Curs 5 - agenda

- I/O cu fişiere
- Argumentele funcţiei main
- Tablouri de pointeri
- Funcţiile ca argumente
- Funcţii cu număr variabil de argumente
- Măsurarea timpului de execuţie

Fişiere

- Un fişier poate fi privit ca un “stream” (flux) de caractere.
- Un fişier are un nume
- Pentru a putea fi accesat un fişier trebuie “deschis”
- Sistemul trebuie sa ştie – programatorul ii spune – ce operaţii pot fi făcute cu un fişier:
  - se deschide pentru citire – fişierul trebuie să existe
  - se deschide pentru scriere – fişierul se crează
  - se deschide pentru adăugare – fişierul există şi se modifică
- După prelucrare fişierul trebuie închis

Fişiere. Structura FILE

- Starea curentă a unui fişier este descrisă într-o structură numită FILE, definită în **stdio.h**
- Programatorul poate folosi fişiere fără să cunoască în detaliu structura FILE

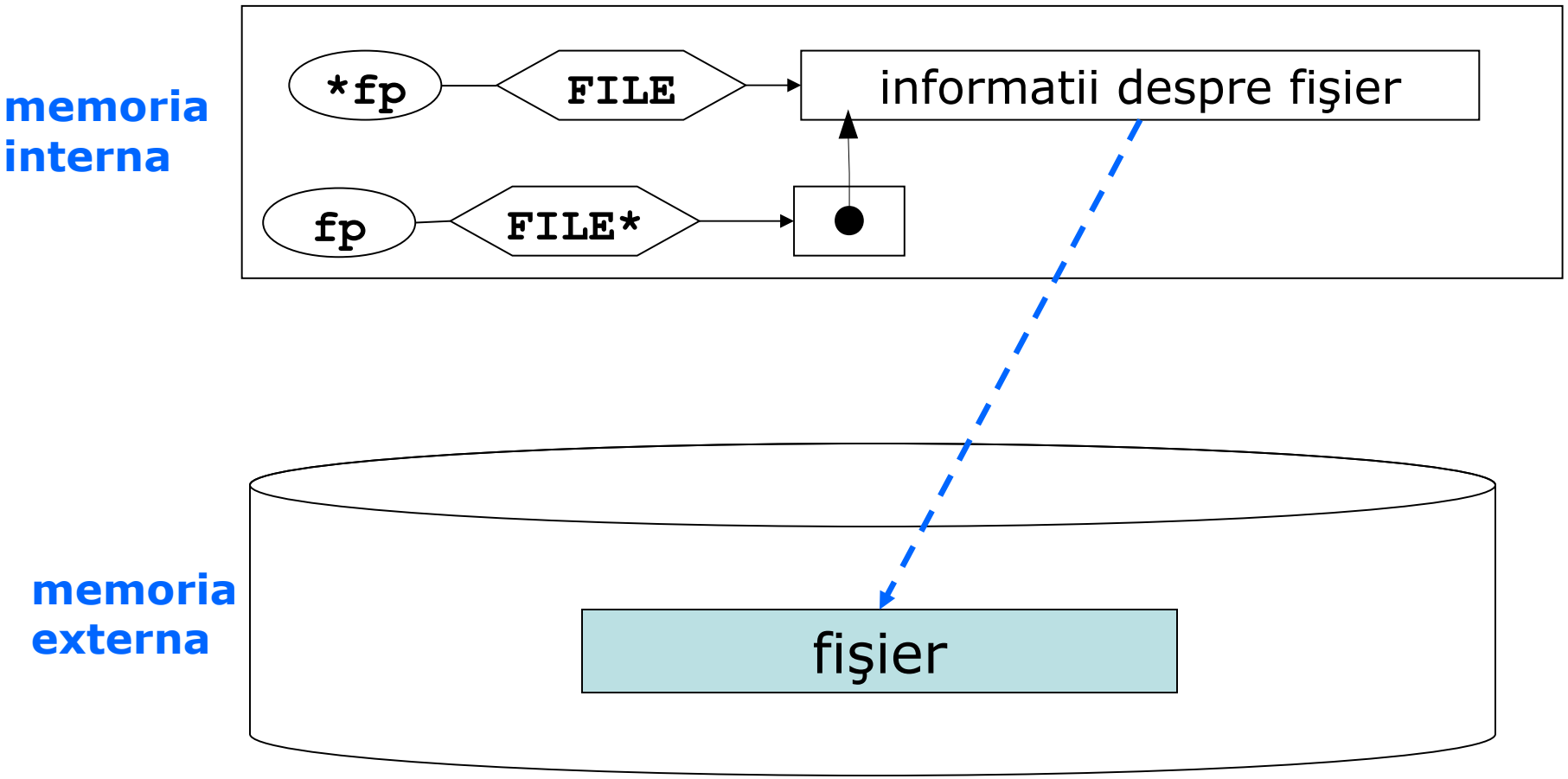
```
struct _iobuf {
        char *_ptr;
        int   _cnt;
        char *_base;
        int   _flag;
        int   _file;
        int   _charbuf;
        int   _bufsiz;
        char *_tmpfname;
        };
typedef struct _iobuf FILE;
```

