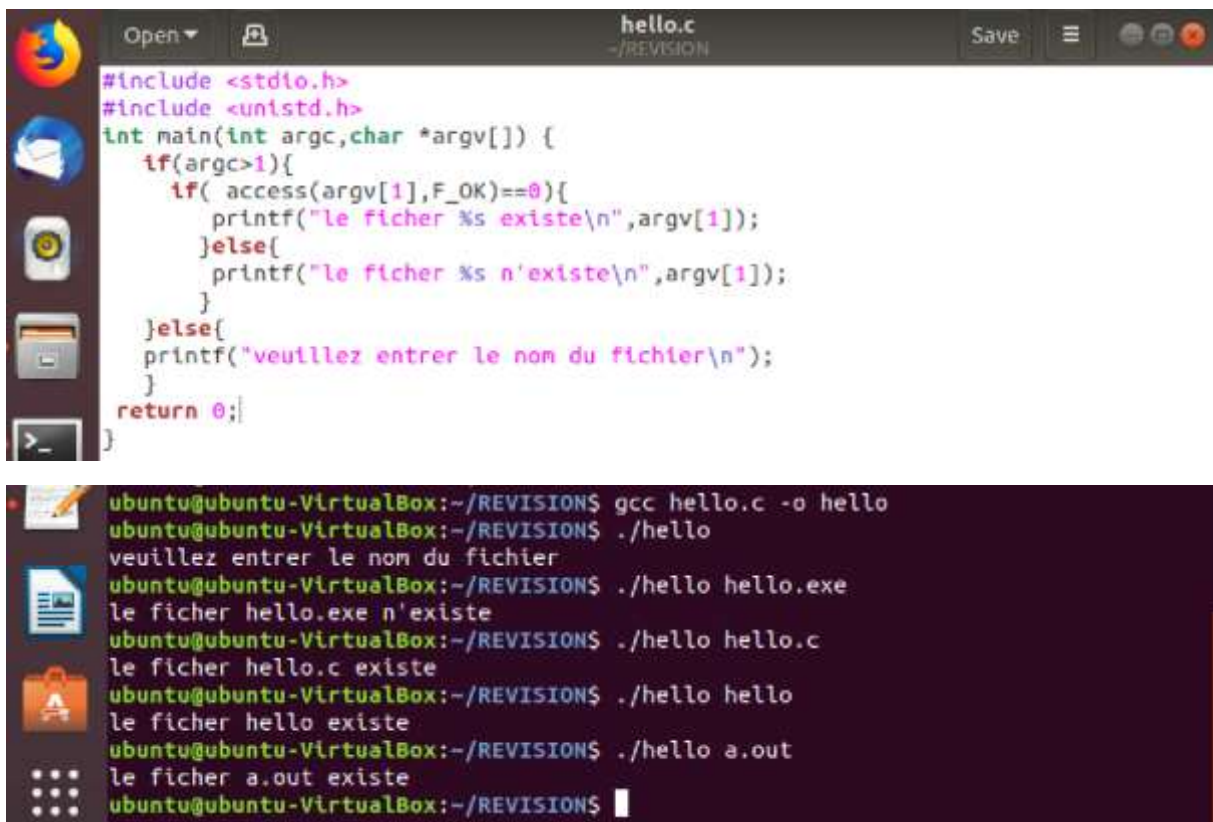


Compte rendu TP2

Eercice1 :

FLAG	ROLE
F_OK	Vérifie l'existence d'un fichier
R_OK	Vérifie l'accès en lecture
W_OK	Vérifie l'accès en écriture
X_OK	Vérifie l'accès en exécution

