0. Limbajul C-ansi: pointeri, tablouri, IO

Contents

0.1. Pointeri

0.1.1. Aritmetica de pointeri, echivalenţa pointeri - tablouri

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

Să considerăm **p** un pointer la un anumit tip de dată **T** (declarat `T ---*p---`) 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 **int** 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 întreg **i** care indică câte locatii cu variabile de tip **T** pot fi plasate intre adresele **p1** si **p2**. 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. Fie **T** un tip de date. Să considerăm declaraţiile şi secvenţa de instrucţiuni:

```
T t[...], *p; // Declararea tabloului t si a pointerului p
  . . . Initializarea tabloului t . . .
p = t; // p are aceeasi valoare ca si t, inceputul de tablou.
```

Tabelul de mai jos indică câte patru specificări echivalente de elemente ale tabloului sau adrese ale acestora:

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));
}
```

0.1.2. Citirea unor linii si ordonarea lor alfabetică, varianta 1

Ne propunem să rezolvăm următoarea problemă: Se citeste de la intrarea standard un număr întreg **n**, urmat de citirea citirea a maximum **n** linii. Aceste linii sunt depuse într-un vector cu elemente (pointeri la) stringuri având **n** elemente. După terminarea citirilor, tabloul de linii se ordonează alfabetic, se tipăreşte tabloul ordonat şi se eliberează spaţiile alocate dinamic. Sursa programului este:

```
// Se citeste un intreg n urmat de citirea a maximum n linii.
// Se construie un vector alocat dinamic de n elemente cu aceste stringuri
// Se ordoneaza alfabetic liniile citite si se tiparesc.
// Se elibereaza toate spatiile alocate dinamic
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main() {
    char **t, *p, linie[1000];
    int i, j, n, nef;
    printf("n = ?"); fgets(linie, 1000, stdin);
    n = atoi(linie);
    t = (char**)malloc(n * sizeof(char*));
    for (i = 0; i < n; i++) {
        printf("Linie %i = ?", i);
        if (fgets(linie, 1000, stdin) == NULL) break;
        p = (char *)malloc(strlen(linie) + 1);
        strcpy(p, linie);
        t[i] = p;
    }
    nef = i;
    for (i = 0; i < nef - 1; i++)
        for (j = i + 1; j < nef; j++)
            if (strcmp(t[i], t[j]) > 0) {
                p = t[i];
                t[i] = t[j];
```

```
                t[j] = p;
            }
    printf("\n");
    for (i = 0; i < nef; i++) printf("%s",t[i]);
    for (i = 0; i < nef; i++) free(t[i]);
    free(t); // Atentie la ordinea de eliberare!
    return 0;
}
```

0.1.3. Citirea unor linii si ordonarea lor alfabetică, varianta 2

Este vorba de (aproape) aceeaşi problemă ca cea de mai sus, cu deosebirea că nu se limitează numărul de linii ce se vor citi. Rezolvarea, de această dată, se face memorând liniile într-o listă simplu înlănţuită. După citirea fiecărei linii, se parcurge lista liniilor deja citite, iar linia curentă se inserează în locul în care liniile dinaintea ei sunt mai mici în ordine alfabetică, iar cele de după mai mari sau egale. In acest fel, la terminarea citirilor liniile sunt deja ordonate alfabetic. Sursa programului este:

```
// 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 elibera toate spatiile ocupate dinamic.
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct node {char *linie; struct node *next;} NODE;
NODE *cap;
void add(char *linie) {
    NODE *nod, *pn, *qn;
    nod = (NODE*)malloc(sizeof(NODE));
    nod->linie = (char*)malloc(strlen(linie) + 1);
    strcpy(nod->linie, linie);
    for (pn = cap, qn = NULL; pn != NULL; qn = pn, pn = pn->next)
        if (strcmp(nod->linie, pn->linie) <= 0) break;
    if (qn == NULL) cap = nod; else qn->next = nod;
    nod->next = pn;
}
int main () {
    char linie[1000];
    NODE *pn;
    cap = NULL;
    for ( ; ; ) {
        printf("Linie = ?");
        if (fgets(linie, 1000, stdin) == NULL) break;
        add (linie);
    }
    printf("\n");
    for (pn = cap; pn != NULL; pn = pn->next) printf("%s",pn->linie);
    for (pn = cap; pn != NULL; pn = pn->next) {
        free(pn->linie);
        free(pn);
    }
    return 0;
}
```

0.1.4. Citirea unor linii si ordonarea lor alfabetică, varianta Go

Solutia Go nu cere limitari ale lungimilor liniilor, nici limitari ale numarului de linii.

```
// Se citesc de la stdin linii, care se depun in slice-ul linii.
```

```
// Dupa terminarea citirilor, se sorteaza linii si se scrie pe stdout.
package main
import (
    "bufio"
    "fmt"
    "io"
    "os"
    "sort"
    "strings"
)
func main() {
    linii := []string{}
    in := bufio.NewReader(os.Stdin)
    for {
        linie, err := in.ReadString('\n')
        if err == io.EOF { break }
        linie = strings.Replace(linie, "\n", "", -1)   // Efectiv Unix
        linie = strings.Replace(linie, "\r\n", "", -1) // Efectiv Windows
        linii = append(linii, linie)
    }
    sort.Strings(linii)
    for i := 0; i < len(linii); i++ { fmt.Println(linii[i]) }
}
```

0.1.5. Citirea unor linii si ordonarea lor alfabetică, varianta Python

Solutia Python nu cere limitari ale lungimilor liniilor, nici limitari ale numarului de linii.

