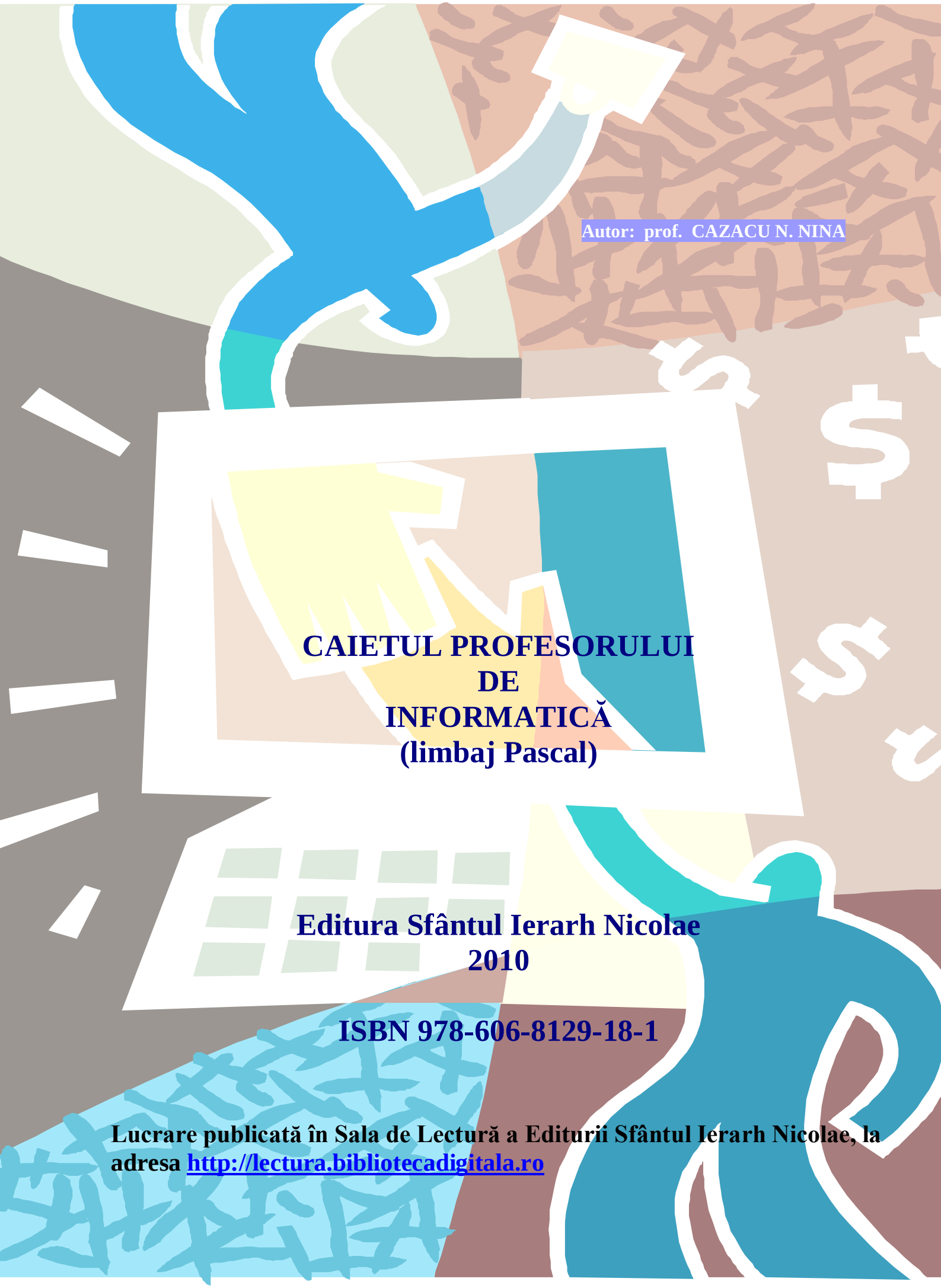


Autor: prof. CAZACU N. NINA

CAIETUL PROFESORULUI DE INFORMATICĂ (limbaj Pascal)

Editura Sfântul Ierarh Nicolae
2010

ISBN 978-606-8129-18-1

Lucrare publicată în Sala de Lectură a Editurii Sfântul Ierarh Nicolae, la adresa http://lectura.bibliotecadigitala.ro

Cuprins

Introducere

Lucrarea este structurata in 6 capitole, cuprinzand problemele principale in derularea lor, conforma Programei, fiecare capitol tratand o sectiune a cerintelor intr-un mod care se doreste cat mai accesibil. Majoritatea subiectelor sunt tratate in pseudocod, codul fiind asociat.

Dorinta noastra este sa-i ajutam pe cei care, urmand acest profil de specialitate, nu sunt neaparat elevi de exceptie, ci doar elevi, si, spunem noi viitori informaticieni.

In finalul lucrarii este prezentata o bibliografie completa, deoarece majoritatea algoritmilor nu sunt originali, ci selectati din manualele recomandate de diverse edituri la profilul de specialitate.

Mult Succes!

prof. Nina Cazacu

Autor: CAZACU N. NINA LICEUL C. A. ROSETTI,, BUCURESTI

CAPITOLUL I

ALGORITMI

I.1 NOTIUNEA DE ALGORITM (RETETA, METODA, PROCEDEU)

Definitie:
ALGORITM este o secventa finita de operatii, ordonata si complet definita, care plecand de la date(intrari) produce rezultate (iesiri).

Descrierea unui algoritm se face prin intermediul unor propozitii care sunt de fapt comenzi executabile, in cazul nostru adresate calculatorului.Fiecare propozitie specifica o operatie (actiune) care se aplica datelor algoritmului determinand modificarea acestora.

Caracteristicile unui algoritm:

a) sa fie bine definit, deci operatiile cerute sa fie specificate riguros si fara ambiguitate;
b) sa fie finit, adica sa se termine dupa executarea unui numar finit de operatii;
c) sa fie general pentru rezolvarea unei clase de probleme.