2/.



```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[]) {
    if(argc>1){
        if( access(argv[1], F_OK)==0){
            printf("le fichier %s existe\n", argv[1]);
        }else{
            printf("le fichier %s n'existe\n", argv[1]);
        }
    }else{
        printf("veuillez entrer le nom du fichier\n");
    }
    return 0;
}
```

```
ubuntu@ubuntu-VirtualBox:~/REVISION$ gcc hello.c -o hello
ubuntu@ubuntu-VirtualBox:~/REVISION$ ./hello
veuillez entrer le nom du fichier
ubuntu@ubuntu-VirtualBox:~/REVISION$ ./hello hello.exe
le fichier hello.exe n'existe
ubuntu@ubuntu-VirtualBox:~/REVISION$ ./hello hello.c
le fichier hello.c existe
ubuntu@ubuntu-VirtualBox:~/REVISION$ ./hello hello
le fichier hello existe
ubuntu@ubuntu-VirtualBox:~/REVISION$ ./hello a.out
le fichier a.out existe
ubuntu@ubuntu-VirtualBox:~/REVISION$
```

Pour tester tous les droits :



```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[]) {
    if(argc>1){
        if( access(argv[1], R_OK|W_OK|X_OK)==0){
            printf("le fichier %s a TOUS LES DROITS \n", argv[1]);
        }else{
            printf("le fichier %s n'a pas LES DROITS \n", argv[1]);
        }
    }else{
        printf("veuillez entrer le nom du fichier\n");
    }
    return 0;
}
```

```

ubuntu@ubuntu-VirtualBox:~/REVISIONS$ gcc hello.c -o hello
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./hello tp
le fichier tp n'a pas acces en execution
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ gcc hello.c -o hello
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./hello tp
le fichier tp n'a pas LES DROITS
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ █

```

Exercice2 :

FLAG	ROLE
O_RDONLY	Ouvrir le fichier en lecture seul
O_RDWR	Ouvrir le fichier en lecture écriture
O_WRONLY	Ouvrir le fichier en écriture seul, si le fichier est rempli j'écrase son contenu car le curseur est placé au début du fichier
O_APPEND	Ouvrir le fichier en écriture et le curseur se positionne à la fin du fichier pour continuer à écrire
O_CREAT	Créer un fichier s'il n'existe pas s'il existe il ne va rien faire donc on risque d'écraser le contenu du fichier déjà créé et pour cela on utilise O_EXCL si le fichier existe on aura une erreur
O_TRUNC	Elle permet d'effacer le contenu d'un fichier mais le fichier existera toujours

2/.

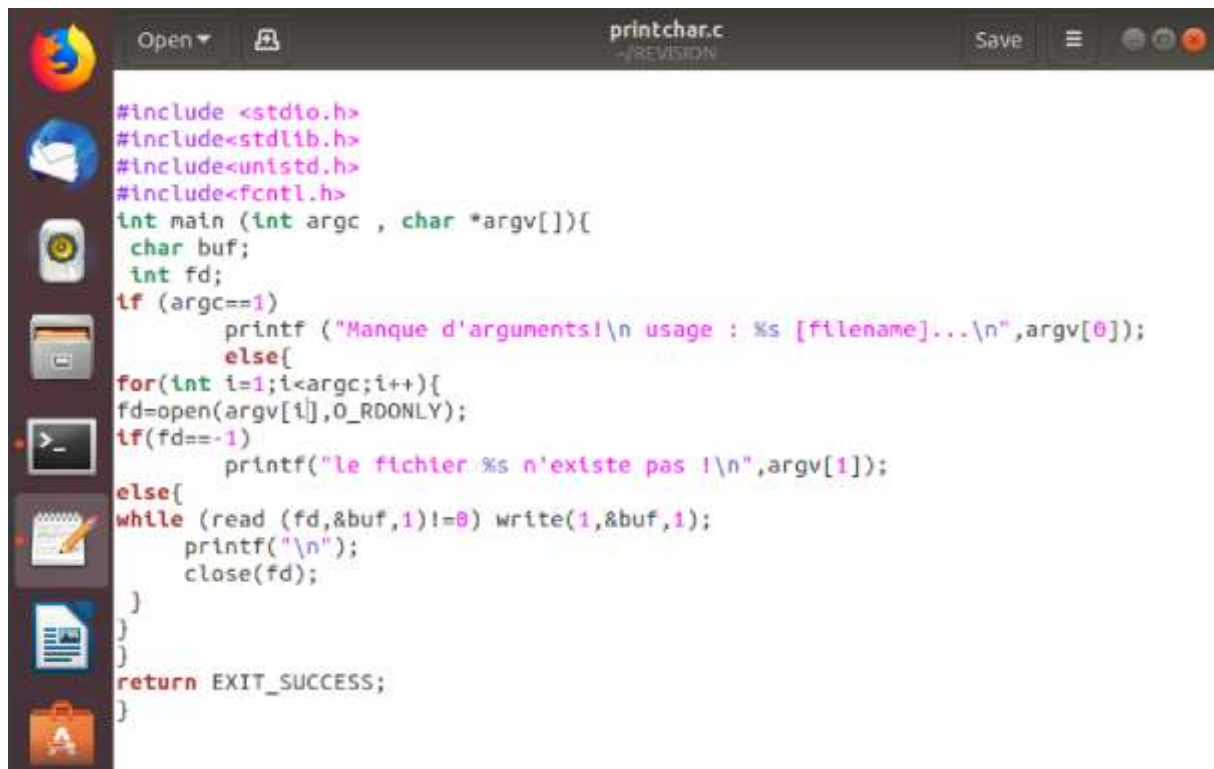
Le programme commence par vérifier si le nombre d'arguments si il est égale à 1 si oui un message sera affiché à l'utilisateur qui indique qu'on a un manque d'arguments, si ce n'est pas le cas on ouvre le 1^{er} fichier entre en argument en lecture seule tq le descripteur du fichier sera sauvegarder dans la variable fd de type int, on vérifie si fd est égale à -1 le fichier n'existe pas sinon on affiche le premier caractère du 1^{er} fichier entrée en paramètres en utilisant les fonctions read et write

