1 Operatori C

first 1	`++`	Suffix increment	Left to right
	`--`	Suffix decrement	
	`()`	Function call	
	`[]`	Array subscripting	
	`.`	Element selection by reference	
	`->`	Element selection through pointer	
2	`++`	Prefix increment	Right to left
	`--`	Prefix decrement	
	`+`	Unary plus	
	`-`	Unary minus	
	`!`	Logical NOT	
	`~`	Bitwise NOT (One's Complement)	
	`(type)`	Type cast	
	`*`	Indirection (dereference)	
	`&`	Address-of	
	`sizeof`	Size-of	
3	`*`	Multiplication	Left to right
	`/`	Division	
	`%`	Modulo (remainder)	
4	`+`	Addition	Left to right
	`-`	Subtraction	
5	`<<`	Bitwise left shift	Left to right
	`>>`	Bitwise right shift	
6	`<`	Less than	Left-to-right
	`<=`	Less than or equal to	
	`>`	Greater than	
	`>=`	Greater than or equal to	
7	`==`	Equal to	Left-to-right
	`!=`	Not equal to	
8	`&`	Bitwise AND	Left-to-right
9	`^`	Bitwise XOR (exclusive or)	Left-to-right
10	`|`	Bitwise OR (inclusive or)	Left-to-right
11	`&&`	Logical AND	Left-to-right
12	`||`	Logical OR	Left-to-right
13	`?:`	Ternary conditional (see ?:)	Right-to-left
14	`=`	Direct assignment	Right-to-left
	`+=`	Assignment by sum	
	`-=`	Assignment by difference	
	`*=`	Assignment by product	
	`/=`	Assignment by quotient	
	`%=`	Assignment by remainder	
	`<<=`	Assignment by bitwise left shift	
	`>>=`	Assignment by bitwise right shift	
	`&=`	Assignment by bitwise AND	
	`^=`	Assignment by bitwise XOR	
	`|=`	Assignment by bitwise OR	
15	`,`	Comma	Left-to-right

2 Exemple de programe introductive C

2.1 Compilare C sub Unix (Windows, gcc cu MinGW)

Forma generala a comenzii de compilare este:

gcc -o nume | -c -g -l surseC si eventual module rezultate din alte compilari

In forma simpla compilarea se face:

gcc sursa.c

Rezulta fisierul executabil a.out care se poate lansa:

./a.out argumente

Principalele optiuni gcc sunt:

-o nume compileaza si editeaza legaturi si genereaza un fisier executabil
   nume

-c compileaza sursa nume.c dar nu face editarea de legaturi. Rezultatul
   este fisierul obiect nume.o

-lNumeBiblioteca cauta in bibliotecile standard Unix biblioteca
  NumeBiblioteca si extrage din ea modulele care sunt necesare in program.

-g produce extracod pentru depanarea programului in timpul executiei.

Exemple de programe C simple: argvenvp.c, pi.c, plus10.c, bubbleInt.c, bubbleString.c, binaryInt.c, binaryString.c

2.2 Afiseaza argumentele liniei de comanda si variabilele de mediu

Sursa argvenvp.c este:

```
// Afiseaza numarul de argumente din linia de comanda,
// argumentele liniei de comanda si variabilele de mediu
#include <stdio.h>
main (int argc, char *argv[], char *envp[]) {
    int i;
    printf ("%d\n", argc);
    for (i = 0; argv[i] /* echivalent: i < argc */ ; i++)
        printf ("%s\n", argv[i]);
    printf ("\n");
    for (i = 0; envp[i]; i++)
        printf ("%s\n", envp[i]);
}
```

Rularea:

```
gcc argvenvp.c

a.exe unu 2 Trei patru 5
6
a
unu
2
```

```
Trei
patru
5

ACTIVEMQ_HOME=c:\apache-activemq-5.9.0
ALLUSERSPROFILE=C:\ProgramData
ANT_HOME=c:\apache-ant-1.8.3
APPDATA=C:\Users\Florin\AppData\Roaming
COMMANDER_DRIVE=C:
COMMANDER_EXE=C:\totalcmd\TOTALCMD64.EXE
COMMANDER_INI=C:\Users\Florin\AppData\Roaming\GHISLER\wincmd.ini
COMMANDER_PATH=C:\totalcmd
CommonProgramFiles=C:\Program Files (x86)\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
COMPUTERNAME=RLF
ComSpec=C:\Windows\system32\cmd.exe
FP_NO_HOST_CHECK=NO
HOMEDRIVE=C:
HOMEPATH=\Users\Florin
JAVA_HOME=c:\jdk1.7.0_51
JBOSS_HOME=c:\jboss-as-7.1.1.Final
JETTY_HOME=jetty-distribution-8.1.7.v20120910
LOCALAPPDATA=C:\Users\Florin\AppData\Local
LOGONSERVER=\\RLF
MYSQL_HOME=c:\xampp\mysql
NUMBER_OF_PROCESSORS=8
OS=Windows_NT
Path=C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Progr
am Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;c:\jdk1.7.
0_51\bin;c:\jdk1.7.0_51\jre\bin;c:\apache-ant-1.8.3\bin;c:\Python27;c:\jboss-as-
7.1.1.Final\bin;c:\xampp\mysql\bin;C:\Program Files (x86)\OpenVPN\bin;c:\MinGW\bin;c:\Python27\Scripts;c:\MyBin
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_ARCHITEW6432=AMD64
PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 42 Stepping 7, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=2a07
ProgramData=C:\ProgramData
ProgramFiles=C:\Program Files (x86)
ProgramFiles(x86)=C:\Program Files (x86)
ProgramW6432=C:\Program Files
PROMPT=$P$G
PSModulePath=C:\Windows\system32\WindowsPowerShell\v1.0\Modules\
PUBLIC=C:\Users\Public
PYTHON_HOME=c:\Python27
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\Windows
TEMP=C:\Users\Florin\AppData\Local\Temp
TMP=C:\Users\Florin\AppData\Local\Temp
USERDOMAIN=RLF
USERNAME=Florin
USERPROFILE=C:\Users\Florin
VBOX_INSTALL_PATH=C:\Program Files\Oracle\VirtualBox\
windir=C:\Windows
```

2.3 2. Calculul lui PI

Sursa pi.c este:

```
// Calculeaza aproximativ PI in trei variante:
// 1. Folosind functia standard atan
// 2. Folosind seria armonica patrata SUM(1/n^2) -> PI^2/6
// 3. Primele 800 cifre ( vezi http://crypto.stanford.edu/pbc/notes/pi/code.html).
#include <stdio.h>
#include <math.h>
main() {
    printf("PI (cu atan) = %12.10f\n", 4 * atan(1.0));
    int n;
    double pi;
    for (n=2, pi= 1.0; ; n++) {
        double t = (double)1.0/(double) n /(double) n;
        if (t < 0.000000000001) break;
        pi += t;
    }
    pi = sqrt(6.0 * pi);
    printf("PI (cu %d termeni ai seriei) = %12.10f\n", n, pi);
    printf("PI (cu 800 cifre) = ");
    int r[2800 + 1];
    int i, k;
    int b, d;
    int c = 0;
    for (i = 0;i < 2800; i++) {
        r[i] = 2000;
    }
    for (k = 2800; k > 0; k -= 14) {
        d = 0;
        i = k;
        for (;;) {
            d += r[i] * 10000;
            b = 2 * i - 1;
            r[i] = d % b;
            d /= b;
            i--;
            if (i == 0) break;
            d *= i;
        }
        printf("%.4d", c + d / 10000);
        c = d % 10000;
    }
    printf("\n");
}
```