2. Algoritmul lucreaza cu anumite obiecte. Acestea sunt:

- datele;
- expresiile;
- operatiile.

Definitie :
Datele sunt entitati asupra carora opereaza calculatorul, furnizate in scopul obtinerii unor rezultate.
Datele pot avea valoare fixa sau modificabila, primele numindu-se constante, cele din urma variabile. Intre date pot interveni operatii iar rezultatul sa fie o expresie .

Operatiile nu pot interveni decat intre elementele compatibile. Datele se introduc sub forma unui text, adica o secventa de caractere ce apartin unui anumit set de caractere, cel mai uzual fiind codul ASCII. In interiorul calculatorului informatia este grupata in CUVINTE, fiecare CUVANT avand o lungime de n biti. El este memorat de n dispozitive cu 2 valori (0 sau 1) in 2^n moduri (stari), unde n are o valoare fixa, de obicei n=8.

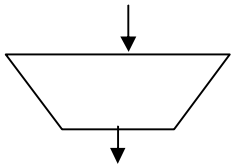
I.2 PRINCIPIILE PROGRAMARII STRUCTURATE

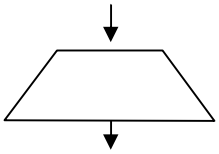
Formele conventionale de reprezentare a algoritmilor

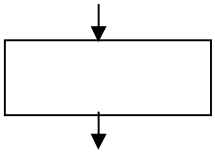
- scheme logice (organigrame);
- limbaje pseudocod.

Schemele logice utilizeaza sageti pentru fluxul controlului algoritmilor (succesiunile posibile ale actiunilor); sagetile leaga diferite forme geometrice care simbolizeaza actiunile.
Limbajele pseudocod folosesc “cuvinte- cheie” si cateva reguli simple de aliniere.
Definitie:
Cuvintele- cheie sunt cuvinte cu inteles prestabilit care identifica operatiile ce se executa.

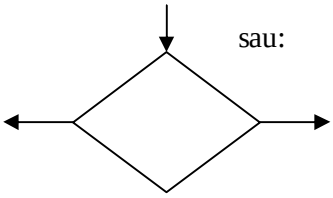
SCHEME LOGICE

1. Bloc de citire:  (valori citite)

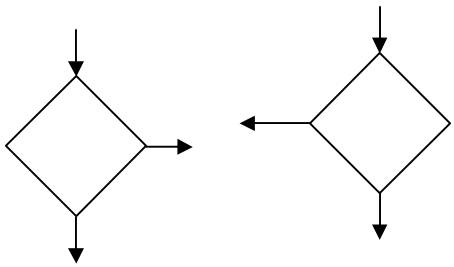
2. Bloc de scriere:  (valori rezultate)

3. Bloc de atribuire: 

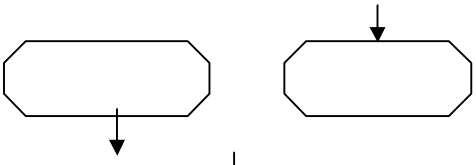
4. Bloc de decizie sau salt conditionat(cu 2 iesiri)



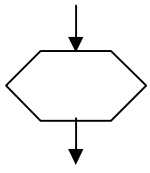
sau:



5. Bloc de inceput/ sfarsit organigrama:



6.Bloc de instructiuni (subalgoritm)



PSEUDOCOD

Prezinta doua tipuri de enunturi: → limba engleza

→ limba romana

exemplu: daca x>0 atunci scrie x
if x>0 then write x etc.

TEOREMA LUI JACOPINI: (TEOREMA DE STRUCTURA)

Orice algoritm avand o intrare si o iesire poate fi reprezentat ca o combinatie a 3 structuri:

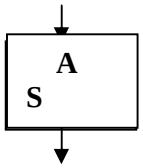
a) secventa (succesiunea de 2 sau mai multe operatii);
b) decizia (alegerea unei optiuni din 2 posibile);
c) ciclul cu test initial (repetarea unei actiuni atat timp cat o conditie este indeplinita).

In plus, derivate din a, b, c, mai sunt inca 3 variante utilizate de catre Programarea structurata:

d) selectia (alegerea din mai multe cazuri posibile);
e) ciclul cu text final →
f) ciclul cu contor → iteratii

SCHEMA LOGICA A STRUCTURILOR DE BAZA

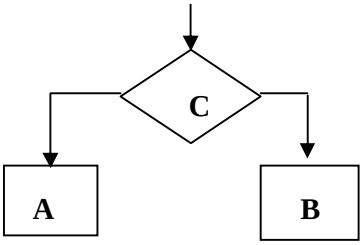
I.2.1 Secventa (structura de calcul)



A este o transformare de date:

- una sau mai multe atribuiri, sau calculare urmata de atribuire ca de exemplu : $x \leftarrow 0$, $y \leftarrow \cos(x)+\sin(x)$ etc.
- una sau mai multe enunturi: citire, afisare ca de exemplu : readln(x); write(x+y); (in interiorul Programului, nu introducere de date initiale)

I.2.2 Decizia



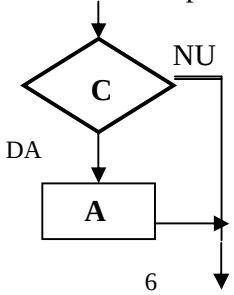
Se evalueaza conditia C: daca C este adevarata atunci se executa secventa A, daca nu se executa secventa B.

PSEUDOCOD: daca C atunci A altfel B respectiv : if C then A else B

Varianta forma scurta :

Se evalueaza conditia C : daca C este adevarata atunci se executa secventa A.

