PDS TD ① dw 17/09

Exo 1

meth bourrin :

```
#! bin / bash
echo " NAME_MAX $ NAME_MAX "
____ " PATH_MAX $PATH_MAX"
```

non bourrin :

```
# include <Unix.h>

int main(int argc, char **argv) {

    printf("%s", NAME_MAX);

    printf("%s", PATH_MAX);
    return 0;
}
```

Exo 2 :

code ASCII ↑ % macces -x /bin/ls ; echo $?

```
int getopt (int argc, char * argv [], char *optstring);
```

var. gl. :

```
extern int optind
# include <unistd.h>
int main (int argc, char **argv){
char *optstring;
int opt = getopt (argc, **argv, *optstring);
```

if (opt == 'w') {
return access(x argc +2 ,

```
# nocorrect option 255        # usage 254
# include <unistd.h>
int main (int argc , char ** argv){
  if (argc >=3 ) {
     <2 ==> == 2
   printf("erreur : on suppose une option , un fichier");
   return usage
  }
  switch ( getopt ()){
     case 'x' : mode=x_ok ; break;
     — 'w' : mode = W_ok ; —;
     — 'r' : mode = R_ok; —;
case -1: fprintf(
stderr, "pas d'option
entrée");
return nocorrect
option
     default : fprintf(stderr, "option : r, w, x);
  return ! access(argv [optind ], mode );
}
```

Exo3: Maccess. sh

```
#!/bin/bash
no correct option = $(255);
usage = $ 254
maccess ;
if [ $? == usage ];
then echo " test usage: sucess";
else ____________ refused";
fi ;
exit(1);
```

PDS TD② du 1?/09

Exo 3:

```
fillDir ( char * path ) {
    char *dirs;    int i, P;  P = strlength(path) ;
    dirs = malloc( sizeof (char) x (strlength(    path)+1);
    for ( i=0; i<P; i++) {
        if ( path[i] = ' ' ) {
                dirs[i] = '\0'; }
        else  dir[i] = path[i]; }
    path[i]= '\0';
}
```

Exo 4:

PDS

TD ~~cour~~ ① du 24/09

Exo 7

```
int is_root (char *path){
    struct stat pathstat;
    static ino_t rootino;
    if (rootino == 0){
        stat("/", &rootstat);
        stat(path, &pathstat);
        return rootino == rootstat.ino_t;
    }
    return rootino == pathstat.ino_t
}
```

Exo 8 ;

```
void print_name_in_parent (char *current, char *father){
    struct dirent *direntry;
    ~~dirent = malloc(sizeof(struct dirent));~~
    struct stat currentstat;
    stat(current, &currentstat);
    Dir *dir;
    dir = opendir(father);
    while (direntry = readdir(dir)) {
        if direntry -> ino_t = currentstat.ino_t {
            printf("%s", direntry -> d_name);
            break}
    }
    closedir(dir);
```

-> test nécessaire

Exercice 9:

```
void print_node_dirname(char * current)
{
    if (is_root(current))
        return;
    char father[PATH_MAX];
    strcpy(current, father, PATH_MAX);
    print_node_dirname(strncat(current, "/..", PATH_MAX));
    printf("/");
    print_name_in_parent(current, father);
}
```

Exercice 10:

```
void pwd(void) {
    if (is_root("."))
        printf("/");
    else
        print_node_dirname(".");
    return;
}
```

Exercice 11:

```
int du_file(const char * path){
    struct stat pathstat;
    fstat(path, &pathstat);
    if (S_ISREG(pathstat.st_mode))
        return pathstat.st_size;
    if (S_ISDIR(pathstat.st_mode)){
        Dir * dir; off_t size = 0;
        struct dirent * direntry;
        dir = opendir(path);
```

Exo 11 suite

```
size += pathstat.st_size;
while (direntry = readdir(dir)) {
        size += du_file(direntry->t_name);
}
return size;
}
```

Exo 20 à 22 ;

```
# define BUFSIZE 1024
struct kFILE {
int offset;   int id ;  int pos  ; int  inputpos ; int maxin ;
        char buf [ BUFSIZE ];
        char input [ ———— ];
};

typedef struct kfile KFILE;
```

par csq on peut retirer ses variables
↳ on doit réadapter le code;

/* interdire le mask lect et écriture */

```
KFILE * kopen ( char * path, ~~ )
        KFILE * file ;
        file = malloc ( sizeof(KFILE));
        file -> pos = 0; file -> inputpos = 0 ;
        file -> id = open ( path, ~~ );

        return file
}
```

file -> max input = 0 ;

```
void kputc(KFILE *file, char c)
    if(file->pos < BUFSIZE)
        file->buf[file->pos++] = c;
    else
        kfflush(file);

}

char kgetc(KFILE *file){
    if(file->inputpos == file->maxin){
file->maxin = read(file->id, file->input, BUFSIZE);
file->inputpos = 0;}
        return file->input[file->inputpos++]

}

void kseek(KFILE *file, int offset){
    file->offset = Pseek(file->id, offset)
}
```

PDS

TD ② du 08/10

```
void kflush (KFILE * file)
  file->offset +=
    write ( file ->id, file -> buf, file -> pos - 1);
      file -> pos = 0;

}

void    kclose ( KFILE * file )
      if ( file -> pos != 0)
            kflush ( file );
      close ( file -> id );
      free (file);
}
```

?