Rularea:

```
gcc pi.c   (pe unele platforme se compileaza cu optiunea -lm )

./a.out
PI (cu atan) = 3.1415926536
PI (cu 1000001 termeni ai seriei) = 3.1415916987
PI                      (cu                      800                      cifre)                      =
31415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679821480865132
82306647093844609550582231725359408128481117450284102701938521105 55
96446229489549303819644288109756659334461284756482337867831652712019091456485669234603486104543266482133936072602
49141273724587006606315588174881520920962829254091715364367892590360011330530548820 4665
21384146951941511609433057270365759591953092186117381932611793105118548074462379962749567351885752724891227938183
01194912983367336244065664308602139494639522473719070217986094370277053921717629 3176752
38467481846766940513200056812714526356082778577134275778960917363717872146844090122495343014654958537105079227968
92589235420199561121290219608640344181598136297747713099605187072113499999983729 7804995
10597317328160963185
```

2.4 Adunarea a doua numere mari, cifra cu cifra, reprezentate ca stringuri.

Sursa plus10.c este:

```
// Adunarea, cifra cu cifra in baza 10, a doua numere pozitive cu maximum 100
// cifre fiecare, date ca siruri de cifre ASCII
#include <stdio.h>
#include <string.h>
#include <ctype.h>
main () {
    char a[101], b[101], c[101];
    int m, n, k, i, t;
    fgets(a, 101, stdin);
    fgets(b, 101, stdin);
    for ( i=0; i<strlen(a); i++)
        if (!isdigit(a[i])) {
            a[i] = 0; // se retin doar primele cifre
            break;
        }
    for ( i=0; i<strlen(b); i++)
        if (!isdigit(b[i])) {
            b[i] = 0; // se retin doar primele cifre
            break;
        }
    m = strlen(a);
    n = strlen(b);
    k = (n < m? m : n) + 1;
    c[k] = 0; // restul locurilor vor fi completate dupa adunare
    // Se aliniaza sirurile a si b deplasandu-le spre deapta
    for (i = k; i >= k - m; a[i] = a[i - k + m], i--);
    for (i = k; i >= k - n; b[i] = b[i - k + n], i--);
    // Se completeaza cu zerouri la stanga
    for (i = 0; i < k - m; a[i] = '0', i++);
    for (i = 0; i < k - n; b[i] = '0', i++);
    for (i = k - 1, t = 0; i >= 0; i--) {
        t = a[i] - '0' + b[i] - '0' + t;
        c[i] = t % 10 + '0';
        t = t / 10;
    }
    // Se elimina zerourile nesemnificative
    for (i = 0; a[i] == '0' && i < m ; a[i++] = ' ');
    for (i = 0; b[i] == '0' && i < n ; b[i++] = ' ');
    for (i = 0; c[i] == '0' && i < k ; c[i++] = ' ');
    printf("%s +\n%s =\n%s\n", a, b, c);
}
```

Rularea:

```
gcc plus10.c

./a.out
01234567890123456789
999999999999999999999
    1234567890123456789 +
 999999999999999999999 =
1000123456789012345678 8
```

2.5 Citirea unui sir de intregi si sortarea lui prin bubble sort

Sursa bubbleInt.c este:

```
// Citeste de la intrarea standard maximum 100 numere intregi,
// cate unul pe linie. Citirea se termina la tastarea
```

```
// EOF (^d sau ^z) sau la limita de 100 numere.
// Se sorteaza sirul prin metoda bulelor
// si se tipareste.
#include <stdio.h>
main () {
    int sir[100],i, j, x, n, u, dr;
    for ( n=0; ; ) {
        if (fscanf(stdin, "%d", &x) == EOF) break;
        if (n >= 100) break;
        sir[n++] = x;
    }
    printf("Sirul initial: ");
    for (i=0; i<n; fprintf(stdout, "%d ", sir[i]), i++);
    dr = n - 1;
    for ( ; ; ) {
        for (i=0, u=-1; i<dr; i++) {
            if (sir[i] > sir[i+1]) {
                x = sir[i];
                sir[i] = sir[i+1];
                sir[i+1] = x;
                u = i;
            }
        }
        if ( u <= 0) break;
        dr = u;
    }
    printf("\nSirul ordonat: ");
    for (i=0; i<n; fprintf(stdout, "%d ", sir[i]), i++);
}

Rularea:

gcc bubbleInt.c

./a.out
8
6
-1
0
6
6
^d
Sirul initial: 8 6 -1 0 6 6
Sirul ordonat: -1 0 6 6 6 8
```

2.6 Citirea unui vector de linii si ordonarea lui folosind bubble sort

```
The source bubbleString.c is:

// Citeste de la intrarea standard un sir de maximum 100 linii,
// cu maximum 50 caractere fiecare (cele mai lungi se scurteaza).
// Citirea se termina la tastareaEOF (^d sau ^z) sau la limita de 100 linii.
// Se sorteaza alfabetic sirul prin metoda bulelor
// si se tipareste.
# include <stdio.h>
# include <string.h>
main () {
    char sir[100][50],i, j, x[50], n, u, dr;
    for ( n=0; ; ) {
        if (fgets(x, 50, stdin) == NULL) break;
```

```
        if (n >= 100) break;
        x[strlen(x) - 1] = 0; // Se sterge \n de la sfarsit (fgets il lasa)
        strcpy(sir[n++], x);
    }
    printf("Sirul initial:\n");
    for (i=0; i<n; fprintf(stdout, "%s\n", sir[i]), i++);
    dr = n - 1;
    for ( ; ; ) {
        for (i=0, u=-1; i<dr; i++) {
            if (strcmp(sir[i], sir[i+1]) > 0) {
                strcpy(x, sir[i]);
                strcpy(sir[i], sir[i+1]);
                strcpy(sir[i+1], x);
                u = i;
            }
        }
        if ( u <= 0) break;
        dr = u;
    }
    printf("\nSirul ordonat:\n");
    for (i=0; i<n; fprintf(stdout, "%s\n", sir[i]), i++);
}

gcc bubbleString.c

./a.out
masina
valiza
nu stiu ce
avion
5
^d
Sirul initial:
masina
valiza
nu stiu ce
avion
5

Sirul ordonat:
5
avion
masina
nu stiu ce
valiza
```

2.7 Cautare binara intr-un vector ordonat de intregi

Sursa binaryInt.c este:

```
// Cauta binar intr-un sir ordonat o valoare v data la linia de comanda
#include <stdio.h>
main (int argc, char *argv[]) {
    int sir[] = {1,2,4,5,7,8,9};
    int m, s, d, v, n;
    v = atoi(argv[1]);
    int g = 0;
    n = sizeof(sir) / sizeof(int);
    printf("Se cauta %d in sirul urmator:\n", v);
    for (s=0; s<n; printf("sir[%d]=%d ",s, sir[s]), s++);
```

```
    printf("\n");
    for ( s = 0, d = n-1; s <= d ; ) {
        m = (s + d) / 2;
        if (sir[m] == v) {g=1;break;}
        if (sir[m] <= v) s = m + 1; else d = m - 1;
    }
    if (g == 0)
        printf("NU! %d ar trebui sa fie aproape de pozitia %d\n", v, m);
    else
        printf("Da, %d este pe pozitia %d in sir\n", v, m);
}
```