PSEUDOCOD : daca C atunci A respectiv : if C then A

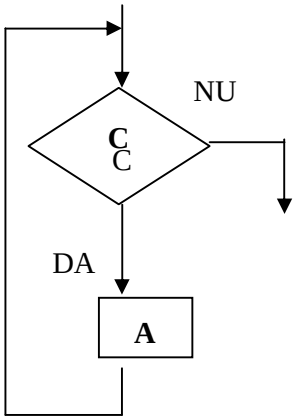


I. 2.3 Ciclul (bucla, iteratia) :

- cu test initial

Se evalueaza conditia C : daca C este adevarata, se executa secventa A, apoi ciclul se reia: se evalueaza C, daca C este adevarata atunci se executa A.

PSEUDOCOD : cat timp C executa A while C do A

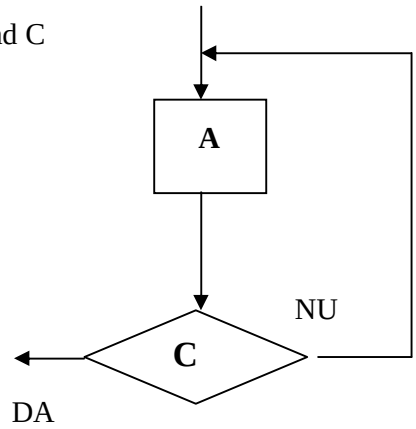


- cu test final

Se executa secventa A. Se evalueaza conditia C. Cat timp conditia C nu este adevarata se reia ciclul : se executa A, se evalueaza C.

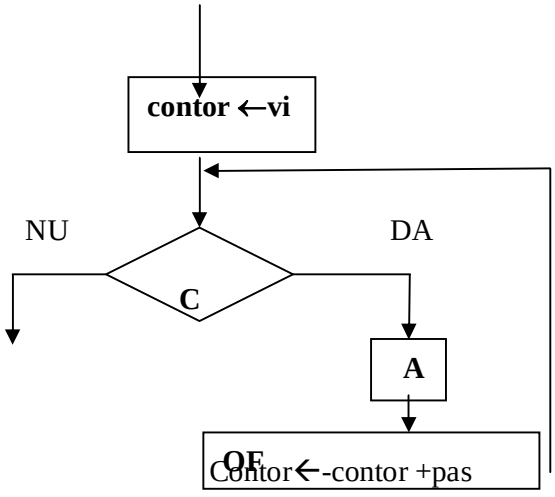
PSEUDOCOD :

repeta A pana cand C repeat A until C



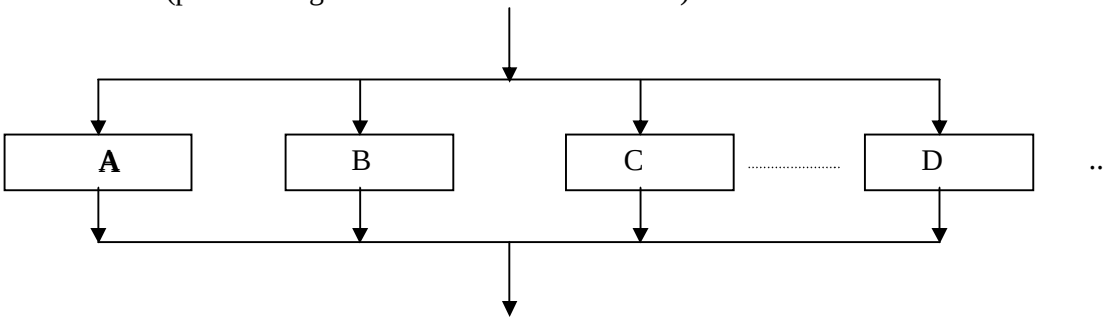
- cu contor

Secventa A se executa in timp ce un numarator (contor) ia valori intre o valoare initiala vi si respectiv o valoare finala vf.
Intre valori consecutive contorul creste cu valoarea pas.



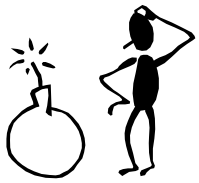
PSEUDOCOD
pentru contor = vi, vf, pas executa A sau : for contor:=vi to (downto) vf do A

I.2.4 Selectia (permite alegerea din mai multe alternative)



Se considera expresia e, numita selector, se evalueaza comparandu-se rezultatul cu o anumita constanta dintr-un sir dat, v1,v2..vn; in functie de acest rezultat, este executata una din secventele A, B, C,..D

PSEUDOCOD	l. romana	l. engleza	
	Selectie(e)	CASE(e)	
	v1: A	v1: A	e = selector(expresie cu valoare ordinala)
	v2 :B	v2 : B	
	v3: C	v3: C	
	…	…	
	vn : D	vn: D	
	altfel E	else E	
	sfarsit selectie	endCASE	



I.3 ELEMENTE DE BAZA ALE LIMBAJULUI PASCAL

I.3.1. STRUCTURA PROGRAMELOR PASCAL este compusa din trei sectiuni:

a) Antet (titlu)
b) Sectiune declarativa
c) Sectiune executabila

de exemplu :

```
Program exemplu;   { antetul, care poate lipsi}
        {sectiunea declarativa}
uses crt;
var  ch: char;
        {sectiunea executabila}
        begin
         write('apasati o tasta:'); readln(ch);
        end.
```

I.3.2 DESCRIEREA VOCABULARULUI LIMBAJULUI PASCAL CU AJUTORUL DIAGRAMELOR DE SINTAXA.

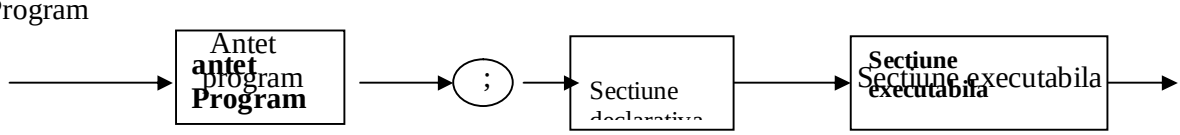
Definitie: Diagrama de sintaxa este un ansamblu de elemente grafice: elipse, cercuri, dreptunghiuri, avand fiecare un rol bine definit.
Elipsele incadreaza "terminale" (nemodificabile), iar dreptunghiurile incadreaza "atribute"; cercurile sunt utilizate pentru "separatori". Legaturile se realizeaza prin sageti .

