10). Read the current directory and display the name of the files, no of files in current directory.

Ans:-

```
     #include <dirent.h>
#include <stdio.h>

int main() {
    DIR *d;
    struct dirent *dir;
    int count = 0;
    d = opendir(".");
    if (d) {
        while ((dir = readdir(d)) != NULL) {
            printf("%s\n", dir->d_name);
            count++;
        }
        closedir(d);
    }
    printf("Number of files: %d\n", count);
    return 0;
}
```

Output:-

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PPL Practical Journal - 1 (1) (1).pdf
pr3.c
pr3.exe
pr9.c
pr9.exe
pra8.c
pra8.exe
PRATICAL.docx
program10.c
program10.exe
tempCodeRunnerFile.c
Number of files: 18
PS C:\Users\Arjun Virendra Yadav\OneDrive\Desktop>
```

11). Write a C program to implement the following unix/linux command (use fork, pipe and exec system call) ls –l | wc –l

Ans:-

```
        #include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main() {
    int fd[2];
    pid_t pid1, pid2;

    // create pipe
    if (pipe(fd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // fork first child (ls -l)
    pid1 = fork();
    if (pid1 == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid1 == 0) {
        // redirect stdout to write end of pipe
        dup2(fd[1], STDOUT_FILENO);
        // close unused read end of pipe
        close(fd[0]);
        // execute ls -l
        execlp("ls", "ls", "-l", NULL);
        perror("execlp");
        exit(EXIT_FAILURE);
    }

    // fork second child (wc -l)
    pid2 = fork();
    if (pid2 == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid2 == 0) {
        // redirect stdin to read end of pipe
        dup2(fd[0], STDIN_FILENO);
        // close unused write end of pipe
        close(fd[1]);
        // execute wc -l
        execlp("wc", "wc", "-l", NULL);
        perror("execlp");
        exit(EXIT_FAILURE);
    }

    // parent process
    // close both ends of pipe
    close(fd[0]);
    close(fd[1]);
```

```
    // wait for both children to finish
    wait(NULL);
    wait(NULL);

    return 0;
}
```

Output:-

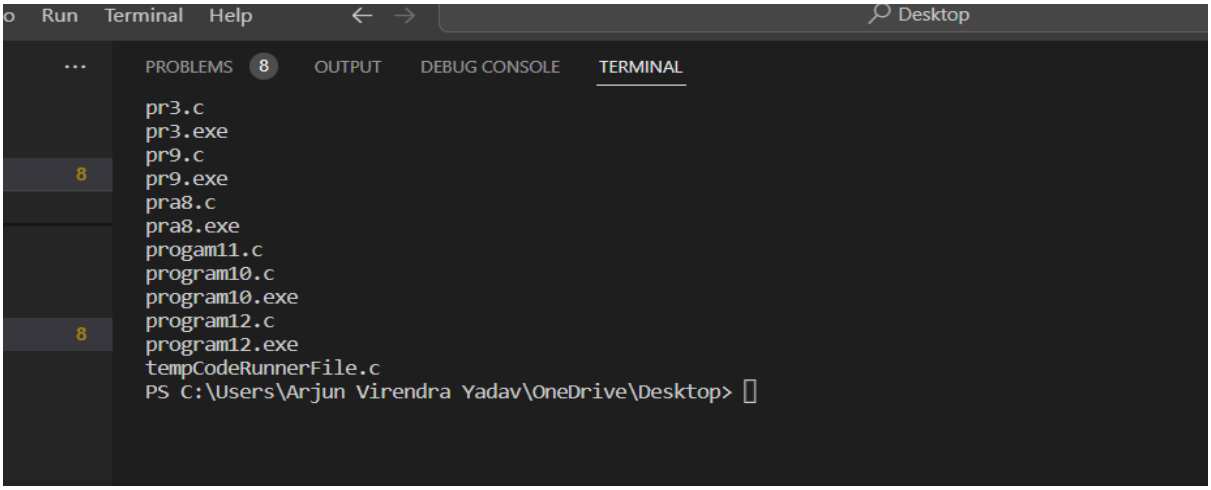
```
student@SCMIRT-32:~/Desktop$ gcc program1111.c
student@SCMIRT-32:~/Desktop$ ./a.out
16
student@SCMIRT-32:~/Desktop$
```

12). Write a C program to display all the files from current directory which are created in particular month

Ans :-

```
      #include <dirent.h>
#include <stdio.h>
#include <sys/stat.h>
#include <time.h>

int main() {
    DIR *d;
    struct dirent *dir;
    struct stat st;
    int month = 4; // change to desired month (1-12)
    char month_str[4][4] = {"Jan", "Feb", "Mar", "Apr", "May",
"Jun",
                            "Jul", "Aug", "Sep", "Oct", "Nov",
"Dec"};

    d = opendir(".");
    if (d) {
        while ((dir = readdir(d)) != NULL) {
            if (stat(dir->d_name, &st) == -1) {
                perror("stat");
                continue;
            }
            int file_month = localtime(&st.st_ctime)->tm_mon;
            if (file_month == month-1) {
                printf("%s\n", dir->d_name);
            }
        }
        closedir(d);
    } else {
        perror("opendir");
    }
    return 0;
}
```

Output:-



13). Write a C program to display all the files from current directory whose size is greater that n Bytes Where n is accept from user.

Ans:-