Fişiere. Structura FILE

- Un obiect de tip FILE înregistrează informaţiile pentru a controla un stream:
  - Indicatorul pentru poziţia in fişier
  - Un pointer la zona buffer asociată
  - Un indicator de eroare care înregistrează dacă se produc erori de citire/scriere (codificat în _flag)
  - Un indicator *end-of-file* ce înregistrează dacă s-a atins sfârşitul de fişier (codificat în _flag)

- Când se deschide un fişier, sistemul de operare îl asociază cu un *stream* şi păstrează informaţiile despre acest *stream* într-un obiect de tip FILE
- Un pointer la FILE "face legătura" cu fişierul sau cu stream-ul asociat fişierului

Fişiere


memoria interna
*fp
FILE
informatii despre fişier
fp
FILE*
memoria externa
fişier

Fişiere. Structura FILE

```
FILE *inf, *outf, *f;
```

- În **stdio.h** sunt definiţi pointerii:
  - **stdin**: fişierul standard de intrare
  - **stdout**: fişierul standard de ieşire
  - **stderr**: fişierul standard pentru erori

- Programatorul nu trebuie să deschidă explicit fişierele standard

Funcţia `fopen()`

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

- Realizează cele necesare gestionării unui fişier:
  - Dacă se execută cu succes(**`filename`** poate fi accesat), crează un stream şi întoarce pointer la FILE asociat acestui stream
  - Dacă **`filename`** nu poate fi accesat, întoarce NULL

```
mode ::= "r"     | "w"  | "a"  |"r+"  |"w+"  |"a+"
        |"rb"    |"wb"  |"ab"  |"r+b" |"w+b" |"a+b"
        |"rb+"   |"wb+" |"ab+"
```

- Indicatorul de poziţie este pus la începutul fişierului (în modul "r" sau "w") sau la sfârşit (în modul "a")
- Modul "a+" este pentru actualizare:
  - Citirea nu poate fi urmată de scriere dacă nu s-a ajuns la EOF sau nu s-a intervenit cu o funcţie de poziţionare
  - Scrierea nu poate fi urmată de citire dacă nu se intervine cu apel la **`flush()`** sau la o funcţie de poziţionare

Funcţiile `close()`, `flush()`, `freopen()`

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

- Realizează cele necesare pentru a închide un fişier: goleşte buffer-ul şi întrerupe orice legătură între fişier şi pointerul fp
  - Daca se execută cu succes, returnează zero
  - Dacă apare o eroare sau fişierul este deja închis, se returneaza EOF