```
# Se citesc de la stdin linii, care se depun in lista linii.
# Dupa terminarea citirilor, se sorteaza linii si se scrie pe stdout.
import fileinput
def main():
    linii = []
    for linie in fileinput.input():
        linie = linie.replace("\n", "")   # Efectiv Unix
        linie = linie.replace("\r\n", "") # Efectiv Windows
        linii.append(linie)
    linii.sort()
    print (linii)
    for linie in linii: print(linie)
main()
```

0.2. Tablouri C alocare statică şi dinamică

0.2.1. Tablouri bidimensionale statice

Tablourile cu elemente de tip **T** bidimensionale în c sunt declarate sub forma **`T t[m][n] - - -`**, unde **T** este tipul elementelor de tablou, **t** este numele variabilei tablou, iar **m** şi **n** sunt constante întregi. Constanta **m** indică numărul de linii, iar **n** este numărul de coloane. In memorie, începând cu adresa **t** sunt rezervate **m*n** locaţii consecutive de tip **T**, în care elementele sunt reprezentate linie după linie.

Principalele dificultăţi ale acestui mod de reprezentare sunt:

- Obligaţia de a preciza dimensiunile **m** şi **n** cu valori maximale, deşi dimensiunile reale pot rezulta din calcule şi sunt mai mici decât dimensiunile **m** şi **n**.
- Transmiterea unui astfel de tablou ca şi parametru este o sursă puternică de erori dacă se dau dimensiunile reale în locul celor maximale alocate.

Exemplul următor defineşte o funcţie **list** care primeşte un tablou de întregi şi listează matricea. In program se alocă un tablou cu **3** linii şi **5** coloane. Programul apelează această funcţii pentru mai multe dimensiuni reale. In comentarii se văd efectele acestor apelări. Programul este:

```
#include <stdio.h>
void list(int m, int n, int a[m][n]) {
    // Ok daca m <= cu cel alocat (3) si n == cu cel alocat (5)
    // In caz contrar, sunt mari sanse sa interpreteze ca int o locatie aiurea.
    printf("\n");
    for (int i=0; i<m; i++) {
        for (int j=0; j<n; j++) printf(" %02d", a[i][j]);
        printf("\n");
    }
}
int main() {
    int a[3][5] = {{00,01,02,03,04},{10,11,12,13,14},{20,21,22,23,24}};
    list(3, 5, a); // 00 01 02 03 04/10 11 12 13 14/20 21 22 23 24
    list(2, 5, a); // 00 01 02 03 04/10 11 12 13 14
    list(1, 5, a); // 00 01 02 03 04
    list(3, 4, a); // 00 01 02 03/04 10 11 12/13 14 20 21
    list(3, 3, a); // 00 01 02/03 04 10/11 12 13
    list(3, 2, a); // 00 01/02 03/04 10
    list(3, 1, a); // 00/01/02
    list(2, 3, a); // 00 01 02/03 04 10
    list(1, 4, a); // 00 01 02 03
    list(4, 5, a); // 00 01 02 03 04/10 11 12 13 14/20 21 22 23 24/ ? ? ? ? ?
    list(3, 6, a); // 00 01 02 03 04 10/11 12 13 14 20 21/22 23 24 ? ? ?
    list(4, 4, a); // 00 01 02 03/04 10 11 12/13 14 20 21/22 23 24 ?
    list(15, 1, a); // 00/01/02/03/04/10/11/12/13/14/20/21/22/23/24
    list(1, 15, a); // 00 01 02 03 04 10 11 12 13 14 20 21 22 23 24
    list(1, 20, a); // 00 01 02 03 04 10 11 12 13 14 20 21 22 23 24 ? ? ? ? ?
   return 0;
}
```

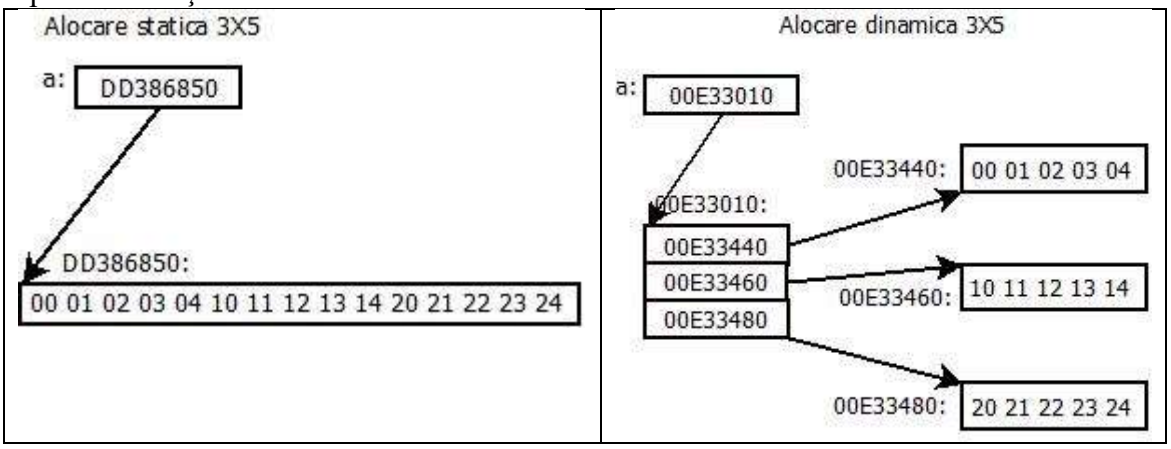
0.2.2. Tablouri bidimensionale dinamice

In baza echivalenţei dintre tablouri şi pointeri, se pot aloca tablouri dinamice! Sursa următoare alocă dinamic un tablou de 3X5 ca în exemplul precedent. Programul este:

