LECTIA 13
INTRARI SI IESIRI

	Obiective	
	Prezentare generala	

OBIECTIVE

- "**intrari/iesiri standard** "si functiile care permit utilizatorului sa schimbe date prin intermediul componentelor periferice cu diferite programe.

PREZENTARE GENERALA

Prin **intrari/iesiri** intelegem un set de operatii care permit schimbul de date intre un program si un periferic. In general operatia de introducere a datelor de la un periferic se numeste **citire**, iar cea de iesire pe un periferic **scriere**. Numim **terminal standard** terminalul de la care sa lansat programul .De obicei terminalele standard sunt de tip display. In acest caz operatia se numeste si **afisare**. Limbajul C nu dispune de instructiuni specifice pentru operatiile de **intrare/iesire**. Ele pot fi apelate folosind functii special construite pentru acest scop.

Utilizatorul poate el insusi sa construiasca astfel de functii, in cazul in care nu sunt utilizabile cele existente. Functiile de biblioteca, utilizate mai frecvent pentru realizarea operatiilor de intrare/iesire folosind terminalul standard sunt :

- pentru intrari: **getch()**,**getche()**,**gets()** *si* **scanf()**;
- pentru iesiri: **putch**,**puts** *si* **printf().**

FUNCTII gets() si puts()

Functia **gets()** poate fi folosita pentru a introduce de la terminalul standard o succesiune de caractere terminata prin actionarea tastei Enter. Citierea se face fara ecou. Se pot citi numai caractere ale codului ASCII. Functia are ca parametru adresa de inceput a zonei de memorie in care se pastreaza caracterele citite. De obicei ,in aceasta zona de memorie este alocata unui tablou unidimensional de tip char. Functia **gets()** returneaza adresa de inceput a zonei de memorie in care s-au pastrat caracterele.

Functia **puts()** afisaza la terminalul standard un sir de caractere ale codului ASCII. Ea returneaza codulultimului caracter al sirului de caractere afisat .

Ele au urmatoarele prototipuri :

```
char *gets(char *str);
int puts(char *str);
```

Functiile **gets**() si **puts**() au prototipurile in fisierul **stdio.h** .

FUNCTIA printf()

Functia **printf()** poate fi folosita pentru a afisa date pe ecranul terminalului standard sub controlul unor formate.

```
printf(control , param 1 ,param 2,...,param n)
```

unde:

- **control**: este un sir de caractere
- **parametru** 1, .., parametru n - sunt expresii.

Fiecare specificatie de conversie e introdusa prin caracterul % si incheiata printr-un caracter de conversie. Intre % si caracterul de conversie pot fi:

- un semn minus care specifica alinierea la stanga in camp a argumentului convertit;
- un sir de digiti ce specifica o lungime minima a campului. Numarul convertit va fi introdus in camp la cel putin aceasta lungime sau mai mare , daca este necesar. Daca argumentul are mai putine caractere decat dimensiunea campului , el va fi aliniat la dreapta sau la stanga si va fi completat cu zero sau spatii pana la lungimea campului. Caracterul de completare utilizat in mod implicit e blank, iar daca dimensiunea campului a fost specificata prin leading zero, este zero;
- o virgula care separa lungimea campului de un alt sir de digiti;
- un sir de digiti ce specifica numarul maxim de caractere acceptate dintr-un sir sau numarul de pozitii dupa virgula zecimala.

Caracterele de conversie si semnificatiile lor sunt:

- **d,i** argumentul e convertit in zecimal;
- **o** argumentul e convertit in octal fara semn;
- **x,X** argumentul e convertit in hexazecimal fara semn;
- **u** argumentul e convertit in zecimal fara semn;
- **c** argumentul e preluat ca un singur caracter;
- **s** argumentul e un sir de caractere terminat cu caracterul nul;
- **e** argumentul e luat ca virgula flotanta sau dubla precizie si convertit in notatia stiintifica in care lungimea sirului de cifre de dupa virgula flotanta este specificata de precizie ;
- **f** argumentul e luat ca virgula mobila sau dubla precizie si convertit in notatia normala;

- **g** este similar lui e sau f , dar se bazeaza pe precizie.
- **%** caracterul %.

FUNCTIA scanf()

scanf() citeste caractere de la intarea standard , le interpreteaza conform formatului specificat in control si memoreaza rezultatele in celelalte argumente, care sunt pointeri ce indica unde vor fi depuse datele convertite.