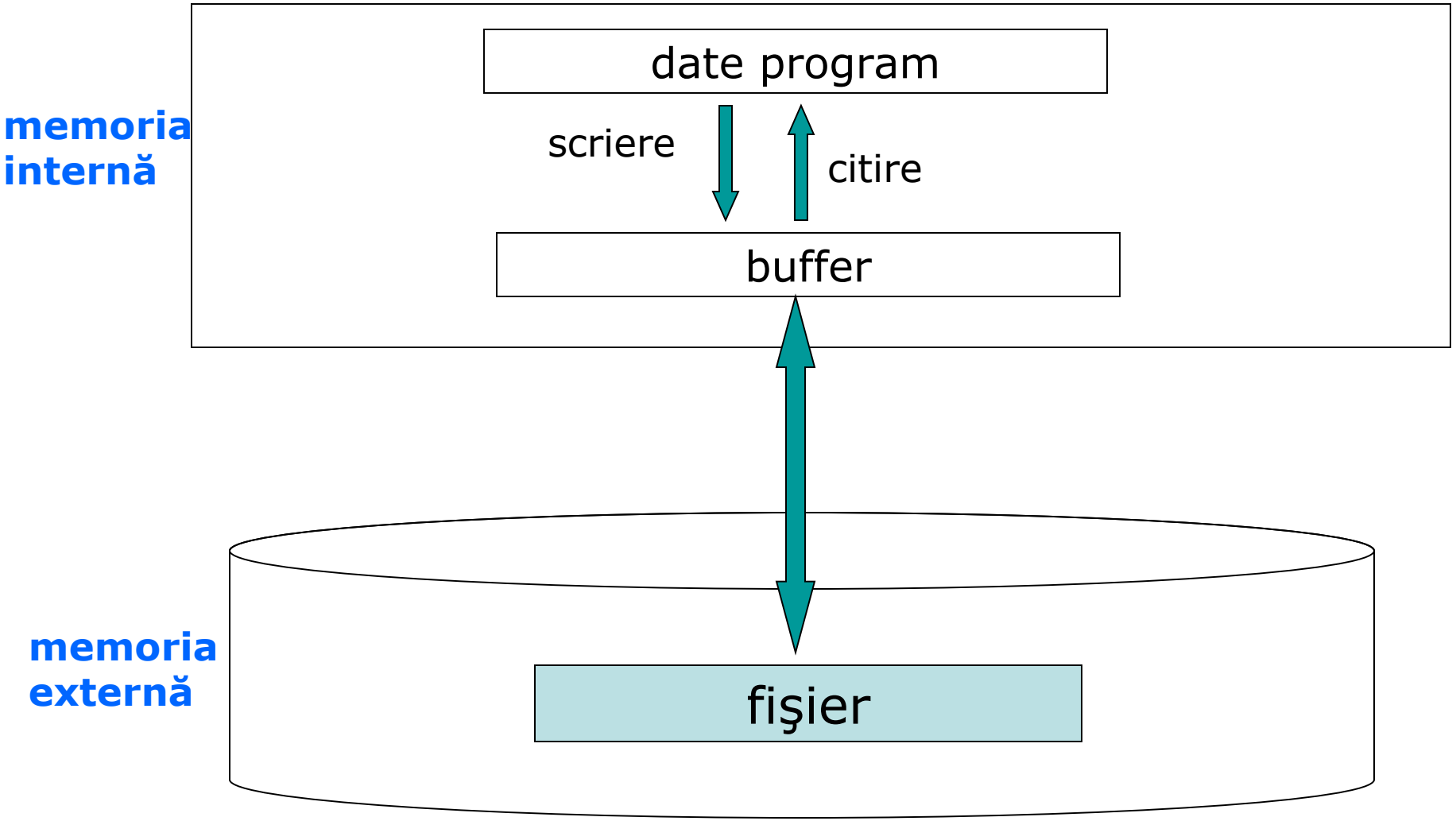
```
int fflush(FILE *fp);
```

- Golirea bufferului: datele din buffer sunt scrise în fişier (daca fp nu este NULL). Se întoarce 0 în caz de succes şi EOF altfel

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

- Este închis fişierul asociat pointerului **`fp`** şi se deschide filename iar fp se asociază acestuia

Fişiere – citire/scriere



Funcţiile fprintf, printf(), sprintf()

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

- Apelul returnează numărul de conversii realizate cu succes
- În şirul **format** apar specificatorii de conversie introduşi prin caracterul %
- La apel, corespondenţa argument --- specificator de conversie
- Caracterele ce nu fac parte din specificatorii de conversie sunt scrise în stream-ul de ieşire

```
fprintf(ofp, "a = %d, b = %f, c = %s.\n", a, b, c);
```

Funcţiile **fprintf, printf(), sprintf()**

specificator_de_conversie ::= **%**{*modificator*}$_{opt}$
{*marime_camp*}$_{opt}$ {**.***precizie*}$_{opt}$ *caracter_de_conversie*

caracter_de_conversie ::=**c**|**d**|**i**|**u**|**o**|**x**|**X**|**e**|**E**|**f**|**g**|**G**|**s**|**p**|**n**|**%**

modificator ::= **h**|**l**|**L**|**-**|**+**|**#**|**0**

marime_camp ::= *numar_intreg_fara_semn*

precizie ::= *numar_intreg_fara_semn*

Funcţiile `fscanf()`, `scanf()`, `sscanf()`

```
int fscanf(FILE *stream, const char *format, ...);
int scanf(const char *format, ...);
int sscanf(const char *str, const char *format, ...);
```