I.3.3 VOCABULARUL LIMBAJULUI (setul de caractere, identificatorii, separatorii, comentariile)

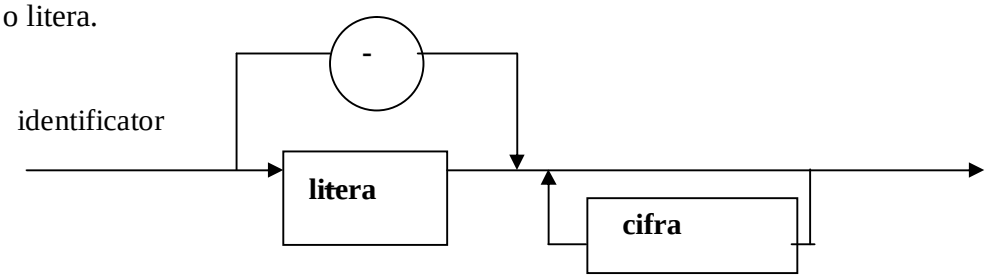
I.3.3.1 Setul de caractere al limbajului Pascal

	- minus
2	* inmultit
3	/ impartit
4	= egal
5	^ adresa
6	< mai mic
7	> mai mare
8	(
9	)
litera	{
a	}
A	[
b	]
B	.
.....	,
y	:
Y	;
z	'
Z	# {diez}
_ (underline)	$ {dolar}

Observatie: Limbajul Pascal nu distinge majusculele de minuscule!!!
Diagrama de sintaxa:
Program



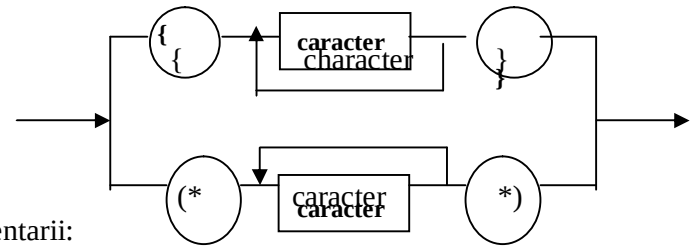
Definitii:
I.3.3.2 Identificatorii sunt siruri de caractere alfabetice, numerice sau _, primul caracter fiind totdeauna o litera.



Nu pot fi folosite ca identificatori anumite siruri de caractere, apartinand limbajului, numite cuvinte cheie , care completeaza vocabularul : absolute, aand, array, sm, const, begin, case, constructor, destructor, div, do, downto, else, end, external, file, for, forward, function, in, if, implementation, in, inline, interface, interrupt, label, mod, nil, not, object, on, of, or, packed, procedure, Program, record, repeat, set, shr, shl, string, then, to, type, unit, until, uses, var, virtual, while, with, xor.

I.3.3.3 Separatorii sunt : ; , virgula, eof,eoln, cu mentiunea ca ultimile doua nu sunt printabile

I.3.3.4 Comentariile sunt texte explicative, care nu sunt compilate (executate)



Exemple de comentarii:

{ acesta este un comentariu}

(*si acesta este un comentariu*)

I.3.4 CONSTANTE

- CONSTANTA INTREAGA : -125, $A1, $FFFF, 17

MAXINT=32.767(cel mai lung intreg)

MAXLONGINT=2.147.483.647(cel mai mare intreg dublu)

Literele sunt utilizate pentru a descrie constantele intregi scrise in baza 16 (sistemul hexazecimal):

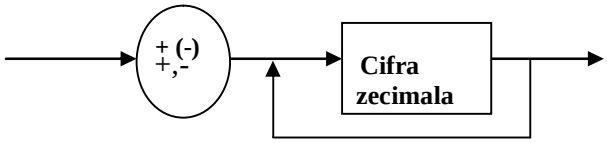
0 1 2 3 4 5 6 7 8 9 A B C D E F

(10 11 12 13 14 15)

Un intreg poate fi definit drept constanta intr-un Program, utilizand cuvantul cheie “const”, semnul de egalitate , ca in exemplul urmator:

Const a=10;

Constanta intreaga formata din cifre zecimale :



- CONSTANTA REALA:
- Exemple : 5.135, 5.23e+10, 11e-20(notatie exponentiala)

- CONSTANTA CARACTER

Exemple :‘a’, ‘A’, ‘9’, #66 (caracterul ‘B’)

- CONSTANTA SIR DE CARACTERE (secventa de maxim 256 de caractere)

Exemple : ^A; #65#13 ; ‘SIR’

Eercitiu: Scrieti diagramele de sintaxa ale exemplelor de mai sus

La fel ca si intregii, si celelalte constante se vor putea defini cu ajutorul cuvantului const :

```
Const    ci=10; cc='a'; cb=true; csir='string'; cr=2.3;
```

-CONSTANTE SIMBOLICE (desemnate printr-un identificator)

Aceste constante nu pot fi modificate in timpul executiei programului asigurand independenta

acestuia relativa la tentativa de a le atribui ulterior.

a) definite de utilizator (ca in exemplele de mai sus)

```
const CT=41;
    e= 3.14;   luna='septembrie';
```

b) predefinite (definite in unit-ul SYSTEM): MAXINT, MAXLONGINT, NIL, TRUE, FALSE
PI este numarul π, aproximat cu 12 zecimale:3,14159265358979323846264338327 95

I.3.5 NOTIUNEA DE TIP DE DATA

Definitie :

Data este orice entitate asupra careia poate opera calculatorul. Tipul de data este multimea informatio-nala definita printr-un nume din care-si poate lua valorile o anumita data.

