Programarea calculatoarelor
Limbajul C

CURS 10

Fişiere

Fişier

- Colecție de date memorate pe un suport extern (floppy, harddisk, etc) identificată printr-un nume.
- Entități ale sistemului de operare: numele lor respectă convențiile sistemului, fără legătură cu un limbaj de programare anume
- Conținutul:
  - texte (ex. programe sursă)
  - numere
  - alte informații binare: programe executabile, numere în format binar, imagini sau sunete codificate numeric s.a.
- Numărul de elemente ale unui fişier este variabil (poate fi nul).
- Se folosesc pentru
  - date inițiale sau rezultate mai numeroase
  - păstrarea permanentă a unor date de interes pentru anumite aplicații

Operarea cu fişiere

- De obicei "fişier" = fişier disc (pe suport magnetic sau optic)
- Noțiunea de fişier este mai generală şi include orice flux de date (stream) din exterior spre memorie sau dinspre memoria internă spre exterior.
- “Stream” (flux de date, canal) sinonim cu “file” (fişier): pune accent pe aspectul dinamic al transferului de date
- Programatorul se referă la un fişier printr-o variabilă; tipul acestei variabile depinde de limbajul folosit şi chiar de funcțiile utilizate (în C).
- Asocierea dintre numele extern (un şir de caractere) şi variabila din program se face la deschiderea unui fişier, printr-o funcție standard.

Tipuri de fişiere în C

- Fişiere text
  - conțin o succesiune de linii, separate prin NewLine
  - fiecare linie are 0 sau mai multe caractere tipăribile şi/sau tab
- Fişiere binare
  - conțin o succesiune de octeți

Fişiere text

- Caracter terminator de linie:
  - fişierele Unix/Linux: un singur caracter terminator de linie ‘\n’
  - fişierele Windows şi MS-DOS: caracterele ‘\r’ şi ’\n’ (CR,LF) ca terminator de linie
- Un fişier text poate fi terminat printr-un caracter terminator de fişier (Ctrl-Z = EOF = -1)
  - nu este obligatoriu acest terminator
- Sfârşitul unui fişier disc poate fi detectat şi pe baza lungimii fişierului (număr de octeți), memorată pe disc.
- Se realizează conversia automată din/în format extern (şir de caractere) în/din format intern (binar virgulă fixă sau virgulă mobilă)

Fişiere binare

- Pot conține
  - numere în reprezentare internă (binară)
  - articole (structuri de date)
  - fişiere cu imagini grafice, în diverse formate, etc
- Citirea şi scrierea se fac fără conversie de format.
- Pentru fiecare tip de fişier binar este necesar un program care să cunoască şi să interpreteze corect datele din fişier (structura articolelor).
- Este posibil ca un fişier binar să conțină numai caractere, dar funcțiile de citire şi de scriere pentru aceste fişiere nu cunosc noțiunea de linie; ele specifică un număr de octeți care se citesc sau se scriu la un apel al funcției “fread” sau “fwrite”.

Operarea cu fişiere

1. *se defineşte o variabilă* de tip FILE * pentru accesarea fişierului;
   - FILE * un tip structură definită în stdio.h
   - conține informații referitoare la fişier şi la tamponul de transfer de date între memoria centrală şi fişier (adresa, lungimea tamponului, modul de utilizare a fişierului, indicator de sfârşit, de poziție în fişier)
2. *se deschide fişierul* pentru un anumit mod de acces, folosind funcția de biblioteca fopen, care realizează şi asocierea între variabila fişier şi numele extern al fişierului
3. se prelucrează fişierul - citire/scriere cu funcțiile specifice
4. se închide fişierul folosind funcția de biblioteca fclose.

Deschiderea unui fişier

FILE *fopen(const char *numefişier, const char *mod);

- Deschide fişierul cu numele dat pentru acces de tip mod
- Returnează pointer la fişier sau NULL dacă fişierul nu poate fi deschis
- Valoarea returnată este memorată în variabila fişier, care a fost declarată (FILE *) pentru accesarea lui.