```
#include <stdio.h>
#include <stdlib.h>
void list(int m, int n, int **a) {
    // Ok daca n <= cu cel alocat (5) si m <= cu cel alocat (3)
    // In caz contrar, sunt mari sanse de Segmentation fault (core dumped),
    // deoarece se fac adresari prin pointeri inexistenti.
    // Uneori este posibil sa interpreteze ca int o locatie aiurea.
    printf("\n");
    for (int i=0; i<m; i++) {
        for (int j=0; j<n; j++) printf(" %02d", a[i][j]);
        printf("\n");
    }
}
int main() {
    int **a;
    a = (int**)malloc(3*sizeof(int*));
    printf("%d %d %d\n",a, sizeof(int**), sizeof(int));
    for (int i = 0; i < 3; i++) {
        a[i] = (int*)malloc(5*sizeof(int));
        printf("%d\n",a[i]);
        for (int j = 0; j < 5; j++) a[i][j] = 10*i + j;
    }
    //int a[3][5] = {{00,01,02,03,04},{10,11,12,13,14},{20,21,22,23,24}};
    list(3, 5, a); // 00 01 02 03 04/10 11 12 13 14/20 21 22 23 24
    list(2, 5, a); // 00 01 02 03 04/10 11 12 13 14
```

```
    list(1, 5, a); // 00 01 02 03 04
    list(3, 4, a); // 00 01 02 03/04 10 11 12/13 14 20 21
    list(3, 3, a); // 00 01 02/10 11 12/20 21 22
    list(3, 2, a); // 00 01/10 11/20 21
    list(3, 1, a); // 00/10/20
    list(2, 3, a); // 00 01 02/10 11 12
    list(1, 4, a); // 00 01 02 03
    list(4, 5, a); // Segmentation fault sau numere aiurea din zona nealocata
    list(3, 6, a); // Segmentation fault sau numere aiurea din zona nealocata
    list(4, 4, a); // Segmentation fault sau numere aiurea din zona nealocata
    list(15, 1, a); // Segmentation fault sau numere aiurea din zona nealocata
    list(1, 15, a); // Segmentation fault sau numere aiurea din zona nealocata
    list(1, 20, a); // Segmentation fault sau numere aiurea din zona nealocata
    return 0;
}
```

In primele 8 linii din funcţia main este ilustrat modul în care se poate aloca dinamic o matrice de 3 linii şi 5 coloane. Evident, în funcţiile **malloc**, în locul constantelor 3 şi 5 pot apărea orice variabile sau expresii întregi! Următoarele linii din main apelează funcţia **list** la fel ca la exemplul precedent. Comentariile arată efectele apelurilor. Se vede că în acest caz se pot folosi corect şi submatrice ale celei alocate, preluând mai puţine prime linii şi / sau prime coloane. Figurile următoare prezintă alocarea statică, respectiv dinamică pentru acelaşi tablou.



0.2.3. Generalizări: tablouri dinamice nerectangulare şi / sau multidimensionale

Alocarea dinamică a tablourilor bidimensionale poate fi generalizată uşor. Iată câteva posibile generalizări:

- Tablouri bidimensionale nerectangulare (nu toate liniile au acelaşi număr de coloane).
- Tablouri tridimensionale, quatrodimensionale, multidimensionale etc.
- Tablouri multidimensionale nerectangulare.
- etc.

Pentru tablourile bidimensionale nerectangulare de dimensiune **mXn** există două rezolvări simple:

1. Se alocă în plus un tablou de dimensiune **m** care să reţină lungimea fiecărei linii.
2. **Folosirea ca terminatori a pointerilor NULL**. Această rezolvare este foarte generală şi se pretează perfect la limbajele C / C++: La pointerul ****T (**int** în cazul nostru) care reţine adresele începuturilor de linii i se alocă **m+1** locaţii de tip *T în loc de **m** spaţii. Pe ultima poziţie se depune un pointer **NULL** şi astfel numărul de linii este reţinut automat - până la întâlnirea pointerului **NULL**. Pentru fiecare linie se alocă o locaţie în plus (**n+1** locuri sau lungimea efectivă a liniei plus 1) şi în ultima locaţie se pune **NULL** sau o valoare care nu poate să apară în cadrul matricei **MAXINT**, **MAXLONG**, **MAXDOUBLE** etc. De multe ori valorile elementelor de tablou pot fi structuri, stringuri etc. In acest caz ca element al matricei apare un pointer spre structura / stringul respectiv, iar ca terminator al liniei se poate folosi pointerul **NULL**.

Pentru tablourile tridimensionale indicatorul de tablou este de tip ***T. Acesta punctează la începuturile straturilor care sunt structuri de tip **T (tablouri bidimensionale) şi fiecare urmează schema de reprezentare a tablourilor bidimensionale.

Pentru tablourile quatrodimensionale se pleacă de la tipul ****T şi se urmează schema de mai sus. De aici generalizarea este clară. Folosirea terminatorilor **NULL** la aceste scheme de alocare uşurează mult prelucrarea.

0.3. Fişiere text şi fişiere binare

In limbajul CS se folosesc sintagmele de fişier text şi fişier binar. Fără pretenţia de a da definiţii exacte, încercăm să lămurim ce se înţelege, îndeobşte, prin aceste două tipuri de fişiere:

- Fişiere *text*, sunt cele al căror conţinut poate fi afişat pe un ecran sau poate fi tipărit pe o imprimantă. El este format dintr-o succesiune de octeţi, fiecare conţinând codul unui caracter tipăribil: literă mare, literă mică, cifră, simbol special. Codificarea caracterelor se face folosind unul dintre sistemele de codificare standard: ASCII (pe 7 biţi), UNICODE (pe 8, 16, 32 biţi). La acest tip de fişiere articolul este format dintr-o *linie*. Două linii sunt separate fie prin *'\n'* – cazul Unix, *\r* în cazul MacOS, *'\r\n'* cazul Windows. Un fişier text se termină întotdeauna cu caracterul funcţional EOF (End Of File). Fiecare limbaj de programare are funcţii specifice de lucru cu fişiere text: **readln**, **writeln** etc.
- Fişiere *binare*, formate din şiruri de octeţi consecutivi fără nici o semnificaţie pentru afişare. Semnificaţia fiecărui octet este numai internă. Spre exemplu, fisiere avi, mp3, fişierele obiect rezultate in urma compilări şi fişierele executabile rezultate din editări de legături sunt fişiere binare. Evident că înşiruirea de biţi şi octeţi este "înţeleasă" de către CPU. Incercarea de a tipări direct un astfel de fişier nu are nici un sens!

In fapt, orice fişier "text" poate fi tratat ca şi un fişier binar dacă este accesat cu funcţii specifice fişierelor binare (care nu sunt din categoria celor text!).

Identificarea fişierelor text codate în altceva decât ASCII / 7 biţi este cunoscută sub numele de codificare UNICODE. De fapt acesta este un standard, iar codificările lui sunt UTF-8, UTF-16, UTF-32. Marcarea codificărilor UNICODE se face punând în primii octeţi ai fişierului o configuraţie de valori care să identifice tipul de codare. Această configuraţie (şir de octeţi) poartă numele de **BOM (Byte Order Mark)**. După caz, ea poate fi:

- EF BB BF, pentru UTF-8
- FE FF (maşini big-endian) respectiv FF FE (maşini little-endian), pentru UTF-16
- 00 00 FE FF (maşini big-endian) respectiv FF FE 00 00 (maşini little-endian) pentru UTF-32

0.4. Operaţii IO în C

0.4.1. Principalele funcţii

Exista doua posibilitati de efectuare a operatiilor I/O asupra unui fişier din programe C:

- Prin functiile standard C (**fopen, fclose, fgets, fprintf, fread, fwrite, fseek, sprints, sscanf** etc.) existente in bibliotecile standard C; prototipurile acestora se afla in fisierul header **<stdio.h>** (nivelul superior de prelucrare al fisierelor). 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 functii standardizate POSIX (**open, close, read, write, lseek, dup, dup2, fcntl** etc.) care reprezinta puncte de intrare in nucleul Unix si ale caror prototipuri se afla de regula in fisierul

header **<unistd.h>**, dar uneori se pot afla si in **<sys/types.h>**, **<sys/stat.h>** sau **<fcntl.h>** (nivelul inferior de prelucrare al fisierelor). 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 functii o presupunem cunoscuta deoarece face parte din standardul C (ANSI). Functiile din aceasta categorie repereaza orice fisier printr-o structura **FILE ***, pe care o vom numi descriptor de fisier.

Functiile din a doua categorie constituie **apeluri sistem Unix pentru lucrul cu fisiere**. Ele (antetul lor) sunt cuprinse in standardul POSIX. Functiile din aceasta categorie repereaza orice fisier printr-un intreg nenegativ, numit **handle**, dar atunci cand confuzia nu este posibila il vom numi tot descriptor de fisier. Pentru a obtine detalii despre formatele de fisiere si despre functii sau comenzi specifice formatelor de fisiere se poate consulta **$ man 5 nume**

0.4.2. Interclasarea a n fişiere text ordonate alfabetic

