CURS 10.Operaţii cu fişiere

Un fişier este o colecţie de date de acelaşi tip, numite înregistrări, memorate pe suport extern (hard – disc,CD,...). Avantajele utilizării fişierelor sunt o consecinţă a proprietăţilor memoriilor externe. Fişierele permit memorarea unui volum mare de date persistente. Un fişier are o înregistrare care marchează sfârşitul de fişier. În cazul fişierelor de intrare de la tastatură sfârşitul de fişier se generează prin CTRL+Z. El poate fi pus în evidenţă folosind constanta simbolică EOF definită în `stdio.h`.
Prelucrarea fişierelor implică un număr de operaţii specifice acestora. Orice fişier, înainte de a fi prelucrat, trebuie să fie deschis. De asemenea la terminarea prelucrării unui fişier acesta trebuie închis.
Operaţiile de deschidere – închidere se pot realiza prin intermediul unor funcţii speciale de bibliotecă.

Alte operaţii frecvente:
- *Crearea unui fişier;*
- *Actualizarea unui fişier;*
- *Poziţionarea într-un fişier;*
- *Consultarea unui fişier;*
- *Adăugarea de înregistrări într-un fişier;*
- *Stergerea unui fişier.*

Sistemul de I/O din C separă printr-o anumită abstractizare programatorul de echipament. Forma abstractă se numeşte stream iar instrumentul efectiv care se poate utiliza este *fişierul.*

In C, conform standardului, se poate lucra cu fişiere fie la nivel binar, fie la nivel de caracter. Diferenţa constă în modul în care sunt tratate inregistrările din fişier. Astfel, dacă acestea sunt tratate la nivel de caracter, atunci operaţiile de IO se vor realiza prin utilizarea unor structuri de tipul şir de caractere, referite fie sub formă de variabile de tip şir de caractere (char[]) fie pointeri spre caractere (char*).

Dacă accesul la fişier se face in mod binar, atunci datele se vor referi prin pointeri de tipul void*. Prin intermediul acestora, practic se poate face referire la orice tip de dată din program. Astfel, fişierele tratate la nivel binar pot salva date de orice tip. Evident, în comparaţie cu tratarea fişierelor la nivel şir de caractere, nu avem un delimitator de înregistrare (precum caracterul \0), astfel încât, la fiecare operaţie de scriere / citire trebuie să se specifice lungimea înregistrării, adică numărul de octeţi scrişi / citiţi.

Trebuie să remarcăm faptul că limbajul de programare C unifică modul de lucru cu fişierele. Astfel, intrările / ieşirile sunt tratate unitar, fie că vorbim de fişiere de pe disc fie că vorbim despre dispozitivele standard de intrare / ieşire precum stdin, stdout sau stderr. Astfel, funcţiile de citire / scriere din dispozitivele standard, din fişiere sau şiruri de caractere precum scanf / fscanf / sscanf respectiv printf / fprintf / sprintf sunt realizate in mod unitar şi au utilizare similară.

10.1. Lucrul cu fişiere la nivel de şiruri de caractere

Deschiderea unui fişier

La acest nivel se utilizează funcţia *fopen* pentru deschiderea unui fişier. Ea returnează un pointer spre tipul FILE (tipul fişier), tip definit în fişierul stdio.h. In caz de eroare, funcţia fopen returnează pointerul NULL. Funcţia fopen realizează o alocare de memorie şi returnează zona de memorie alocată în cazul in care sistemul de operare reuşeste să deschidă fişierul specificat în modul de deschidere solicitat de funcţie.

Prototipul funcţiei *fopen* este:
```
FILE *fopen(const char *cale, const char *mod);
```
unde:

cale reprezintă calea pe disc a fişierului care se deschide.
mod este un pointer spre un şir de caractere care defineşte modul de prelucrare al fişierului după deschidere. Acest şir se defineşte în felul următor:

”r” deschidere în citire (read); fişierul trebuie să existe.
”w” deschidere în scriere (write); se deschide un fişier gol. Dacă fişierul deja există pe disc, conținutul acestuia este șters și se crează un fișier nou
”a” deschidere pentru adăugare (append); datele vor fi scrise la finalul fişierului. Dacă fişierul nu există, atunci acesta este creat.
”r+” deschidere pentru modificare (citire sau scriere); fişierul trebuie să existe
“w+” crează un fişier gol pentru scriere / citire;
”a+” deschide un fişier pentru citire şi adăugare.