On commence par vérifier si read est différente de 0 car le fichier n'est pas vide on lui donne comme paramètres le descripteur du fichier ouvert la variable buf qui va enregistrer ce qu'on veut lire et le nombre d'octet à lire dans notre cas c'était 1 si elle est différente de 0 car le fichier n'est pas vide on appelle write qui va avoir comme paramètres d'entrée le nombre d'octet le buf d'ont on a enregistré ce qu'on a lu et le nombre de caractère qu'on veut afficher qu'on on finit on ferme le fichier en utilisant close(fd)

3/. Execution :

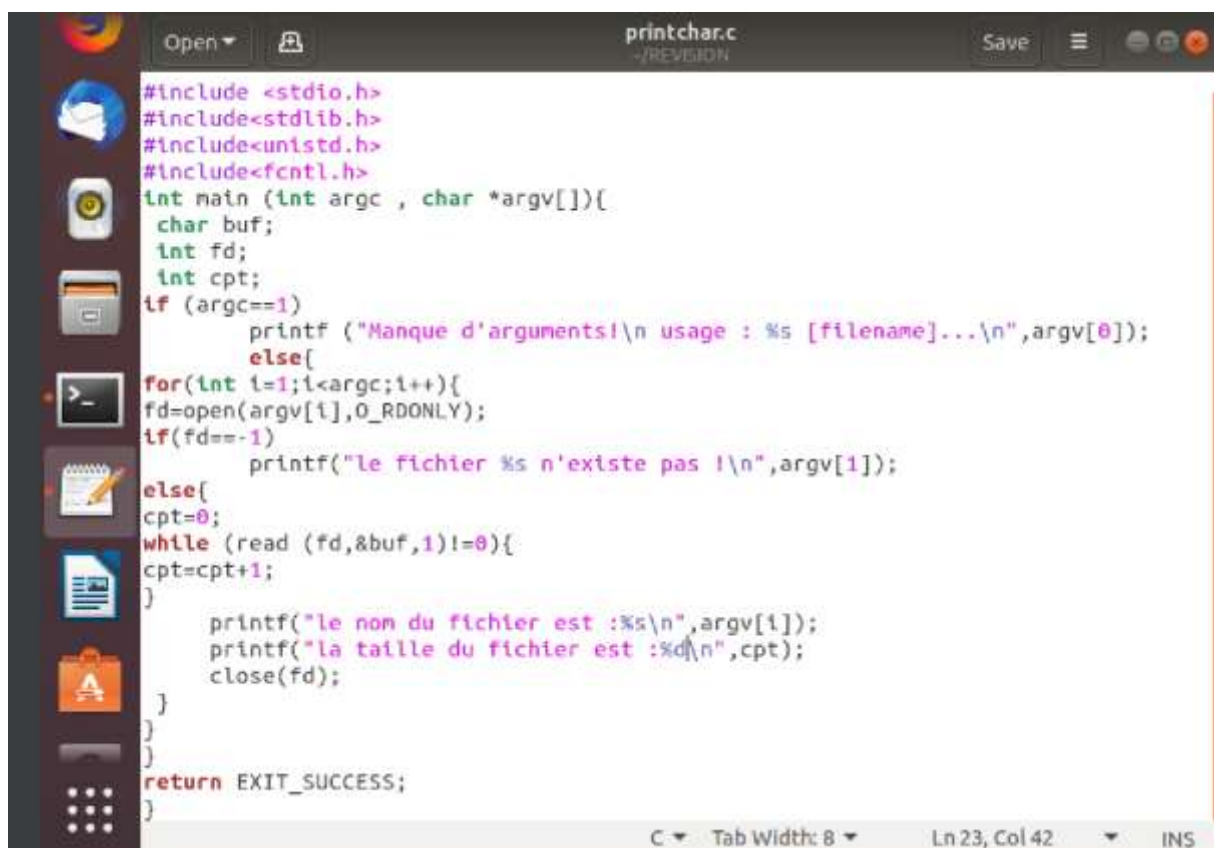
On aura le premier carc qui sera afficher

