

1.

Pentru prima problema am deschid mai intai fisierul din care urmeaza sa citesc.

Am citit numarul de linii din fisier.

Am citit linia pe care vreau sa o printez.

Am verificat daca linia este valida.

Daca a fost valida am sarit peste celelalte redundante cu un while care imi

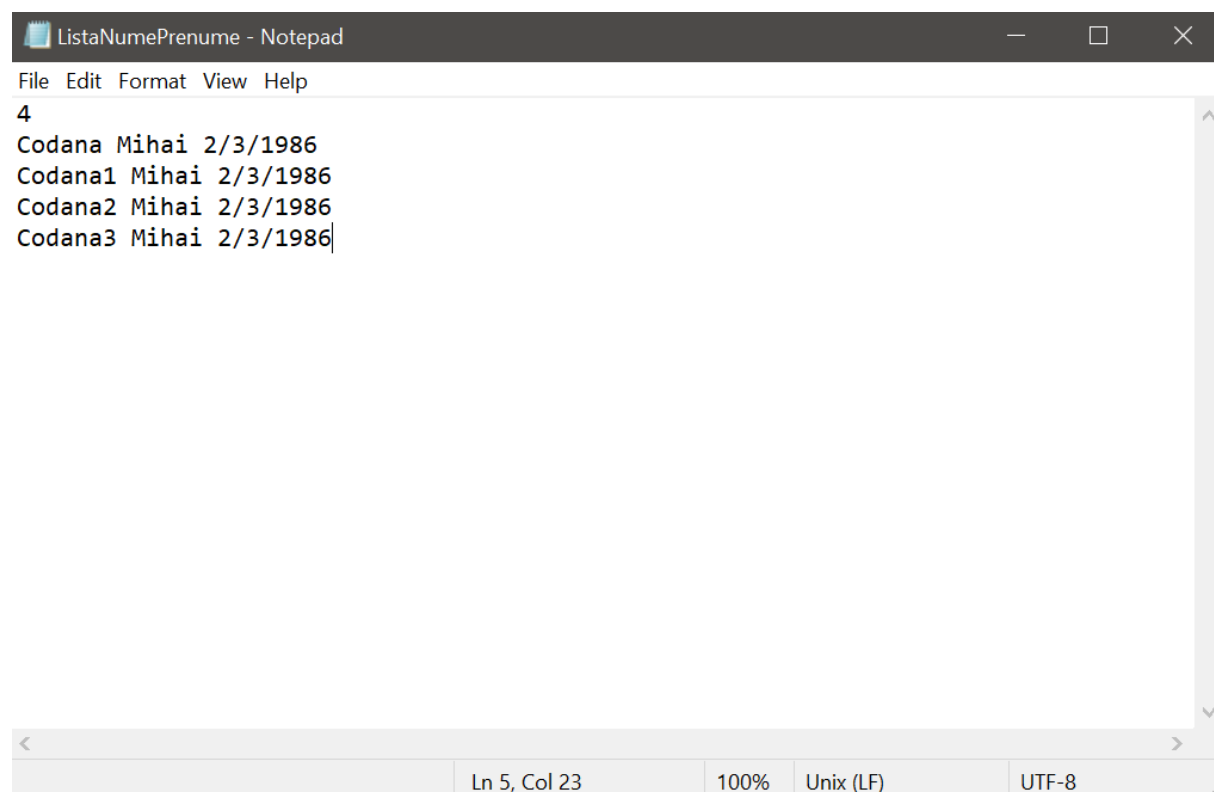
itereaza pana cand printed_line care este variabila dorita va fi 1

Apoi citim linia dorita si cu fscanf despartim numele prenumele si data.

iar pentru data deoarece are / intre numere am tokenizat cu un algoritm care merge

cu un pointer din "/" in "/" pana cand nu mai are ce sa gaseasca.