Ea are urmatorul prototip:

```
int scanf(const char control,...)
```

Sirul de control contine specificatii de conversie care sunt utilizate pentru o interpretare directa a secventelor de intrare. Sirul de control poate sa contine:

- spatii, taburi, caractere de linie noua care sunt ignorate
- caractere ordinare
- specificatii de conversie continand caracterul % si caracterul de suprimare, un numar optional de specificare a lungimii maxime a campului si un caracter de conversie.

Caracterul de conversie indica interpretarea campului de la intrare, argumentul corespunzator trebuind sa fie un pointer. Urmatoarele caractere de conversie sunt legale:

- **d,i** un intreg zecimal e asteptat la intrare;
- **o** un intreg e asteptat la intrare;
- **x** un intreg hexazecimal e asteptat la intrare;
- **h** un intreg short e asteptat la intrare;
- **c** un singur caracter e asteptat la intrare; argumentul corespunzator trebuie sa fie un pointer la caracter;
- **s** un sir de caracter e asteptat la intrare; argumentul trebuie sa fie un pointer al unui tablou de caractere;
- **f** un numar in virgula flotanta e asteptat; argumentul corespunzator trebuie sa fie un pointer la un camp float. Caracterul de conversie e este un sinonim al lui f. Formatul prezentat la intrare pentru un float e alcatuit dintr-un semn optional, un sir de numere care pot sa contina si un punct zecimal si un camp de exponent care e format din E sau e, urmat de un intreg cu semn.

FUNCTIILE sscanf() si sprintf()

Functiile **scanf()** si **printf()** au corespondente functiile **sscanf()** si **sprintf()**, care executa acelasi tip de conversii, dar care opereaza prin intermediul unui sir si nu prin intrarea, respectiv iesirea, standard.

Formatul general este:

```
sprintf(char*string,const char *control,...)
sscanf(char *string,const char *control,...)
```

MACROURILE putchar() si getchar()

Aceste macrouri sunt definite in fisierul ***stdio.h***. Macroul getchar permite citirea cu ecou a caracterelor de la terminalul standard. Se pot caractere ASCII, nu si caractere corespunzatoare tastelor speciale.

```
getchar();
```

Macroul putchar afiseaza un caracter al codului ASCII. El afiseaza codul caracterului afisat sau eroare .

```
putchar(expresie);
```

FUNCTIA putch()

Functia **putch()** afisaza un caracter pe ecranul terminalului standard.Ea are un parametru care determina imaginea afisata la terminal.

```
putch(expresie);
```

Intrari/Iesiri orientate spre fisiere

Aceste operatii se realizeaza pin intermediul unor functii din biblioteca standard a limbajului. Aceste functii pot fi aplicate in mod eficient la o gama larga de aplicatii. Ele asigura o portabilitate buna a programelor, fiind implementate intr-o forma compatibila pe diferite sisteme de operare.

In continuare ne vom referi la functiile din biblioteca standard **I/O** care au o utilizare frecventa in diferite aplicatii.

Datele introduse de la tastatura unui terminal se considera ca formeaza un fisier de intrare. Datele care se afiseaza pe terminal formeaza un fisier de iesire.

Prin intrarea standard se intelege tastatura terminalului de la care s-a lansat programul. Iesirea standard este ecranul aceluiasi terminal. Prelucrarea fisierelor implica un numar de operatii specifice acestora:

- deschiderea de fisiere

- inchiderea
- creare
- citirea (consultarea) inregistrarilor unui fisier
- actualizarea unui fisier
- adaugarea de inregistrari intr-un fisier
- pozitionarea intr-un fisier
- stergerea unui fisier

Tratarea fisierelor se poate face la doua nivele. Primul nivel face apel direct la sistemul de operare. Acesta este nivelul inferior de prelucrare a fisierelor. Al doilea nivel se realizeaza prin intermediul unor proceduri specializate de prelucrare a fisierelor care utilizeaza structuri speciale de tip fisier. Acesta este nivelul superior de prelucrare a fisierelor.

Nivelul inferior de prelucrare a fisierelor

Deschiderea unui fisier

Deschiderea unui fisier se realizeaza cu ajutorul functiei **open()**. Functia **open**(), in forma cea mai simpla, se apeleaza printr-o expresie de atribuire de forma:

```
df=open(...);
```

unde: `df`=variabila de tip `int`.

Prototipul functiei **open**() :

`int open (const char*` cale, int acces);

unde:
- `cale :` este un pointer spre un sir de caractere care defineste calea spre fisierul care se deschide
- `acces` : este o variabila de tip intreg care poate avea una din valorile urmatoare:

- `O_RDONLY` - fisierul se deschide numai la citire
- `O_WRONLY` - fisierul se deschide numai la scriere
- `O_RDWR` - fisierul se deschide in citire/scriere
- `O_APPEND` - fisierul se deschide pentru adaugarea de inregistrari la sfirsit
- `O_BINARY` - fisierul se prelucreaza binar
- `O_TEXT` - fiserul este de tip text (se prelucreaza pe caractere)

Pentru a crea un fisier nou se utilizeaza functia **creat()** in locul functiei `open`.