- Apelul returnează numărul de conversii realizate cu succes, respectiv EOF dacă stream-ul de intrare este vid
- În şirul **`format`** apar specificatorii de conversie introduşi prin caracterul `%`
- La apel, corespondenţa argument --- specificator de conversie. Argumentele trebuie sa fie pointeri sau adrese
- Caracterele ce nu fac parte din specificatorii de conversie trebuie să apară în stream-ul de intrare

Exemplu

```
#include <stdio.h>
int main(){
    char *name[3] =  {"Ionescu", "Popescu", "Georgescu"};
    int i;
    FILE *ofp;

    ofp = fopen("rezultate.txt", "w");
    if (!ofp) {
       printf("Eroare fisier!\n");
       return 1;
    }
    for (i = 0; i < 3; ++i)
       fprintf(ofp, "%d %s\n", i, name[i]);
    fclose(ofp);
    return 0;
}
```

Exemplu

```
#include <stdio.h>
int main(){
    FILE *ifp;
    ifp = fopen("rezultate.txt", "r");
    if (!ifp) {
       printf("Eroare fisier!\n");
       return 1;
    }
    char name[20];
    int i, n;
    for (i = 0; i < 3; ++i) {
       fscanf(ifp, "%d %s\n", &i, &name);
       printf(" %d %s\n", n, name);
    }
    fclose(ifp);
    return 0;
}
```

Funcţii de intrare/ieşire caracter

```
int fgetc(FILE *stream);
int getc(FILE *stream);
int getchar(void);
char* fgets(char *s, int n, FILE *stream);
char* gets(char *s);
```

- **`getc()`** este implementată ca macro
- **`getchar()`** este echivalentă cu **`getc(stdin)`**
- **`gets(s)`** pune în **`s`** caracterele citite din **`stdin`** până la **`newline`** sau **`EOF`**. În loc de **`newline`** pune la sfârşit **`'\0'`**; **`fgets()`** păstrează **`newline`**

Funcţii de intrare/ieşire caracter

```
int fputc(int c, FILE *stream);
int putc(int c, FILE *stream);
int putchar(int c);
int fputs(const char *s, FILE *stream);
int puts(const char *s);
int ungetc(int c, FILE *stream);
```

- **fputc(c, pf)** converteste **c** la **unsigned char**, îl scrie în **pf** şi întoarce **(int)(unsigned char) c** sau **EOF** la eroare
- **putc()** este macro echivalent cu **fputc()**
- **fputs(s, pf)** copie şirul **s** terminat cu **'\0'** în **pf** fără să pună şi **'\0'**; **puts()** adaugă **'\n'**
- **ungetc(c, pf)** pune înapoi valoarea **(unsigned char)c** în stream-ul asociat lui **pf** (**c** nu este **EOF**)

Exemplu

```
/* Copiere fisier cu modificarea literelor mici in
litere mari */
char    file_name[MAXSTRING];
int     c;
FILE    *ifp, *ofp;

fprintf(stdout, "\nIntrodu numele unui fisier:  ");
scanf("%s", file_name);
ifp = fopen(file_name, "r");
if (!ifp) {
   printf("Eroare la deschiderea fisierului\n");
   return 1;
}
ofp = fopen("test.out", "w");
while ((c = getc(ifp))!=EOF) {
   if (islower(c)) c = toupper(c);
      putc(c, ofp);
}
```

Citirea/scrierea blocurilor de date

```
size_t fread(void *ptr, size_t size,
     size_t nelem, FILE *stream);
```

- Se citesc cel mult **`nelem*size`** octeţi (caractere) din stream-ul asociat în tabloul pointat de **`ptr`**. Se întoarce numărul de elemente transferate în tablou.

```
size_t fwrite(const void *ptr, size_t size,
  size_t nelem, FILE *stream);
```

- Se citesc cel mult **`nelem*size`** octeţi (caractere) din tabloul **`ptr`** şi se scriu în fişierul asociat cu **`stream`**. Se întoarce numărul elementelor din tablou transferate cu succes.