Rularea:

```
gcc binaryInt.c

./a.out 6
Se cauta 6 in sirul urmator:
sir[0]=1 sir[1]=2 sir[2]=4 sir[3]=5 sir[4]=7 sir[5]=8 sir[6]=9
NU! 6 ar trebui sa fie aproape de pozitia 4

./a.out 7
Se cauta 7 in sirul urmator:
sir[0]=1 sir[1]=2 sir[2]=4 sir[3]=5 sir[4]=7 sir[5]=8 sir[6]=9
Da, 7 este pe pozitia 4 in sir
```

2.8 Cautare binara intr-un vector ordonat stringuri

Sursa binaryString.c este:

```
// Cauta binar intr-un sir ordonat o valoare v data la linia de comanda
#include <stdio.h>
#include <string.h>
main (int argc, char *argv[]) {
    char* sir[] = {"avion", "caruta", "masina", "tramvai", "vapor"};
    int m, s, d, n, c;
    char v[100];
    strcpy(v, argv[1]);
    int g = 0;
    n = sizeof(sir) / sizeof(char*);
    printf("Se cauta %s in sirul urmator:\n", v);
    for (s=0; s<n; printf("sir[%d]=%s ",s, sir[s]), s++);
    printf("\n");
    for ( s = 0, d = n-1; s <= d ; ) {
        m = (s + d) / 2;
        c = strcmp(sir[m], v);
        if (c == 0) {g=1;break;}
        if (c < 0) s = m + 1; else d = m - 1;
    }
    if (g == 0)
        printf("NU! %s ar trebui sa fie aproape de pozitia %d\n", v, m);
    else
        printf("Da, %s este pe pozitia %d in sir\n", v, m);
}
```

Rularea:

```
gcc binaryString.c

./a.out ceva
```

```
Se cauta ceva in sirul urmator:
sir[0]=avion sir[1]=caruta sir[2]=masina sir[3]=tramvai sir[4]=vapor
NU! ceva ar trebui sa fie aproape de pozitia 1

./a.out caruta
Se cauta caruta in sirul urmator:
sir[0]=avion sir[1]=caruta sir[2]=masina sir[3]=tramvai sir[4]=vapor
Da, caruta este pe pozitia 1 in sir
```

3 Declaraţii definiţii utilizări de variabile şi funcţii

Orice entitate C: constantă, variabilă, funcţie trebuie mai întâi să fie **declarată** sau **definită** şi apoi **utilizată**.

- Declararea înseamnă pentru anunţarea tipului şi numelui; în cazul declarării funcţiilor se specifică prototipul (tipul rezultat şi lista parametrilor.
- Definirea înseamnă pentru variabile alocarea spaţiului de memorie (la declaraţii statice se face implicit); în cazul funcţiilor se precizează şi corpul acestora.
- Utilizarea se poate face fie dacă definirea se face înainte de utilizare, fie se face doar o declarare, urmând ca definirea să urmeze în textul programului.

3.1 Definirea de funcţii; transmiterea parametrilor

In C **nu** există funcţie în funcţie! Corpurile funcţiilor sunt distincte.
Variabilele declarate / definite între funcţii se văd în toate corpurile funcţiilor începând cu locul anunţării.
O variabilă dintre funcţii poate fi declarată **extern**, asta însemnând că definirea se face înafara textului sursă curent.
Parametrii funcţiilor sunt transmişi prin valoare. Aceasta înseamnă că se depune în stivă valoarea fiecăui parametru. **Orice funcţie întoarce un rezultat.**
Singurele tipuri ce pot fi transmise ca parametri şi întoarce ca rezultat sunt:

- `char ( + unsigned char)`
- `int ( + unsigned int, short int, unsigned short int, long, unsigned long)`
- `float (+ double)`
- `void` - absenta oricarui tip
- `pointer` - la orice tip de date primitive sau definite de utilizator. Parametrii functie se transmit tot prin pointeri (numele de functie este un pointer). Vezi exemplul.

In consecinţă, utilizatorul trebuie să aibă grijă ca pentru **struct**, **union** şi tablouri să folosească pentru transmitere pointeri.

3.2 Exemple de apeluri de funcţii, fişierul functii.c:

```
#include <stdio.h>
#include <string.h>
typedef struct student {int varsta; char nume[10];} STUDENT;
void incv(int x) {x++; /* Efectul nu se propaga inafara functiei! */}
void incp(int *x) {(*x)++;}
void array(int x[], int i, int v) {x[i] = v;}
void structp(STUDENT *x, int v, char *n) {x->varsta = v; strncpy(x->nume, n, 9);}
float x2(float x) {return x*x;}
float integrala(float a, float b, int n, float (*f)(float)) {
```

```
    float h, s;
    int i;
    h = (b-a)/n;
    s = ((*f)(a) + (*f)(b))/2.0;
    for (i=0; i<n; s += (*f)(a + i * h), i++);
    return h * s;
}
main () {
    int v = 5;
    int t[10];
    STUDENT student;
    incv(v);
    printf("Valoare %d\n", v);
    incp(&v);
    printf("Adresa %d\n", v);
    array(t, 2, 77);
    printf("Array %d\n", t[2]);
    structp(&student, 21, "Ionescu");
    printf("%s, %d ani\n", student.nume, student.varsta);
    printf("%10.6f\n", integrala(0.0, 1.0, 10000, x2));
}
```

3.3 Acelaşi exemplu, dar "rupt" în trei surse:

Fişierul header **declaratii.h**

```
typedef struct student {int varsta; char nume[10];} STUDENT;
void incv(int x);
void incp(int *x);
void array(int x[], int i, int v);
void structp(STUDENT *x, int v, char *n);
float x2(float x);
float integrala(float a, float b, int n, float (*f)(float));
```

Fisierul sursa cu definitii de functii **definitii.c**

```
void incv(int x) {x++;}
void incp(int *x) {(*x)++;}
void array(int x[], int i, int v) {x[i] = v;}
void structp(STUDENT *x, int v, char *n) {x->varsta = v; strncpy(x->nume, n, 9);}
float x2(float x) {return x*x;}
float integrala(float a, float b, int n, float (*f)(float)) {
    float h, s;
    int i;
    h = (b-a)/n;
    s = ((*f)(a) + (*f)(b))/2.0;
    for (i=0; i<n; s += (*f)(a + i * h), i++);
    return h * s;
}
```

Fisierul sursa **functii2.c** care le reuneste:

```
#include <stdio.h>
#include <string.h>
#include "declaratii.h"
main () {
    int v = 5;
    int t[10];
```

```
    STUDENT student;
    incv(v);
    printf("Valoare %d\n", v);
    incp(&v);
    printf("Adresa %d\n", v);
    array(t, 2, 77);
    printf("Array %d\n", t[2]);
    structp(&student, 21, "Ionescu");
    printf("%s, %d ani\n", student.nume, student.varsta);
    printf("%10.6f\n", integrala(0.0, 1.0, 10000, x2));
}
# include "definitii.c"
```

4 Operaţii I/O şi lucrul cu fişiere în C

4.1 Operaţii I/O

Există două posibilităţi de efectuare a operaţiilor I/O din programe C:

- Prin funcţiile standard C (`fopen, fclose, fgets, fprintf, fread, fwrite, fseek, sprints, sscanf` etc.) existente în bibliotecile standard C; prototipurile acestora se află în fişierul header `<stdio.h>` (*nivelul superior de prelucrare al fişierelor*). Pentru orice detalii legate de aceste functii, ca si pentru alte functii inrudite cu acestea, se pot consulta manualele Unix `$ man numefunctie` sau `$ man 3 numefunctie`
- Prin funcţii standardizate POSIX (`open, close, read, write, lseek, dup, dup2, fcntl`, etc.) care reprezintă puncte de intrare în nucleul Unix şi ale căror prototipuri se află de regulă în fişierul header `<unistd.h>`, dar uneori se pot afla şi în `<sys/types.h>, <sys/stat.h>` sau `<fcntl.h>` (*nivelul inferior de prelucrare al fişierelor*). Pentru orice detalii legate de aceste functii, ca si pentru alte functii inrudite cu acestea, se pot consulta manualele Unix: `$ man numefunctie` sau `$ man 2 numefunctie`

Prima categorie de funcţii o presupunem cunoscută deoarece face parte din standardul C (ANSI). Funcţiile din această categorie reperează orice fişier **printr-o structură `FILE *`,** pe care o vom numi *descriptor de fişier*.