5./



```
#include <stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
int main (int argc , char *argv[]){
    char buf;
    int fd;
    if (argc==1)
        printf ("Manque d'arguments!\n usage : %s [filename]...\n",argv[0]);
    else{
        for(int i=1;i<argc;i++){
            fd=open(argv[i],O_RDONLY);
            if(fd==-1)
                printf("le fichier %s n'existe pas !\n",argv[i]);
            else{
                while (read (fd,&buf,1)!=0) write(1,&buf,1);
                printf("\n");
                close(fd);
            }
        }
        return EXIT_SUCCESS;
    }
}
```

Exercice 5 :



```
#include <stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
int main (int argc , char *argv[]){
    char buf;
    int fd;
    int cpt;
    if (argc==1)
        printf ("Manque d'arguments!\n usage : %s [filename]...\n",argv[0]);
    else{
        for(int i=1;i<argc;i++){
            fd=open(argv[i],O_RDONLY);
            if(fd==-1)
                printf("le fichier %s n'existe pas !\n",argv[i]);
            else{
                cpt=0;
                while (read (fd,&buf,1)!=0){
                    cpt=cpt+1;
                }
                printf("le nom du fichier est :%s\n",argv[i]);
                printf("la taille du fichier est :%d\n",cpt);
                close(fd);
            }
        }
        return EXIT_SUCCESS;
    }
}
```

TP2 part2

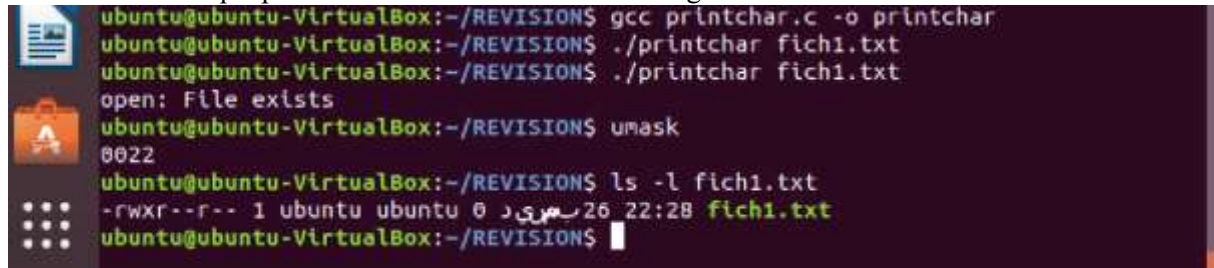
Exercice1 :