```
// Interclaseaza fisierele text, cu liniile ordonate alfabetic, ale caror nume
// sunt date la linia de comanda
#include <stdio.h>
#include <string.h>
#define MAXFILE 100
#define MAXNRFILES 20
#define MAXLINIE 1000
int main(int c, char **argv) {
    FILE *fi[MAXNRFILES];
    char lMax[MAXLINIE], lMin[MAXLINIE], liniaCurenta[MAXNRFILES][MAXLINIE];
    int n, i;
    lMax[0] = 0x7f; // Cea mai mare linie (nu e in fisiere)
    lMax[1] = 0;
    lMin[0] = 0;// Cea mai mica linie
    for (i = 1, n = 0; argv[i]; i++) {
        fputs(argv[i], stdout);
        fi[n] = fopen(argv[i], "r");
        if (fi[n] == NULL) continue; // Probabil nume eronat
        liniaCurenta[n++][0] = 0;// La deschidere se pune linia vida
    } // Terminat de deschis fisierele de intrare
    for (;; ) { // Ciclul principal de interclasare
        for (i = 0; i < n; i++) { // Citiri de linii din unele fisiere
            if (strcmp(lMin, liniaCurenta[i]) != 0) continue;// Nu citeste
            if (fgets(liniaCurenta[i], MAXLINIE, fi[i]) != NULL) continue;
            strcpy(liniaCurenta[i], lMax);
            fclose(fi[i]);// S-a terminat fisierul
        }
        strcpy(lMin, lMax); // Alege cea mai mica linie dintre curente
        for (i = 0; i < n; i++)
            if (strcmp(lMin, liniaCurenta[i]) > 0)
                strcpy(lMin, liniaCurenta[i]);
        if (strcmp(lMin, lMax) == 0) break;// Terminat interclasarile
        // Scrierea in iesire:
        //  (1) iesire fara linii multiple, se scrie doar lMin
        //  (2) iesire cu linii multiple, se scriu cele egale cu lMin
        for (i = 0; i < n; i++) { // (2)
            if (strcmp(lMin, liniaCurenta[i]) != 0) continue;// Nu scrie
            fputs(lMin, stdout);
        }
    } // Terminat interclasarile
    return 0;
} // main
```

0.4.3. Interclasarea a n fişiere text ordonate alfabetic - Go