Funcţiile din a doua categorie constituie apeluri sistem Unix pentru lucrul cu fişiere şi fac obiectul secţiunii care urmează. Ele (antetul lor) sunt cuprinse în standardul POSIX. Funcţiile din această categorie reperează orice fişier **printr-un întreg nenegativ, numit *handle*,** dar atunci când confuzia nu este posibilă îl vom numi tot *descriptor de fişier.*.

Informatii despre formatele fisierelor Pentru a obtine detalii despre formatele de fisiere si despre functii sau comenzi specifice formatelor de fisiere se poate consulta

```
$ man 5 nume
```

4.1.1 Apelul sistem open

Prototipul funcţiei sistem este:

```
int open (char *nume, int flag [, unsigned int drepturi ]);
```

Funcţia `open` întoarce un întreg - *handle* sau *descriptor de fişier*, folosit ca prim argument de către celelalte funcţii POSIX de acces la fişier. In caz de eşec `open` întoarce valoarea -1 şi poziţionează corespunzător variabila `errno`. In cele ce urmează vom numi `descr` acest număr.

`nume` - specifică printr-un string C, calea şi numele fişierului în conformitate cu standardul Unix.

Modul de deschidere este precizat de parametrul de deschidere `flag`. Principalele lui valori posibile sunt:

- O_RDONLY deschide fişierul numai pentru citire
- O_WRONLY deschide fişierul numai pentru scriere
- O_RDWR deschide fişierul atât pentru citire şi pentru scriere
- O_APPEND deschide pentru adaugarea - scrierea la sfîrşitul fişierului
- O_CREAT creează un fişier nou dacă acesta nu există, sau nu are efect dacă fişierul deja există; următoarele două constante completează crearea unui fişier
- O_TRUNC asociat cu O_CREAT (şi exclus O_EXCL vezi mai jos) indică crearea necondiţionată, indiferent dacă fişierul există sau nu
- O_EXCL asociat cu O_CREAT (şi exclus O_TRUNC), în cazul în care fişierul există deja, `open` eşuează şi semnalează eroare
- O_NDELAY este valabil doar pentru fişiere de tip pipe sau FIFO şi vom reveni asupra lui când vom vorbi despre pipe şi FIFO.

In cazul în care se folosesc mai multe constante, acestea se leagă prin operatorul |, ca de exemplu: O_CREAT | O_TRUNC | O_WRONLY.

Parametrul `drepturi` este necesar doar la crearea fişierului şi indică drepturile de acces la fişier (prin cei 9 biţi de protecţie) şi acţionează în concordanţă cu specificarea `umask`.

4.1.2 Apelul sistem close

Inchiderea unui fişier se face prin apelul sistem:

```
int close (int descr);
```

Parametrul `descr` este cel întors de apelul `open` cu care s-a deschis fişierul. Funcţia întoarce 0 la succes sau -1 în caz de eşec.

4.1.3 Apelurile sistem read şi write

Acestea sunt cele mai importante funcţii sistem de acces la conţinutul fişierului. Prototipurile lor sunt:

```
int read (int descr, void *mem, unsigned int noct);
int write (int descr, const void *mem, unsigned int noct);
```

Efectul lor este acela de a citi (scrie) din (în) fişierul indicat de `descr` un număr de `noct` octeţi, depunând (luând) informaţiile în (din) zona de memorie aflată la adresa `mem`. In majoritatea cazurilor de utilizare, `mem` referă un şir de caractere (`char* mem`).

Cele două funcţii întorc numărul de octeţi efectiv transferaţi între memorie şi suportul fizic al fişierului. De regulă, acest număr coincide cu `noct`, dar sunt situaţii în care se transferă mai puţin de `noct` octeţi.

Aceste două operaţii sunt atomice - indivizibile şi neintreruptibile de către alte procese. Deci, dacă un proces a început un transfer între memorie şi suportul fişierului, atunci nici un alt proces nu mai are acces la fişier până la terminarea operaţiei `read` sau `write` curente.

Operaţia se consideră terminată dacă s-a transferat cel puţin un octet, dar nu mai mult de maximul dintre `noct` şi ceea ce permite suportul fişierului. Astfel, dacă în fişier există doar `n < noct` octeţi rămaşi necitiţi atunci se vor transfera în memorie doar `n` octeţi şi `read` va întoarce valoarea `n`. Dacă în suportul fişierului există cel mult `n < noct` octeţi disponibili, atunci se depun din memorie în fişier doar `n` octeţi şi `write` va întoarce valoarea `n`. In ambele situaţii, pointerul curent al fişierului avansează cu numărul de octeţi transferaţi. Dacă la lansarea unui `read` nu mai există nici un octet necitit din fişier - s-a întâlnit marcajul de sfârşit de fişier, atunci funcţia întoarce valoarea 0.

Dacă operaţia de citire sau de scriere nu se poate termina - a apărut o eroare în timpul transferului - funcţia întoarce valoarea -1 şi se poziţionează corespunzător variablia `errno`.

4.1.4 Apelul sistem lseek

Funcţia sistem `lseek` facilitează accesul direct la orice octet din fişier. Evident, pentru aceasta trebuie ca suportul fişierului să fie unul adresabil. Prototipul acestei funcţii sistem este:

```
long lseek (int descr, long noct, int deUnde);
```

Se modifică pointerul curent în fişierul indicat de `descr` cu un număr de `noct` octeţi. Punctul de unde începe numărarea celor `noct` octeţi este indicat de către valoarea parametrului `deUnde`, astfel:

- de la începutul fişierului, dacă are valoarea SEEK_SET (valoarea 0)
- de la poziţia curentă dacă are valoarea SEEK_CUR (valoarea 1)
- de la sfârşitul fişierului dacă are valoarea SEEK_END (valoarea 2)

4.2 Acces la sistemul de fisiere

Unix permite efectuarea din C a principalelor operaţii asupra sistemului de fişiere, oferind în acest sens câteva apeluri sistem. Ele sunt echivalente cu comenzile Shell care efectuează aceleaşi operaţii. Aceste operaţii sunt definite prin specificări POSIX corespunzătoare.