- numefis: numele fişierului
- mod: şir de caractere (între 1 şi 3 caractere):
  - "r" - readonly , este permisă doar citirea dintr-un fişier existent
  - "w" - write, crează un nou fişier, sau dacă există deja, distruge vechiul conținut
  - "a" - append, deschide pentru scriere un fişier existent (scrierea se va face în continuarea informației deja existente în fişier, deci pointerul de acces se plasează la sfârşitul fişierului)
  - “+” = permite scrierea şi citirea din acelasi fişier - actualizare (ex: "r+", "w+", "a+").
  - "t" sau "b" = tip fişier ("text", "binary"), implicit este "t“

Nume extern fişier

- Poate include următoarele:
  - Numele unității de disc sau partiției disc (ex: A:, C:, D:, E:)
  - "Calea" spre fişier: succesiune de nume de fişiere catalog (director), separate printr-un caracter ('\' în MS-DOS şi MS-Windows, sau '/' în Unix şi Linux)
  - Numele propriu-zis al fişierului
  - Extensia, care indică tipul fişierului şi care poate avea între 0 şi 3 caractere în MS-DOS.
- Sistemele MS-DOS şi MS-Windows nu fac deosebire între litere mari şi litere mici, în cadrul numelor de fişiere
- Atenție! pentru separarea numelor de cataloage dintr-o cale se vor folosi:
  - "\\", pentru a nu se considera o secvență de caractere "Escape"
  - sau caracterul ‘/’.
  - char *numef = "C:\\WORK\\T.TXT";
  - char *numef = “c:/work/t.txt”;

Închiderea unui fişier

- **int fclose(FILE *fp);**
  - închide fişierul şi eliberează zona tampon
  - în caz de succes întoarce 0.
  - altfel, întoarce **EOF**.

- Atenție!
  - Închiderea unui fişier disc este absolut necesară pentru fişierele în care s-a scris ceva!
  - Poate lipsi dacă s-au făcut doar citiri din fişier!

Exemplu

```
#include <stdio.h>
int main ( ) {
  FILE * f;       // pentru referire la fişier
     // deschide un fişier text ptr citire
  f = fopen ( “c:\\t.txt", "rt“ );
  printf ( f == NULL ? "Fişier negasit" : " Fişier gasit");
  if (f)              // dacă fişier existent
    fclose(f);    // închide fişier
  return 0;
}
```

Prelucrarea fişierelor text

- se poate face fie la nivel de linie, fie la nivel de caracter:
  - **int fgetc (FILE *fp)**

    întoarce următorul caracter din fp ca un unsigned char convertit la int, sau EOF dacă s-a întâlnit sfârşitul de fişier sau în caz de eroare.
  - **char *fgets (char *s, int n, FILE *fp)**

    citeşte maxim n-1 caractere sau până la '\n' inclusiv, şi le depune în s, adaugă la sfârşit '\0' şi returnează adresa şirului. La eroare întoarce valoarea NULL.
  - **int fputc(int c,FILE *fp)**

    scrie caracterul cu codul ascii c în fişier
  - **char *fputs(char *s,FILE *fp)**

    scrie şirul în fişier, fara caracterul '\0'. La eroare întoarce EOF.

Exemplu: citire şi afişare linii dintr-un fişier

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  FILE *fp;
  char s[80];
  if ( (fp=fopen("c:\\test.c","r")) == NULL ) {
     printf ( "Nu se poate deschide la citire fişierul!\n“ );
      exit (1);
  }
  while ( fgets(s,80,fp) != NULL )
     printf ( "%s", s);
   fclose (fp);
   return 0;
}
```

Exemplu: scriere sub formă de litere mici caracterele dintr-un fişier în alt fişier

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
int main () {
   FILE * f1, * f2;  int ch;
   f1= fopen ("c:\\1cc\\in.txt", "r");
   f2= fopen ("c:\\1cc\\out.txt", "w");
   if ( f1==0 || f2==0) {
      puts (" Eroare la deschidere fişiere \n");
      system("pause");
      return 1;
   }
   while ( (ch=fgetc(f1)) != EOF)       // citeste  din f1
      fputc ( tolower(ch),f2);          // scrie în f2
   fclose(f1);
   fclose(f2);
   system("pause");
   return 0;
}
```

Intrări/ieşiri cu conversie de format

- Datele numerice pot fi scrise în fişiere disc
  - în format intern, binar (mai compact)
  - transformate în şiruri de caractere (cifre zecimale, semn ş.a): fişier text ocupă mai mult spațiu
- Formatul şir de caractere necesită şi caractere separator între numere, dar poate fi citit cu programe scrise în orice limbaj sau cu orice editor de texte sau cu alt program utilitar de vizualizare fişiere!

