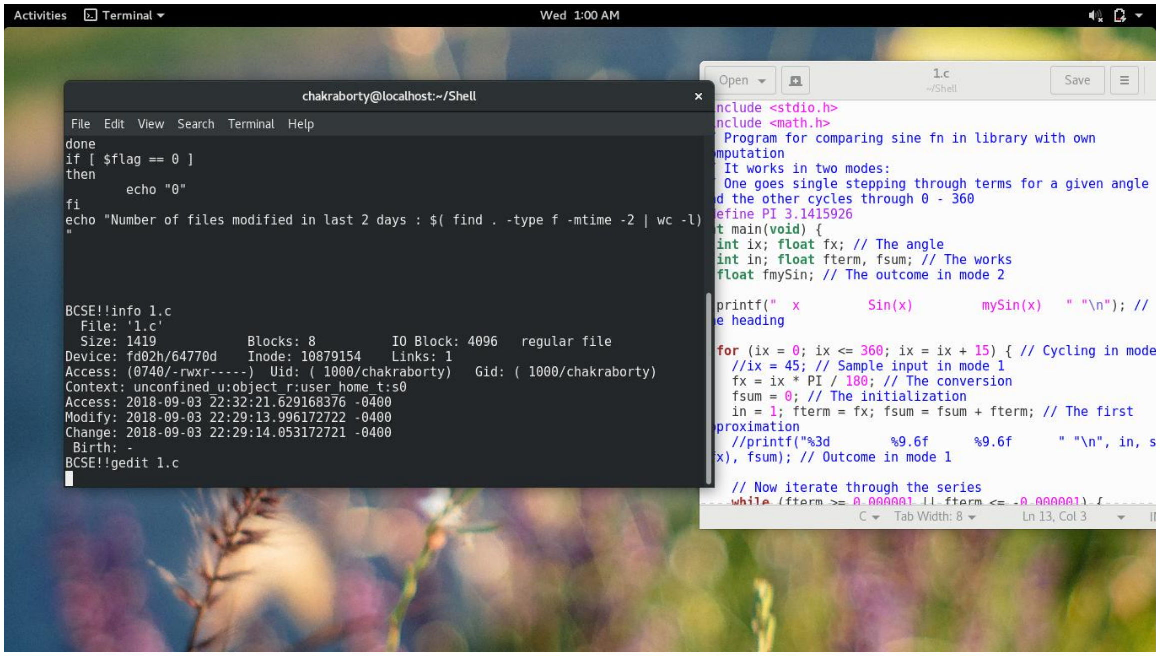
3.


Activities
Terminal
Wed 12:59 AM
chakraborty@localhost:~/Shell
File Edit View Search Terminal Help

```
***********************************************
           Good Morning
***********************************************
BCSE!!content prog2.sh
#!/usr/bin/bash
echo "Number of files = $( find . -type f | wc -l)"
echo "Number of files in each subdirectory -"
directory="./*"
flag=0
for file in $directory
do
        if [ -d "$file" ] && [ "$file" != "$directory" ]
        then
                flag=1
                echo "$file $( find $file -type f | wc -l )"
        fi
done
if [ $flag == 0 ]
then
        echo "0"
fi
echo "Number of files modified in last 2 days : $( find . -type f -mtime -2 | wc -l)"



BCSE!!info 1.c
  File: '1.c'
  Size: 1419            Blocks: 8          IO Block: 4096   regular file
Device: fd02h/64770d    Inode: 10879154    Links: 1
Access: (0740/-rwxr-----)  Uid: ( 1000/chakraborty)   Gid: ( 1000/chakraborty)
Context: unconfined_u:object_r:user_home_t:s0
Access: 2018-09-03 22:32:21.629168376 -0400
Modify: 2018-09-03 22:29:13.996172722 -0400
Change: 2018-09-03 22:29:14.053172721 -0400
 Birth: -
BCSE!!
```

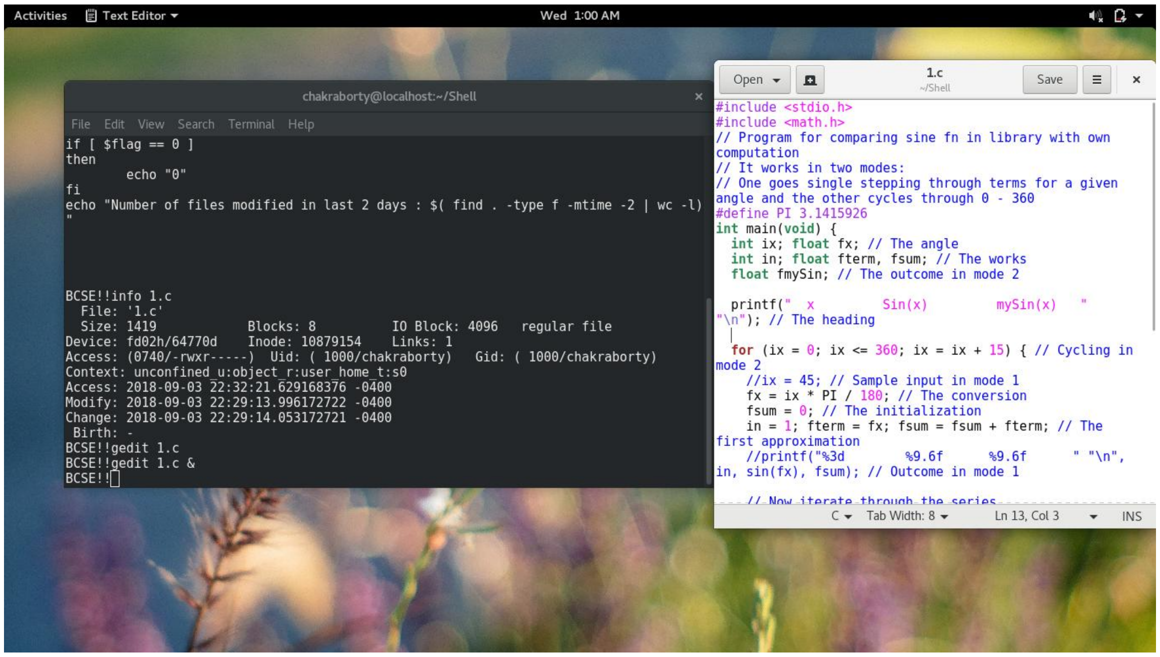
"content" prints content of file on screen
"info" displays information about file specified