```
// Interclaseaza fisierele text, cu liniile ordonate alfabetic, ale caror nume
// sunt date la linia de comanda
package main
import (
    "bufio"
    "fmt"
    "io"
    "os"
    "strings"
)
func main() {
    f := []*os.File{}
    fi := []*bufio.Reader{}
    liniaCurenta := []string{}
    lMax, lMin := "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~", ""
    var n, i int
    for i = 1; i < len(os.Args); i++ {
        file, err := os.Open(os.Args[i])
        if err != nil { continue }
        fmt.Print(os.Args[i], " ")
        f = append(f, file)
        fi = append(fi, bufio.NewReader(file))
        liniaCurenta = append(liniaCurenta, lMin)
    }
    n = len(fi)
    fmt.Println(lMin)
    fmt.Println(n)
    fmt.Println(lMax)
    for {
        for i = 0; i < n; i++ {
            if lMin != liniaCurenta[i] { continue }
            linie, err := fi[i].ReadString('\n')
            if err == io.EOF {
                liniaCurenta[i] = lMax
                f[i].Close()
                continue
            }
            linie = strings.Replace(linie, "\n", "", -1)             // Unix
            liniaCurenta[i] = strings.Replace(linie, "\r\n", "", -1) // Windows
        }
        lMin = lMax
        for i = 0; i < n; i++ {
            if lMin > liniaCurenta[i] { lMin = liniaCurenta[i] }
        }
        if lMin == lMax { break }
        // Scrierea in iesire:
        //  (1) iesire fara linii multiple, se scrie doar lMin
        //  (2) iesire cu linii multiple, se scriu cele egale cu lMin
        for i = 0; i < n; i++ { // (2)
            if lMin != liniaCurenta[i] { continue }
            fmt.Println(lMin)
        }
    }
}
```

0.4.4. Interclasarea a n fişiere text ordonate alfabetic - Python

```
# Interclaseaza fisierele text, cu liniile ordonate alfabetic, ale caror nume
# sunt date la linia de comanda
import os
import sys
```

```
def main():
    f = []
    liniaCurenta = []
    lMax, lMin = "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~", ""
    n, i = 0, 0
    for arg in sys.argv[1:]:
        file = open(arg)
        if file == None: continue
        print(arg+" ", end='')
        f.append(file)
        liniaCurenta.append(lMin)
    n = len(f)
    print(lMin)
    print(str(n))
    print(lMax)
    while True:
        for i in range(n):
            if lMin != liniaCurenta[i]: continue
            linie = f[i].readline()
            if linie == "":
                liniaCurenta[i] = lMax
                f[i].close()
                continue
            linie = linie.replace("\n", "")             # Unix
            liniaCurenta[i] = linie.replace("\r\n", "") # Windows
        lMin = lMax
        for i in range(n):
            if lMin > liniaCurenta[i]: lMin = liniaCurenta[i]
        if lMin == lMax: break
        # Scrierea in iesire:
        #  (1) iesire fara linii multiple, se scrie doar lMin
        #  (2) iesire cu linii multiple, se scriu cele egale cu lMin
        for i in range(n): # (2)
            if lMin != liniaCurenta[i]: continue
            print(lMin)
main()
```

0.4.5. Oglindirea conţinutului unui fişier: soluţia 1

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. In prima soluţie se transferă octet cu octet, iar în soluţia a doua (secţiunea următoare) se face transfer pe blocuri (în cazul nostru de câte 10000 octeţi). Codul (completat cu calculul duratei operatiei) este:

```
// 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>
int main(int argc, char *argv[]) {
    int f;
    long s, d;
    char bs, bd;
    struct stat stare;
    time_t start;
    start = time(NULL);
```

```
    stat(argv[1], &stare);
    f = open(argv[1], O_RDWR);
    for (s = 0, d = stare.st_size-1; s < d; s++, d--) {
        lseek(f, s, SEEK_SET);
        read(f, &bs, 1);
        lseek(f, d, SEEK_SET);
        read(f, &bd, 1);
        lseek(f, s, SEEK_SET);
        write(f, &bd, 1);
        lseek(f, d, SEEK_SET);
        write(f, &bs, 1);
    }
    close(f);
    printf("Durata: %d\n",(int)(time(NULL)-start));
    return 0;
}
```

0.4.6. Oglindirea conţinutului unui fişier: soluţia 1 Go

```
package main
import "os"
func main() {
    var s, d int64
    t, _ := os.Stat(os.Args[1])
    d = t.Size() - 1
    bs := make([]byte, 1)
    bd := make([]byte, 1)
    f, _ := os.OpenFile(os.Args[1], os.O_RDWR, 0666)
    for s=0; s < d; {
        _,_ = f.Seek(s, 0)
        _,_ = f.Read(bs)
        _,_ = f.Seek(d, 0)
        _,_ = f.Read(bd)
        _,_ = f.Seek(s, 0)
        _,_ = f.Write(bd)
        _,_ = f.Seek(d, 0)
        _,_ = f.Write(bs)
        s++
        d--
    }
    f.Close()
}
```

0.4.7. Oglindirea conţinutului unui fişier: soluţia 1 Python

```
import sys
import os
s = 0
d = os.stat(sys.argv[1]).st_size - 1
f = open(sys.argv[1], "rb+")
while s < d:
    f.seek(s, 0)
    bs = f.read(1)
    f.seek(d, 0)
    bd = f.read(1)
    f.seek(s, 0)
    f.write(bd)
    f.seek(d, 0)
    f.write(bs)
    s = s + 1
    d = d - 1
f.close()
```

0.4.8. Oglindirea conţinutului unui fişier: soluţia buffer

Problema de mai sus are o soluţie mult mai eficientă folosind citirea pe blocuri:. Este util sa se retina din solutia a doua functiile construite de noi **Read** si **Write**. Ele citesc / scriu, eventual prin repetarea read / write, exact **n** octeţi. Utilizatorul trebuie să se asigure în prealabil că schimbul celor **n** octeţi este posibil. Aceste funcţii s-ar putea dovedi utile in multe situatii.