Exemplu

```
#include <stdio.h>
int main(){
    float tab[3] = {7.50, 9.75, 8.25};
    int i;
    FILE *ofp;
    ofp = fopen("note.dat", "wb");
    if (!ofp) { printf("Eroare fisier!\n"); return 1; }
    fwrite((const char *) &tab, 1, sizeof(tab), ofp);
    fclose(ofp);
    for (i = 0; i < 3; ++i) tab[i] = 0.0;

    FILE *ifp;
    ifp = fopen("note.dat", "rb");
    fread((char *) &tab, 1, sizeof(tab), ifp);

    for (i = 0; i < 3; ++i) printf("%f ", tab[i]);
    fclose(ifp);
    return 0;
}
```

Funcţii de acces aleator

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

- Poziţia indicatorului pentru următoarea operaţie este stabilită la **`offset`** octeţi faţă de **`place`**.
- Valoare lui **`place`** poate fi:
  - **`SEEK_SET`** sau 0
  - **`SEEK_CUR`** sau 1
  - **`SEEK_END`** sau 2
- Exemple:
  - poziţionarea la sfârşitul fişierului

    ```
    fseek(fp, 0, SEEK_END)
    ```
  - poziţionarea la caracterul precedent

    ```
    fseek(fp, -1, SEEK_CUR)
    ```
  - poziţionarea la începutul fişierului

    ```
    fseek(fp, 0, SEEK_SET)
    ```

Funcţii de acces aleator

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

- Returnează valoarea curentă a indicatorului de poziţie în fişierul **fp**; la fişierele binare este numărul de octeţi de la începutul fişierului, pentru cele text depinde de sistem.

```
int fsetpos(FILE *fp, const fpos_t *pos);
```

- Setează indicatorul de poziţie la valoarea pointată de **pos** şi întoarce 0 dacă s-a realizat cu succes

```
int fgetpos(FILE *fp, fpos_t *pos);
```

- Indicatorul de poziţie al fisierului **fp** este memorat la **pos** şi poate fi folosit ulterior
- Este returnat 0 în caz de succes

Funcţii de acces aleator

```
void rewind(FILE *fp);
```

- **`rewind(fp)`** este echivalent cu

```
(void) fseek(fp, 0L, SEEK_SET);
```

```
int remove(const char *filename);
int  rename(const char *old, const char *new);
```

Exemplu

```
/* Afisarea unui fisier de la sfarsit */
/*..*/
char   file_name[MAXSTRING];
int    c;
FILE   *ifp;

fprintf(stdout, "\nInput a file name:  ");
scanf("%s", file_name);
ifp = fopen(file_name, "rb");
fseek(ifp, 0, 2);   // pozitionare la sfarsit
fseek(ifp, -1, 1);  // pozitionare la ultimul octet
while (ftell(ifp) > 0) {
      c = getc(ifp);
      putchar(c);
      fseek(ifp, -2, 1); //octetul anterior
}
```

Funcţii pentru controlul erorilor

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

- Întoarce o valoare nenulă dacă indicatorul end-of-file este setat pentru **`fp`**

```
int ferror(FILE *fp);
```

- Întoarce o valoare nenulă dacă indicatorul de eroare este setat pentru **`fp`**

```
void clearerr(FILE *fp);
```

- Resetează indicatorii de eroare şi end-of-file pentru **`fp`**

```
void perror(const char *s);
```

- Tipăreşte un mesaj de eroare la **`stderr`**: se scrie şirul **`s`** apoi mesajul de eroare. Apelul **`perror(errno)`** scrie doar mesajul de eroare

Parametri în linia de comandă

```
D:\AlgsiProg\Exemple>suma.exe f_in f_out
```

- funcţia **`main()`** cu argumente

```
int main(int nr_arg, char *arg[]) { ... }
```

- testarea numărului de argumente

```
if (nr_arg != 3) {
    printf("Linie de comanda gresita.\n%s%s%s",
       "Trebuie sa introduceti", arg[0],
       "fisier_intrare fisier_iesire.\n");
    exit(1);
}
```

- utilizarea argumentelor

```
finp = fopen(arg[1],"r");
fout = fopen(arg[2],"w");
```

Argumentele funcţiei `main()`: variabile de mediu

```
#include <stdio.h>
void main(int argc, char *argv[], char *env[])
{
      int i;
      for (i=0; env[i] != NULL; i++)
            printf("%s\n", env[i]);
}
```

```
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=CG
ComSpec=C:\WINDOWS\system32\cmd.exe
ProgramFiles=C:\Program Files
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\DOCUME~1\Cristian\LOCALS~1\Temp
...
```