```
       #include <dirent.h>
#include <stdio.h>
#include <sys/stat.h>

int main() {
    DIR *d;
    struct dirent *dir;
    struct stat st;
    long min_size;

    printf("Enter minimum file size in bytes: ");
    scanf("%ld", &min_size);

    d = opendir(".");
    if (d) {
        while ((dir = readdir(d)) != NULL) {
            if (stat(dir->d_name, &st) == -1) {
                perror("stat");
                continue;
            }
            if (st.st_size > min_size) {
                printf("%s\n", dir->d_name);
            }
        }
        closedir(d);
    } else {
        perror("opendir");
    }
    return 0;
}
```

Output:-

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Arjun Virendra Yadav\OneDrive\Desktop>  & 'c:\Users\Arjun Virendra Yadav\.vscode\extensions\ms-vscode.cpptools-1.14.5-win32-x64\debugAda
pters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-3ozusbnw.i2m' '--stdout=Microsoft-MIEngine-Out-0buytbqy.iu2' '--stderr=Microsoft-
MIEngine-Error-frs24ldb.apu' '--pid=Microsoft-MIEngine-Pid-1msbycxw.dt2' '--dbgExe=C:\Program Files\CodeBlocks\MinGW\bin\gdb.exe' '--interpreter=mi'

Enter minimum file size in bytes: 1
assppppp.docx
file_with_hole.txt
PPL Practical Journal - 1 (1) (1).pdf
pr3.c
pr3.exe
pr9.c
pr9.exe
pra8.c
pra8.exe
PRATICAL.docx
progam11.c
program10.c
program10.exe
program12.c
program12.exe
program13.c
program13.exe
tempCodeRunnerFile.c
PS C:\Users\Arjun Virendra Yadav\OneDrive\Desktop>
```

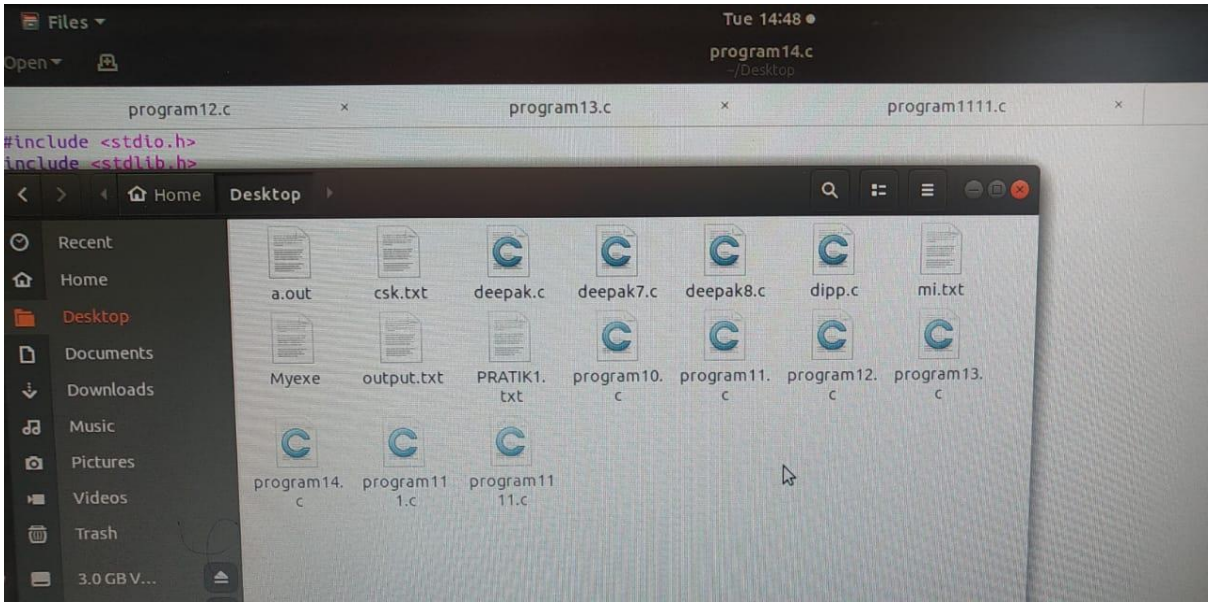
14). Write a C program to implement the following unix/linux command

i. ls –l > output.txt

Ans:-

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   system("ls -l > output.txt");
   return 0;
}
```

Output:-



15). Write a C program which display the information of a given file similar to given by the unix / linux command ls –l

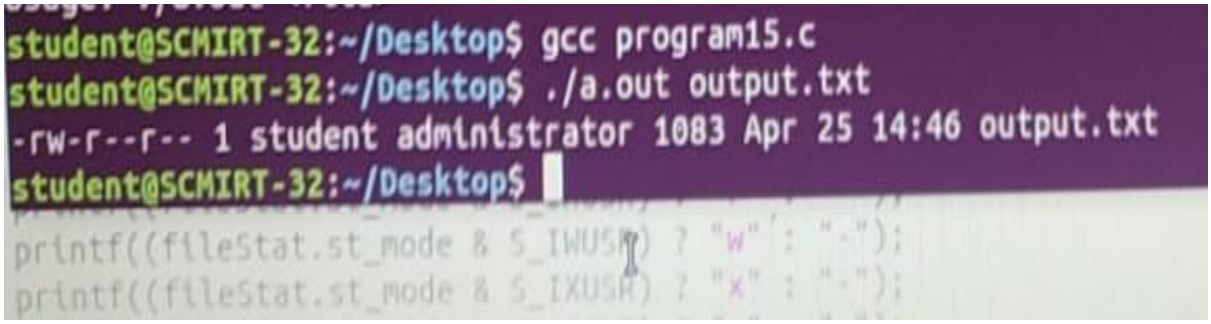
Ans:-

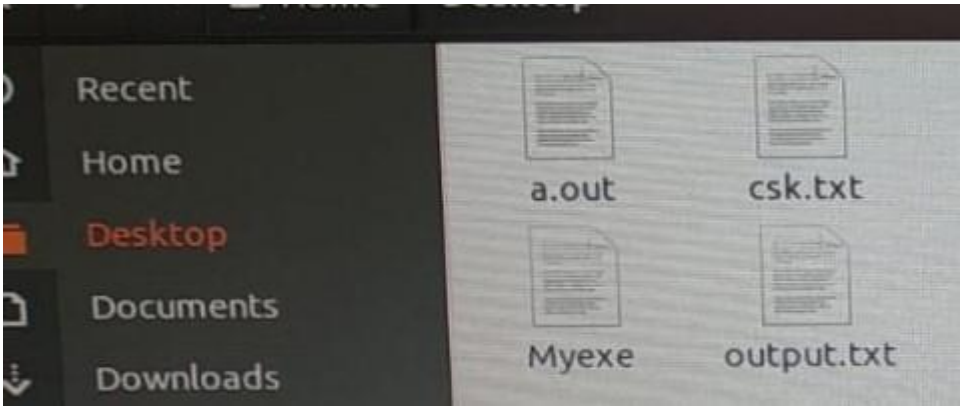
```
                    #include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <time.h>
#include <pwd.h>
#include <grp.h>

int main(int argc, char *argv[])
{
   struct stat fileStat;

   if (argc < 2) {
      fprintf(stderr, "Usage: %s <file>\n", argv[0]);
      exit(EXIT_FAILURE);
   }

   if (stat(argv[1], &fileStat) < 0) {
      perror("stat");
      exit(EXIT_FAILURE);
   }

