File in C programing

FILE *fp, *fin, *fout;

FILE – cuvântul cheie care semnifică o structură dedate de tip fișier;
fp – descriptor de fișier;
* - pointer spre tipul FILE.

fp = fopen(const char * cale, const char * mod);

cale – calea de directoare + nume valid de fișier, cuprinse între ghilimele.
mod – modul de prelucrarea a fișierului după deschiderea acestuia, cuprinse, de afemenea, între ghilimele.
Poate avea următoarele valori:
”r” – fișierul se va deschide pentru citire;
”w”- fișierul se va deschide pentru scriere;
”a” - fișierul se va deschide pentru adăugare de înregistrări, poziționarea se face la sfârșitul fișierului;
”rb” - fișierul se va deschide pentru citire binară;
”wb” - fișierul se va deschide pentru scriere binară;
”r+b” - fișierul se va deschide pentru citire și scriere binară.

Un fișier inexistent, deschis în modul ”w” va fi creat(dacă este posibil) în momentul deschiderii, iar un fișier existent deschis în modul ”w” va fi creat din nou, ceea ce înseamnă că vechiul său conținut se va șterge. Unui fișier inexistent deschis în modul ”r” i se va asocia pointerul NULL.

Declarare:

```
FILE *fp;
fp = fopen("program.txt","w"); // deschiderea fișierului pentru scriere

fp = fopen("program.txt","r");  // deschiderea fișierului pentru citire
```

Verificarea existenței unui fișier:

```
if(fp == NULL)
 {
   printf("Error!");
   exit(1);
 } else{}
```

```
if (!fp)
      return 1;
```

```
if (!fp)
   return printf("Error");
```

Citirea unui număr din fișier:

```
while (!feof (fp)){
   fscanf(fp,"  %d", &num);  // fscanf(fp,"%lf", &n);  număr double
   printf(" %d",num);
}
```

Scrierea unui număr în fișier:

```
fprintf(fp,"%d",num);
```

Citirea unui string din fișier:

```
while (fgets(str,50, fp)!= NULL)
      printf("%s", str);
```

Închiderea unui fișier:

fclose(fp);

```
#include <stdio.h>
int main(){
        FILE *fp;
        fp = fopen(“input.txt”, “r”);
        printf(“%d”, EOF);
        fclose(fp);
return(0);
}
```

Valoarea echivalentă a EOF este -1.

EOF – character constant care indică sfârșit de fișier. Citirea lui se testează cu funcție **feof()**, ce returnează o valoare nenulă dacă a fost detectat caracterul EOF și zero dacă nu a fost citit caracterul.

Modalități de detectare a sfârșitului de fișier:

Fișiere textuale:

if (ch == EOF) break ;
printf(”%c”, ch) ;

Fișiere binare :

feof(), poate fi folosit și pentru fișiere textuale.
if(feof(fp) == 1)
printf(”End of file”) ;

File I/O: opening a text file

```
#include<stdio.h>

        int main()
        {
                FILE *fp;
                int x;

                fp =fopen("output.txt", "w");

                if (!fp)
                        return 1;

                for (x=1; x<=10; x++)
                        fprintf(fp,"%d\n", x);

                fclose(fp);

                return  0;
        }
```

File I/O: reading a text file

```
#include<stdio.h>

        int main()
        {
                FILE *fp;
                char str[50];
```

```
                fp = fopen("input.txt","r");
                if (!fp)
                        return 1;

                while (fgets(str, 50, fp)!=NULL)
                        printf("%s", str);

                fclose(fp);
                return 0;
        }
```

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int num, sum = 0;
   FILE *fp;
   fp = fopen("program.txt","w");

   if(fp == NULL)
   {
      printf("Eroare!");
      exit(1);
   } else{

    for(int i=5; i<=15; i++)
        fprintf(fp,"%3d",i);
}
        fclose(fp);
fp = fopen("program.txt","r");

   while (!feof (fp)){
        fscanf(fp,"%3d", &num);
        printf("%3d",num);
    }
    fclose(fp);