```
// 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 s, d;
    char t;
    for (s = 0, d = n - 1; s < d; s++, d--) {
        t = sir[s];
        sir[s] = sir[d];
        sir[d] = 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; ; ) {
        i = write(f, p, c);
        if (i == c) return;
        c -= i;
        p += i;
    }
}
int main(int argc, char *argv[]) {
    int f, n;
    long s, d, l, m;
    char bs[MAX], bd[MAX];
    struct stat stare;
    time_t start;
    start = time(NULL);
    stat(argv[1], &stare);
    l = stare.st_size;
    m = l / 2;
    n = MAX;
    if (n > m) n = m;
    f = open(argv[1], O_RDWR);
    for (s = 0, d = l - n; s + n <= m; s += n, d -= n) {
        if (m - s < n) n = m - s;
        lseek(f, s, SEEK_SET);
```

```
        Read(f, bs, n);
        lseek(f, d, SEEK_SET);
        Read(f, bd, n);
        oglinda(bs, n);
        oglinda(bd, n);
        lseek(f, s, SEEK_SET);
        Write(f, bd, n);
        lseek(f, d, SEEK_SET);
        Write(f, bs, n);
    }
    close(f);
    printf("Durata: %d\n",(int)(time(NULL)-start));
    return 0;
}
```

0.4.9. Oglindirea conţinutului unui fişier: soluţia buffer Go

```
package main
import "os"
func oglinda (sir [] byte) []byte {
    s, d := 0, len(sir) - 1
    for ; s < d; {
        t := sir[s]
        sir[s] = sir[d]
        sir[d] = t
        s++
        d--
    }
    return sir
}
func Read(f *os.File, n int) []byte {
    sir := make([]byte, 0)
    c := n
    for {
        t := make([]byte, c)
        c,_ = f.Read(t)
        sir = append(sir, t...)
        if len(sir) == n { return sir }
        c = n - len(sir)
    }
}
func Write(f *os.File, sir []byte) {
    t := sir
    for {
        c,_ := f.Write(t)
        if len(t) == c { return }
        t = t[c:]
    }
}
func main() {
    const MAX = 10000
    var s, d, m, l int64
    t,_ := os.Stat(os.Args[1])
    l = t.Size()
    m = l / 2
    n := MAX
    if int64(n) > m { n = int(m) }
    s = 0
    d = l - int64(n)
    f,_ := os.OpenFile(os.Args[1], os.O_RDWR, 0666)
    for s=0; s + int64(n) <= m; {
        if m - s < int64(n) { n = int(m - s) }
        f.Seek(s, 0)
        bs := Read(f, n)
        f.Seek(d, 0)
```

```
        bd := Read(f, n)
        f.Seek(s, 0)
        Write(f, oglinda(bd))
        f.Seek(d, 0)
        Write(f, oglinda(bs))
        s += int64(n)
        d -= int64(n)
    }
    f.Close()
}
```

0.4.10. Oglindirea conţinutului unui fişier: soluţia buffer Python

```
import sys
import os
def oglinda (sir):
    s, d = 0, len(sir) - 1
    res = bytearray(sir)
    while s < d:
        t = res[s]
        res[s] = res[d]
        res[d] = t
        s = s + 1
        d = d - 1
    return bytes(res)
def Read(f, n):
    sir = bytes()
    c = n
    while True:
        t = f.read(c)
        sir = sir + t
        if len(sir) == n: return sir
        c = n - len(sir)
def Write(f, t):
    f.write(t)
def main():
    MAX = 5000
    l = os.stat(sys.argv[1]).st_size
    m = l / 2
    n = MAX
    if n > m: n = int(m)
    s = 0
    d = l - n
    f = open(sys.argv[1], "rb+")
    while s + n <= m:
        if m - s < n: n = m - s
        f.seek(s, 0)
        bs = Read(f, n)
        f.seek(d, 0)
        bd = Read(f, n)
        f.seek(s, 0)
        f.write(oglinda(bd))
        f.seek(d, 0)
        f.write(oglinda(bs))
        s = s + n
        d = d - n
    f.close()
main()
```

0.5. Manipularea fişierelor în sisteme de fişiere

0.5.1. Principalele prototipuri de funcţii

Iata 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);`

0.5.2. Parcurgerea recursivă a fişierelor dintr-un director şi descendenţi

```
// La linia de comanda se da un nume de director urmate de tipuri de fisiere
// Se face rezumatul fisierelor de tipurile specificate din directorul dat.
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <time.h>
#define MAX 1024
#define TRUE 1
#define FALSE 0
char *types[MAX];
time_t start, lastTime = 0; // 31536000 secunde / an
int nr = 0;
int endWith(char *nume, char *tip) {
    int ln, lt;
    ln = strlen(nume);
    lt = strlen(tip);
    if (strcmp(nume + ln - lt, tip) == 0) return TRUE;
    return FALSE;
}
int isType(char *nume) {
    int i;
    if (types[0] == 0) return TRUE;
    for (i = 0; types[i]; i++)
        if (endWith(nume, types[i])) return TRUE;
    return FALSE;
}
void bs2s(char nou[], char vechi[]) {
    // Este efectiva doar la Windows. Pentru Unix este inefectiv
    char *p;
    int i, j;
    strcpy(nou, vechi);
    for ( ; ; ) {
        p = strchr(nou, '\\');
        if (p == NULL) return;
        for (i = 0; p[i] == '\\'; i++);
        p[0] = '/';
        if (i == 1) continue;
```