   // File permissions
   printf((S_ISDIR(fileStat.st_mode)) ? "d" : "-");
   printf((fileStat.st_mode & S_IRUSR) ? "r" : "-");
```

```
printf((fileStat.st_mode & S_IWUSR) ? "w" : "-");
printf((fileStat.st_mode & S_IXUSR) ? "x" : "-");
printf((fileStat.st_mode & S_IRGRP) ? "r" : "-");
printf((fileStat.st_mode & S_IWGRP) ? "w" : "-");
printf((fileStat.st_mode & S_IXGRP) ? "x" : "-");
printf((fileStat.st_mode & S_IROTH) ? "r" : "-");
printf((fileStat.st_mode & S_IWOTH) ? "w" : "-");
printf((fileStat.st_mode & S_IXOTH) ? "x" : "-");

// Number of links
printf(" %ld", fileStat.st_nlink);

// Owner name
printf(" %s", getpwuid(fileStat.st_uid)->pw_name);

// Group name
printf(" %s", getgrgid(fileStat.st_gid)->gr_name);

// File size
printf(" %ld", fileStat.st_size);

// Last modified time
struct tm *tm = localtime(&fileStat.st_mtime);
char modifiedTime[20];
strftime(modifiedTime, sizeof(modifiedTime), "%b %d %H:%M", tm);
printf(" %s", modifiedTime);
```

```
    // File name
    printf(" %s\n", argv[1]);


    return 0;
}
```

Output:-





16). Write a C program that behaves like a shell (command interpreter). It has its own prompt say "NewShell$". Any normal shell command is executed from your shell by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following command.

i) count c - print number of characters in file

ii) count w - print number of words in file

iii) count l - print number of lines in file

Ans:-

```
        #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAX_COMMAND_LENGTH 100
#define MAX_ARGUMENTS 10

void count_command(char *command, char *filename) {
   FILE *fp;
   char ch;
   int count = 0, lines = 0, words = 0;

   fp = fopen(filename, "r");
   if (fp == NULL) {
      printf("Error: unable to open file %s\n", filename);
```

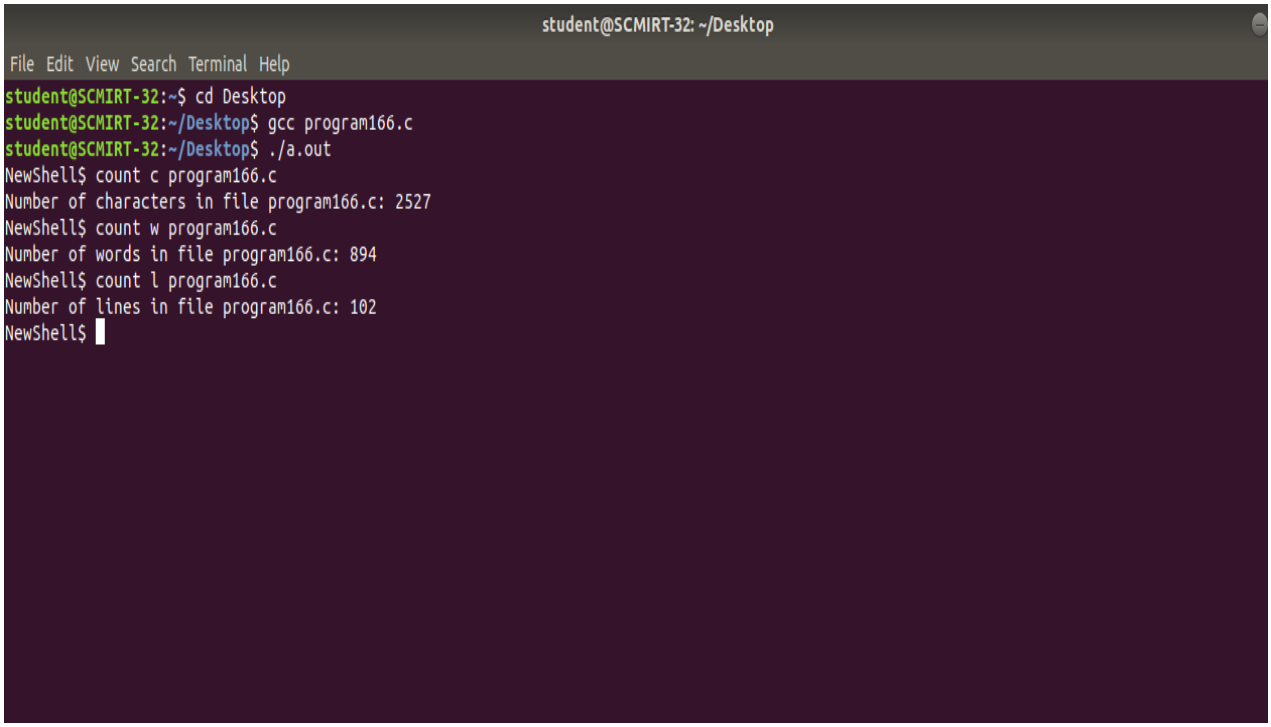
```
        return;
    }

    while ((ch = fgetc(fp)) != EOF) {
        count++;
        if (ch == '\n') {
            lines++;
        }
        if (ch == ' ' || ch == '\n' || ch == '\t') {
            words++;
        }
    }

    fclose(fp);

    if (strcmp(command, "c") == 0) {
        printf("Number of characters in file %s: %d\n", filename, count);
    } else if (strcmp(command, "w") == 0) {
        printf("Number of words in file %s: %d\n", filename, words);
    } else if (strcmp(command, "l") == 0) {
        printf("Number of lines in file %s: %d\n", filename, lines);
    } else {
        printf("Error: invalid count command\n");
    }
}
```