- int fprintf (FILE *fp, const char *format,...)

identică cu printf cu deosebirea că scrie într-un fişier.

- int fscanf (FILE *fp, const char *format,...)

realizează citirea cu format dintr-un fişier; analog scanf

- Exemplu:
  - fscanf (fp, ”%d%f ”,&a, &b);
  - fprintf (fp, ”a= %d \t b=%f \n”, a, b);

Testare sfârşit de fişier

- int feof(FILE *fp)
  - testează dacă s-a ajuns la *end-of-file* al fişierului referit de fp
  - returnează 0 dacă nu s-a detectat sfârşit de fişier la ultima operație de citire, respectiv o valoare nenulă (adevarată) pentru sfârşit de fişier.
- Atenție! Se va scrie în fişierul de ieşire şi –1 - rezultatul ultimului apel al funcției “fgetc”:

```
while ( ! feof(f1))
                fputc(fgetc(f1),f2);
```

- Soluția preferabilă pentru ciclul de citire-scriere caractere este testarea rezultatului funcției de citire:

```
while ( (ch=fgetc(f1)) != EOF)
    fputc ( ch, f2);
```

Exemplu

- Într-un fişier de tip text sunt păstrate valorile reale ale unei măsuratori sub forma: nr_măsuratori '\n' val1 '\n' val2 '\n' val3 ... Să se scrie programul care afişează numărul de măsurători şi valorile respective, după care adaugă la fişier noi măsuratori până la introducerea valorii 0. Valorile citite se afişeaza în format ştiințific.

Rezolvare

```
# include <math.h>
# include <stdio.h>
# include <stdlib.h>
# include <ctype.h>
# define MAX  100

double convf (char *s) {
  double val=0.0, putere;
  int i=0,semn;
  while ( isspace(s[i]) )
              i++;
  semn = (s[i]=='-')?-1:1;
  if (s[i]=='+‘ || s[i]=='-‘ )
              i++;
  for ( val=0.0; isdigit(s[i]); i++)
            val = 10*val + s[i]-'0‘;
```

Rezolvare

```
if (s[i]=='.') {
        i++;
        for (putere=1.0; isdigit(s[i]); i++) {
             val = 10*val + s[i]-'0';
             putere *= 10;
        }
        val /= putere;
  } /*sfirsit parte zecimala*/
  val *= semn;
  return val;
}
void loadmat (FILE *fp, int *nrm, double *mas){
  int i=0;
  fscanf (fp,"%d", nrm);
  if (*nrm>MAX) *nrm = MAX;
  for ( ; i<*nrm; i++)
    fscanf (fp, "%lf", &mas[i]);
}
```

Rezolvare

```
void afiseaza(double *mas,int nrm){
  int i;
  for (i=0; i<nrm; i++)
    printf ("Masuratoarea %d = %6.2e\n", i+1, mas[i]);
}

int main(){
  FILE *fp;
  double masur[MAX], mas_noua;
  char nume_fis[12], s[20];
  int nr_mas;
  printf ("nume fisier:");
  gets (nume_fis);
  if ( (fp = fopen(nume_fis, "r+") ) == 0 ){
     printf ("nu exista fisierul\n");
     exit (1);
  }
```

Rezolvare

```
  loadmat (fp,&nr_mas,masur);
  afiseaza (masur,nr_mas);

  fseek (fp, 0L, SEEK_END);           // pozitionare la sfarsit
  printf ("\nmasuratoarea %d =",++nr_mas);
  while ( (mas_noua=convf(gets(s))) && nr_mas<MAX-1){
    printf ("\nmasuratoarea %d =", ++nr_mas);
    fprintf (fp, "%lf\n", mas_noua);
  }
  fseek (fp, 0L, SEEK_SET);  // pozitionare la inceput
  fprintf (fp, "%d", --nr_mas);

  fclose (fp);
  system ("pause");
  return 0;
}
```

Exerciții

- Program pentru numărarea liniilor şi cuvintelor dintr-un fişier text. Cuvintele sunt şiruri de orice caractere separate între ele prin (oricâte) spații albe. Se va folosi funcția de biblioteca "strtok".