Dacă se deschide un fişier inexistent cu modul ”w” sau ”a”, atunci el este deschis în creare. Dacă se deschide un fişier existent cu modul ”w”, atunci conţinutul vechi al fişierului se pierde şi se va crea unul nou cu acelaşi nume.
Menţionăm că *stdin*, *stderr*, *stdaux* şi *stdprn* sunt pointeri spre tipul FILE şi permit ca funcţiile de nivel superior de prelucrare a fişierelor să poată trata intrarea standard şi ieşirile standard pe terminal şi imprimantă la fel ca şi fişierele pe celelelate suporturi. Singura deosebire constă că aceste fişiere nu se deschid şi nici se închid de către programator. Ele sunt deschise automat la lansarea în execuţie a programului şi se închid la apelul funcţiei *exit*. Variabilele stdin, stderr, stdaux și stderr există în program ca și variabile globale, iar declarațiile lor sunt introduse prin stdio.h

Citire / scriere de caractere

Fişierele pot fi scrise şi citite caracter cu caracter, folosind două funcţii simple şi anume *putc* pentru scriere şi *getc* pentru citire. Funcţia *putc* are prototipul:

```
int putc(int c, FILE *pf);
```

unde:

- c este codul ASCII al caracterului care se scrie in fişier;
- pf este pointerul spre tipul FILE a cărui valoare a fost returnată de funcţia *fopen* la deschiderea fişierului în care se scrie. In particular, *pf* poate fi unul din pointerii:
  `stdout` (ieşire standard);
  `stderr` (ieşire pe terminal în caz de eroare);
  `stdaux` (comunicaţie serială);
  `stdprn` (ieşire paralelă la imprimantă).

Funcţia *putc* returnează valoarea lui c respectiv -1 în caz de eroare.
Funcţia getc are prototipul

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

unde pf este pointerul spre tipul FILE a cărui valoare a fost returnată de funcţia fopen la deschiderea fişierului. Ea returnează codul ASCII al caracterului citit sau EOF la sfârşit de fişier sau eroare. In particular, *pf* poate fi pointerul *stdin* (intrare de la tastatură).

Similar cu acestea, sunt funcțiile fputc și fgetc.

Pentru lucrul cu intrarea / ieșirea standard, se pot folosi funcțiile getchar sau putchar, cu sintaxă similară cu getc si putc, fără să fie necesară transmiterea pointerului stdin sau stdout.

Inchiderea unui fişier

După terminarea prelucrării unui fişier, acesta urmează a fi închis. In acest caz se utilizează funcţia *fclose*. Ea are prototipul

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

unde pf este pointerul spre tipul FILE a cărui valoare a fost definită la deschiderea fişierului prin intermediul funcţiei *fopen*. Funcţia fclose returnează:

0 la închiderea normală a fişierului
EOF: în caz de eroare.

Pentru toate fișierele deschise prin fopen trebuie să existe un apel fclose. fclose are 2 roluri:

- eliberează resursele alocate la nivelul sistemului de operare
- eliberează memoria alocată pentru pointerul de tip FILE* prin intermediul funcției fopn.

Exemplu. Programul copiază intrarea standard la ieşirea standard stdout, folosind funcţiile *getc* şi *putc*.

```
#include <stdio.h>
int main() /* copiaza intrarea stdin la iesirea stdout */
{   int  c;
    while((c = getc(stdin))  != EOF) putc(c,stdout);
}
```

Exemplu. Programul scrie la ieşirea stdout caracterele unui fişier a cărui cale este argumentul din linia de comandă (pentru detalii în legătură cu parametrii transmişi prin linia de comandă). Dacă nu există un argument în linia de comandă, atunci se citeşte de la intrarea standard.

```
#include <stdio.h>

int main(int argc,char *argv[]) /* listeaza la ieºirea stdout un fisier a carui cale este argumentul din linia de comanda  */
{
  FILE  *pf;
  int c;
  if(argc == 1)  /* nu exista argument in linia de comanda */
            pf = stdin;
  else   // se deschide fisierul a carui cale se afla în argv[ 1] //
                  if((pf = fopen( argv[1], "r"))==NULL) {
                              printf("nu se poate deschide fisierul  %s/n",*argv);
                              return -1;
                  }
          while((c = getc(pf)) != EOF )
                  putchar(c);
          fclose(pf);
          return 0;
}
```

Operaţiile de intrare/ieşire cu format

Biblioteca standard a limbajului C conţine funcţii care permit realizarea operaţiilor de intrare/ieşire cu format. Se pot utiliza funcţiile *fscanf* şi fprintf, prima pentru citire cu format dintr-un fişier, iar a doua pentru scriere cu format într-un fişier.

Funcţia *fscanf* este asemănătoare cu funcţia *scanf*. Ea are un parametru în plus faţă de scanf, un pointer spre tipul FILE care defineşte fişierul din care se face citirea. Acest pointer este primul parametru al funcţiei *fscanf*. Ceilalţi parametri au aceeaşi semnificaţie ca şi în cazul funcţiei *scanf.*
Rezultă că funcţia fscanf poate fi apelată printr-o expresie de atribuire de forma
`nr=fscanf(pf,control,par1, par2,...parn);`
unde:

- pf este un pointer spre tipul FILE şi valoarea lui a fost definită prin apelul funcţiei fopen (aceasta defineşte fişierul din care se face citirea) iar ceilalţi parametri sunt identici cu cei utilizaţi la apelul funcţiei scanf .