```
int main() {
    char command[MAX_COMMAND_LENGTH];
    char *arguments[MAX_ARGUMENTS];
    char *token;
    pid_t pid;
    int i, status;

    while (1) {
        // Print prompt
        printf("NewShell$ ");
        fflush(stdout);

        // Read command from user input
        fgets(command, MAX_COMMAND_LENGTH, stdin);

        // Replace newline character with null character
        command[strlen(command) - 1] = '\0';

        // Tokenize command into arguments
        token = strtok(command, " ");
        i = 0;
        while (token != NULL && i < MAX_ARGUMENTS) {
            arguments[i] = token;
            i++;
            token = strtok(NULL, " ");
        }
```

```
    arguments[i] = NULL;

    // Handle count command separately
    if (strcmp(arguments[0], "count") == 0 && arguments[1] != NULL &&
arguments[2] != NULL) {
        count_command(arguments[1], arguments[2]);
        continue;
    }

    // Fork child process to execute command
    pid = fork();

    if (pid == -1) {
        perror("fork");
        exit(1);
    } else if (pid == 0) {
        // Child process

        // Execute command
        execvp(arguments[0], arguments);

        // If execvp returns, there was an error
        perror("execvp");
        exit(1);
    } else {
        // Parent process
```

```
        // Wait for child process to finish
        wait(&status);
    }
  }

  return 0;
}
```

Output:-



20. Write a C program which receives file names as command line arguments and display those

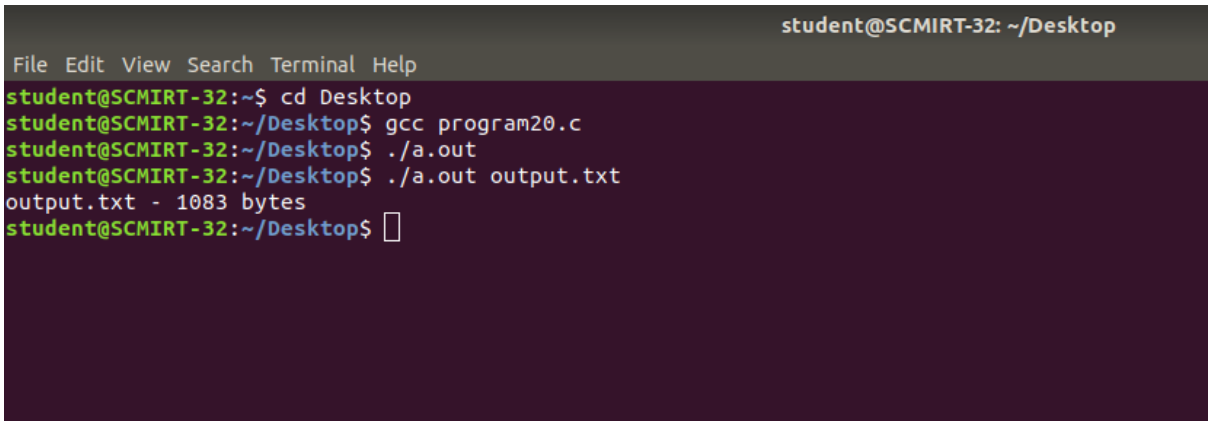
filenames in ascending order according to their sizes

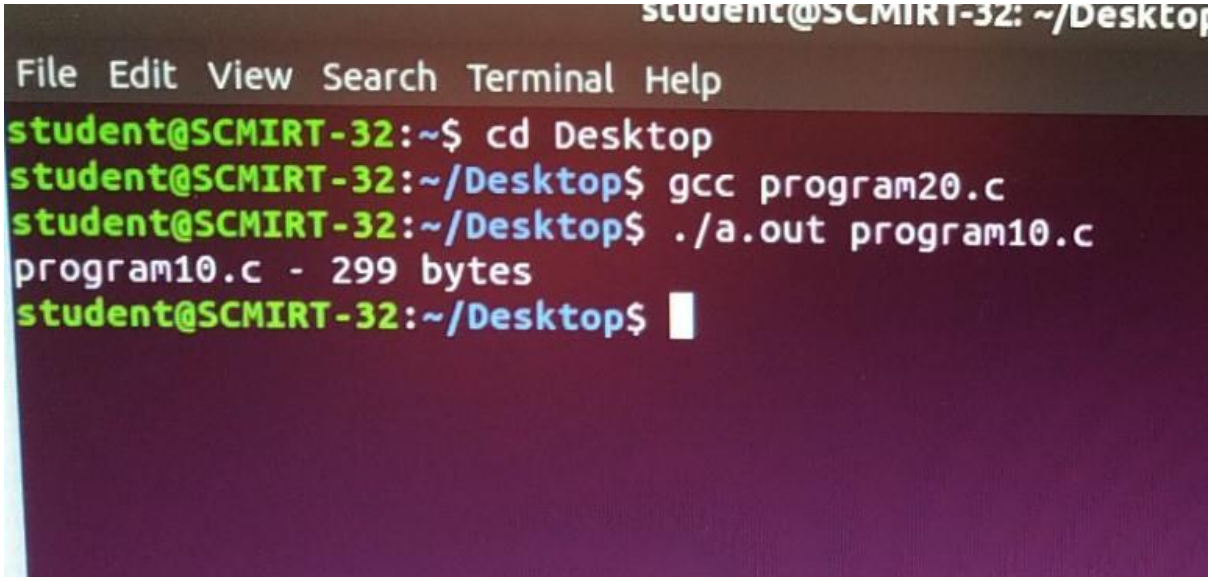
Ans:-

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int compare(const void *a, const void *b) {
   struct stat *stat_a = (struct stat *) a;
   struct stat *stat_b = (struct stat *) b;
   return stat_a->st_size - stat_b->st_size;
}

int main(int argc, char *argv[]) {
   struct stat *stats = malloc(sizeof(struct stat) * argc);
   if (stats == NULL) {
      fprintf(stderr, "Failed to allocate memory.\n");
      return EXIT_FAILURE;
   }

   for (int i = 1; i < argc; i++) {
      if (stat(argv[i], &stats[i]) != 0) {
         fprintf(stderr, "Failed to get file size for %s.\n", argv[i]);
         return EXIT_FAILURE;
      }
```

```
    }

    qsort(&stats[1], argc - 1, sizeof(struct stat), compare);

    for (int i = 1; i < argc; i++) {
        printf("%s - %ld bytes\n", argv[i], stats[i].st_size);
    }

    free(stats);
    return EXIT_SUCCESS;
}
```