De regulă, aceste funcţii au prototipurile într-unul dintre fişierele header:

```
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
```

4.2.1 Manevrarea fişierelor în sistemul de fişiere

Iată prototipurile celor mai importante dintre aceste apeluri sistem:

```
int chdir (const char *nume);
char *getcwd(char *mem, int dimensiune);
int mkdir (const char *nume, unsigned int drepturi);
int rmdir (const char *nume);
int unlink(const char *nume);
int link(const char *numevechi, const char *numenou);
int symlink(const char *numevechi, const char *numenou);
int chmod (const char *nume, unsigned int drepturi);
int stat  (const char *nume, struct stat *stare);
int mknod(const char *nume, unsigned int mod, dev_t dev);
int chown(const char *nume, unsigned int proprietar,
          unsigned int grup);
int access(const char *nume, int permisiuni);
int rename(const char *numevechi, const char *numenou);
```

- `chdir` schimbă directorul curent în cel specificat prin `nume`. Pe sistemele BSD şi Unix System V release 4 mai este prezent şi apelul `fchdir` care face acelaşi lucru ca şi `chdir`, doar că directorul este specificat printr-un descriptor de fişier deschis, nu prin nume.
- `getcwd` copiază în zona de memorie indicată de `mem`, de lungime `dimensiune` octeţi, calea absolută a directorului curent. Dacă calea absolută a directorului are lungimea mai mare decât `dimensiune`, se returnează NULL şi `errno` primeşte valoarea ERANGE.
- `mkdir` creează un nou director, având calea şi numele specificate prin `nume` şi drepturile indicate prin întregul `drepturi`, din care se reţin numai primii 9 biţi.
- `rmdir` şterge directorul specificat prin `nume` (acest director trebuie să fie gol).
- `unlink` şterge fişierul specificat prin `nume`.
- `link` creează o legătură hard cu numele `numenou` spre un fişier existent, `numevechi`. Numele nou se poate folosi în locul celui vechi în toate operaţiile şi nu se poate spune care dintre cele două nume este cel iniţial.
- `symlink` creează o legătură simbolică (soft) cu numele `numenou` spre un fişier existent, `numevechi`. O legătură soft este doar o referinţă la un fişier existent sau inexistent. Dacă şterg o legătură soft se şterge doar legătura, nu şi fişierul original, pe când dacă şterg o legătură hard, se şterge fişierul original.
- `chmod` atribuie drepturile de acces `drepturi` la fişierul specificat prin `nume`. Pe sistemele BSD şi Unix System V release 4 mai este prezent şi apelul `fchmod` care face acelaşi lucru ca şi `chmod`, doar că fişierul este specificat printr-un descriptor de fişier deschis, nu prin nume.
- `stat` depune la adresa `stare` informaţii privind fişierul specificat prin `nume` (inod-ul fişierului, drepturile de acces, id-ul proprietarului, id-ul grupului, lungimea în octeţi, numărul de blocuri ale fişierului, data ultimului acces la fişier, etc. – vezi exemplul de mai jos). Sistemele BSD şi Unix System V release 4 au şi apelul `fstat` care face acelaşi lucru ca şi `stat`, doar că fişerul este specificat printr-un descriptor de fişier deschis, nu prin nume.
- `mknod` creează un fişier simplu sau un fişier special desemnat prin `nume`. Parametrul `mod` specifică printr-o combinaţie de constante simbolice legate prin simbolul '|' atât drepturile de acces la fişierul nou creat, cât şi tipul fişierului care poate fi unul dintre următoarele:
  - S_IFREG (fişier normal)
  - S_IFCHR (fişier special de tip caracter)
  - S_IFBLK (fişier special de tip bloc)
  - S_IFIFO (*pipe* cu nume sau FIFO – vezi capitolul 5)
  - S_IFSOCK (Unix domain socket – folosit pentru comunicarea locală între procese)

Dacă tipul fişierului este S_IFCHR sau S_IFBLK, atunci `dev` conţine numărul minor şi numărul major al fişierului special nou creat; altfel, acest parametru se ignoră.

- `chown` schimbă proprietarul şi grupul din care face parte proprietarul unui fişier specificat prin `nume`. Noul proprietar al fişierului va fi cel indicat de parametrul `proprietar`, iar noul grup al fişierului va fi cel indicat de parametrul `grup`. Pe sistemele BSD şi Unix System V release 4 mai este prezent şi apelul `fchown` care face acelaşi lucru ca şi `chown`, doar că fişierul este specificat printr-un descriptor de fişier deschis, nu prin nume.
- `access` verifică dacă procesul curent are dreptul specificat de `permisiuni` relativ la fişierul specificat prin `nume`. `permisiuni` va conţine una sau mai multe valori legate prin '|' dintre următoarele: R_OK - citire, W_OK - scriere, X_OK – execuţie şi F_OK – existenţă fişier. Verificarea se face cu UID-ul şi GID-ul reale ale procesului, nu cele efective (vezi capitolul 5).
- `rename` redenumeşte fişierul specificat de `numevechi` în `numenou`.

4.2.2 Creat, truncate, readdir

Următoarele trei apeluri nu au corespondent direct printre comenzile Shell:

```
int creat(const char *pathname, unsigned int mod);
int truncate(const char *nume, long int lung);
#include <dirent.h> - - - struct dirent *readdir(DIR *dir);
```

Apelul sistem `creat` este echivalent cu următorul apel sistem:

```
open (const char *nume, O_CREAT|O_WRONLY|O_TRUNC);
```

Apelul sistem `truncate` trunchiază fişierul specificat prin `nume` la exact `lung` octeţi.

Funcţia `readdir` nu este o funcţie sistem, ci este funcţia POSIX pentru parcurgerea subdirectoarelor şi fişierelor dintr-un director. Ea întoarce într-o structură de tipul `dirent` următorul subdirector sau fişier din directorul dat de parametrul `dir`. Această funcţie încapsulează de fapt apelul sistem `getdents`. Alte funcţii în legătură cu `readdir` şi specificate de standardul POSIX sunt: `opendir`, `closedir`, `rewinddir`, `scandir`, `seekdir` şi `telldir`. Nu intrăm în detalierea acestor funcţii deoarece ele nu sunt funcţii sistem.

Majoritatea apelurilor de mai sus întorc valoarea 0 în caz de succes şi -1 (plus setarea corespunzătoare a variabilei `errno`) în caz de eroare. Excepţie de la această regulă fac apelurile: `getcwd` care returnează NULL în caz de eroare şi setează corespunzător `errno` şi `readdir` care întoarce NULL în caz de eroare.

4.3 Exemple de programe de lucru cu fisiere

4.3.1 Un exemplu de lucru cu fisiere text: numararea propozitiilor.

La linia de comanda se da numele unui fisier text. Se cere sa se numere cate propozitii sunt in acest fisier. Solutia este una simplista: presupunand ca orice propozitie se termina fie cu punct, fie cu semn de exclamare fie cu semn de intrebare, vom numara, pur si simplu, cate astfel de caractere apar in fisier.