Funcţia *fscanf*, ca şi *scanf,* returnează numărul câmpurilor citite din fişier.

La întâlnirea sfârşitului de fişier se returnează valoarea EOF definită în fişierul stdio.h. Pentru ps=stdin, functia *fscanf* este identică cu *scanf.*
Funcţia *fprintf* este analoagă cu funcţia *printf.* Primul ei parametru este un pointer spre tipul FILE şi el defineşte fişierul în care se scriu datele. Ceilalţi parametri sunt identici cu cei ai funcţiei *printf.*

Funcţia fprintf, ca şi funcţia printf, returnează numărul caracterelor scrise în fişier sau -1 în caz de eroare. Pentru sf=stdout, funcţia *fprintf* este identică cu *printf*. De asemenea, se pot utiliza pointerii standard obişnuiţi:

`stderr`	- afişarea mesajelor de eroare pe terminal
`stdaux`	- comunicaţie serială
`stdprn`	- ieşirea paralelă la imprimantă.

Menţionăm că la comunicaţia serială se poate conecta o imprimantă serială.

Intrări/ieşiri de şiruri de caractere

Biblioteca standard a limbajului C conţine funcţiile *fgests* şi *fputs* care permit citirea respectiv scrierea într-un fişier ale cărui înregistrări sunt şiruri de caractere.
Funcţia *fgets* are prototipul
```
char *fgets(char *s, int n, FILE *pf);
```
unde:

- s este pointerul spre zona în care se face citirea caracterelor
- n-1 este numărul maxim de caractere care se citesc iar pf este pointerul spre tipul FILE care defineşte fişierul din care se face citirea.

Citirea se incheie fie dacă se citesc n-1 caractere, fie dacă se întâlneşte delimitatorul de sfârşit de linie.

De obicei s este numele unui tablou de tip *char* de dimensiune cel puţin n. s trebuie să fie alocat înainte de inceperea execuţie funcţiei fgets şi trebuie să aibă capacitatea de minim n octeţi.
Sirul se termină cu `\0` (caracterul NULL) care este inserat de funcţia fgets la capătul şirului de caractere citit. La întâlnirea caracterului `\n`, citirea se opreşte. In acest caz, în zona receptoare se transferă caracterul `\n` şi apoi caracterul NULL. In mod normal, funcţia returnează valoarea pointerului s. La întâlnirea sfârşitului de fişier se returnează valoarea NULL.

Funcţia *fputs* scrie într-un şir fişier un şir de caractere care se termină prin `\n`. Ea are prototipul
`int fputs(const char *s,FILE *pf);` unde:

- s este pointerul spre zona care conţine şirul de caractere care se scrie;
- pf este pointerul spre tipul FILE care defineşte fişierul în care se scrie.
  Funcţia *fput* returnează codul ASCII al ultimului caracter scris sau -1 în caz de eroare.

Aceste funcţii sunt realizate folosind funcţia getc pentru fgets şi putc pentru fputs.

10.2 Prelucrarea fişierelor binare

Fişierele organizate ca date binare (octeţii nu sunt consideraţi ca fiind coduri de caractere) pot fi prelucrate la acest nivel folosind funcţiile *fread* şi *fwrite*.

In acest caz, se consideră că înregistrarea este o colecţie de date structurate numite articole. La o citire, se transferă într-o zonă specială, numită zonă tampon, un număr de articole care se presupune că au o lungime fixă. In mod analog, la scriere se transferă din zona tampon un număr de articole de lungime fixă. Cele două funcţii au prototipurile de mai jos:

```
size_t fread(void *ptr, size_t dim, size_t nrart, FILE *pf);
size_t fwrite(const void *ptr, size_t dim, size_t nrart, FILE *pf);
```

unde:

- ptr este pointerul spre zona tampon ce conţine articolele citite / scrise (înregistrarea citită / scrisă);
- dim este un întreg ce reprezintă lungimea unui articol;
- *nrart* este un întreg ce reprezintă numărul articolelor care se transferă;
- pf este un pointer spre tipul FILE care defineşte fişierul din care se face citirea.

De obicei, dim este dimensiunea (sizeof) a tipului către care pointează pointerul ptr.

Ambele funcţii returnează numărul articolelor transferate sau -1 în caz de eroare.

