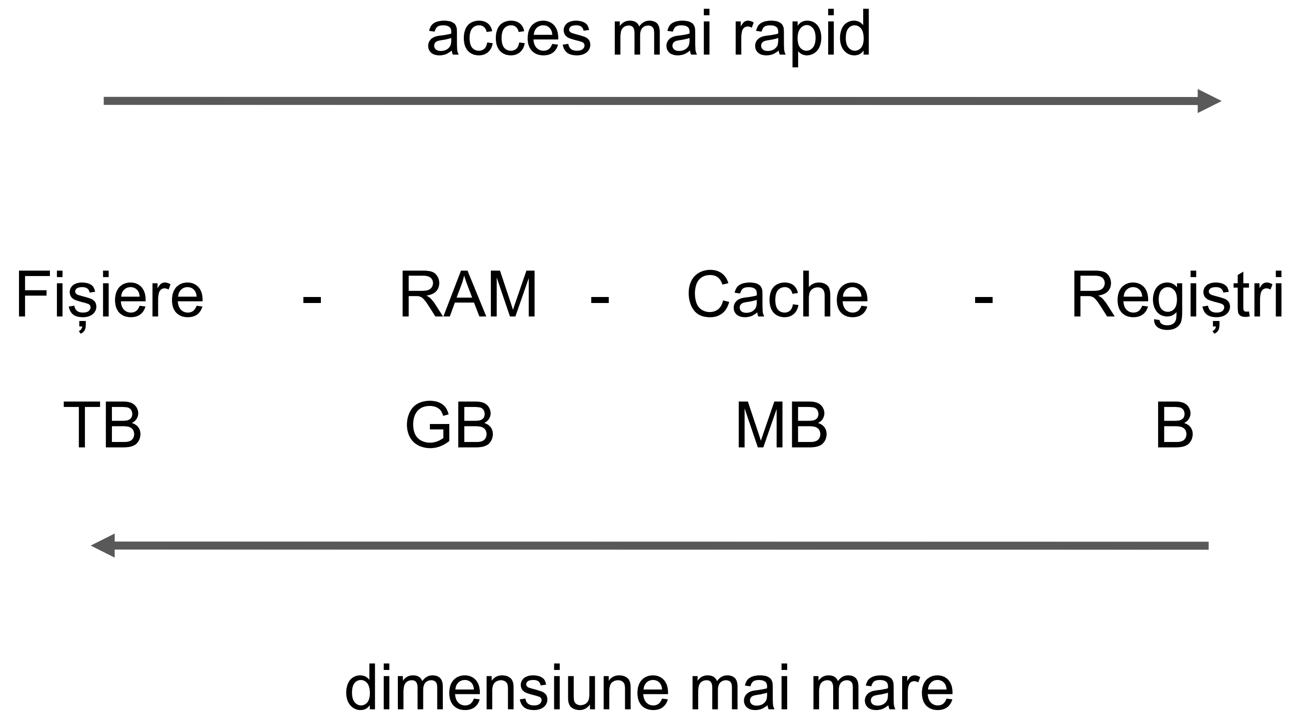
Programarea Calculatoarelor Cursul 11

Fișiere

Obiective

- Cum putem să citim din și să scriem în fișiere?
  - introducere
  - funcții importante
  - tipuri de fișiere: text, binar
  - exemple de prelucrare date

Stocare date



Fișiere

- Un fișier este o resursă folosită pentru stocarea datelor pe un dispozitiv de stocare (harddisk, solid state disk, DVD)
  - memorare pe termen lung
  - acces mai lent comparativ cu accesul la date din memoria RAM
- Poate fi abstractizat ca un șir de bytes
- În afara de câteva operații de pregătire citirea și scrierea se întâmplă la fel ca citirea de la tastatură și scrierea pe ecran
  - și funcțiile de procesare reflectă acest lucru
- Necesită interacțiune cu sistemul de operare
  - sursă de erori
- Prelucrarea lor se face folosind funcțiile din biblioteca standard de intrare/ieșire `stdio.h` și structura `FILE`

Structura FILE

- Există 3 fișiere create automat pentru orice program
  - `stdin` - intrarea standard - legat de tastatură
  - `stdout` - ieșirea standard - legat de ecran
  - `stderr` - ieșirea standard pentru erori - legat de ecran
- Funcțiile de IO scanf/printf lucrează direct cu aceste fișiere
- Pentru operații pe alte fișiere trebuie să deschidem acel fișier
- Starea unui fișier este memorată într-o structură de tip FILE
- Orice operație are nevoie de un pointer la un fișier valid