```
        for (j = i; p[j]; p[j - i + 1] = p[j], j++);
    }
}
void aFile(char numed[]) {
    char numef[MAX];
    DIR *dir;
    struct dirent *d;
    struct stat status;
    dir = opendir(numed);
    if (dir != NULL) {
        while (d = readdir(dir)) {
            if (d->d_name[0] == '.') continue;
            strcpy(numef, numed);
            strcat(numef, "/");
            strcat(numef, d->d_name);
            aFile(numef);
        }
    }
    closedir(dir);
    stat(numed, &status);
    if (isType(numed) == FALSE)     return;
    if (!S_ISREG(status.st_mode))   return;
    if (status.st_mtime < lastTime) return;
    bs2s(numef, numed);
    printf("%d\t%s\n", ++nr, numef);
}
int main(int argc, char *argv[]) {
    int i, j;
    char dir[MAX], absDir[MAX];
    char *help[] = {
        "Se afiseaza numele fisierelor de anumite tipuri create dupa un moment dat.",
        "Sunt trei feluri de argumente si se pot scrie in orice ordine:",
        " - directorul de start, in absenta .; incepe cu / sau ./ sau ../",
        " - momentul de start, in absenta 0; numarat in secunde de la 1970_01_01,
31536000 secunde / an",
        " - tipurile fisierelor create dupa momentul de start; incep cu . urmat de
litere si / sau cifre",
        "      In absenta tipurilor sunt date nuele tuturor fisierelor create dupa
momentul de start.",
        "De exemplu, fisierele c si java din directorul parinte dupa 2020 se obtin
prin:",
        "./lsRecursiv 1576800000 .java .c ../", NULL
    };
    dir[0] = '.';
    dir[1] = 0;
    start = time(NULL);
    for (i=0; help[i]; printf("%s\n", help[i]), i++);
    for (i=1, j=0; argv[i]; i++) {
        if (argv[i][0] == '/' ||
           (strlen(argv[i]) >= 2 && argv[i][0] == '.' && argv[i][1] == '/') ||
           (strlen(argv[i]) >= 3 && argv[i][0] == '.' && argv[i][1] == '.'
||argv[i][2] == '/'))
             strcpy(dir, argv[i]);
        else if (argv[i][0] == '.') types[j++] = argv[i];
        else if (isdigit(argv[i][0])) lastTime = atol(argv[i]);
        else ; // toate celelalte stringuri se ignora
    }
    realpath(dir, absDir);
    bs2s(dir, absDir);
    types[j] = NULL;
    printf("Director start: %s\nMoment de start: %ld\nExtensii: ",dir, lastTime);
    for (i=0; types[i]; i++) printf("%s ",types[i]);
    printf("\n");
    aFile(absDir);
    printf("Durata in secunde: %ld\n",time(NULL)-start);
    fflush(stdout);
    return 0;
}
```

0.5.3. Parcurgerea recursivă a fişierelor dintr-un director şi descendenţi - Go

```
package main
import (
    "os"
    "time"
    "path/filepath"
    "strconv"
    "strings"
    "fmt"
)
var types = []string{}
var lastTime = int64(0)
func isType(nume string) bool {
    if len(types) == 0 {return true }
    for _,typ := range types {
        if strings.HasSuffix(nume, typ) { return true }
    }
    return false
}
func bs2s(vechi string) string {
    nou := strings.ReplaceAll(vechi, "\\\\", "\\")
    nou = strings.ReplaceAll(nou, "\\", "/")
    return nou
}
func aFile(absDir string) {
    nr := 0
    filepath.Walk(absDir, func(path string, info os.FileInfo, err error) error {
        if info.IsDir() { return nil }
        if !isType(path) { return nil }
        t,_ := os.Stat(path)
        ts := strconv.FormatInt(t.ModTime().UnixNano(), 10)
        ti,_ := strconv.ParseInt(ts, 10, 64)
        if ti < lastTime { return nil }
        nr++
        fmt.Printf("%d\t%s\n", nr, bs2s(path))
        return nil
    })
}
func main() {
    help := `Se afiseaza numele fisierelor de anumite tipuri create dupa un moment
dat.
Sunt trei feluri de argumente si se pot scrie in orice ordine:
 - directorul de start, in absenta .; incepe cu / sau ./ sau ../
 - momentul de start, in absenta 0; numarat in secunde de la 1970_01_01, 31536000
secunde / an
 - tipurile fisierelor create dupa momentul de start; incep cu . urmat de litere si /
sau cifre
      In absenta tipurilor sunt date nuele tuturor fisierelor create dupa momentul de
start.
De exemplu, fisierele c si java din directorul parinte dupa 2020 se obtin prin:
    ./lsRecursiv 1576800000 .java .c ../`
    dir := "."
    start := time.Now()
    fmt.Printf("%s\n", help)
    for _,arg := range os.Args[1:] {
        if len(arg) >= 3 && (arg[:1] == "/" || arg[:2] == "./" || arg[:3] =="../") {
            dir = arg
        } else if arg[:1] == "." {
            types = append(types, arg)
        } else if '0' <= arg[0] && arg[0] <= 9 {
                lastTime,_ = strconv.ParseInt(arg, 10, 64)
        }
```

```
    }
    absDir,_ := filepath.Abs(dir)
    fmt.Printf("Director start: %s Moment de start: %d Extensii: %v", bs2s(absDir),
lastTime, types)
    fmt.Printf("\n")
    aFile(absDir)
    t := time.Now()
    d := t.Sub(start) / 1000000
    fmt.Printf("Durata in secunde: %d\n", d)
}
```

0.5.4. Parcurgerea recursivă a fişierelor dintr-un director şi descendenţi - Python