Prototip:

`int creat(const char*` calea, int mod);

unde:
- `calea` este un pointer spre un sir de caractere care defineste calea spre fisierul care se deschide in creare.
- `mod` este un intreg care poate fi definit folosind constantele simbolice de mai jos :

- `S_IREAD` : proprietarul poate citi fisierul
- `S_IWRITE` : proprietarul poate scrie un fisier
- `S_IEXE` : proprietarul poate executa programul continut in fisierul respectiv.

Functia **creat(),** ca si functia **open()**, returneaza descriptorul de fisier sau -1 in caz de eroare.

Fisierele de **I/O** standard se deschid automat la lansarea programului in executie.

Citirea dintr-un fisier

Operatia de citire a unei inregistrari dintr-un fisier deschis se ralizeaza functia **read()**. Aceasta returneaza numarul de octeti dintr-un fisier sau -1 la eroare.

Prototip:

`int read(int` df, void *buf, unsigned lung);

unde:
- `df` este descriptorul de fisier returnat de functia open la deschiderea fisierului respectiv
- `buf` este pointerul spre zona de memorie in care se pastreaza inregistrarea citita dintr-un fisier
- `lung` este lungimea in octeti a inregistrarii citite.

La un apel al functiei **read()** se citesc cel mult `lung` octeti. Cand se citesc inregistrari de pe disc, o valoare utilizata frecvent pentru `lung` este 512. Utilizarea functiei **read()** implica includerea fisierului **io.h**

Scrierea intr-un fisier

Pentru a scrie o inregistrare intr-un fisier folosim functia **write()**. Fisierul trebuie sa fie in prealabil deschis cu functia **open()** sau **creat()**. Functia **write()** este asemanatoare cu functia **read()** si are acelasi

prototip. Diferenta consta in aceea ca realizeaza transferul datelor in sens invers. Functia **write()**poate fi folosita pentru a scrie la cele doua iesiri standard, folosind descriptorii de fisier de valoare 1 sau 2. Utilizarea functiei **write()** implica includerea fisierului **`io.h`**.

Pozitionarea intr-un fisier

Inregistrarile se scriu una dupa alta pe suportul fisierului. Acest mod de scriere si acces la inregistrarile fisierului se numeste acces secvential. Apar situatii in care noi dorim sa scriem si sa citim inregistrari intr-o ordine diferita de cea secventiala. In acest caz se spune ca accesul la fisier este aleator. O pozitionare oriunde in fisierul respectiv este posibila pe suporturile de disc magnetic si se realizeaza cu functia **lseek().**

Prototipul functiei este :

`long lseek(int` df, long deplasament, int origine);

unde:
- `df` este descriptorul de fisier
- `deplasament` defineste numarul de octeti peste care se va deplasa capul de citire/scriere al discului
- `origine` are una din valorile:

- 0 - `deplasamentul` se considera de la inceputul fisierului
- 1 - `deplasamentul` se considera din pozitia curenta a capului de citire/scriere
- 2 - `deplasamentul` se considera de la sfirsitul fisierului.

Utilizarea functiei **lseek()** implica includerea fisierului **io.h**.

Inchiderea unui fisier

Inchiderea unui fisier se realizeaza automat daca programul se termina prin apelul functiuni **exit()**. Programatorul poate inchide un fisier folosind **close()**.

Prototipul functiei **close**():

`int close (int` df);

unde:
- `df` : descriptorul fisierului care se inchide.

Utilizarea functiei **close()** implica includerea fisierului **io.h.**

NIVELUL SUPERIOR DE PRELUCRARE A FISIERELOR

La acest nivel operatiile de prelucrare a fisierelor se executa uilizindu-se functii specializate de gestiune a fisierelor. Toate functiile din aceasta clasa au prototipurile in fisierul **stdio.h**.

Deschiderea unui fisier

Pentru a deschide un fisier se utilizeaza functia **fopen()**. Ea returneaza un pointer spre tipul FILE sau pointerul in caz de eroare.
Prototipul:

`FILE *fopen(const char *calea,const char *mod);`

unde:
- `calea` are aceeasi semnificatie ca si in cazul functiei **open()**
- `mod` este un pointer spre un sir de caractere care defineste modul de prelucrare al fisierului dupa deschidere. Acest sir de caractere se defineste astfel:

- `"r"` - deschidere in citire (read)
- `"w"` - deschidere in scriere (write)
- `"a"` - deschidere pentru adaugare (append)
- `"r+"` - deschidere pentru modificare
- `"rb"` - citire binara
- `"wb"` - scriere binara
- `"r+b"` - citire/scriere binara.