Structura FILE

```
typedef struct _iobuf
{
    char*   _ptr;
    int _cnt;
    char*   _base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char*   _tmpfname;
} FILE;
```

- definiție de tip din `stdio.h`
- datele sunt stocate la pointerul `_ptr`
- programatorul interacționează cu fișier prin funcții = nu trebuie să cunoaștem semnificația câmpurilor

```
#define EOF (-1)
#define SEEK_CUR 1
#define SEEK_END 2
#define SEEK_SET 0
#define FILENAME_MAX 260
#define FOPEN_MAX 20
```

- constanta care semnifică sfârșitul fișierului
- constante folosite la funcția `fseek`
- lungimea maximă pentru numele unui fișier
- număr maxim de fișiere care pot fi deschise simultan

Fișiere

- Mod de prelucrare
  - mod text
    - datele sunt salvate sub formă de caractere direct interpretabile de om
    - ocupă de obicei mai mult spațiu
  - mod binar
    - datele sunt salvate exact cum sunt stocate în memorie
    - ocupă de obicei mai puțin spațiu
    - se poate interpreta doar dacă se cunoaște exact structura
      - putem ascunde datele
    - toate fișierele sunt de fapt fișiere binare
      - un fișier text nu este altceva decât fișierul binar corespunzător reprezentării sub formă de string a datelor

Procesare fișiere

- Presupune pași
  - deschidere fișier - obținere pointer FILE valid
  - prelucrarea datelor din fișier - folosind funcții care lucrează cu pointer la FILE
  - închiderea fișierelor
- Interacțiune cu sistemul de operare
  - pentru prelucrare fișier avem nevoie
    - cale corectă către fișier
    - drepturi de acces pentru operații
  - funcțiile de prelucrare fac legătura cu fișier interacționând cu sistemul de operare
  - se recomandă verificarea valorilor returnate de funcții pentru erori

Funcția fopen

```
FILE* fopen(const char* filename, const char* mode);
```

- Încearcă deschiderea fișierului cu numele dat
- Parametrii
  - `const char* filename` - calea pe disk către fișier
  - `const char* mode` - string care specifică modul de prelucrare
- Valoare returnată
  - Un pointer valid la o structură FILE
  - Pointerul NULL în caz de eșec
- Erori posibile
  - cale incorectă
  - nu avem drepturi suficiente

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Funcția fopen - moduri de prelucrare

- String-ul `mode` are maxim 3 caractere
  - primul caracter (obligatoriu) poate fi r, w, a
    - r - pentru citire din fișier (read)
    - w - pentru scriere într-un fișier nou sau suprascriere (write)
    - a - pentru adăugare la un fișier (append)
  - al doilea caracter (opțional) +
    - poate lipsi
    - înseamnă că se dorește atât citirea din cât și scrierea în fișier
  - al treilea caracter (opțional) b
    - poate lipsi
    - prezența lui semnifică modul de prelucrare în binar

Deschidere fișiere - exemple simple

```c
#include <stdio.h>

int main(){

    FILE* pf = fopen("file.txt", "r");
    if (pf == NULL)
        return -1;



    FILE* pf2;
    pf2 = fopen("..//date.out", "wb");
    if (pf2 == NULL){
        puts("Nu se poate deschide");
        return -2;
    }

    return 0;
}
```

- deschidem fișierul text pentru citire
- dacă nu se poate deschide
- returnăm cod de eroare
- de obicei nu se poate reveni de la o eroare de tip intrare/ieșire la fișiere

- deschidem fișierul date.out din directorul părinte
- mod de prelucrare: scriere în mod binar
- dacă nu se poate deschide
- afișăm un mesaj de eroare
- returnăm cod de eroare
- de obicei nu se poate reveni de la o eroare de tip intrare/ieșire la fișiere

Specificare nume fișier

- Calea către fișier poate fi
  - numele fișierului + extensie (este tot cale relativă, vezi mai jos)
    - se prelucrează fișierul din directorul curent
  - cale relativă - se furnizează locația fișierului relativ la directorul curent, numele fișierului și extensia
    - directorul curent este directorul proiectului sau locația de unde se lansează executabilul
    - pentru a specifica directorul părinte se utilizează ..
    - se poate folosi de mai multe ori ../../file.txt
  - cale absolută - se furnizează calea completă către fișier incluzând partiția, toate directoarele, numele fișierului și extensia
    - pe Windows fiecare caracter de backslash \ din string-ul care reprezintă calea trebuie dublată \\ - este caracter special
    - dacă apare spațiu în numele directorului sau al fișierului trebuie să încadrăm calea între ghilimele cu \"

