Count – de cate ori apare un subsir in fisier

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

int main(int argc, char *argv[])
{
    if (argc < 3) {
       puts("Argumente insuficiente");
       return -1;
    }

    int descriptor = open(argv[1], O_RDONLY);

    if (descriptor < 0) {
       printf("Fisierul %s nu s-a putut deschide\n", argv[1]);
       return -1;
    }

    int dimensiune_sir = strlen(argv[2]);
    int count = 0;
    char* buffer = malloc(dimensiune_sir*sizeof(char));

    while (read(descriptor, buffer, dimensiune_sir) == dimensiune_sir) {
       buffer[dimensiune_sir] = '\0';

       if (strcmp(buffer, argv[2]) == 0) {
          count++;
       }
       lseek(descriptor, -1 * dimensiune_sir + 1, SEEK_CUR);
    }

    printf("Sirul %s apare de %d ori\n", argv[2], count);

    free(buffer);
    close(descriptor);

       return 0;
}
```

Inlocuieste 2 siruri de pe anumite pozitii cu cele date in fisier

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

int main(int argc, char *argv[])
{
    if (argc < 4 ) {
        puts("Argumente insuficiente in linia de comanda");
        return -1;
    }
    if (argc % 2 != 0) {
        puts("Sirul nu reprezinta perechi de numere");
        return -1;
    }

    int descriptor = open(argv[1], O_RDWR);
    if (descriptor < 0) {
        printf("Fisierul %s nu s-a putut deschide\n", argv[1]);
        return -1;
    }

    int lungime_fisier = lseek(descriptor, 0, SEEK_END);
    int i;
    int prima_pozitie;
    int a_doua_pozitie;
    char ch1, ch2;

    for (i = 2; i < argc - 1; i += 2) {
        prima_pozitie = atoi(argv[i]);
        a_doua_pozitie = atoi(argv[i+1]);

        if (prima_pozitie > lungime_fisier || a_doua_pozitie > lungime_fisier ||
prima_pozitie < 1 || a_doua_pozitie < 1){
            printf("Pozitia %s sau/si pozitia %s nu se gaseste in fisier", argv[i],
argv[i+1]);
            continue;
        }

        //citesc
        lseek(descriptor, prima_pozitie - 1, SEEK_SET);
        read(descriptor, &ch1, 1);
        lseek(descriptor, a_doua_pozitie - 1, SEEK_SET);
        read(descriptor, &ch2, 1);

        //scriu
        lseek(descriptor, prima_pozitie - 1, SEEK_SET);
        write(descriptor, &ch2, 1);
        lseek(descriptor, a_doua_pozitie - 1, SEEK_SET);
        write(descriptor, &ch1, 1);
    }

    close(descriptor);

    return 0;
}
```

Scote legaturiile subfisierelor unor foldere si sterge folderul

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

int is_special_directory(char* name) {
    if (strcmp(name, "..") == 0 || strcmp(name, ".") == 0) {
        return 1;
    }
    return 0;
```

```c
}

void remove_link(char* name, int inod) {

    DIR* director = opendir(name);

    if (director == NULL) {
        return;
    }

    chdir(name);

    struct dirent* entry;

    while ((entry = readdir(director)) != NULL) {
        if (is_special_directory(entry->d_name)){
            continue;
        }

        struct stat info;
        stat(entry->d_name, &info);

        if (S_ISDIR(info.st_mode)) {
            remove_link(entry->d_name, inod);
        }

        if (S_ISREG(info.st_mode) && info.st_ino == inod) {
            unlink(entry->d_name);
        }
    }

    chdir("..");
    rmdir(name);
    closedir(director);

}

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

    if (argc < 3) {
        puts("Argumente insuficiente");
        return -1;
    }

    struct stat info;
    if (stat(argv[1], &info)  == -1) {
        printf("Fisierul %s nu exista", argv[1]);
        return -1;
    }

    int inod = info.st_ino;

    remove_link(argv[2], inod);

    return 0;
}
```

Cauta fisierul (nu director) cu nume palindromic de length al numelui maxim

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

```c
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <dirent.h>

int is_special_directory(char* name) {
    if (strcmp(name, "..") == 0 || strcmp(name, ".") == 0) {
        return 1;
    }
    return 0;
}

int is_palindrom(char* name) {

    int descriptor = open(name, O_RDONLY);

    if (descriptor < 0){
        return 0;
    }

    char ch_stanga, ch_dreapta;

    int stanga = lseek(descriptor, 0, SEEK_SET);
    read(descriptor, &ch_stanga, 1);
    int dreapta = lseek(descriptor, -1, SEEK_END);
    read(descriptor, &ch_dreapta, 1);

    int i = 1;

    while (stanga < dreapta) {
        if (ch_stanga != ch_dreapta){
            close(descriptor);
            return 0;
        }

        stanga = lseek(descriptor, i, SEEK_SET);
        read(descriptor, &ch_stanga, 1);
        dreapta = lseek(descriptor, -1 * i - 1, SEEK_END);
        read(descriptor, &ch_dreapta, 1);

        i++;
    }

    close(descriptor);
    return 1;
}

void find_largest_palindrom(char* name, int* max_size, char* max_name) {

    DIR* director = opendir(name);

    if (director == NULL) {
        return;
    }

    chdir(name);

    struct dirent* entry;

    while ((entry = readdir(director)) != NULL) {
        if (is_special_directory(entry->d_name)){
            continue;
        }

        struct stat info;
        stat(entry->d_name, &info);

        if (S_ISDIR(info.st_mode)) {
            find_largest_palindrom(entry->d_name, max_size, max_name);
        }

        if (S_ISREG(info.st_mode) && is_palindrom(entry->d_name) && info.st_size >
*max_size) {
            *max_size = info.st_size;
            strcpy(max_name, entry->d_name);
        }
    }

    chdir("..");
    closedir(director);

}

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

    if (argc < 2) {
        puts("Argumente insuficiente");
        return -1;
    }

    int max_size = -1;
    char name[NAME_MAX];

    find_largest_palindrom(argv[1], &max_size, name);

    printf("Cel mai mare fisier palindromic este fisierul %s cu dimensiunea %d\n",
name, max_size);

    return 0;
}
```