Solutia:

```
/*
```

```
Numara propozitiile dintr-un fisier text (numarand . ! ? )
*/
#include <stdio.h>
#include <string.h>
main (int argc, char* argv[]) {
    char linie[1000], *p, *q;
    int propozitii = 0;
    FILE* f = fopen(argv[1], "r");
    for (;; ) {
        p = fgets(linie, 1000, f);
        if (p == NULL) break;
        for (p = linie, q = NULL;; ) {
            q = strchr(p, '.');
            if (q == NULL) break;
            propozitii++;
            p = q + 1;
        }
        for (p = linie, q = NULL;; ) {
            q = strchr(p, '!');
            if (q == NULL) break;
            propozitii++;
            p = q + 1;
        }
        for (p = linie, q = NULL;; ) {
            q = strchr(p, '.');
            if (q == NULL) break;
            propozitii++;
            p = q + 1;
        }
    }
    fclose(f);
    printf("In \"%s\" sunt %d propozitii\n", argv[1], propozitii);
}
```

4.3.2 Copierea unui fişier

Ca exemplu de folosire a apelurilor sistem `open, close, read` şi `write` vom scrie un program care face acelaşi lucru ca şi comanda shell `cp`, adică copiază conţinutul unui fişier într-altul. Cele două fişiere se vor da ca parametrii în linia de comandă. Dacă al doilea fişier (fişierul destinaţie) nu există, el se va creea, iar dacă există deja, el se va suprascrie. Sursa programului este:

```
/*
Copiaza un fisier in altul:
copy sursa destinatie
 */
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

main(int argc, char* argv[]) {
    int fd_sursa, fd_dest, n, i;
    char buf[100], *p;
    if (argc!=3) {
        fprintf(stderr, "Eroare: trebuie dati 2 parametrii.\n");
        exit(1);
```

```
    }
    //deschidem primul fisier în modul read-only
    fd_sursa = open(argv[1], O_RDONLY);
    if (fd_sursa<0) {
        fprintf(stderr, "Eroare: %s nu poate fi deschis.\n", argv[1]);
        exit(1);
    }
    /* Deschidem al doilea fisier în modul write-only. Daca el nu
     * exista, se va creea, sau daca exista va fi trunchiat. 0755
     * specifica drepturile de acces ale fisierului nou creat
     * (citire+scriere+executie pentru proprietar, citire+executie
     * pentru grup si altii).
     */
    fd_dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0755);
    if (fd_dest<0) {
        fprintf(stderr, "Eroare: fisierul %s nu poate fi deschis.\n",
                argv[2]);
        exit(1);
    }
//citim din fisierul fd_sursa bucati de maxim 100 octeti si le
//scriem în fisierul fd_dest, pâna când nu mai avem ce citi.
    for ( ; ; ) {
        n=read(fd_sursa, buf, sizeof(buf));
        if (n == 0) break; // S-a terminat de citit fisierul
        p = buf;
        for(; n > 0; ) { // Poate nu se poat scrie odata n octeti
            i = write(fd_dest, p, n);
            if (i == n) break;
            p += i;
            n -= i;
        }
    }
    //închidem cele doua fisiere
    close(fd_sursa);
    close(fd_dest);
}
```

4.3.3 Exemplu de lucru cu fisiere binare: oglindirea unui fisier.

La linia de comanda se da un nume de fisier. Se cere sa se realizeze oglindirea acestui fisier - primul octet al fisierului se schimba cu ultimul, al doilea cu penultimul s.a.m.d pana se ajunge la jumatatea fisierului.

Prezentam doua variante de rezolvare si invitam studentii sa le testeze pe ambele si sa observe diferentele intre codurile C si intre vitezele de executie.

Este de asemenea util sa se retina din solutia a doua functiile **Read** si **Write**, care s-ar putea dovedi utile si in alte situatii.

Solutia1:

```
/*
 Oglindeste continutul unui fisier binar dat la linia de comanda.
 Oglindirea se realizeaza citind caracter cu caracter.
 A se confrunta cu executia programului similar oglindan.c
 */
#include <stdio.h>
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
main(int argc, char *argv[]) {
    int f;
    long ps, pd, m;
    char os, od;
    struct stat stare;
    time_t start;
    start = time(NULL);
    stat(argv[1], &stare);
    f = open(argv[1], O_RDWR);
    m = stare.st_size / 2;
    for (ps=0, pd=stare.st_size-1; ps+1 <= m; ps++, pd--) {
        lseek(f, ps, SEEK_SET);
        read(f, &os, 1);
        lseek(f, pd, SEEK_SET);
        read(f, &od, 1);
        lseek(f, ps, SEEK_SET);
        write(f, &od, 1);
        lseek(f, pd, SEEK_SET);
        write(f, &os, 1);
    }
    close(f);
    printf("Durata: %d\n",(int)(time(NULL)-start));
}
```

Solutia2:

```
/*
 Oglindeste continutul unui fisier binar dat la linia de comanda.
 Oglindirea se realizeaza citind blocuri de octeti consecutivi.
 A se confrunta cu executia programului similar oglinda1.c
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#define MAX 10000
void oglinda(char sir[MAX], int n) {
    int ps, pd;
    char t;
    for (ps = 0, pd = n - 1; ps < pd; ps++, pd--) {
        t = sir[ps];
        sir[ps] = sir[pd];
        sir[pd] = t;
    }
}
void Read(int f, char *t, int n) {
    char *p;
    int i, c;
    for (p = t, c = n;;) {
        i = read(f, p, c);
        if (i == c)
            return;
        c -= i;
```

```
        p += i;
    }
}
void Write(int f, char *t, int n) {
    char *p;
    int i, c;
    for (p = t, c = n; c;) {
        i = write(f, p, c);
        if (i == c)
            return;
        c -= i;
        p += i;
    }
}
main(int argc, char *argv[]) {
    int f, m, n;
    long ps, pd;
    char os[MAX], od[MAX];
    struct stat stare;
    time_t start;
    start = time(NULL);
    stat(argv[1], &stare);
    n = MAX;
    m = stare.st_size / 2;
    if (n > m) n = m;
    f = open(argv[1], O_RDWR);
    for (ps=0, pd=stare.st_size-n; ps+n <= m; ps+=n, pd-=n) {
        if (m - ps < n) n = m - ps;
        lseek(f, ps, SEEK_SET);
        Read(f, os, n);
        lseek(f, pd, SEEK_SET);
        Read(f, od, n);
        oglinda(os, n);
        oglinda(od, n);
        lseek(f, ps, SEEK_SET);
        Write(f, od, n);
        lseek(f, pd, SEEK_SET);
        Write(f, os, n);
    }
    close(f);
    printf("Durata: %d\n",(int)(time(NULL)-start));
}
```

4.3.4 Obţinerea tipului de fişier prin apelul sistem stat

In această secţiune vom da un exemplu de utilizare a lui `stat`. Exemplul se referă la afişarea tipului de fişier pentru fişierele ale căror nume sunt date ca argumente la linia de comandă.

Programul `tipfis.c` exemplifică folosirea apelului sistem `stat` pentru determinarea tipului de fişier recunoscut de către sistem. Tipurile de fişiere le-am prezentat într-o secţiune precedentă.

```
/*
 * Tipareste tipurile fisierelor date in linia de comanda
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <string.h>
main (int argc, char *argv[]) {
    int i;
    struct stat statbuff;
    char tip[40];
    strcpy(tip, "");
    for (i = 1; i < argc; i++) {
        printf ("%s: ", argv[i]);
        if (stat (argv[i], &statbuff) < 0)
        fprintf (stderr, "Eroare stat");
        switch (statbuff.st_mode & S_IFMT) {
            case S_IFDIR:
            strcpy(tip, "Director");
            break;
            case S_IFCHR:
            strcpy(tip, "Special de tip caracter");
            break;
            case S_IFBLK:
            strcpy(tip, "Special de tip bloc");
            break;
            case S_IFREG:
            strcpy(tip, "Obisnuit");
            break;
            case S_IFLNK:
            /* acest test nu va fi adevarat niciodata deoarece stat
             * verifica în cazul unei legaturi simbolice, fisierul
             * pe care îl refera legatura si nu legatura în sine. Il
             * scriem aici pentru completitudine. Ca sa verificam
             * tipul unei legaturi simbolice putem folosi functia
             * lstat asemanatoare cu stat si disponibila pe
             * versiunile BSD si Unix System V.
             */
            strcpy(tip, "Legatura simbolica");
            break;
            case S_IFSOCK:
            strcpy(tip, "Socket");
            break;
            case S_IFIFO:
            strcpy(tip, "FIFO");
            break;
            default:
            strcpy(tip, "Tip necunoscut");
        } //switch
        printf ("%s\n", tip);
    } //for
} //main
```

5 Pointeri; alocare dinamica

5.1 Pointeri

5.1.1 Aritmetica de pointeri

In C sunt permise o serie de operatii aritmetice cu pointeri, astfel:

Să considerăm **p** un pointer la un anumit tip de dată **T** şi **i** un întreg.