Cu ajutorul functiei **fopen()** se poate deschide un fisier inexistent in modul `"w"` sau `"a"`. Pentru a utiliza aceste fisiere se vor folosi urmatorii pointeri spre tipul FILE:

- `stdin` : pentru a citi de la intrarea standard
- `stdout` : pentru a afisa pe ecranul de la iesirea standard
- `stderr` : pentru afisarea erorilor la iesirea standard
- `stdprn` : iesirea paralela pe imprimanta
- `stdaux` : comunicatie seriala.

Prelucrarea pe caractere a unui fisier

Fisierele pot fi scrise si citite caracter cu caracter folosind doua functii simple.

Functia **`putc()`** pentru scriere, al carei prototip este:

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

unde:
- `c` este codul ASCII al caracterului care se scrie in fisier
- `pf` este un pointer spre tipul FILE a carui valoare a fost returnata de functia **fopen**() la deschiderea fisierului in care se face scrierea. `pf` poate fi unul din pointerii :

- `stdout` - iesire standard
- `stderr` - iesire standard pentru eroare
- `stdprn` - iesire paralela la imprimanta
- `stdaux` - iesire seriala.

Functia **putc()** returneaza valoarea lui `c` sau -1 la eroare.

Macroul **putchar()** se defineste cu ajutorul functiei **putc**() astfel:

`#define putchar(c) putc(c,stdout)`

Functia **getc()** pentru citire, cu prototipul:

`int getc(FILE *pf);`

unde:
- `pf` este pointerul de tipul FILE si poate fi unul din pointerii:

- `stdin` - intrare standard
- `stdaux` - intrare seriala

Functia **getc()** returneaza codul ASCII al caracterului citit din fisier.

Macroul **getchar()** se defineste in fisierul **stdio.h**, astfel:

`#define getchar() getc(stdin)`

Inchiderea unui fisier

Inchiderea unui fisier se realizeaza cu ajutorul functiei **`fclose()`**. Prototip:

`int fclose(FILE *pf);`

unde:
- `pf` este pointerul spre tipul FILE.

Functia returneaza valorile:

0 - la inchiderea normala
1 - la eroare.

Intrari/iesiri de siruri de caractere

Functia **fgets()** permite citirea inregistrarilor care sunt siruri de caractere intr-un fisier. Prototip:

`char *fgets(char *s,int n, FILE *pf);`

unde:
- `s` este pointerul spre zona in care se pastreaza caracterele citite din fisier
- `n` este dimensiunea in octeti a zonei in care se citesc caracterele din fisier
- `pf` este pointerul spre tipul FILE a carui valoare s-a definit la deschiderea fisierului.

Citirea caracterelor se intrerupe la intilnirea caracterului '`\n`'. In zona spre care se pointeaza `"s"` se pastreaza caracterul '`\n`' daca acesta a fost citit din fisier, iar apoi se memoreaza caracterul nul ('`\0`'). La intilnirea sfirsitului de fisier functia returneaza valoarea zero.

Functia **fputs()** permite scrierea inregistrarilor care sunt siruri de caractere intr-un fisier. Prototip:

`int fputs(const char *s,FILE *pf);`

unde:
- `s` este pointerul spre inceputul zonei de memorie care contine sirul de caractere care se scrie in fisier
- `pf` este pointerul spre tipul FILE a carui valoare a fost definita la deschiderea fisierului prin apelul lui **fopen().**

Functia **fputs()** returneaza codul ASCII al ultimului caracter scris in fisier sau -1 la eroare.

Intrari/iesiri cu format

Biblioteca standard a limbajului **C** contine functii care permit realizarea operatiilor de intrare/iesire cu format. In acest scop se pot utiliza functiile **fscanf()** si **fprintf()**.

Functia **fscanf()** citeste date dintr-un fisier si le converteste pastrind rezultatele acestor conversii in conformitate cu parametrii existenti la apelul functiei, in timp ce **scanf()** realizeaza acelasi lucru dar utilizind date din memorie. Functia **fprintf()** converteste date din format intern in format extern si apoi le scrie intr-un fisier, spre deosebire de functia **sprintf()** care realizeaza aceleasi conversii, dar rezultatele se pastreaza in memorie.