Exemplu strtok:

```
char *p, *sep= ="\t \r\n" ;
while ( (p= strtok (p,sep)) != NULL) {
        …
        p=p+strlen(p)+1;
    }
```

Funcții de acces la fişiere binare

- Un fişier binar este format în general din articole de lungime fixă, fără separatori între articole.
- Un articol poate conține
  - un singur octet
  - un număr binar (pe 2,4 sau 8 octeți)
  - o structură cu date de diferite tipuri.
- Funcțiile de acces pentru fişiere binare "fread" şi "fwrite" pot citi sau scrie unul sau mai multe articole, la fiecare apelare.
- Transferul între memorie şi suportul extern se face fără conversie sau editare (adăugare de caractere la scriere sau eliminare de caractere la citire).

Funcții intrare/iesire (fişiere binare – b)

- **size_t fread (void *ptr, size_t size, size_t nmemb, FILE *fp)**
- citeşte la adresa **ptr** cel mult **nmemb** elemente de dimensiune **size** din fişierul referit de **fp**:

  int a[10];

  fread (a, sizeof(int), 10, fp);

- **size_t fwrite (void *ptr, size_t size, size_t nmemb, FILE *fp)**
- scrie în fişierul referit de **fp** cel mult **nmemb** elemente de dimensiune **size** de la adresa **ptr**:

  fwrite(a, sizeof(int),10,fp);

- Rezultatul funcțiilor "fread" şi "fwrite" este numărul de articole efectiv citite sau scrise
- Este diferit de argumentul 3 numai la sfârsit de fişier (la citire) sau în caz de eroare de citire/scriere.

assert

- **void assert (int *expr*);**
- Permite ca informații de diagnostic să fie scrise la fişierul standard de eroare.
- Dacă expr este 0 (fals), atunci expresia expr, numele fişierului şi linia în care a apărut sunt trimise la fişierul standard de eroare după care execuția programului este oprită:

  Assertion failed: *expr*, **file** *filename*, **line** *line-number*
- Exemplu:

```
#include<assert.h>
void open_record(char *record_name) {
    assert ( record_name != NULL );
    /* Rest of code */
}
int main(void) {
    open_record(NULL);
}
```

Exemplu: operații cu un fişier de elevi (nume şi medie)

```
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>
#include<string.h>
typedef struct {
  char nume[25];
  float medie;
} Elev;
     // creare fişier cu nume dat
void creare(char * numef) {
  FILE * f;  Elev s;
  f=fopen(numef,"wb");
  assert (f != NULL);
  printf ("Nume şi medie ptr. fiecare student :\n");
  while ( scanf ("%s%f", s.nume, &s.medie) !=  EOF)
     fwrite (&s, sizeof(s), 1, f);
  fclose (f);
}
```

Exemplu

```
     // afisare conținut fişier pe ecran
void listare (char* numef) {
 FILE * f;  Elev e;
 f = fopen (numef, "rb");  assert (f != NULL);
 printf ("Nume şi medie: \n");
 while ( fread (&e, sizeof(e), 1, f ) ==1 )
   printf ("%-25s %6.2f \n", e.nume, e.medie);
 fclose (f);
}
       // adaugare articole la sfârşitul unui fişier existent
void adaugare (char * numef) {
  FILE * f;  Elev e;
  f = fopen (numef, "ab");  assert (f != NULL);
  printf ("Adaugare nume şi medie:\n");
  while (scanf ("%s%f", e.nume, &e.medie) != EOF)
     fwrite (&e, sizeof(e), 1, f);
  fclose (f);
}
```