Exemplu: numărarea cuvintelor

- declararea pointerilor către fluxurile asociate

```
FILE *f_inp, *f_out;
```

- testarea numărului de argumente ale funcţiei **main()**

```
if (nr_arg != 3) {
    fprintf(stderr, ...);
    exit(1);
}
```

- deschiderea fişierului de intrare

```
if ((f_inp = fopen(arg[1], "r"))==NULL) {
    fprintf(stderr, ...);
    exit(1);
}
```

- deschiderea fişierului de ieşire

```
f_out = fopen(arg[2], "w");
```

Numărarea cuvintelor

- un cuvânt poate fi numărat când se "păşeşte" prima dată în el sau când este "părăsit"
- o variabilă de tip "flag" **`in_cuv`** ne va ajuta să ştim dacă suntem în interiorul unui cuvânt sau în spaţiul de separare
- funcţia **`este_sep(c)`** testează dacă un caracter **`c`** este separator de cuvinte
- fiecare cuvânt este scris într-un fişier de ieşire pe o linie separată

Numărarea cuvintelor

```
in_cuv = 0; nr_cuv = 0;
while ((c=getc(f_inp)) != EOF) {
  if (!este_sep(c)) putc(c, f_out);
  if (este_sep(c) && in_cuv) {
    in_cuv = 0;
    putc('\n', f_out);
  }
  else if (!este_sep(c) && !in_cuv) {
    nr_cuv++;
    in_cuv = 1;
  }
}
```

Tablouri de pointeri

```
#include <stdio.h>
#include <string.h>
void schimba(char**, char**);
void sort_cuvinte(char**, int);
void sort_cuvinte(char* w[], int n){
   int i, j;
   for (i = 0; i < n-1; ++i)
      for (j = i+1; j < n; ++j)
         if (strcmp(w[i], w[j]) > 0)
            schimba(&w[i], &w[j]);
}
void schimba(char** p, char** q){
   char* temp;
   temp = *p;
   *p = *q;
   *q = temp;
}
```

Tablouri de pointeri

```
int main (void){
  int i;
  char* luna[] ={ "ianuarie", "februarie",
          "martie", "aprilie", "mai", "iunie",
          "iulie", "august", "septembrie",
          "octombrie", "noiembrie","decembrie"};
  printf("\nLunile anului sunt: ");
  for (i = 0; i < 12; ++i)
     printf("%s ", luna[i]);
  sort_cuvinte(luna, 12);
  printf("\n\nLunile anului ordonate alfabetic
                                        sunt: ");
  for (i = 0; i < 12; ++i)
     printf("%s ", luna[i]);
  return 0;
}
```

Funcţiile ca argumente

Numele unei funcţii este pointer ce are ca valoare adresa de început a codului ei

```
printf("Adresa codului: %p", main);
```

Un parametru funcţie:

```
int f(double g(double x), int m);
int f(double g(double), int);
int f(double (*g)(double), int);
int f(double (*)(double), int);
int f(double (*)(double x), int);
```

Funcţiile ca argumente

```
double f(double), sin(double);
double sum_square(double (*)(double), int, int);

int main(void){
  printf("%s%.7f",
    "Primul apel: ", sum_square(sin, 2, 13));
  printf("%s%.7f",
   "Al doilea apel: ", sum_square(f, 1, 10000));
  return 0;
}
```

Funcţiile ca argumente

```
double sum_square(double (*f)(double), int m, int n)
{
   int k;
   double sum = 0.0;
   for (k = m; k <= n; ++k)
      sum += f(k) * f(k);     /* (*f)(k)*(*f)(k) */
   return sum;
}

double f(double x)
{
   return 1.0 / x;
}
```

Pointeri la funcţii

- Declaraţia tipului “pointer la funcţii”:

tip (**nume_pointer*)(*lista_tipuri*);

```
double (*pf)(int, double);
double f(int, double);
pf = f;
```

- Declaraţia tipului “tablou de pointeri la funcţii”:

tip (**nume_pointer*[*exp_const*])(*lista_tipuri*);

```
double (*pf[5])(int, double);
double f(int, double);
pf[3] = f;
```

Pointeri la funcţii

```
int g(){
      double (*pf)(int, double);
      double f(int, double);
      double (*tpf[5])(int, int);
      double h(int, int);
      tpf[3] = h;
      pf = f;
}
```