Intre timp am afisat numele prenumele si data de nastere pe vreme ce o tokenizam



```
File Edit Format View Help
4
Codana Mihai 2/3/1986
Codana1 Mihai 2/3/1986
Codana2 Mihai 2/3/1986
Codana3 Mihai 2/3/1986|
```

Ln 5, Col 23 100% Unix (LF) UTF-8

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

```
#include <string.h>
```

```
#define bufferLine 1024
```

```

int main(void)
{
    /* Deschidem fisierul sub format read text */
    FILE* file_in;
    file_in = fopen("ListaNumePrenume.txt", "rt");

    /* citim numarul de linii din fisier */
    int number_of_lines;
    fscanf(file_in, "%d", &number_of_lines);

    /* Citim ce linie vrem sa afisam */
    int printed_line;
    printf("Ce linie doresti sa afisezi din fisier? Introdu: ");
    scanf("%d", &printed_line);

    /* Daca linia nu este in fisier iesim*/
    if (printed_line > number_of_lines) {
        printf("Linie vida\n");
        return -1;
    }

    /* Citim liniile redundante pana cand vrem sa ajungel la linia dorita */
    char line[bufferLine];
    while (printed_line != 1) {
        /* Citim linie cu linie */
        fgets(line, sizeof(line), file_in);
        line[strlen(line) - 1] = '\0';
        printed_line--;
    }

    /* Stocam numele prenumele si data*/

```

```

char nume[bufferLine];
char prenume[bufferLine];
char data[bufferLine];
fscanf(file_in, "%s %s %s", nume, prenume, data);

/* Afisam */
printf("%d\n", number_of_lines);
printf("%s\n", nume);
printf("%s\n", prenume);
const char s[2] = "/";
char *token;

/* luam ziua */
token = strtok(data, s);

/* apoi luna si anul si afisam */
while( token != NULL ) {
    printf("%s\n", token );
    token = strtok(NULL, s);
}
fclose(file_in);
return 0;
}

```

2. Pseudocod:

Functia read_random merge de la 0 la 256 pe linii si coloane

si creez un numar random pe care il pun in matrice.

Functia allocmatrix alocata spatiu pentru matrice si returneaza matricea.

Functia FreeMatrix dezaloca spatiul alocat pentru memorie

Functie write_img scrie intr un fisier al carui nume e dat ca parametru in fisier datele din zona ROI

Functia negate_image merge pe zona de interes si aplica formula :x-255 unde x

este valoarea de la pozitia i si j.

Functia BlacWhite face acelasi lucru si aplica un if care zice daca valoarea de la i si j este mai mare de 127 o face 255 daca nu o face 0.

GrayimageProcessing este o functie generica care apeleaza alte functii la randul ei

Citim de la tastatura numele

Citim de la statura prenumele

Citim de la tastatura ziua

Citim de la tastatura luna

le aplicam atoi la zi si luna sa le facem inturi

Ne alocam matricea.

creem valorile random

Scriem zona de interes

aplicam negate si afisam

aplicam blackwhite si afisam
dezalocam matricea.

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

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

```
#include <string.h>
```

```
#include <time.h>
```

```
#define N 256
```

```
#define M 256
```

```
struct region{
```

```
    int x0;
```

```
    int y0;
```

```
    int x1;
```

```
    int y1;
```

```
};
```

```
/* Functie cu care citim numere random si le punem in matrice */
```

```
void read_random(unsigned char **matrix)
```

```
{;
```

```
    for (int i = 0; i < N; i++) {
```

```
        for (int j = 0; j < M; j++) {
```

```
            int num = (rand() % (256 - 0 + 1)) + 0;
```

```
            matrix[i][j] = num;
```

```
        }
```

```
    }
```

```
}
```

```
/* AllocMatrix este o functie care aloca memorie pentru o matrice
```

```
pe linii apoi pe coloane */
```

```
unsigned char** allocMatrix()
```

```
{
```

```
    unsigned char **matrix = (unsigned char**)malloc(N * sizeof(unsigned char*));
```

```
    for (int i = 0; i < N; ++i) {
```

```
        matrix[i] = (unsigned char*)malloc(M * sizeof(unsigned char));
```

```
    }
```

```
    return matrix;
```

```
}
```

```
/* Freematrix este o functie care dezaloca spatiul pentru memoria
```

```
matricii*/
```

```
void freeMatrix(unsigned char **matrix)
```

```
{
```

```
    for (int i = 0; i < N; ++i) {
```

```
        free(matrix[i]);
```

```
    }
```

```

    free(matrix);
}

/* Deschidem fisierul si scriem datele*/
void write_img(unsigned char** Img, struct region ROI, char *nume)
{
    FILE* file_out;
    file_out = fopen(nume, "wt");

    for (int i = ROI.x0; i >= ROI.x1; i--) {
        for (int j = ROI.y0; j >= ROI.y1; j--) {
            fprintf(file_out, "%u ", Img[i][j]);
        }
        fprintf(file_out, "\n");
    }
    fclose(file_out);
}

/* negam imaginea dupa formula din cerinta*/
int Negate_Image(unsigned char** Img, int row, int col, struct region ROI)
{
    for (int i = ROI.x0; i >= ROI.x1; i--) {
        for (int j = ROI.y0; j >= ROI.y1; j--) {
            Img[i][j] = 255 - Img[i][j];
        }
    }
    return 0;
}

/* Facem pixelii albi sau negrii in functie de valoarea lor */
int BlackWhite_Image (unsigned char** Image, int row, int col, struct region ROI)