Output:-





21. Write a C program which create a child process which catch a signal sighup, sigint and sigquit. The Parent

process send a sighup or sigint signal after every 3 seconds, at the end of 30 second parent send sigquit signal

to child and child terminates my displaying message "My DADDY has Killed me!!!".

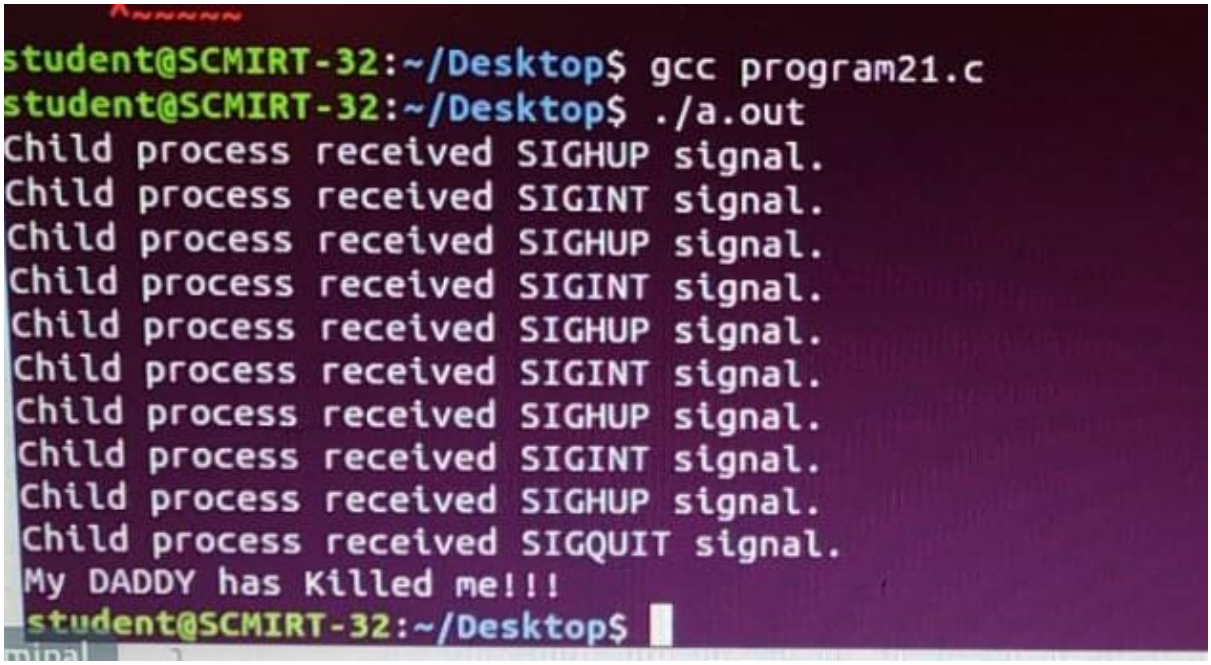
Ans:-

```
 #include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>


void sighup_handler(int signum) {

   printf("Child process received SIGHUP signal.\n");

}


void sigint_handler(int signum) {

   printf("Child process received SIGINT signal.\n");

}


void sigquit_handler(int signum) {

   printf("Child process received SIGQUIT signal.\n");

   printf("My DADDY has Killed me!!!\n");

   exit(0);

}


int main() {
```

```
pid_t pid;
int i;

// Create child process
pid = fork();

if (pid == -1) {
    perror("fork");
    exit(1);
} else if (pid == 0) {
    // Child process

    // Set up signal handlers
    signal(SIGHUP, sighup_handler);
    signal(SIGINT, sigint_handler);
    signal(SIGQUIT, sigquit_handler);

    // Wait for signals
    while (1) {
        sleep(1);
    }
} else {
    // Parent process

    // Send SIGHUP or SIGINT signals every 3 seconds for 30 seconds
    for (i = 0; i < 10; i++) {
```

```
        sleep(3);
        if (i % 2 == 0) {
            kill(pid, SIGHUP);
        } else {
            kill(pid, SIGINT);
        }
    }

    // Send SIGQUIT signal to child process
    kill(pid, SIGQUIT);
    sleep(1);
  }

  return 0;
}
```