Acces direct la datele dintr-un fişier

- posibilitatea de a citi sau scrie oriunde într-un fişier, printr-o poziționare prealabilă înainte de citire sau scriere
- În C poziționarea se face pe un anumit octet din fişier, iar funcțiile standard permit accesul direct la o anumită adresă de octet din fişier.
- Funcțiile pentru acces direct din <stdio.h> permit operațiile următoare:
  - Pozitionarea pe un anumit octet din fişier ("fseek").
  - Citirea poziției curente din fişier ("ftell").
  - Memorarea poziției curente şi poziționare ("fgetpos", "fsetpos").
- Poziția curentă în fişier este un număr de tip *long*, pentru a permite operații cu fişiere foarte lungi!

Funcții pentru acces direct

- **long int ftell (FILE *fp)**
  - Întoarce valoarea indicatorului de poziție
  - Pentru fişier binar: numărul de octeți de la începutul fişierului
  - Fişier text: o valoare ce poate fi utilizată de fseek pentru a seta indicatorul de poziție în fişier la această poziție.
- **int fseek (FILE *fp, long int offset, int poziție)**

  poziționează indicatorul de poziție la valoarea dată de offset față de poziție:
  - SEEK_SET = 0 - Căutarea se face de la începutul fişierului
  - SEEK_CUR = 1 - Căutare din poziția curentă
  - SEEK_END = 2 - Căutare de la sfârşitul fişierului
- Intr-un fişier text poziționarea este posibilă numai fată de începutul fişierului, iar poziția se obține printr-un apel al funcției “ftell”!

Exemple

- poziționarea la sfârşitul fişierului: **fseek (fp, 0, SEEK_END)**
- poziționarea la caracterul precedent: **fseek (fp, -1, SEEK_CUR)**
- poziționarea la inceputul fişierului: **fseek (fp, 0, SEEK_SET)**

- Atenție! Poziționarea relativă la sfârşitul unui fişier nu este garantată nici chiar pentru fişiere binare, astfel că ar trebui evitată!
- Ar trebui evitată şi poziționarea față de poziția curentă cu o valoare negativă, care nu funcționează în toate implementările!

Funcții pentru acces direct

- **int fgetpos (FILE *fp, fpos_t *poziție)**

  memorează starea curentă a indicatorului de poziție al fluxului referit de **fp** în **poziție;**

  întoarce 0 dacă operația s-a realizat cu succes!
- **int fsetpos (FILE *fp, const fpos_t *poziție)**

  setează indicatorul de poziție al fluxului referit de **fp** la valoarea data de **poziție**
- **void rewind (FILE *fp)**

  setează indicatorul de poziție al fluxului referit de **fp** la începutul fişierului

Funcție care modifică conținutul mai multor articole din fişierul de elevi creat anterior

```
      // modificare conținut articole, dupa cautarea lor
void modificare (char * numef) {
  FILE * f; Elev e;  char nume[25];
  long pos;  int  ef;
  f = fopen(numef,"rb+");  assert (f != NULL);
  do {
     printf ("Nume cautat: "); scanf ("%s",nume);
        if (strcmp(nume, ".") == 0)  break;     // datele se termină cu un punct
              // cauta "nume" în fişier
        fseek (f, 0, 0);                              // readucere pe inceput de fişier
        while ( (ef=fread (&e, sizeof(e), 1, f)) ==1 )
                  if (strcmp (e.nume, nume)==0) {
                            pos= ftell(f) - sizeof(e);
                            break;
                  }
```

Exemplu

```
      if ( ef < 1) break;
      printf ("noua medie: "); scanf ("%f", &e.medie);
      fseek (f, pos, 0);                  // pe inceput de articol gasit
       fwrite (&e, sizeof(e), 1, f); // rescrie articol modificat
  } while (1);
  fclose (f);
}

int main(){
    char name[]="c:elev.txt“;
    creare (name);
    listare (name);
    adaugare (name);
    modificare (name);
    listare (name);
    system("pause");
    return 0;
}
```

Fişiere predefinite

- Există trei *fluxuri predefinite*, care se deschid automat la lansarea unui program:
  - stdin - fişier de intrare, text, este intrarea standard - tastatura
  - stdout - fişier de ieşire, text, este ieşirea standard - ecranul monitorului.
  - stderr - fişier de iesire, text, este ieşirea standard de erori - ecranul monitorului.
- pot fi folosite în diferite funcții
- practic se folosesc în funcția "fflush" care goleşte zona tampon ("buffer") asociată unui fişier.