```

```

{
    for (int i = ROI.x0; i >= ROI.x1; i--) {
        for (int j = ROI.y0; j >= ROI.y1; j--) {
            if (Image[i][j] > 127) {
                Image[i][j] = 255;
            } else {
                Image[i][j] = 0;
            }
        }
    }
}
return 0;
}

```

/* Functie Generica care apeleaza cele doua functii de procesare */

```

int Gray_Image_Processing (unsigned char** Img, int n, int m, struct region ROI, int (* pfunc)
(unsigned char**, int, int, struct region)) {
    pfunc(Img, n, m, ROI);
}

```

```

int main(void) {

```

```

    char nume[1024];

```

```

    char prenume[1024];

```

```

    char ziua[10];

```

```

    char luna[10];

```

```

    struct region my_region;

```

/* Citim numele prenumele si ziua de nastere si calculam zona de interes*/

```

    printf("numele: ");

```

```

    scanf("%s", nume);

```

```

    my_region.x0 = strlen(nume) % 5 + 10;

```

```

printf("prenumele: ");
scanf("%s", prenume);
my_region.x1 = strlen(prenume) % 5 + 10;

printf("ziua de nastere: ");
scanf("%s", ziua);
my_region.y1 = atoi(ziua) % 5 + 10;

printf("luna de nastere: ");
scanf("%s", luna);
my_region.y0 = atoi(luna) % 5 + 10;

printf("%d -> %d", my_region.x0, my_region.x1);
printf("%d -> %d", my_region.y0, my_region.y1);
/* Creem matricea */
unsigned char **matrix = allocMatrix();

/* citim din fisier */
read_random(matrix);

/* Afisam matricea */
write_img(matrix, my_region, "Imagine.txt");

/* Negam elementele dupa formula si afisam */
Gray_Image_Processing(matrix, N, M, my_region, &Negate_Image);
write_img(matrix, my_region, "image_neg.txt");

/* Facem elementele albe sau negre dupa formula si afisam */
Gray_Image_Processing(matrix, N, M, my_region, &BlackWhite_Image);
write_img(matrix, my_region, "image_bw.txt");

```



```
/* O dezalocam */  
free(matrix);  
return 0;  
}
```

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

```
struct item{  
    char N[20];  
    int ID;  
};
```

```
int main() {  
    /* arrayul generic */  
    void unic_table = (void)malloc(5 * sizeof(void));  
  
    /* primul element */  
    int first_number = 1;  
    memcpy(unic_table, (void*)&first_number, sizeof(first_number));  
  
    /* al doilea element */  
    float double_number = 1.1f;  
    memcpy(unic_table + sizeof(first_number), (void*)&double_number, sizeof(double_number));  
  
    /* elementul structura */
```

```
struct item third_number;

third_number.ID = 1;

strcpy(third_number.N, "Natalia");

memcpy(unic_table + sizeof(first_number) + sizeof(double_number), (void*)&third_number,
sizeof(third_number));


return 0;
}
```