Clasificarea tipurilor de date:

a) NESTRUCTURATE(SIMPLE) : reale si ordinale(predefinite in SYSTEM, sau definite de utilizator)
Datele simple ordinale sunt : { datele predefinite: intreg , boolean, char.
datele definite de utilizator: enumerat, subdomeniu

Tipurile ordinale definesc o multime finita si ordonata de valori, in timp ce tipul real, desi este tot pre-definit, nestructurat, nu este ordinal.

b) STRUCTURATE: tablou, string, articol(inregistrare), multime, fisier.

c) REPER(POINTER, REFERINTA)

I.3.6 INFORMATII GENERALE ASUPRA TIPURILOR DE DATE

I.3.6.1 TIPUL REAL: ocupa 6 octeti(11-12 cifre) primul bit fiind bitul de semn

variante: (in mediul Turbo, optiunea *options/compiler/numeric processing* este pe ON)
single(ocupa 4 octeti primul bit fiind bitul de semn)
double(ocupa 8 octeti primul bit fiind bitul de semn)
extended(ocupa 10 octeti primul bit fiind bitul de semn)

Expresiile aritmetice care folosesc tipul real au ca operatori : +,-,*,/ si functii standard: abs(x), sqr(x), sin(x), cos(x), arctan(x), ln(x), exp(x) (explicit :exp(x)= e^x), int(x), frac(x).

Efectul aplicarii functiilor mentionate este sugerat de denumire :

abs (x) - calculeaza valoarea absoluta a lui x
sqr(x) - calculeaza patratul lui x
sin(x), cos(x), arctan(x), ln(x), exp(x) sunt analoge functiilor matematice sinus, cosinus,arctangenta, logaritm natural, exponentiala cu baza e (baza logaritmului natural), deci au acelasi efect.
int(x) - calculeaza partea intreaga a lui x, rezultatul fiind de tip real
frac(x) - calculeaza partea fractionara a lui x
PI - este numarul π, aproximat cu 12 zecimale:3,14159265358979323846264338327 95
(se considera ca functie cu rezultat real, fara argument)

I.3.6.2 TIPUL INTREG: ocupa 2 octeti primul bit fiind bitul de semn, cu intervalul de valori posibile [-MAXINT-1, MAXINT];

Variante: shortint(ocupa 1 octet, primul bit fiind bitul de semn, cu intervalul de valori posibile [-128, 127]);
byte(ocupa 1 octet fara semn, cu intervalul de valori posibile [0, 255];
word (ocupa 2 octeti fara semn, cu intervalul de valori posibile [0, 65.535]);
longint (ocupa 4 octeti, primul bit fiind bitul de semn), cu intervalul de valori posibile [-MAXLONGINT-1, MAXLONGINT];
comp(ocupa 8 octeti fara semn) , intervalul de valori posibile $[-2^{63}, 2^{63}-1]$

Expresiile aritmetice care folosesc tipul intreg au ca operatori : +,-,*,/, div, mod, shl, shr, not, or, xor, and, operatori relationali (<,>,=,<=,>=,<>), precum si functii standard: abs(x), sqr(x), trunc(x), round(x), succ(x), pred(x), ord(x), ultimele fiind specifice oricarui tip ordinal.
trunc(x) – calculeaza partea intreaga a unui numar real, cu rezultat intreg
round(x) – calculeaza rotunjirea lui x , prin adaos, cu rezultat intreg
succ(x) – calculeaza succesorul lui x
pred(x) – calculeaza precedentul lui x
ord(x) - se aplica unui caracter si intoarce numarul intreg asociat acestuia in codul ASCII

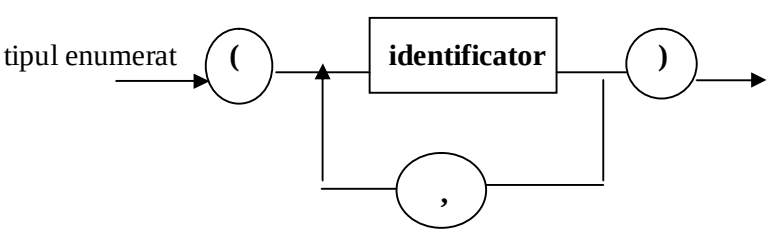
I.3.6.3 TIPUL BOOLEAN - ocupa 1 octet, are 2 valori: false si true (corespunzator cu 0 si 1)
Expresiile care folosesc tipul logic au operatori(pe biti) care sunt : and, or, xor, not, si respectiv operatori relationali care sunt : <, >, =, <=, >=, <>. Functiile standard utilizate care lucreaza cu acest tip sunt : ord() ord(true)=1, ord(false)=0;
odd() → odd(i)= 0, daca i=par si odd(i)= 1, daca i este impar, unde i este intreg oarecare

I.3.6.4 TIPUL CHAR - ocupa 1 octet.Exista 256 de caractere care formeaza codul numit ASCII, fiecaruia corespunzandu-i un numar din intervalul [0, 255] (Anexa_1)
exemple: #13, ^a, ‘a’, ‘A’. (de exemplu : caracterului ‘A’ i se asociaza numarul 65)
Operatorii care se pot aplica datelor de acest tip sunt comuni datelor de tip ordinal, operatorii relationali (<,>,=,<=,>=,<>) Functiile standard care utilizeaza date de tip caracter sunt de asemenea tipice oricaror date de tip ordinal : succ(c), pred(c), ord(c), unde c este un caracter , si in plus functia : chr(i), i intreg din intervalul [0, 255])
ord(c) sunt functii inverse una celeilalte: chr(ord(c))=c si ord(chr(i))=i
Compararea a doua caractere corespunde compararii numerelor de ordine, asociate lor in setul extins de caractere, deci in codul ASCII : ‘A’<’B’, deoarece 65<66 (ord(‘A’)< ord(‘B’))
Caractere de control: #7(asociaza un sunet), #8(deplasare stanga ←),#10(deplasare jos ↓);
Efecte speciale : #9(deplasare peste cateva coloane ,<tab>), #13(deplasare la cap de rand,<home>);