Erori des întâlnite

- La deschiderea fișierului putem avea erori
  - când calea către fișier este incorectă
    - se recomandă preluarea ei direct de la sistemul de operare
    - se înlocuiește atent fiecare \ cu \\ (pe Windows)
    - se verifică locația de unde a fost lansat executabilul
  - nu am inclus extensia în cale
    - extensia poate fi ascunsă (în File Explorer pe Windows)
    - extensia unui fișier poate fi orice - fără extensie, .in, .txt. dat
    - este doar o convenție care indică formatul fișierului
  - dacă nu avem drepturi
    - fișierul nu poate fi scris
      - este deschis deja de o altă aplicație
      - are drepturi de citire doar - read-only eng.
- Se recomandă verificarea pointerului returnat de fopen

Funcția fscanf

```
int fscanf(FILE* pf, const char* format, ...);
```

- Funcția citește din fișier conform specificatorilor de format din șirul de caractere `format` și stochează valorile la adresele care urmează
- Parametrii
  - `FILE* pf` - pointer valid la un fișier / structură FILE
  - `const char* format` - specificatori de format (analog ca la scanf)
  - … - adresele de variabilelor unde se face citirea
- Valoare returnată
  - Numărul de argumente citite cu succes
  - constanta EOF dacă s-a produs o eroare înaintea primei citiri
- Erori posibile
  - pointer la fișier invalid
  - pointer invalid folosit pentru destinație

Funcția fprintf

```
int fprintf(FILE* pf, const char* format, ...);
```

- Funcția scrie în fișier conform specificatorilor de format din șirul de caractere `format` expresiile care urmează (0 sau mai multe)
- Parametrii
  - `FILE* pf` - pointer valid la un fișier / structură FILE
  - `const char* format` - specificatori de format (analog ca la printf)
  - … - expresiile unde se face citirea
- Valoare returnată
  - Numărul de caractere scrise cu succes
  - o valoare negativă în caz de eroare
- Erori posibile
  - pointer la fișier invalid

Funcția fclose

```
int fclose(FILE* pf);
```

- Închide fișierul identificat prin pointerul `pf`
- Parametrii
  - `FILE* pf` - pointer valid la un fișier / structură FILE
- Valoare returnată
  - 0 în caz de succes
  - constanta EOF în caz de eroare
- Erori posibile
  - pointer la fișier invalid
  - fișierul a fost închis deja

IO fișiere text - exemplu simplu

```c
#include <stdio.h>

int main(){
    FILE* pf = fopen("file.txt", "w");
    if (pf == NULL)
        return -1;

    int x = 7;
    fprintf(pf, "%d\n%d", x, 5);
    fclose(pf);

    pf = fopen("file.txt", "r");
    int y;
    fscanf(pf, "%d%d", &x, &y);
    printf("x = %d y = %d\n", x, y);
    fclose(pf);
    return 0;
}
```

- deschidem fișierul text pentru scriere
- dacă nu se poate deschide
- returnăm cod de eroare

- scriem două valori pe rânduri diferite
- închidem fișierul

- refolosim variabila pentru a obține un pointer nou pentru citire
- citim 2 întregi - observăm că se sare peste caracterele albe (spații, tab-uri, linii noi)
- închidem fișierul

IO fișiere text recomandări

- Trebuie să privim citirea din/scrierea în fișiere exact ca citirea de la tastatură respectiv scrierea pe ecran
  - funcțiile de procesare reflectă acest lucru
  - doar adăugăm litera f și furnizăm pointer la FILE
- De multe ori NU este necesară citirea rând cu rând și separarea cuvintelor
  - se pot citi direct numere cu %d (sau specificatorul potrivit)
  - se sare peste spații, linii noi (caractere albe) în mod automat
  - se pot sări peste alte caractere nedorite fie prin includerea lor în specificatorul de format, fie prin citirea unui string adițional
- Dacă nu se știe numărul de elemente existente în fișier
  - se urmărește valoarea returnată de `fscanf` pentru a sesiza sfârșitul fișierului = trebuie să fie egală cu numărul de argumente