Le programme exige que le nombre d'argument soit 2 le premier c'est celui de l'exécutable et le deuxième du fichier qu'on a besoin on cas d'erreur un message sera afficher et on sort en exécutant la commande exit (1) qui signifie exit_success on définit une variable mode de type mode_t qui va contenir des droits d'accès

On ouvre le fichier entre en argument en mode écriture seul en utilisant aussi les flags O_CREAT et O_EXCL qui vont permettre de créer le fichier si le fichier n'existe pas si le fichier existe on va générer une erreur grâce à O_EXCL donc le descripteur de fichier sera -1 du coup un message d'erreur va être affiché au niveau du terminal et cela grâce à perror qui est une fonction qui prend en entrée une chaîne de caractères exemple open et affiche sur le terminal le message d'erreur selon le contenu de errno

Si on refait l'exécution un message d'erreur sera affiché car à la première exécution le fichier n'existe pas du coup il sera créé en lui attribuant les modes définis dans mode - umask lors de la deuxième exécution un message d'erreur sera exécuter grâce à perror

La valeur du masque par défaut est 0022 et on la obtient grâce à la commande umask



```
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ gcc printchar.c -o printchar
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./printchar fich1.txt
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./printchar fich1.txt
open: File exists
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ umask
0022
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ls -l fich1.txt
-rwxr--r-- 1 ubuntu ubuntu 0 26 22:28 fich1.txt
ubuntu@ubuntu-VirtualBox:~/REVISIONS$
```

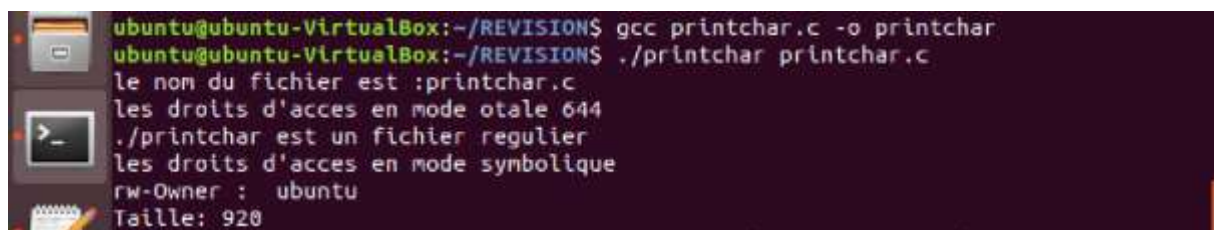
Les valeurs du mode sont : rwx rw- rw-

Les valeurs du mask sont : rwx r - - r - -

La valeur du mask est : 000 010 010 → 0 le droit n'est pas masqué / 1 le droit est masqué car les droits write sont enlevés du groupe et du other car on a des 1

Exercice 2 :

1/.



```
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ gcc printchar.c -o printchar
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./printchar printchar.c
le nom du fichier est :printchar.c
les droits d'accès en mode octale 644
./printchar est un fichier régulier
les droits d'accès en mode symbolique
rw-Owner : ubuntu
Taille: 920
```

2/.

```
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ gcc printchar.c -o printchar
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./printchar printchar.c
-rw-r--r-- ubuntu ubuntu 1181 140169920693856:140727991578280 printchar.c
ubuntu@ubuntu-VirtualBox:~/REVISIONS$
```

Exercice 3 :