Tablou de pointeri la funcţii

```
/* Tablou pointeri la functii */
#include<assert.h>
#include<math.h>
#include<stdio.h>
#define N 4
typedef double dbl;
typedef dbl (*pointf)(dbl);
dbl bisection(pointf f, dbl a, dbl b);
dbl f1(dbl);
dbl f2(dbl);
dbl f3(dbl);
int cnt = 0;
const dbl eps = 1e-10;
```

Tablou de pointeri la funcţii

```
dbl bisection(pointf f, dbl a, dbl b){
   dbl m = (a + b)/2.0;
   ++cnt;
   if (f(m) == 0.0 || b - a < eps)
      return m;
   else if (f(a)*f(m) < 0.0)
            return bisection(f, a, m);
         else
            return bisection(f, m, b);
}

dbl f1(dbl x){return (x*x*x - x*x + 2.0*x - 2.0);}
dbl f2(dbl x){return (sin(x) - 0.7*x*x*x + 3.0);}
dbl f3(dbl x){return (exp(0.13*x) - x*x*x);}
```

Tablou de pointeri la funcţii

```
int main(void){
  int i_cnt, i, nf_calls;
  dbl a = -100.0, b = 100.0;
  dbl root, val;
  pointf f[N] = {NULL, f1, f2, f3};
  for (i = 1; i < N; ++i) {
    assert(f[i](a)*f[i](b) <= 0.0);
    i_cnt = cnt;
    root = bisection(f[i], a, b);
    nf_calls = cnt - i_cnt;
    val = f[i](root);
    printf("f[%d] are rad. aprox. = %.10f\n", i, root);
    printf("Numarul de apeluri bisection(): %d\n", nf_calls);
    printf("Valoarea f[%d](root) : %.10f\n", i, val);
  }
  return 0;
}
```

Funcţii cu număr variabil de argumente

- Este posibilă utilizarea funcţiilor ce au număr variabil de argumente:

```
suma(3, 6, 9, 4);
suma(5, 8, 5, 6, 3, 9);
max(2, 44, 22);
max(5, a, b, c, d, e);
scanf("%d %c %f", &n, &a, &x);
scanf("%d", &m);
```

- Declaraţie:
  - primele argumente (cel puţin unul) sunt fixe.
  - celelate argumente sunt declarate prin mecanismul "elipsa":

```
int suma(int nr_arg, ...);
int max(int nr_arg, ...);
int scanf(const char *_format, ...);
```

Funcţii cu număr variabil de argumente

În definiţia funcţiei sunt utilizate patru construcţii definite în fişierul **`stdarg.h`** :

- **`va_list`** tip pointer (pentru parcurgerea listei de argumente).
- **`va_start()`** iniţializează parcurgerea la primul argument.
- **`va_arg()`** oferă argumentul următor.
- **`va_end()`** termină parcurgerea listei, cu eliberarea memoriei

Exemplul 1

```
#include <stdarg.h>
#include <stdio.h>
int suma(int nargs,...){
   va_list args;                    //declaratie
   int i, total = 0;
   va_start(args, nargs);           //initializare
   for (i = 0; i < nargs; i++)
     total += va_arg(args, int); //val.arg.+ inaintare
   va_end(args);                   // eliberare
   return total;
}

int main(){
   printf("%d\n", suma(7, 1,2,3,4,5,6,7));     //=28
   printf("%d\n", suma(3, 1,2,3));             //=6
   return 0;
}
```

Exemplul 1. Explicaţii

`va_list args;`

declaraţie: **`args`** este declarată variabilă pointer (de tip **`void*`**)

`va_start(args, nargs);`

iniţializare: variabila **`args`** se iniţializează cu adresa argumentului următor lui **`nargs`** (din lista de argumente a funcţiei)

`va_arg(args, int);`

funcţie ce returnează valoarea argumentului curent şi modifică pointerul **`args`**; acesta va pointa la următorul argument

`va_end(args)`

se eliberează memoria alocată pentru pointerul **`args`**

Exemplul 2

```c
#include <stdio.h>
#include <limits.h>
#include <stdarg.h>
int max(int nargs,...){
    va_list args;
    int i, maxloc = INT_MIN, x;
    va_start(args, nargs);
    for (i = 0; i < nargs; i++)
        if ((x = va_arg(args, int)) > maxloc) maxloc = x;
    va_end(args);
    return maxloc;
}
int main(){
    printf("%d\n", max(7, 1,2,3,4,5,6,7));
    printf("%d\n", max(3, 1,2,3));
    printf("%d\n", max(0));
    return 0;
}
```