Expresiile **p + i** şi **p - i** (i poate fi pozitiv sau negativ) au ca rezultat tot un pointer cu valoarea mai mare sau mai mică cu **i * sizeof(T)** decât **p**. De exemplu daca p este de tip intreg reprezentat pe 4 octeti, atunci p + 3 indica o adresa cu 12 (3 locatii a cate 4 octeti) octeti mai mare decat adresa p.

O expresie **p1 - p2**, unde **p1** si **p2** sunt pointeri de un anumit tip **T** are ca rezultat un intreg **i** care indica cate locatii cu variabile de tip **T** pot fi plasate intre adresele **p1** si **p2**.
Avem deci relaţia adresa din p2 este adresa cu valoarea **p1 + (p1 - p2) * sizeof(T)**. De exemplu, daca p1 si p2 sunt pointeri de tip double ce se reprezinta pe 8 octeti, iar p2 - p1 are valoarea 3, atunci adresa p2 este cu 24 octeti mai mare decat adresa p1.

Utilizatorul trebuie sa gestioneze pointerii fata de tipul lor, NU trebuie sa tina cont de lungimea de reprezentare a tipului.

5.1.2 Echivalenta intre tablouri si pointeri

Fie T un tip de date si sa consideram declaratiile si secvenţa de instructiuni:

```
T t[...], *p;
 . . . Initializarea tabloului t . . .
p = t;
```

Elementele tabloului `t`:	`t[0]`
	`p[0]`
	`*p`
	`*t`
		`t[1]`
		`p[1]`
		`*(p+1)`
		`*(t+1)`
			`t[2]`
			`p[2]`
			`*(p+2)`
			`*(t+2)`
				`t[3]`
				`p[3]`
				`*(p+3)`
				`*(t+3)`
					...

Adresele elementelor tabloului `t`:	`&t[0]`
	`&p[0]`
	`p`
	`t`
		`&t[1]`
		`&p[1]`
		`p+1`
		`t+1`
			`&t[2]`
			`&p[2]`
			`p+2`
			`t+2`
				`&t[3]`
				`&p[3]`
				`p+3`
				`t+3`
					...

Această echivalenţă este cunoscută sub numele de **echivalenţa dintre pointeri şi tablouri**

Puteti testa folosind de exemplu programul:

```
#include <stdio.h>
main () {
    long t[10], *p;
    int i;
    for (i=0; i<10; t[i++]=i);
    p = t;
    for (i=0; i<10; i++)
        printf("%d %d %d %d\n",t[i],p[i],*(p+i),*(t+i));
}
```

5.2 Alocare dinamică

C permite folosirea spaţiului heap pentru a aloca variabile dinamice şi de a le elibera. Sunt în esenţă două funcţii destinate acestui scop, specificate în headerul **<stdlib.h>**.

5.2.1 Funcţia malloc

Prototipul ei este:

```
void * malloc (size_t nOcteti);
```

Funcţia alocă `nOcteti` in zona heap şi întoarce un pointer la începutul acestei zone, sau NULL dacă alocarea nu este posibilă. Evident, utilizatorul trebuie să convertească pointerul întors la tipul de date dorit. De exemplu, dacă avem un tip de date STUDENT şi dorim să rezervăm o variabilă dinamică cu 100 de locuri pentru date de acest tip, atunci se poate folosi:

```
#include <stdlib.h>
 . . .
STUDENT *p;
 . . .
p = (STUDENT*)malloc(100 * sizeof(STUDENT));
 . . . se pot folosi *(p+5) . . . p[5]  . . . (conform celor spuse mai sus) etc.
```

5.2.2 Funcţia free

Prototipul ei este:

```
void free(void *p);
```

Eliberează spaţiul din heap către care punctează `p`, iniţializat anterior prin `malloc`.

5.3 Exemple de utilizare a pointerilor si variabilelor dinamice

5.3.1 Utilizarea de liste înlănţuite

Vom rezolva urmatoarea problema: Se citesc de la intrarea standard un sir de linii. Dupa citirea fiecarei linii, aceasta se insereaza intr-o lista simplu inlantuita, in pozitia care sa respecte ordinea alfabetica (sortare prin insertie). Dupa terminarea citirilor lista se va tipari. Apoi se vor sterge din ea primul element, ultimul si cel din mijloc, dupa care se va tipari din nou. La sfarsit se vor elibera toate spatiile dinamice ocupate de lista.

Sursa C completa este urmatoarea:

```
/*
Se citesc de la intrarea standard un sir de linii.
Dupa citirea fiecarei linii, aceasta se insereaza
intr-o lista simplu inlantuita, in pozitia care sa
respecte ordinea alfabetica (sortare prin insertie).
Dupa terminarea citirilor lista se va tipari.
Apoi se vor sterge din ea primul element, ultimul
si cel din mijloc, dupa care se va tipari din nou.
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct node {char *sir; struct node *next;} NODE;
NODE *cap;
```

```
void add(char *linie) {
    char *pc;
    NODE *nod, *pn, *qn;
    pc = (char*)malloc(strlen(linie) + 1);
    strcpy(pc, linie);
    nod = (NODE*)malloc(sizeof(NODE));
    nod->sir = pc;
    for (pn = cap, qn = NULL; pn != NULL; qn = pn, pn = pn->next) {
        if (strcmp(nod->sir, pn->sir) <= 0) break;
    }
    if (qn == NULL) cap = nod; else qn->next = nod;
    nod->next = pn;
}
void del(int n) {
    int i;
    char *pc;
    NODE *pn, *qn;
    if (cap == NULL) return;
    for (i = 0, pn = cap, qn = NULL; i < n; i++, qn = pn, pn = pn->next);
    if (pn == NULL) return;
    if (qn == NULL) cap = pn->next; else qn->next = pn->next;
    free(pn->sir);
    free(pn);
}
main () {
    char *pc, linie[100];
    NODE *pn;
    int n = 0;
    cap = NULL;
    for ( ; ; ) {
        pc = fgets(linie, 100, stdin);
        if (pc == NULL) break;
        add (linie);
        n++;
    }
    for (pn = cap; pn != NULL; pn = pn->next) printf("%s",pn->sir);
    del(n);
    del(n / 2);
    del(1);
    for (pn = cap; pn != NULL; pn = pn->next) printf("%s",pn->sir);
    for (pn = cap; pn != NULL; pn = pn->next) {
        free(pn->sir);
        free(pn);
    }
}
```