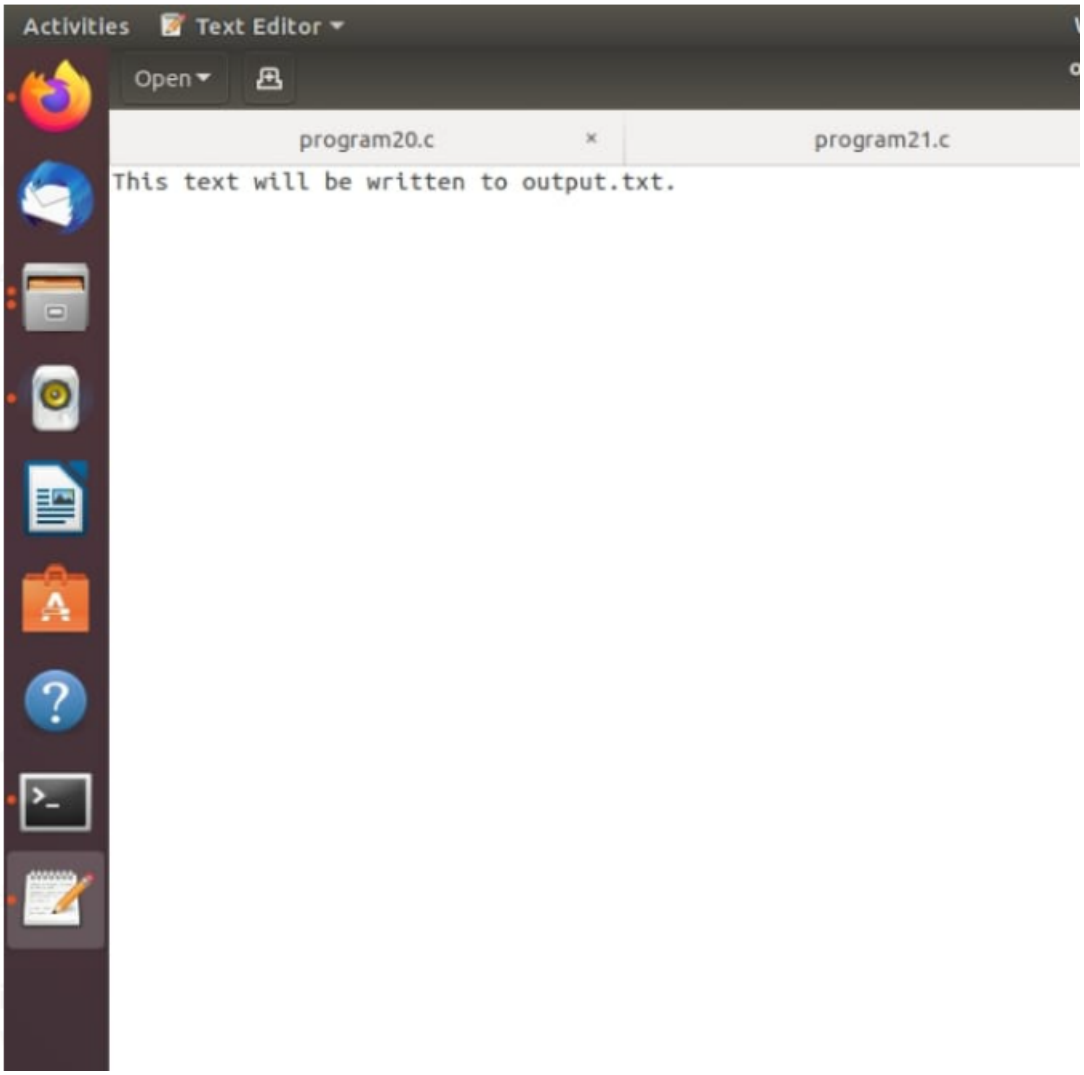
Output:-

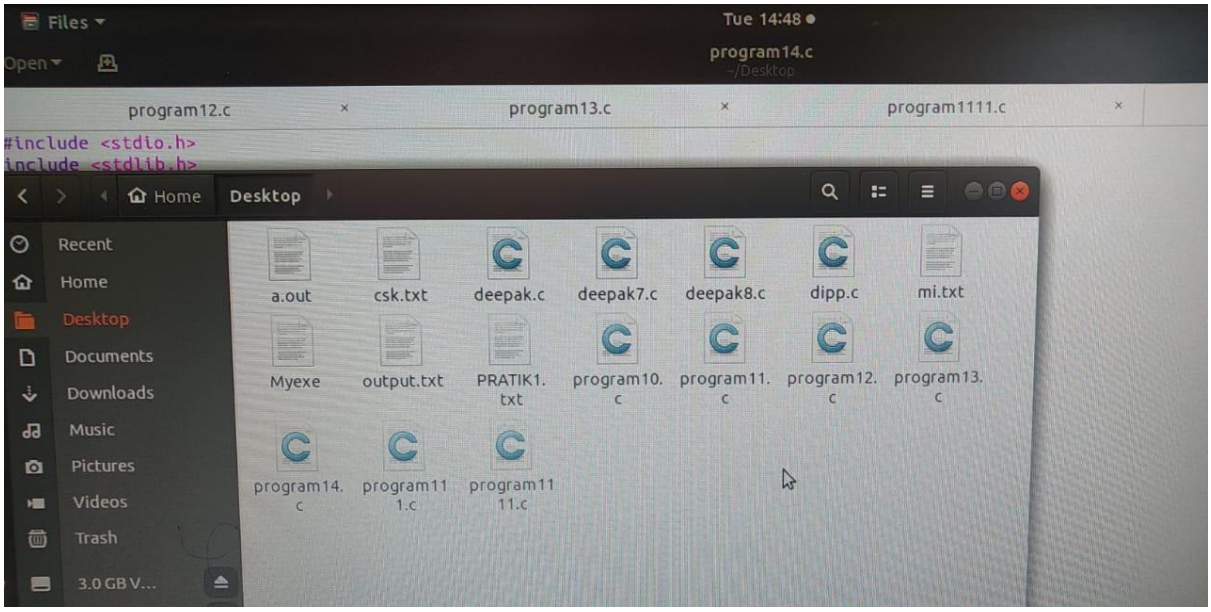


23. Write a C Program that demonstrates redirection of standard output to a file

```
#include <stdio.h>


int main() {
   FILE* fp;


   fp = freopen("output.txt", "w", stdout); // redirect standard output to
output.txt


   if (fp == NULL) {
      fprintf(stderr, "Failed to open file.\n");
      return 1;
   }


   printf("This text will be written to output.txt.\n");


   fclose(fp); // close the file


   return 0;
}
```

Output:-


Activities
Text Editor
Open
program20.c
program21.c
This text will be written to output.txt.