```
import sys
import os
import time
types = []
lastTime = 0
def isType(nume):
    global types
    if len(types) == 0: return True
    for type in types:
        if nume.endswith(type): return True
    return False
def bs2s(vechi):
    nou = vechi.replace("\\\\", "\\")
    nou = nou.replace("\\", "/")
    return nou
def aFile(absDir):
    global lastTime
    global types
    nr = 0
    for root, numed, numef in os.walk(absDir):
        for nume in numef:
            abs = os.path.realpath(os.path.join(root, nume))
            status = os.stat(abs)
            if not isType(abs): continue
            if status.st_mtime < lastTime: continue
            nr += 1
            print(str(nr)+"\t"+bs2s(abs))
def main():
    help = """Se afiseaza numele fisierelor de anumite tipuri create dupa un moment
dat.
Sunt trei feluri de argumente si se pot scrie in orice ordine:
 - directorul de start, in absenta .; incepe cu / sau ./ sau ../
 - momentul de start, in absenta 0; numarat in secunde de la 1970_01_01, 31536000
secunde / an
 - tipurile fisierelor create dupa momentul de start; incep cu . urmat de litere si /
sau cifre
      In absenta tipurilor sunt date nuele tuturor fisierelor create dupa momentul de
start.
De exemplu, fisierele c si java din directorul parinte dupa 2020 se obtin prin:
    ./lsRecursiv 1576800000 .java .c ../"""
    global types
    global lastTime
    dir = "."
    start = time.time()
    print(help)
    for arg in sys.argv[1:]:
        if arg.startswith("/") or arg.startswith("./") or arg.startswith("../"): dir
= arg
        elif arg.startswith("."): types.append(arg)
        elif arg.isdigit(): lastTime = int(arg)
        else: pass #toate celelalte stringuri se ignora
```

```
    absDir = os.path.realpath(dir)
    print("Director start: "+bs2s(str(absDir))+" Moment de start: "+str(lastTime)+"
Extensii: "+str(types))
    aFile(absDir)
    print("Durata in secunde: "+str(time.time()-start))
    sys.stdout.flush()
main()
```

0.6. Probleme propuse

In rezolvarea acestor probleme se va folosi exclusiv alocarea dinamica pentru vectorii si matricele folosite în program.

1. Se da un numar natural n. Se cere sa se genereze toate matricele nxn avand ca elemente numere distincte din 1..nxn astfel incat nici un element sa nu aiba aceeasi paritate cu vecinii sai (vecinii unui element se considera pe directiile N,S,E,V).

2. O trupa de N actori isi propun sa joace o piesa cu M acte astfel incat:
- orice 2 acte au distributia diferita
- in orice act exista cel putin un singur actor
- distributia a doua acte consecutive difera printr-un singur actor

Sa se furnizeze toate solutiile problemei (daca exista).

3. Intr-un oras exista n intersectii legate prin strazi. Legaturile se dau sub forma unei matricie de nxn cu semnificatia A[i][j] = 1 daca intersectiile i si j sunt legate printr-o strada şi 0 in caz contrar. Se dau m politisti. Se cere sa se distribuie un numar minim de politisti in intersectii astfel incat sa fie supravegheate toate strazile.

4. Se da o fotografie specificata printr'o matrice patratica care contine 0 si 1, 0 pentru punctele albe si 1 pentru punctele negre. Se considera fondul alb şi obiectele negre, iar daca doua puncte negre sunt vecine pe linie, coloana sau diagonala, atunci ele apartin aceluiasi obiect. Sa se numere cate obiecte distincte apar in fotografie.

5. Se da un sir x1,..,xn de numere intregi. Se cere sa se ordoneze crescator sirul folosind ca metoda sortarea prin interclasare.

6. Se da o bucata dreptunghiulara de tabla de lungime l si inaltime h, avand pe suprafata ei n gauri de coordonate numere intregi. Se cere sa se decupeze din ea o bucata de arie maxima care nu prezinta gauri. Sunt permise numai taieturi verticale si orizontale.

7. Se da numele unui fisier text la linia de comanda. Se determine cuvintele distincte mai lungi de 5 caractere din acest fisier si sa se determine frecvenţa lor. Cuvintele se vor memora într-o listă în care există două legături: una ce leagă cuvintele în ordine alfabetică şi alta care le leagă în ordinea crescătoare a frecvenţei lor.

8. Split: să se scrie un program care decupează un fişier mare, al cărui nume este dat la linia de comandă, în mai multe fişiere având dimensiunea de MAX caractere (implicit 10000, explicit precizată printr-un parametru la linia de comandă. Părţile decupate din fişierul mare vor avea numele: 00001, 00002, 00003 s.a.m,d. Ele conţin, în ordine, părţile decupate din fişierul mare, toate de lungime MAX, cu excepţia ultimei bucăţi care conţine, de regulă, mai puţini octeţi. Se vor face două implementări: prima va citi octet cu octet din fişierul mare, iar a doua va citi câte o bucată din fişierul mare, cu lungime NU NEAPARAT divizor al lui MAX. Se vor compara timpii de execuţie din cele două implementări.

9. Merge: operaţia inversă din problema precedentă: presupunem că în directorul curent avem fişierele cu numele: 00001, 00002, 00003 s.a.m,d. Ele conţin, în ordine, părţile decupate dintr-un fişier mare, aşa cum am prezentat în problema precedentă. Se cere reconstrucţia acestui fişier mare. Se vor face două implementări: prima va citi octet cu octet din bucăţi, iar a doua va citi câte un tampon cu lungime NU NEAPARAT divizor al lungimii fişierelor bucăţi. Se vor compara timpii de execuţie din cele două implementări.

10. Să se implementeze eficient ciurul lui Eratostene (determinarea numerelor prime până la n) pentru un n foarte mare. Eficienţa se va realiza înlocuind lista de numere întregi cu un şir de biţi reprezentând, prin poziţie, numărul întreg: 0 șters, 1 neşters şi în final număr prim. Implementarea tabloului de biţi se va realiza prin alocare dinamică. Dacă n este foarte mare, atunci vectorul de biţi se va păstra într-un fişier şi se aduc în memorie numai câte o parte (să zicem de câte 10000 biţi) din acest fişier.