IO fișiere text - exemplu complex

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

int main(){
   FILE* pf = fopen("file.txt", "r");
   if (pf == NULL)
       return -1;


   int n = 10;
   int* a = calloc(n, sizeof(int));
   int ai;
   int pos = 0;
   while(fscanf(pf, "%d", &ai) == 1){
       if (pos == n){
           n *= 2;
           a = realloc(a, n * sizeof(int));
           if (a == NULL)
               return -2;
       }
       printf("a[%d] = %d\n", pos, ai);
       a[pos++] = ai;
   }
   fclose(pf);
   return 0;
```

- fișierul de text conține un număr necunoscut de numere întregi separate prin spațiu sau pe rânduri diferite

- deschidem fișierul text pentru citire
- dacă nu se poate deschide
- returnăm cod de eroare

- dimensiunea inițială a tabloului
- tabloul se alocă dinamic fiindcă nu știm dimensiunea
- variabilă temporară pentru citire
- poziția unde inserăm în tablou
- se citește și se verifică codul returnat
- dacă avem deja *n* numere citite
- dublăm dimensiunea
- realocăm tabloul *a*
- dacă nu am putut realoca (rar se întâmplă)
- returnăm cod de eroare

- afișăm pentru verificare
- adăugăm în tablou pe poziția curentă și trecem la cea următoare

Funcția feof

```
int feof(FILE* pf);
```

- Determină dacă s-a ajuns la finalul fișierului <u>după</u> ultima operație
- Parametri
  - `FILE* pf` - pointer valid la un fișier / structură FILE
- Valoare returnată
  - Valoarea nenulă (adevărată) dacă s-a terminat fișierul
  - 0 dacă NU s-a terminat fișierul
- Erori posibile
  - pointer la fișier invalid
- Se recomandă folosirea valorii returnate de la funcțiile de citire pentru determinarea sfârșitului de fișier

Funcția freopen

```
FILE *freopen( const char *filename, const char *mode,
               FILE *fp );
```

- Redirecționează un fișier sau intrarea/ieșirea standard către un fișier
- Parametrii
  - `const char *filename` - cale către fișierul către care se face redirecționarea
  - `const char *mode` - mod de prelucrare fișier (ca la fopen)
  - `FILE *fp` - fișierul de la care se face redirecționarea
- Valoare returnată
  - Copia pointerului la fișierul nou în caz de succes
  - Pointerul NULL în caz de eroare
- Erori posibile
  - pointer la fișier invalid
  - drepturi insuficiente

Funcția fseek

```
int fseek( FILE *fp, long origin, int offset);
```

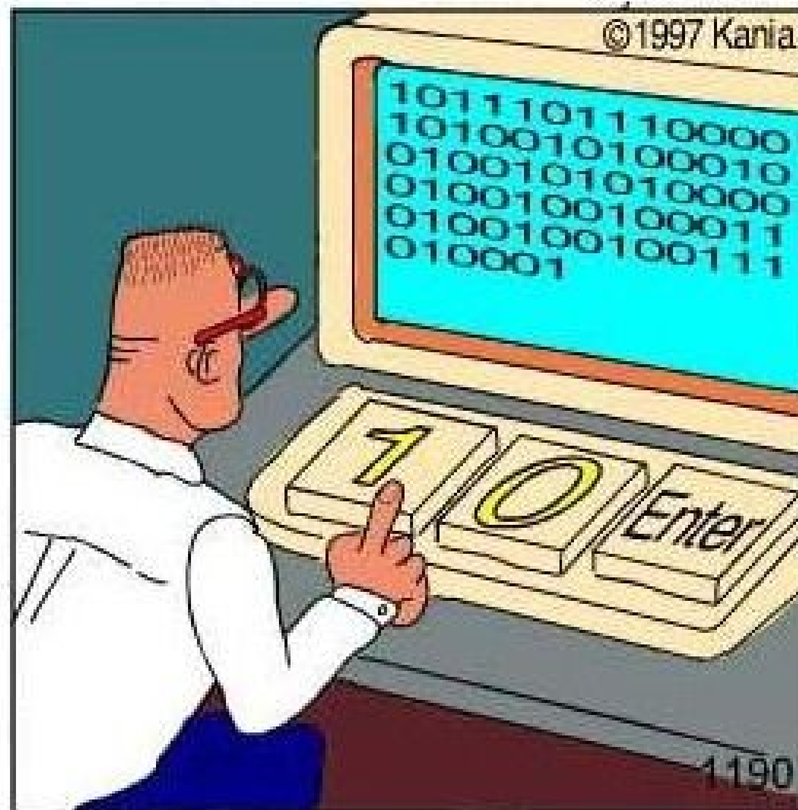
- Mută cursorul pentru prelucrarea fișierului la poziția `offset` relativ față de origine
- Parametrii
  - `FILE* fp` - pointer valid la un fișier / structură FILE
  - `long origin` - originea / punct de referință
  - `int offset` - poziție relativă (poate fi și negativă)
- Valoare returnată
  - valoarea 0 în caz de succes
  - orice valoare nenulă în caz de eșec
- Valori posibile pentru origine (constante definite în `stdio.h`)
  - `SEEK_SET` început fișier,
  - `SEEK_CUR` poziția curentă,
  - `SEEK_END` sfârșit fișier

Funcția ftell