Exemplu. Programul citeşte dde la intrarea *stdin* datele ale căror formate sunt definite mai jos şi le scrie în fişierul *misc.dat* din directorul curent. Formatul datelor de intrare este următorul:

```
Tip     denumire            val     um          cod         pret    cantitate
I       REZISTENTA          010     KG          12345678    1.5     10000.0
#include <stdio.h>
#define MAX 50

typedef struct {
        char tip[2];
        char den[MAX + 1];
        int  val;
        char unit[3];
   long cod;
        float pret;
        float cant;
}  TIP_ARTICOL;


int cit(TIP_ARTICOL  *str) //citeste datele de la intrarea standard si le scrie in structura spre care pointeaza str
{
        int  c,nr;
        float x,y;

        while((nr = scanf("%1s  %50s %3d %2s %1d", str->tip, str->den, &str->val, str->unit, &str->cod))!= 5 ||
                scanf("%f %f", &x, &y) != 2)
        {
                if(nr == EOF ) return EOF;
                printf("rand eronat; se reia \n");
```

```
                // avans pana la newline sau EOF
                while((c = getchar()) != '\n' &&  c != EOF);
                if(c == EOF)  return EOF;
        }   //sfarsit while
        str->pret = x; str->cant = y;
        return nr;
}  // sfarsit cit

int main() // creeaza fisierul misc.dat cu datele citite de la stdin
{
        FILE  *pf;
        TIP_ARTICOL  a[6];
        int  i, n;

        if((pf = fopen("misc.dat","w+")) == NULL)
        {
                printf(" nu se poate deschide in creare fisierul misc.dat\n");
                return -1;
        }
        for(i  =  0; i  <  6;   i++) {
                n = cit(&a[i]);
        }
        if( 6 != fwrite( &a, sizeof(TIP_ARTICOL), 6, pf)) {
                        // eroare la scrierea in fisier
                        printf("eroare la scrierea in fisier\n");
                        return -1;
        }
        fclose(pf);
  // sfarsit main
        return 0;
}
```

Exemplu. Programul listează articolele fişierului misc.dat, creat prin programul anterior, pentru care tip=I.

```
#include <stdio.h>
#define MAX 50

typedef struct {
       char tip[2];
       char den[MAX + 1];
       int  val;
       char unit[3];
    long cod;
       float pret;
       float cant;
}  TIP_ARTICOL;


main()   /* citeste fisierul misc.dat  */
{
       FILE   *pf;
    int  i,n;
       TIP_ARTICOL  a[6];

       if((pf  = fopen("misc.dat","r")) == NULL) {
              printf("nu se poate deschide fisierul misc.dat in citire\n");
              return -1;
```

```
        }
        printf("%60s\n\n\n","INTRARI\n\n");
    //  se citeste o inregistrare
        while((n = fread(a, sizeof(TIP_ARTICOL), 6, pf ))  > 0)
               // se listeaza articolele de tip I dintr-o inregistrare
               for (i = 0; i < n;  i++) {
                   if ( a[i].tip[0]   ==  'I')
                              printf("%s    %3d  %s  %d  %.2f    %.2f\n",  a[i].den,  a[i].val,  a[i].unit,
a[i].cod, a[i].pret, a[i].cant);
               }
        fclose(pf);
        return 0;
}
```

Poziţionarea într-un fişier

Biblioteca standard a limbajului C conţine funcţia fseek, cu ajutorul căreia se poate deplasa capul de citire/scriere al discului în vederea prelucrării înregistrărilor fişierului într-o ordine oarecare, diferită de cea secvenţială (acees aleator). Ea are prototipul

*int fseek(FILE *pf,long deplasament, int origine);*

unde pf este pointerul spre tipul care defineşte fişierul în care se face poziţionarea capului de citire/scriere iar deplasament şi origine au aceeaşi semnificaţie ca şi în cazul funcţiei lseek.

Funcţia *fseek* returnează valoarea zero la pozţionare corectă şi o valoare diferită de zero în caz de eroare.

O altă funcţie utilă în cazul accesului aleator este funcţia *ftell*, care indică poziţia capului de citire în fişier. Ea are prototipul

`long ftell(FILE *pf);` unde:

- pf este pointerul spre tipul FILE care defineşte fişierul în cauză.

Funcţia returnează o valoare de tip *long* care defineşte poziţia curentă a capului de citire/scriere, şi anume reprezintă deplasamentul în octeţi a poziţiei capului faţă de începutul fişierului.

Ştergerea unui fişier

Un fişier poate fi şters apelând funcţia *unlink*. Aceasta are prototipul

`int unlink(const  char  *cale);`

unde cale este un pointer spre un şir de caractere identic cu cel utilizat la crearea fişierului în funcţia de *creat* sau *fopen*.

Funcţia *unlink* returnează valoarea zero la o ştergere reuşită, respectiv -1 în caz de eroare.