Măsurarea timpului de execuţie

```
#include <time.h>
//...
time_t t_inc, t_sf;
//...
t_inc = time(NULL);
/*
  cod pentru care se masoara timpul
*/
t_sf = time(NULL);
printf(..., difftime(t_sf, t_inc));
```

Data şi timpul în `time.h`

```
struct tm {
   int tm_sec;         // secunde : [0..60]
   int tm_min;         // minute : [0..59]
   int tm_hour;        // ore [0..23]
   int tm_mday;        // ziua din luna [1..31]
   int tm_mon;         // luni [0..11]
   int tm_year;        // ani din 1900
   int tm_wday;        // zile din sapt. [0..6]
   int tm_yday;        // zile din an [0..365]
   int tm_isdst;       // indicator
   char *__tm_zone;
   int __tm_gmtoff;
};

#define CLK_TCK CLOCKS_PER_SEC
typedef long clock_t;
typedef long time_t;

clock_t clock(void);
```

- nr. de CPU “clock ticks” ; in secunde: /CLK_TCK

Data şi timpul în `time.h`

```
time_t time(time_t *tod);
```

- Returnează timpul curent exprimat în nr. de secunde care au trecut de la 1 Ianuarie 1970. Daca tod este nenull acesta este memorat în *tod

```
char* ctime(const time_t *cal);
char* asctime(const struct tm *tptr);

struct tm * localtime(const time_t *tod);
```

`ctime(&now)` este echivalent cu:

`asctime(localtime(&now))`

```
double difftime(time_t t1, time_t t2);
```

- Calculează diferenţa **`t2 - t1`** şi o converteşte în nr. de secunde ce au trecut de la momentul **`t1`** până la **`t2`**

Exemplul 1

```c
#include <stdio.h>
#include <time.h>
int main(){
    time_t now;
    now = time(NULL);
    printf("%s%ld\n%s%s%s%s\n",
        " now = ", now,
        " ctime(&now) = ", ctime(&now),
        " asctime(localtime(&now)) = ",
                              asctime(localtime(&now)));
    return 0;
}
/*
                                          now = 1264518532
                    ctime(&now) = Tue Jan 26 17:08:52 2010
     asctime(localtime(&now)) = Tue Jan 26 17:08:52 2010
*/
```

Exemplul 2

```
/*..*/
  time_t t_inc, t_sf; clock_t c_inc, c_sf;
  unsigned long i = 1000000000;
  t_inc = time(NULL);
  c_inc = clock(); // Cat timp a utilizat procesul
  printf("%d %d\n", t_inc, c_inc);
  while(i--); // Cod pentru care se masoara timpul
  t_sf = time(NULL);
  c_sf = clock();
  printf("%d %d\n", t_sf, c_sf);
  printf("Timpul real: %.2f \t", difftime(t_sf, t_inc));
  printf("Timpul utilizator: %.2f \n",
                  (c_sf-c_inc)/(double)CLOCKS_PER_SEC);
/*
1231147371 0
1231147374 2375
Timpul real: 3.00    Timpul utilizator: 2.38
*/
```

Data şi timpul în `time.h`

```
size_t strftime(char *s, size_t n,
   const char *format, const struct tm *tptr);
```

- Funcţia scrie cel mult **`n`** caractere în şirul **`s`** cu formatul **`format`**. În format se dau specificatori pentru conversia elementelor de timp din structura **`tptr`**.

- Specificatorii de conversie:
  - **`%a %A %b %B`** specificatori pentru zile, respectiv luni
  - **`%c`** pentru data şi timp : **`Dec 16 12:34:50 2003`**
  - **`%d`** ziua din luna
  - **`%H %h`** ora din 24, respectiv 12 ore
  - **`%j %m`** ziua, respectiv luna din an
  - **`%M %S`** minute, respectiv secunde după ore
  - **`%p`** AM sau PM
  - **`%U %w`** săptămâna din an, respectiv ziua din săptămână
  - **`%x %X %y %Y %Z`** data, timpul, anul, anul, timp zonal

Exemplul 3

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

int main(){
	char s[100];
	time_t now;
	now = time(NULL);
	strftime(s, 100, "%H:%M:%S on %A, %d %B %Y",
			localtime(&now));
	printf("%s\n", s);
	return 0;
}

/*
17:14:13 on Tuesday, 26 January 2010
*/
```