    return 0;
}
```

Funcțiile fread() și fwrite()

fwrite(&record, size, n, fp);

&record – pointerul la blocul de date care va fi scris în fișier;

size – lungimea în octeți a blocului scris (n*size);

n – numărul de itemi scriși;

fp – pointerul la fișierul în care se scrie.

fread(&record, size, n, fp);

&record – pointerul la blocul de date în care se va scrie datele din fișier;

size – lungimea în octeți a blocului citit (n*size);

n – numărul de itemi citiți;

fp – pointerul la fișierul din care se face citirea.

fgets(str, n, fp); preia într-un șir n-1 caractere dintr-un fișier.

str – pointerul blocului de date în care se va scrie stringul citit din fișier;

n – se va citi n-1 caractere;

fp – pointerul la fișierul din care se face citirea.

fputs(str, fp); scrie un șir de caractere într-un fișier.
str – pointerul blocului de date din care se va prelua stringul;
fp – pointerul la fișierul în care se face afișarea.
fseek(fp, size-movement, position); permite poziționarea aleatoare într-un fișier.
fp – pointerul către fișier;
size-movement - definește numărul de octeți peste care se va deplasa capul de citire/scriere;
position – determină locul în care se va face deplasarea. Poate avea valorile: zero – deplasare la începutul fișierului; unu – deplasare de la poziția curentă a capului decitire; doi – deplasare de la sfârșitul fișierului.

```
#include<stdio.h>
#include<string.h>
#include <stdlib.h>
typedef struct{
    char nume[20];
    int nota;
} record;

void citire(record * pt, int n);
void afisare(record * pt, int n);
void writeBin(record * pt, int n,  FILE * fpb);
void writeTxt(record * pt, int n,  FILE * fpt);
void readBin(record * pt, int n,  FILE * fpb);
void readTxt(record * pt, int n,  FILE * fpt);
void clear_buffer();

int main ()
{
  record * pt;
  FILE * fpb, *fpt;
  int n;
  printf("n="); scanf("%d", &n);

  pt = (record *) malloc (n*sizeof(record));

  citire(pt, n);
  printf("\n-------Afisare tastatura---------");
  afisare(pt, n);

  fpb = fopen("output.bin", "wb");
  fpt = fopen("output.txt", "w");
  if (!fpb){
      return 1;
  } else {
      writeBin(pt, n, fpb);
  }
  if (!fpt){
      return 1;
  } else {
      writeTxt(pt, n, fpt);
  }
  fclose(fpb);
  fclose(fpt);

  fpb = fopen("output.bin", "rb");
  fpt = fopen("output.txt", "r");
```

```
  printf("\n-------Afisare BinFile---------");
  readBin(pt, n, fpb);
  afisare(pt, n);
  printf("\n-------Afisare TxtFile---------");
  readTxt(pt, n, fpt);
  afisare(pt, n);
  fclose(fpb);
  fclose(fpt);

  return 0;
}
void citire(record * pt, int n){
    for(int i = 0; i < n; ++i){
        printf("\nNume: ");
        scanf("%s", (pt+i)->nume);
        clear_buffer();  //fflush(stdin);
        printf("Nota: ");
        scanf("%d", &(pt+i)->nota);
    }
}
void afisare(record * pt, int n){
    for(int i = 0; i < n; ++i){
        printf("\n%s: %d", (pt+i)->nume, (pt+i)->nota);
    }
}
void writeBin(record * pt, int n,  FILE * fpb){
    for(int i = 0; i < n; ++i){
        fwrite((pt+i), sizeof(pt), n, fpb);
    }
}
void writeTxt(record * pt, int n,  FILE * fpt){
    for(int i = 0; i < n; ++i){
        fprintf(fpt, "%s %3d\n", (pt+i)->nume, (pt+i)->nota);
    }
}
void readBin(record * pt, int n,  FILE * fpb){
    for(int i = 0; i < n; ++i){
        fread((pt+i), sizeof(pt), n, fpb);
    }
}
void readTxt(record * pt, int n,  FILE * fpt){
    for(int i = 0; i < n; ++i){
        fscanf(fpt, "%s%3d\n", (pt+i)->nume, &(pt+i)->nota);
    }
}
void clear_buffer(){
    char ch;
    do{
        ch = getchar();
    } while( ch != '\n');
}
```

https://www.codingunit.com/c-tutorial-binary-file-io