```
long ftell(FILE *fp);
```

- Returnează poziția curentă a cursorului pentru prelucrarea fișierului
- Parametrii
  - `FILE* fp` - pointer valid la un fișier / structură FILE
- Valoare returnată
  - valoare pozitivă sau 0 în caz de succes
  - -1 în caz de eroare
- Se poate utiliza pentru a determina mărimea unui fișier
  - posibil nu funcționează corect pentru fișiere ale căror dimensiuni depășesc limita maximă `long` - vezi `fgetpos`

Fișiere binare



Real programmers code in binary.

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Fișiere binare

- Aproape toate fișierele sunt fișiere binare
  - deschiderea lor pentru vizualizare arată conținutul lor interpretat ca un șir de caractere
- Pe Linux nu există diferență între fișier binar și text
- Extensia unui fișier nu contează
  - are menirea doar să ofere informație despre formatul datelor
- În cazul fișierelor binare (recapitulare)
  - datele sunt salvate exact cum se stochează în memorie
  - ocupă de obicei mai puțin spațiu
  - se poate interpreta doar dacă se cunoaște exact structura
    - putem ascunde datele
  - toate fișierele sunt de fapt fișiere binare
    - un fișier text nu este altceva decât fișierul binar corespunzător reprezentării sub formă de string a datelor

Funcția fread

```
size_t fread(void* buffer, size_t size, size_t count, FILE* pf);
```

- Citește `count` elemente de câte `size` bytes din fișier și copiază la zona de memorie `buffer`
- Parametrii
  - `void* buffer` - zona de memorie destinație
  - `size_t size` - mărimea unui singur element
  - `size_t count` - numărul de elemente
  - `FILE* pf` - pointer valid la un fișier / structură FILE
- Valoare returnată
  - numărul de elemente citite cu succes [0, count]
- Erori posibile
  - pointer la fișier invalid
  - buffer nealocat

Funcția fwrite

```
size_t fwrite(const void* buffer, size_t size, size_t count,
              FILE* pf);
```

- Scrie `count` elemente de câte `size` bytes de la zona de memorie `buffer` în fișier
- Parametrii
  - `const void* buffer` - zona de memorie sursă
  - `size_t size` - mărimea unui singur element
  - `size_t count` - numărul de elemente
  - `FILE* pf` - pointer valid la un fișier / structură FILE
- Valoare returnată
  - numărul de elemente scrise cu succes [0, count]
- Erori posibile
  - pointer la fișier invalid
  - buffer nealocat

Fișiere binare - exemplu simplu

```c
#include <stdio.h>

int main(){
    FILE* pf = fopen("in.bin", "rb");
    if (pf == NULL)
        return -1;
    int n;
    fread(&n, sizeof(n), 1, pf);
    int a[n];
    fread(a, sizeof(int), n, pf);
    fclose(pf);