Files
Tue 14:48
Open
program14.c
~/Desktop
program12.c
program13.c
program1111.c
#include <stdio.h>
Home
Desktop
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
3.0 GB V...
a.out
csk.txt
deepak.c
deepak7.c
deepak8.c
dipp.c
mi.txt
Myexe
output.txt
PRATIK1.txt
program10.c
program11.c
program12.c
program13.c
program14.c
program111.c
program1111.c

24. Write a program that illustrates how to execute two commands concurrently with a pipe.

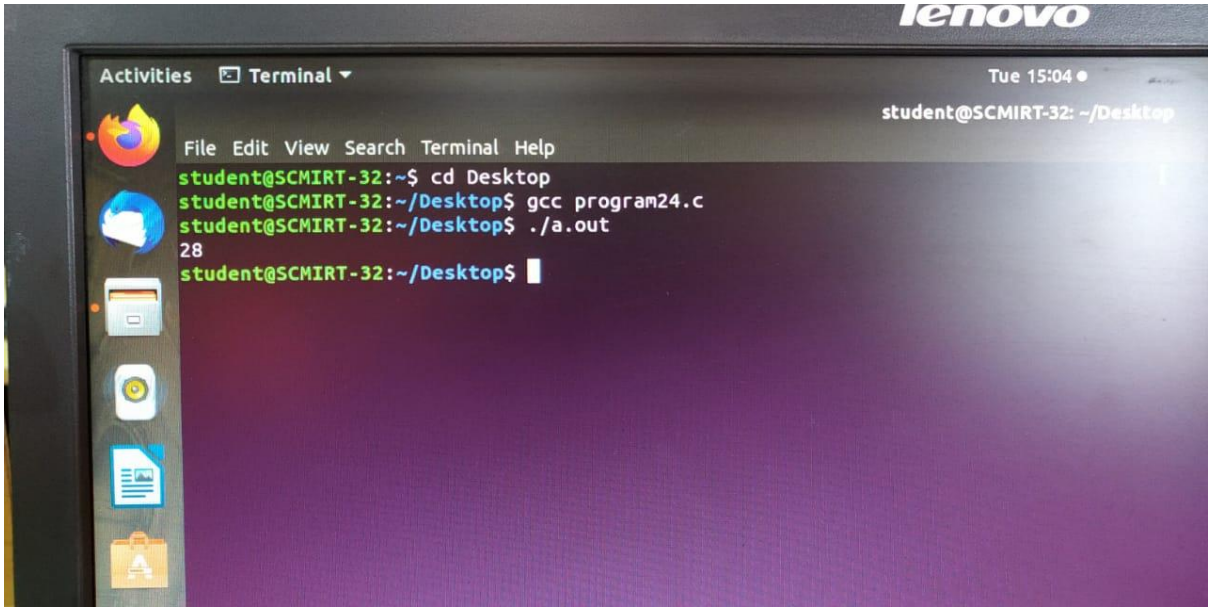
Ans:-

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int pipefd[2];
    pid_t pid1, pid2;

    // Create a pipe
    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // Fork first child process
    pid1 = fork();
    if (pid1 == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid1 == 0) {
        // Child process 1
```

```
    // Close the read end of the pipe
    close(pipefd[0]);

    // Redirect stdout to the write end of the pipe
    dup2(pipefd[1], STDOUT_FILENO);

    // Execute the first command
    execlp("ls", "ls", NULL);

    // Exit the child process if execlp fails
    perror("execlp");
    exit(EXIT_FAILURE);
}

// Fork second child process
pid2 = fork();
if (pid2 == -1) {
    perror("fork");
    exit(EXIT_FAILURE);
} else if (pid2 == 0) {
    // Child process 2

    // Close the write end of the pipe
    close(pipefd[1]);
```

```
        // Redirect stdin to the read end of the pipe
        dup2(pipefd[0], STDIN_FILENO);

        // Execute the second command
        execlp("wc", "wc", "-l", NULL);

        // Exit the child process if execlp fails
        perror("execlp");
        exit(EXIT_FAILURE);
    }

    // Parent process

    // Close both ends of the pipe
    close(pipefd[0]);
    close(pipefd[1]);

    // Wait for both child processes to exit
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    return 0;
}
```

Output:-


lenovo
Activities
Terminal
Tue 15:04
student@SCMIRT-32: ~/Desktop
File Edit View Search Terminal Help
student@SCMIRT-32:~$ cd Desktop
student@SCMIRT-32:~/Desktop$ gcc program24.c
student@SCMIRT-32:~/Desktop$ ./a.out
28
student@SCMIRT-32:~/Desktop$

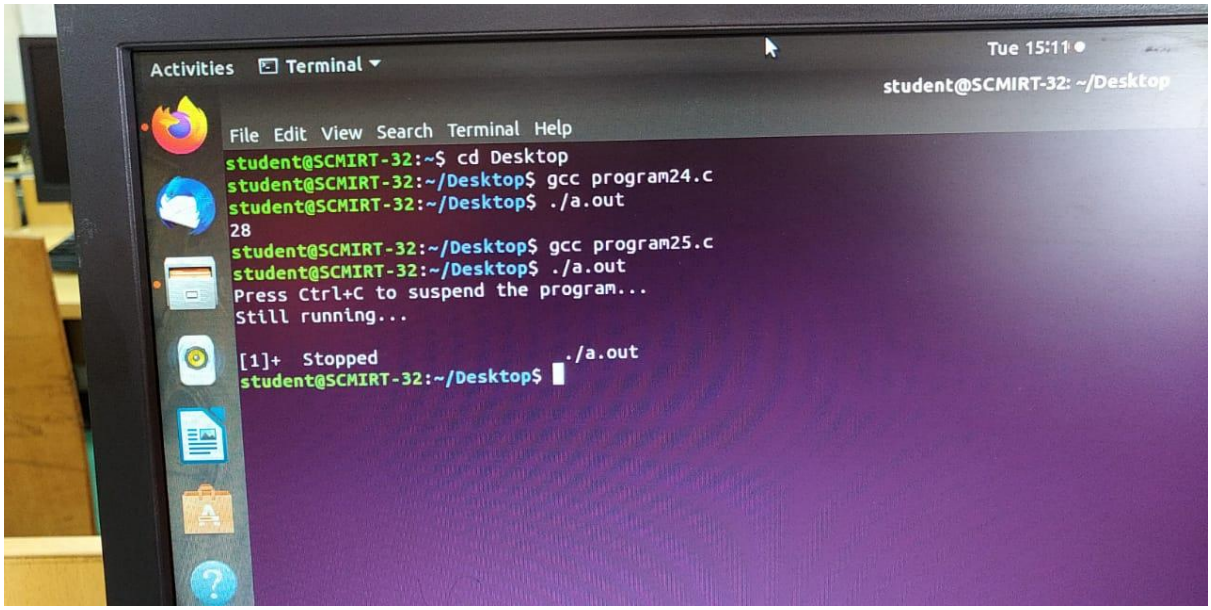
25. Write a C program that illustrates suspending and resuming processes using signals.

Ans:-

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void sigint_handler(int signum) {
    printf("Caught signal %d (SIGINT)\n", signum);
}

int main() {
    struct sigaction sa;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sa.sa_handler = sigint_handler;
    sigaction(SIGINT, &sa, NULL);

    printf("Press Ctrl+C to suspend the program...\n");

    while (1) {
        sleep(1);
        printf("Still running...\n");
        kill(getpid(), SIGSTOP);
        printf("Resuming...\n");
```

```
    }

    return 0;
}
```