I.3.6.5 TIPUL ENUMERAT:
Este definit printr-un sir de obiecte de acelasi fel, asezate in ordine crescatoare, incepand cu pozitia 0, intr-o maniera ca cea urmatoare:

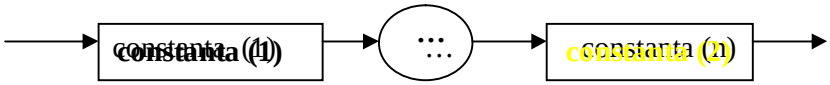
const saptamana= (luni, marti, miercuri, joi, vineri)
0 1 2 3 4

identifi



Datele de tip enumerat nu se pot citi sau afisa, valorile lor putand fi modificate numai prin atribuiri.

I.3.6.6 TIPUL INTERVAL(SUBDOMENIU)
Defineste o submultime a domeniului corespunzator unui tip de data ordinal. Notiunea in sine este analoga celei din matematica, dar cu o notatie diferita:
var natural : 0..MAXINT;(subdomeniu al tipului INTEGER)
cifre :0..9;
litere_mici :‘a’..’z’;
tipul subdomeniu



Operatiile si functiile sunt cele permise de tipul predefinit al componentelor sale.

Exercitii:

Fie var i,j,k:integer;
a, b, c: real; p, q, r : boolean;
majuscula: ‘A’..’Z’;
zi: (luni, marti, miercuri);

Sa se calculeze:

a) chr(ord(majuscula)-ord(‘A’)+ord(‘a’)); unde majuscula=’B’;
b) a+exp(c*ln(abs(b))); succ(succ(pred(marti))); unde b=9.78, c=2.,a=2.
c) abs(j)<=5; d) p or not q or r; unde: p=true, q=false, c=true, j=-3
d) (zi<=miercuri) and (zi>= luni)

Raspunsuri:

a) b
b) 881.17 ; miercuri
c) d) e) true

Codul Pascal corespunzator, care rezolva exercitiile cu exceptia b), partea a doua:

```
var i,j,k:integer;a,b,c:real;p,q,r:boolean;
  majuscula:'A'..'Z';zi:(luni,marti,miercuri);
  begin
  majuscula:='B'; p:=true;q:=false;r:=true;a:=2;b:=2;c:=9.78;j:=3;
  writeln(chr(ord(majuscula)-ord('A')+ord('a')),
       a+exp(c*ln(abs(b))),
       abs(j)<=5,p or not q or r,
       (zi<=vineri) and (zi>=luni));
  readln;
  end.
```

I.4 STRUCTURI DE CONTROL (INSTRUCTIUNI DE BAZA)

Codul instructiunilor de baza este prezentat, insotit de pseudocodul asociat conform cu paragraful 2, fara prescurtari, scriind pe larg cuvintele: C devine conditie; A,B,..D,E devin instructiune_1, instruc-tiune_2, etc. Instructiunile pot fi inlocuite cu secvente de instructiuni (instructiuni compuse).

pseudocod :l.romana	**cod Pascal**
I.4.1. Structura alternativa: (IF) daca conditie atunci instructiune_1 altfel instructiune_2	{instructiunea decizionala} if conditie then instructiune_1 else instructiune_2;
I.4.2. Structura selectiva (CASE) selectie (selector): constanta_1: instructiune_1 constanta_2: instructiune_2 ……. constanta_n: instructiune_n-1 altfel instructiune_n sfarsit selectie	{ instructiunea de selectie} selector=variabila de tip ordinal case selector of constanta_1:instructiune_1; constanta_2 :instructiune_2; …… constanta_n: instructiune_n-1 else instructiune_n; end;

I.4.3. Structuri repetitive

I.4.3.1 Structura repetitiva cu conditie initiala (WHILE) atat timp cat conditie executa instructiune	{instructiunea repetitiva conditionata anterior} while conditie do instructiune;
I.4.3.2 Structura repetitiva cu test final (REPEAT) repeta instructiune pana cand conditie	{instructiunea repetitiva conditionata posterior} repeat instructiune until conditie
I.4.3.3 Structura repetitiva cu contor (FOR)	{instructiunea repetitiva cu contor}

pentru contor=valoare_1, valoare_2, pas executa instructiune	for contor =valoare_1 to (downto) valoare_2 do instructiune
I.4.3.4 Structura de salt neconditionat (GO TO) mergi la eticheta	{instructiunea de salt neconditionat } go to eticheta
I.4.3.5 Structura secventiala (atribuire sau compusa) identificator← expresie	identificator:= expresie; {atribuire} { instructiune compusa } begin instructiune_1; instructiune_2; … end;

I.5. CITIREA SI SCRIEREA DATELOR

I.5.1 Proceduri standard de intrare/iesire

Introducerea si afisarea datelor se realizeaza in limbaj Pascal in mod standard cu ajutorul unor proceduri aflate in unitul SYSTEM, si deci apelate automat de program : read (lista de parametri), readln(lista de parametri), write(lista de parametri), writeln(lista de parametri), unde “lista de parametri” este un sir de identificatori de variabile, separati prin virgula.