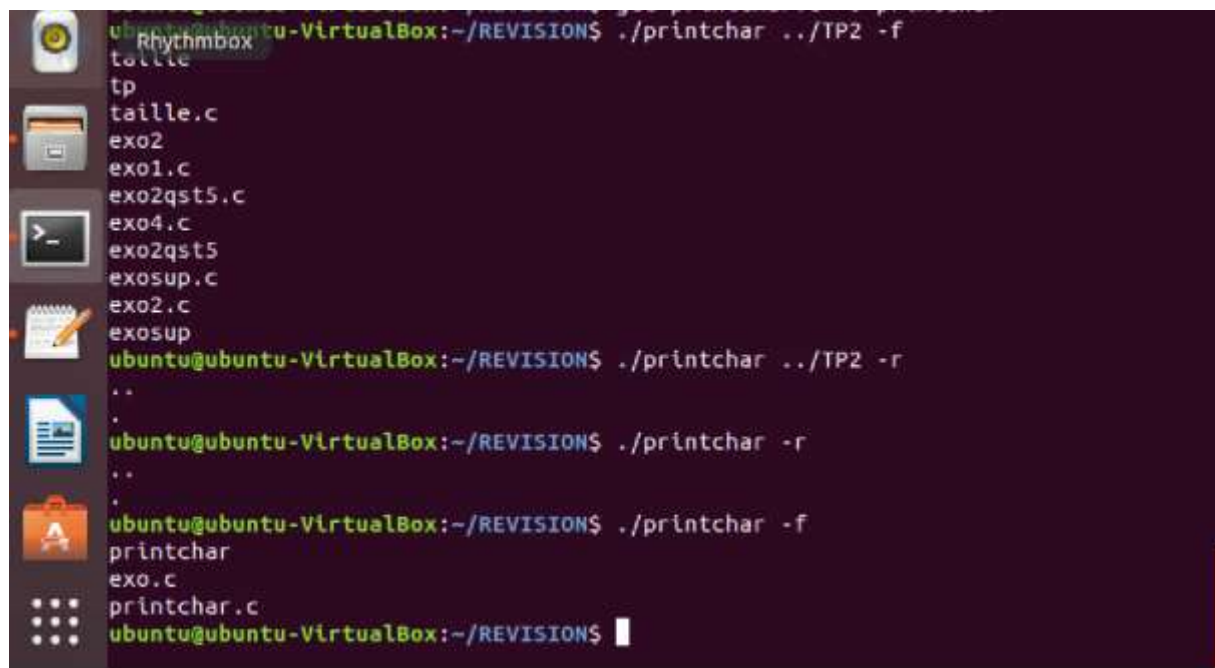
```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>

int main (int argc , char *argv[]){
    char buf;
    int fd;
    int fdd;
    if(argc!=3){
        printf("Manque d'arguments\n");
        exit(1);
    }
    fd=open(argv[1],O_RDWR);
    fdd= open(argv[2],O_CREAT|O_EXCL|O_WRONLY,777);
    if (fd==-1 & fdd==-1){
        perror("error");
        exit(1);
    } else{
        while(read(fd,&buf,sizeof(buf))!=0){
            write(fdd,&buf,1);
        }
        close(fd);
        return EXIT_SUCCESS;
    }
}
```

C Tab Width: 8 Ln 16, Col 48 INS

```
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ gcc printchar.c -o printchar
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./printchar printchar.c exo.c
ubuntu@ubuntu-VirtualBox:~/REVISIONS$
```

Exercice 4 :



The image shows a screenshot of an Ubuntu desktop environment. On the left side, there is a vertical dock containing several application icons: a yellow circle with a black dot, a folder icon, a terminal icon, a notepad icon, a document icon, and a shopping bag icon. The main area of the screen is a terminal window titled 'ubuntu@ubuntu-VirtualBox:~/REVISIONS'. The terminal displays the following commands and their outputs:

```
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./printchar ../TP2 -f
taille
tp
taille.c
exo2
exo1.c
exo2qst5.c
exo4.c
exo2qst5
exosup.c
exo2.c
exosup
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./printchar ../TP2 -r
..
.
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./printchar -r
..
.
ubuntu@ubuntu-VirtualBox:~/REVISIONS$ ./printchar -f
printchar
exo.c
printchar.c
ubuntu@ubuntu-VirtualBox:~/REVISIONS$
```

```
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
#include <unistd.h>

int main (int argc , char *argv[]){
    DIR* dir;
    if (argc>2){
        dir=opendir(argv[1]);
    }else{
        dir=opendir(".");
    }
    int opt = getopt(argc,argv,"rf");
    struct dirent *ent ;
    while ((ent=readdir(dir))!=NULL){
        if((opt=='r')&&(ent->d_type==DT_DIR)) {
            printf("%s\n",ent->d_name);
        }else{
            if ((opt=='f')&& (ent->d_type==DT_REG)){
                printf("%s\n",ent->d_name);
            }
        }
    }
    closedir(dir);
    return 0;
}
```