Prototipul functiei **fscanf**():

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

unde:
- `pf` este un pointer spre tipul FILE a carui valoare a fost definita prin apelul functiei **fopen**(). Acesta defineste fisierul din care se face citirea. Ceilalti parametrii sunt identici cu cei utilizati in functia **scanf** ().

Functia **fprintf()**, ca si functiile **printf()** si **sprintf(),** returneaza numarul caracterelor scrise in fisier sau -1 in caz de eroare.

Eliberarea zonei tampon a unui fisier

In acest scop se poate utiliza functia **fflush()**. Prototip:

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

unde:
- `pf` este pointerul spre tipul FILE care defineste fisierul pentru care videaza zona tampon.

Daca fisierul este deschis la scriere, atunci continutul zonei tampon se scrie in fisierul respectiv. Daca fisierul este deschis la citire, caracterele necitite in zona tampon se pierd, deoarece dupa apelul functiei zona tampon devine goala.

Pozitionarea intr-un fisier

Functia **fseek()** permite deplasarea capului de citire/scriere al discului in vederea prelucrarii inregistrarilor fisierului intr-o ordine diferita decit cea secventiala. Prototip:

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

unde:
- `pf` este pointerul spre tipul FILE care defineste fisierul in care se face pozitionarea capului de citire/ scriere. Ceilalti parametrii se definesc la fel ca in cazul functiei **lseek**().

Functia **fseek()** returneaza valoarea zero la o valoare diferita de zero in caz de eroare.

Functia **ftell()** permite sa cunoasca pozitia curenta a capului de citire/scriere. Prototip:

`long ftell(FILE *pf);`

unde:
- `pf` este pointerul spre tipul FILE care defineste tipul in cauza.

Functia returneaza valoarea care defineste pozitia curenta a capului de citire/scriere. Valoarea returnata este deplasamentul in octeti a pozitiei capului de citire/scriere fata de inceputul fisierului.

Prelucrarea fisierelor binare

Fisierele organizate ca date binare pot fi prelucrate folosind functiile **fread()** si **fwrite()**. La o citire/ scriere se transfera intr-o zona tampon (buffer) un numar de articole care se presupune ca au o lungime fixa. Articolul este o data de tip oarecare.

Prototipul functiei **fread()** este:

`unsigned fread(void *ptr, unsigned dim, unsigned nrart, FILE *pf);`

unde:
- `ptr` este pointerul spre zona tampon care contine articolele citite (inregistrarea citita)
- `dim` defineste dimensiunea unui articol in octeti
- `nrart` defineste numarul de articole din compunerea iregistrarii citite
- `pf` este pointerul spre tipul FILE care defineste fisierul din care se face citirea.

Functia returneaza numarul de articole citite sau -1 in caz de eroare.

Functia **fwrite()** are protipul:

`unsigned fwrite(void *ptr, unsigned dim, unsigned nrart, FILE *pf);`

unde:

- `ptr` este pointerul spre zona tampon care contine articolele care se scriu in fisier
- `dim` defineste lungimea unui articol
- `nrart` defineste numarul de articole din compunerea inregistrarii care se scrie in fisier
- `pf` este pointerul spre tipul FILE care defineste fisierul in care se scrie inregistrarea.

Functia returneaza numarul articolelor scrise in fisier sau -1 in caz de eroare.

STERGEREA UNUI FISIER

Un fisier poate fi sters apelind functia `un` **link()** care are prototipul:

`int unlink(const` char *calea);

unde:
- `calea` este un pointer spre un sir de caractere identic cu cel utilizat la crearea fisierului in functia `creat` sau `fopen`.

Functia returneaza valoarea zero la o stergere reusita si -1 in caz de eroare.

REDIRECTAREA FISIERELOR DE INTRARE/IESIRE STANDARD

Functiile **scanf(), printf(), gets()** si **puts(),** precum si macrourile **getchar()** si **putchar()** pot fi utilizate si cu alte periferice daca, in prealabil, se face o redirectare a fiserelor standard de intrare/iesire. Acest lucru se realizeaza folosind, in linia de comanda a apelului executiei programului, caracterele '`C`' si '`>`'.

'`<`' se foloseste pentru redirectarea fisierelor de intrare (`stdin`).

'`>`' se foloseste pentru redirectarea fisierelor de iesire (`stdout`).

Ele se folosesc astfel:

`<specificator_de_fisier_de_intrare`

si

`>specificator_de_fisier_de_iesire`

Specificatorul de fisier depinde de sistemul de operare utilizat. Fisierul de iesire standard pentru erori **(stderr)** nu poate fi redirectat.