Primele doua se utilizeaza pentru introducerea datelor, celelalte doua pentru afisarea acestora. Deosebirea intre cele doua alternative in aceea ca varianta cu sufixul “ln”(line) realizeaza atat citirea(scrierea) cat si saltul la randul urmator. Transferul in bufferul de memorie se executa la apasarea tastei Enter (<CR>+<LF>)

I.5.2. Afisarea datelor este implicit intr-un format de lungime maxima, numit “stiintific” sau exponential.Apelul urmator

writeln(a); pentru a=5,78 va avea ca rezultat:

5.77999999999884E+0000

ceea ce , matematic reprezinta : 5,77999999999884 * 10 º,dificil de tradus. Alternativa explicita este afisarea cu format, in primul rand pentru datel de tip numeric, dar nu numai. Astfel, vom scrie pentru:

- tipul intreg, caracter, sir de caractere, logic (boolean) :
  writeln(identificator:lungime_de_afisare);
  exemplu:
  a:=10; writeln(a:4);
  Rezultat :blancblanc10 (lungimea de afisare este 4, aliniere la dreapta)
- tipul real:
  writeln(identificator:lungime_de_afisare :lungime_parte_zecimala)
  exemplu :
  a:=9.8 ; writeln(a:4:2);
  Rezultat :9.80 (lungimea de afisare este 4, din care doua cifre zecimale)

I.5.3 Procedurile standard de citire si scriere pot fi apelate fara argumente :

```
Readln; {asteapta apasarea unei taste}
Writeln; {sare un rand (scrie un rand liber)}
```

I.5.4 Observatii :

-Datele de tip enumerat nu pot fi citite si afisate, deci nu constituie argumente ale procedurilor standard de citire sau scriere.

-Datele de tip logic(boolean) nu se pot citi, deci nu sunt apelate de proce-

durile standard de citire.

I.6 EXPRESII

In functie de tipul rezultatului, expresiile pot fi: aritmetice, logice, sir de caractere sau caracter, utilizand operatorii si functiile specifice. Prioritatea lor este(descrescator) :

1º unari (not)

2º binari : a) multiplicativi (*, /, div, mod, shl, shr, and)
b) aditivi (+, -,or, xor)
c) relationali (<,>,=,<=,>=,<>, in)

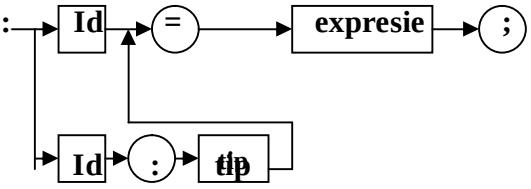
I.7. DEFINIREA CONSTANTELOR, DECLARAREA VARIABILELOR

A) Definirea constantelor simbolice utilizator se realizeaza in sectiunea CONST. Dupa cum am vazut in paragraful 1.3.4, constantele simbolice sunt constante desemnate prin identificatori (Id), reprezentand valori care nu se pot modifica pe parcursul executiei programului, si pot fi : intregi, reale, booleene, sir de caractere.

In plus, in sectiunea CONST se pot defini **constante cu tip** , de fapt, **variabile initializate**, ale caror valori se vor putea modifica in cursul executiei programului.

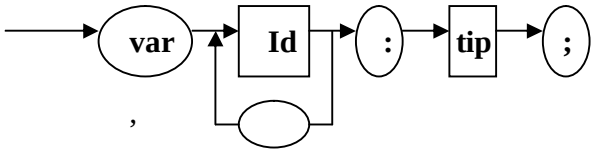
```
CONST a=123.45;
      n=1001;
      impar:boolean=odd(n);
```

sintaxa :



B) Declararea variabilelor

Fiecarei variabile i se asociaza un identificator si un tip de data, ceea ce se realizeaza printr-o declaratie de variabile, de forma :




```
VAR  cod:byte; x,y,z:real;
```

I.8. ALGORITMII SIMPLI, SELECTATI DE PROGRAMA

1. Calcul putere rapida(algoritmul lui Al Kashi)
2. Trecerea unui numar dintr-o baza in alta
3. Verificarea caracterului prim al unui numar natural
4. Algoritmul lui Euclid pentru calculul c.m.m.d.c. si c.m.m.m.c
5. Sirul lui Fibonacci

Rezolvarea se va realiza in pseudocod, dupa prezentarea continutului(textului) fiecarei probleme.

Se cere ca, potrivit pseudocodului , sa se scrie codul Pascal(C++) asociat.

PSEUDOCODURI
(varianta in l.romana)

1. ALGORITMUL LUI AL-KASHI (CALCUL PUTERE RAPIDA)

Se calculeaza b la puterea n, printr-un numar minim de operatii(inmultiri)
p= puterea, b= baza, exp= exponentul, nr=numar inmultiri, n= exponent initial
p←1; nr←0
citeste b si exp
n←exp

```
atat timp cat exp>0 executa
        daca odd(exp) atunci
                        ┌   p←p*b
                        └   exp←exp-1
      altfel    ┌ b←b*b
                └ exp←exp div 2
    nr←nr+1
scrie(p) scrie (nr-1)
```

2. TRECEREA DIN BAZA A IN BAZA B

Se schimba baza de numeratie a numarului n, din a in b. Algoritmul se va referi la numere a nemultiple ale lui b Se va lucra in doua etape:
1) trecerea de la baza a la baza 10 2) trecerea de la baza 10 in baza b.
Citeste n, a , b n=numar, a si b= bazele de numeratie r=rest.

```
ETAPA 1 (trecerea din baza a in baza 10)
citeste n, a, b
p←1 m←0
   ┌ atat timp cat n>0 executa
   │                                  ┌   m←m+(n mod 10) *p
   │                                  │   p←p*a
   └                                  └   n←n div 10
ETAPA2 (trecerea din baza 10 in baza b)
i←0
repeta          ┌   r←m mod b
                │   m←m div b
                │   i←i+1
                └   pozitionare 79-i    scrie r
pana cand m=0
```

3. VERIFICAREA CARACTERULUI PRIM AL UNUI NUMAR !

Algoritmul verifica daca un numar este prim:
semafor=1 n=numarul dat i=contor (divizor posibil!)

```
citeste n
semafor←true
pentru i=2 , trunc(sqrt( n ))  , 1 executa
                        daca n mod i=0  atunci semafor ←false
daca semafor =true atunci scrie'prim'
                              altfel scrie  'nu e prim'
```