    for(int i=0; i<n; i++)
        a[i] = -a[i];
    pf = fopen("out.bin", "wb");
    if (pf == NULL)
        return -2;
    fwrite(&n, sizeof(n), 1, pf);
    fwrite(a, sizeof(int), n, pf);
    fclose(pf);
    return 0;
}
```

- inversează semnul elementelor unui tablou cu n < 1000 int-uri

- deschidem fișierul binar pentru citire
- dacă nu se poate deschide
- returnăm cod de eroare

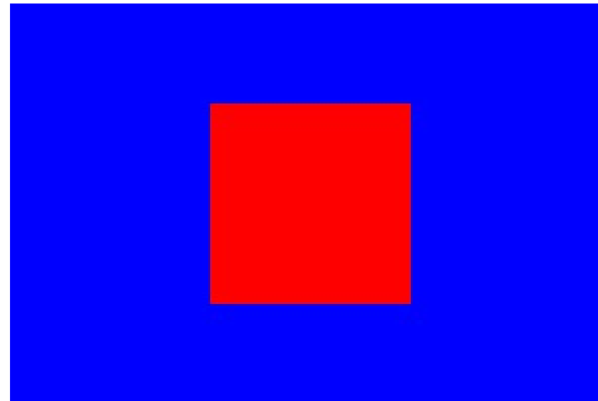
- dimensiunea tabloului
- tabloul se alocă static fiindcă are dimensiune mică
- se citește tot tabloul cu un singur apel
- se închide fișierul

- se schimbă semnul

- deschidem fișierul binar pentru ieșire
- dacă nu se poate deschide
- returnăm cod de eroare
- scriem n
- scriem tabloul întreg cu un singur apel
- se închide fișierul

Analiză problemă - Editarea unei imagini bmp

- Se dă o imagine de tip *bmp* (Bitmap Microsoft) de dimensiune *h* rânduri și *w* coloane
  - se poate crea folosind aplicația Paint sub Windows
- Să se modifice imaginea astfel încât să conțină un pătrat roșu pe un fundal albastru centrat în imagine



Imagine bmp - structuri header - windows.h

```
typedef struct tagBITMAPFILEHEADER {
    WORD bfType;
    DWORD bfSize;
    WORD bfReserved1;
    WORD bfReserved2;
    DWORD bfOffBits;
} BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER {
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    LONG biXPelsPerMeter;
    LONG biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER;

typedef struct tagRGBTRIPLE {
    BYTE rgbtBlue;
    BYTE rgbtGreen;
    BYTE rgbtRed;
} RGBTRIPLE;
```

```
BYTE = char
WORD = short
DWORD = int
LONG = long
```

- Un fișier bmp este un fișier binar
- este compus din structuri definite în `windows.h`
- prima structură este de tip `BITMAPFILEHEADER` - descrie structura fișierului
- a doua structură este de tip `BITMAPINFOHEADER` - descrie dimensiunile imaginii și formatul
- Urmează un tablou de structuri `RGBTRIPLE` care practic stochează pixelii din imagine

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Editarea unei imagini bmp - implementare

```c
#include <stdio.h>
#include <windows.h>
int main(){
    FILE* fp = fopen("img.bmp", "rb+");
    BITMAPFILEHEADER head;
    fread(&head, sizeof(head), 1, fp);
    BITMAPINFOHEADER info;
    fread(&info, sizeof(info), 1, fp);
    int h = info.biHeight;
    int w = info.biWidth;
    RGBTRIPLE* pixels = calloc(sizeof(RGBTRIPLE), h*w);
    fread(pixels, sizeof(RGBTRIPLE), h*w, fp);
    for(int i = 0; i<h*w; i++) {
        pixels[i].rgbtBlue = 255;
        pixels[i].rgbtGreen = 0;
        pixels[i].rgbtRed = 0;
    }
    int h2 = h/2, h4 = h/4, w2 = w/2;
    for(int i = h2 - h4; i < h2 + h4; i++){
        for(int j = w2 - h4; j < w2 + h4; j++){
            pixels[i * w + j].rgbtBlue = 0;
            pixels[i * w + j].rgbtRed = 255;
        }
    }
    fseek(fp, sizeof(head) + sizeof(info), SEEK_SET);
    fwrite(pixels, sizeof(RGBTRIPLE), h*w, fp);
    fclose(fp);
    return 0;
}
```

- structurile necesare sunt în windows.h
- deschide în mod binar citire + scriere
- declarăm header-ul
- și îl citim
- declarăm header-ul info
- și îl citim
- preluăm înălțimea
- și lățimea
- pixelii sunt un șir de tripleți RGBTRIPLE
- se citesc pixelii
- se crează fundalul albastru prin setarea celor trei componente la toți pixeli
- se colorează în centru cu roșu
- pixelii sunt stocați rând după rând
- poziția i * w + j reprezintă rândul i și coloana j
- mutăm cursorul după header-e
- scriem pixelii modificați în fișier
- închidem fișierul