5.3.2 Tablouri statice transmise ca parametri

```
/*
Acest program foloseste o matrice de (m,n) alocata static.
Exemplificam transmiterea matricelor (tablourilor multidimensionale) statice
ca parametri de functii. Testati astfel functionarea corecta a acestui
tip de matrice transmise ca parametri.
*/
#include <stdlib.h>
#include <stdio.h>
#define    m    4
#define    n    7
main() {
```

```
    int t[m][n], i, j;
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            t[i][j] = 10*j+i;
    for(i=0;i<m;i++) {
        printf("\n");
        for (j=0; j<n;j++)
            printf("%02d ",t[i][j]);
    }
    printf("\n");
    f(t);
    printf("\n");
    g(t);
    printf("\n");
    h(t);
    printf("\n");
    l(t);
    printf("\n");
}
f(int t[m][n]) {    // Incercati si cu 20 X 3
    int i,j;
    for(i=0;i<m;i++) {
        printf("\n");
        for (j=0; j<n;j++)
            printf("%02d ",t[i][j]);
    }
}
// Prima dimensiune statica poate sa lipseasca, celelalte trebuie puse
g(int t[][n]) {
    int i,j;
    for(i=0; i<m; i++) {
        printf("\n");
        for(j=0; j<n; j++)
            printf("%02d ",t[i][j]);
    }
}
// Alocarea se face in m*n locatii succesive, linie dupa linie.
h(int *t) {
    int i, j;
    for(i=0; i<m; i++) {
        printf("\n");
        for(j=0; j<n; j++)
            printf("%02d ",*(t+n*i+j));
    }
}
// Daca credeti !? ca ordinea este coloana dupa coloana . . . iata:
l(int *t) {
    int i,j;
    for(i=0;i<m;i++) {
        printf("\n");
        for(j=0;j<n;j++)
            printf("%02d ",*(t+m*j+i));
    }
}
```

5.3.3 Alocare dinamică a unei matrice de m X n

Să presupunem că dorim să alocăm un tablou bidimensional cu m linii şi n coloane. Pentru aceasta, alocăm mai întâi un tablou de m pointeri la tipul T care vor puncta către începuturile de linii. Apoi, vom aloca câte m tablouri cu câte n elemente de tipul T, iar începuturile acestor tablouri vor fi atribuite tablouui de m pointeri ce indică începuturilor de linii. Schema de alocare aduce cu un "ciorchine de struguri": **Pentru situaţiile în care se impun transmiteri de tablouri ca şi parametri, recomandăm călduros utilizarea tablourilor dinamice în locul celor statice.**

Acest mecanism permite definirea de tablouri dinamice cu mai multe dimensiuni. Pentru trei dimensiuni vom folosi ***t, pentru patru dimensiuni ****t etc.

De asemenea se pot crea tablouri care nu sunt neaparat rectangulare (de exemplu cu linii mai lungi sau mai scurte) etc. Evident, utilizatorul este obligat să gestioneze şi să transmită dimensiunile acestor tablouri.

```
// Alocam dinamic un tablou bidimensional de m X n si accesam elemente ale lui
#include <stdlib.h>
#include <stdio.h>
int m, n;
main() {
    int **t, i, j;
    // . . . calculul dimensiunilor m si n . . .
    m = 10;
    n = 7;
    t = (int**)malloc(m * sizeof(int*));
    for (i=0; i<m; i++)
        t[i] = (int*)malloc(n * sizeof(int));
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            t[i][j] = 10*i+j;
    for(i=0;i<m;i++) {
        printf("\n");
        for (j=0; j<n;j++)
            printf("%2d ",t[i][j]);
    }
    printf("\n");
    f(t);
    for(i=0;i<m;i++) {
        printf("\n");
        for (j=0; j<n;j++)
            printf("%2d ",t[i][j]);
    }
}
f (int **t) {
    t[m / 2][ n / 2] = 0;
}
```

5.3.4 Tablou nedreptunghiular de intregi cu retinerea dimensiunilor

```
// Matrice nedreptunghiulare alocate dinamic cu retinerea dimensiunilor
// Vor fi n linii, iar fiecare linie va avea un numar diferit de coloane
#include <stdlib.h>
#include <stdio.h>
main() {
    int **t, *l, i, j, n;
    n = 6+rand()%4;
    l = (int*)malloc(n * sizeof(int));
    t = (int**)malloc(n * sizeof(int*));
```

```
    for (i=0; i<n; i++) {
        l[i] = 5+rand()%4;
        t[i] = (int*)malloc(l[i] * sizeof(int));
    }
    for (i=0; i<n; i++)
        for (j=0; j<l[i]; j++)
            t[i][j] = 10*i+j;
    for(i=0;i<n;i++) {
        printf("\n");
        for (j=0; j<l[i];j++)
            printf("%2d ",t[i][j]);
    }
    printf("\n");
    f(t, n, l);
    for(i=0;i<n;i++) {
        printf("\n");
        for (j=0; j<l[i];j++)
            printf("%2d ",t[i][j]);
    }
}
f (int **t, int n, int *l) {
    t[n-1][l[n-1]-1] = 0;
}
```

5.3.5 Alocare dinamica a spatiului pentru tablouri bidimensionale de caractere

Vom rezolva urmatoarea problema. Se genereaza un numar aleator m mai mic decat o valoare MAX. Se genereaza apoi aleator inca m numere aleatoare, n1, n2, ... ,nm. Plecand de la aceste numere, se aloca dinamic o matrice matc, cu m linii, iar fiecare linie i va avea ni coloane. Elementele matc vor fi caractere.

Se incarca aleator toata matricea cu litere mari dupa care se tipareste. In final se elibereaza intreg spatiul dinamic ocupat.

Sursa C completa este urmatoarea:

```
/*
Se genereaza un numar aleator m mai mic decat o valoare MAX. Se genereaza apoi
aleator inca m numere aleatoare, n1, n2, ... ,nm. Plecand de la aceste numere,
se aloca dinamic o matrice matc, cu m linii, iar fiecare linie i va avea ni
coloane
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX 100
main () {
    char **matc;
    int m, i, j, *n;
    m = rand() % MAX;
    n = (int*) malloc(m * sizeof(int));
    for (i = 0; i < m; i++) n[i] = rand() % MAX;
    matc = (char **) malloc(m * sizeof(char*));
    for (i = 0; i < m; i++) {
        matc[i] = (char*) malloc(n[i] * sizeof(char));
    }
    for (i = 0; i < m; i++)
    for (j = 0; j < n[i]; j++)
```

```
    matc[i][j] = 'A' + rand() % 26;
    for (i = 0; i < m; i++) {
        for (j = 0; j < n[i]; j++) printf("%c", matc[i][j]);
        printf("\n");
    }
    for (i = 0; i < m; i++) free(matc[i]);
    free(matc);
    free(n);
}
```

5.3.6 Tablouri de pointeri la stringuri

```
// Alocam dinamic un vector de stringuri
// Se citeste un intreg n urmat de citirea a n linii.
// Se construie un vector dinamic de n elemente cu aceste stringuri
// Se tipareste elementul din mijloc.
// Se elibereaza toate spatiile
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
main() {
    char **t, *p, linie[1000];
    int i, n;
    fgets(linie, 1000, stdin);
    n = atoi(linie);
    t = (char**)malloc(n * sizeof(char*));
    for (i=0; i<n; i++) {
        fgets(linie, 1000, stdin);
        p = (char *)malloc(strlen(linie) + 1);
        strcpy(p, linie);
        t[i] = p;
    }
    printf("%s\n", t[n / 2]);
    for (i = 0; i < n; i++)
        free(t[i]);
    free(t); // Atentie la ordinea de eliberare!
}
```

Evident, metoda descrisa de mai sus poate fi extinsa cu tablouri de stringuri mai complicate, ca in exemplele de mai sus.