Redirectarea fişierelor standard

- Prin redirectare, fişierele standard se pot asocia cu alte fişiere.
- *Exemplu:*

  fişier_exe <fişier_1 >fişier_2
- În acest caz, preluarea informațiilor se face din fişier_1, iar afişarea informațiilor de ieşire se face în fişier_2.

Observații

- Nu orice apel al unei funcții de citire sau de scriere are ca efect imediat un transfer de date între exterior şi variabilele din program!
- Citirea efectivă de pe suportul extern se face într-o zonă tampon asociată fişierului, iar numărul de octeți care se citesc depind de suport: o linie de la tastatură, unul sau câteva sectoare disc dintr-un fişier disc, etc.
- Cele mai multe apeluri de funcții de I/E au ca efect un transfer între zona tampon (anonimă) şi variabilele din program.
- Funcția “fflush” are rolul de a goli zona tampon folosită de funcțiile de I/E, zonă altfel inaccesibilă programatorului C.
- Are ca argument variabila pointer asociată unui fişier la deschidere, sau variabilele predefinite “stdin” şi “stdout”.

Exemple de situații în care este necesară folosirea funcției “fflush”:

- Citirea unui caracter după citirea unui câmp sau unei linii cu “scanf” :

```
int main () {
  int n; char s[30]; char c;
  scanf (“%d”,&n);                    // sau  scanf(“%s”,s);
// fflush(stdin);              // pentru corectare
  c= getchar();               // sau  scanf (“%c”, &c);
  printf (“%d \n”,c);                 // afiseaza codul lui c
  return 0;
}
```

- va afişa 10 care este codul numeric al caracterului terminator de linie ‘\n’, în loc să afişeze codul caracterului “c”!
- după o citire cu “scanf” în zona tampon rămân unul sau câteva caractere separator de câmpuri (‘\n’,‘\t’,’ ‘), care trebuie scoase de acolo prin fflush(stdin) sau prin alte apeluri “scanf”.

Exemple de situații în care este necesară folosirea funcției “fflush”:

- Funcția “scanf” opreşte citirea unei valori din zona tampon ce conține o linie la primul caracter separator de câmpuri sau la un caracter ilegal în câmp (de ex. literă într-un câmp numeric)!
- In cazul repetării unei operatii de citire (cu “scanf”) după o eroare de introducere în linia anterioară (caracter ilegal pentru un anumit format de citire) în zona tampon rămân caracterele din linie care urmau după cel care a produs eroarea!

```
do {
        printf ("x, y= ");
        err = scanf ("%d%d", &x, &y);
        if ( err == 2 ) break;
        fflush (stdin);
  } while (err != 2);
```

- După citirea unei linii cu funcțiile “gets” sau “fgets” nu rămâne nici un caracter în zona tampon şi nu este necesar apelul lui “fflush”!

Exemple de situații în care este necesară folosirea funcției “fflush”:

- Se va folosi periodic “fflush” în cazul actualizării unui fişier mare, pentru a evita pierderi de date la producerea unor incidente (toate datele din zona tampon vor fi scrise efectiv pe disc):

```
int main () {
  FILE * f;   int c ;   char numef[]="TEST.DAT";
  char x[ ] = "0123456789";
  f=fopen (numef,"w");
  for (c=0;c<10;c++)
      fputc (x[c], f);
   fflush (f);                  // sau fclose(f);
   f=fopen (numef,"r");
   while ( (c=fgetc(f)) != EOF)
      printf ("%c", c);
   return 0;
 }
```

- Este posibil ca să existe diferențe în detaliile de lucru ale funcțiilor standard de citire-scriere din diferite implementări (biblioteci), deoarece standardul C nu precizează toate aceste detalii!