4.



```
done
if [ $flag == 0 ]
then
        echo "0"
fi
echo "Number of files modified in last 2 days : $( find . -type f -mtime -2 | wc -l)
"



BCSE!!info 1.c
  File: '1.c'
  Size: 1419            Blocks: 8          IO Block: 4096   regular file
Device: fd02h/64770d    Inode: 10879154    Links: 1
Access: (0740/-rwxr-----)  Uid: ( 1000/chakraborty)   Gid: ( 1000/chakraborty)
Context: unconfined_u:object_r:user_home_t:s0
Access: 2018-09-03 22:32:21.629168376 -0400
Modify: 2018-09-03 22:29:13.996172722 -0400
Change: 2018-09-03 22:29:14.053172721 -0400
 Birth: -
BCSE!!gedit 1.c
```

```
nclude <stdio.h>
nclude <math.h>
 Program for comparing sine fn in library with own
mputation
 It works in two modes:
 One goes single stepping through terms for a given angle
d the other cycles through 0 - 360
efine PI 3.1415926
t main(void) {
int ix; float fx; // The angle
int in; float fterm, fsum; // The works
float fmySin; // The outcome in mode 2

printf("  x          Sin(x)          mySin(x)    " "\n"); //
e heading

for (ix = 0; ix <= 360; ix = ix + 15) { // Cycling in mode
  //ix = 45; // Sample input in mode 1
  fx = ix * PI / 180; // The conversion
  fsum = 0; // The initialization
  in = 1; fterm = fx; fsum = fsum + fterm; // The first
proximation
  //printf("%3d        %9.6f      %9.6f       " "\n", in, s
x), fsum); // Outcome in mode 1

  // Now iterate through the series
  while (fterm >= 0.000001 || fterm <= -0.000001) {
```



```
if [ $flag == 0 ]
then
        echo "0"
fi
echo "Number of files modified in last 2 days : $( find . -type f -mtime -2 | wc -l)
"



BCSE!!info 1.c
  File: '1.c'
  Size: 1419            Blocks: 8          IO Block: 4096   regular file
Device: fd02h/64770d    Inode: 10879154    Links: 1
Access: (0740/-rwxr-----)  Uid: ( 1000/chakraborty)   Gid: ( 1000/chakraborty)
Context: unconfined_u:object_r:user_home_t:s0
Access: 2018-09-03 22:32:21.629168376 -0400
Modify: 2018-09-03 22:29:13.996172722 -0400
Change: 2018-09-03 22:29:14.053172721 -0400
 Birth: -
BCSE!!gedit 1.c
BCSE!!gedit 1.c &
BCSE!!
```

```
#include <stdio.h>
#include <math.h>
// Program for comparing sine fn in library with own
computation
// It works in two modes:
// One goes single stepping through terms for a given
angle and the other cycles through 0 - 360
#define PI 3.1415926
int main(void) {
  int ix; float fx; // The angle
  int in; float fterm, fsum; // The works
  float fmySin; // The outcome in mode 2

  printf("  x          Sin(x)          mySin(x)    "
"\n"); // The heading

  for (ix = 0; ix <= 360; ix = ix + 15) { // Cycling in
mode 2
    //ix = 45; // Sample input in mode 1
    fx = ix * PI / 180; // The conversion
    fsum = 0; // The initialization
    in = 1; fterm = fx; fsum = fsum + fterm; // The
first approximation
    //printf("%3d        %9.6f      %9.6f       " "\n",
in, sin(fx), fsum); // Outcome in mode 1

    // Now iterate through the series
```

Running "gedit" in foreground(above) and background(below)

References

Websites:

- https://www.geeksforgeeks.org/making-linux-shell-c/
- https://stackoverflow.com/questions/20432595/creating-a-background-process-for-shell-in-c

Books:

- "Advanced Linux Programming" by Mark Mitchell, Jeffrey Oldham,and Alex Samuel