Output:-



26. Write a C program that illustrates inters process communication using shared memory.

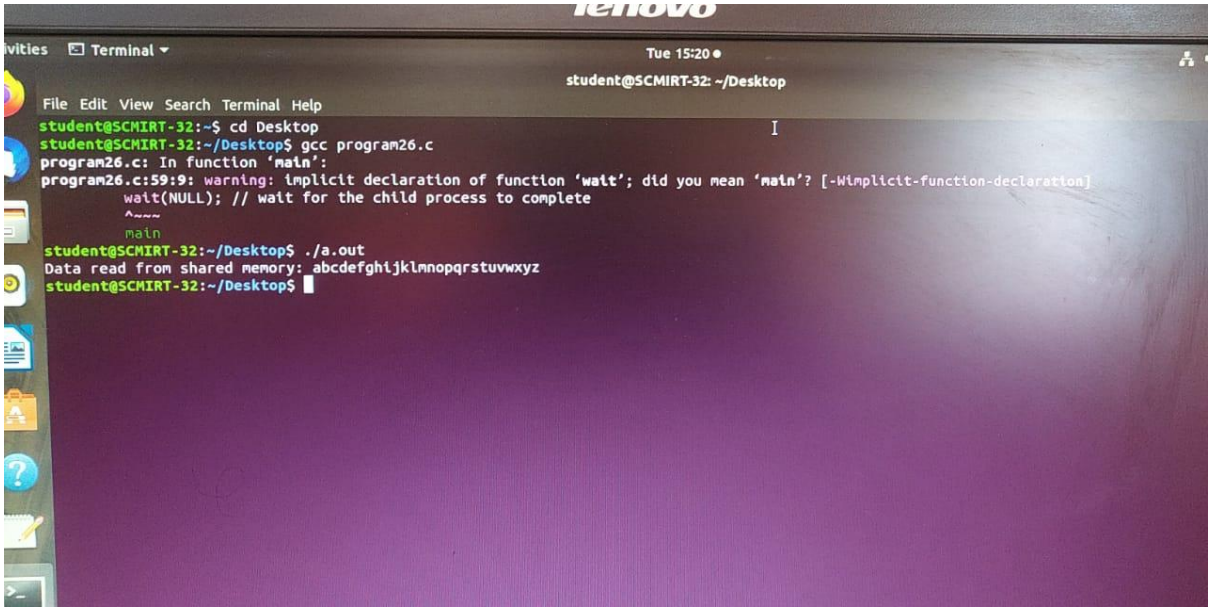
Ans:-

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 1024

int main() {
    int shmid;
    key_t key;
    char *shm;
    char *s;

    key = 1234; // unique key for the shared memory segment

    // create a shared memory segment
    if ((shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
```

```
}

// attach the shared memory segment to our process's address space
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
   perror("shmat");
   exit(1);
}

// write some data to the shared memory segment
s = shm;
for (char c = 'a'; c <= 'z'; c++) {
   *s++ = c;
}
*s = '\0';

// detach the shared memory segment from our process's address space
if (shmdt(shm) == -1) {
   perror("shmdt");
   exit(1);
}

// read the data from the shared memory segment in a separate process
pid_t pid = fork();
if (pid == 0) {
   // child process
   if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
```

```
        perror("shmat");
        exit(1);
    }
    printf("Data read from shared memory: %s\n", shm);
    if (shmdt(shm) == -1) {
        perror("shmdt");
        exit(1);
    }
    exit(0);
} else if (pid > 0) {
    // parent process
    wait(NULL); // wait for the child process to complete
} else {
    // error
    perror("fork");
    exit(1);
}

// delete the shared memory segment
if (shmctl(shmid, IPC_RMID, NULL) == -1) {
    perror("shmctl");
    exit(1);
}

return 0;
}
```

Output:-


Terminal
Tue 15:20
student@SCMIRT-32: ~/Desktop
File Edit View Search Terminal Help
student@SCMIRT-32:~$ cd Desktop
student@SCMIRT-32:~/Desktop$ gcc program26.c
program26.c: In function 'main':
program26.c:59:9: warning: implicit declaration of function 'wait'; did you mean 'main'? [-Wimplicit-function-declaration]
wait(NULL); // wait for the child process to complete
^~~~
main
student@SCMIRT-32:~/Desktop$ ./a.out
Data read from shared memory: abcdefghijklmnopqrstuvwxyz
student@SCMIRT-32:~/Desktop$