4. ALGORITMUL LUI EUCLID

Algoritmul calculeaza cmmdc a doua numere.
Fie a,b= numerele date p= produsul lor r=restul

```
citeste a si b.
r←a mod b
```

```
atat timp cat   r>0  executa
                        a←b
                        b←r
                        r←a mod b

scrie (cmmdc=b)
```

5. SIRUL LUI FIBONACCI

Se citeste n numar natural. Daca n este in sirul lui Fibonacci se va afisa un mesaj corespunzator daca nu se va incerca scrierea numarului citit ca suma de numere ale sirului.
Se stie ca in sir, orice termen al acestui sir este suma a doi termeni consecutivi (precedentii termenului dat) Se dau primele valori din sir : 0 si 1.
De asemenea se foloseste o variabila booleana initializata “true” cu rol de semafor .
Vor fi prezentate doua variante ale acestui algoritm, care nu difera esential Deosebirea este ca prima varianta este scrisa iterativ, fara a utiliza structurarea in timp ce a doua rezolva cerintele problemei intr-o procedura iterativa, apelata din programul principal, al carui pseudocod este de asemenea prezentat.

VARIANTA ITERATIVA

```
citeste n
semafor ←true
repeta
 a←0
 b←1
 c←a+b
atat timp cat c<n executa   a←b
                            b←c
                            c←a+b
Daca c=n atunci
                daca semafor = true atunci scrie “numar Fibonacci”
                  altfel scrie c
                n←0 (iesire fortata)
altfel
      scrie b
      semafor←false
      n←n-b
pana cand n=0
```

```
VARIANTA  STRUCTURATA
procedura fibo(a,b,c, n, semafor)
daca c=n atunci
                  daca semafor = true atunci scrie “numar Fibonacci”
                     altfel scrie c
                  n←0 (iesire fortata)
altfel
      scrie b
      semafor←false
      n←n-b

program principal
citeste n       semafor ←true
    repeta
    a←0
    b←1
   c←a+b
 atat timp cat c<n executa   a←b
                             b←c
                             c←a+b
 fibo(a,b,c,n,semafor)
     pana cand n=0
```

Solutiile cerintei de la &I.7, pg. 14 (coduri Pascal asociate pseudocodului prezentat)

1. CALCUL PUTERE RAPIDA

```
uses crt;
var i,j,nr,n,exp:byte;p,b:longint;
begin
p:=1;nr:=0;
writeln('Baza si exponentul va rog:');
readln(b);readln(exp);n:=exp;
while exp>0 do begin
if odd(exp) {impar} then begin
p:=p*b;exp:=exp-1;end
else
begin
b:=sqr(b);exp:=exp div 2;
end;
nr:=nr+1;
end;
writeln('Rezultatul ',p,' Nr de operatii: ',nr-1);
end.
```

2.TRECEREA UNUI NUMAR NATURAL DIN BAZA A IN BAZA B

Operatia de trecere de schimbare a bazei de numaratie se bazeaza, conform

pseudocodului pe executarea impartirii repetate a numarului dat la baza in care vrem sa-l trecem. In programul care urmeaza se face trecerea din baza zecimala in baza binara, a unui numar natural .

```
Program baza10_baza2;
 uses crt;
 var n,r,i:word;
 begin
 clrscr;
writeln('numarul in baza 10:');
readln(n);i:=0;
gotoxy(40,10);textcolor(14);
write('numarul in baza 2:');gotoxy(79,10);
textcolor(4);
repeat
  r:=n mod 2;
  n:=n div 2;
  i:=i+1;
  gotoxy(79-i,10); write(r);
  until(n=0);
  readln;   end.
```

Generalizand, trecerea de la o baza a la o alta baza b se poate realiza cu ajutorul bazei intermediare 10.

```
Program baza_a_baza_b;
 uses crt;
 var p,n,r,m,i,a,b:word;
 begin
 clrscr;
writeln('numarul si bazele:');
readln(n,a,b);
{trecerea din baza a in baza 10}
p:=1;m:=0;
repeat
m:=m+(n mod 10)*p;
p:=p*a;
n:=n div 10;
until n=0;
write('numarul in baza 10:',m);
{trecerea din baza 10 in baza b}
gotoxy(40,10);textcolor(14);
write('numarul in baza:',b);gotoxy(79,10);
textcolor(4);
i:=0;
repeat
r:=m mod b;
m:=m div b;
write(r); i:=i+1;
gotoxy(79-i,10);
until(m=0);
readln;   end.
```

De exemplu, numarul 1101, scris in baza 2, va fi 111, scris in baza 3, intermediar 13, scris in zecimal.

3. VERIFICAREA CARACTERULUI PRIM AL UNUI NUMAR NATURAL

Verificarea primalitatii unui numar n se va face testand toate numerele mai mici decat √n, in sensul de a fi sau nu divizor al lui n.

```
uses crt;
var sema:boolean;
n,i:word;
begin
clrscr;
sema:=true;
readln (n);
for i:=2 to trunc(sqrt(n)) do
if n mod i=0 then sema:=false;
if not sema then writeln ('nu este prim')
else writeln ('este prim');
end.
```

4. ALGORITMUL LUI EUCLID

Dupa cum se stie, acest algoritm constand in impartiri repetate, ne conduce, cu ultimul rest inainte de restul 0, la obtinerea cmmdc a doua numere.

```
var d,r,p,i:word;
begin
readln (d,i);
p:=d*i;
r:=d mod i;
while r>0 do begin d:=i; i:=r;r:=d mod i;end;
writeln ('c.m.m.d.c=',i);
readln; end.
```

5. SIRUL LUI FIBONACCI

Istoria sirului dateaza din evul mediu, iar autorul problemei era calugar; primii doi termeni sunt 0 si 1 iar fiecare termen urmator al sirului se obtine prin adunarea celor 2 precedenti.Termenii sunt retinuti in variabilele a,b,c. Daca numarul n dat se va afla printre termenii calculati al sirului,atunci el este un numar “Fibonacci”.

```
var a,b,c,n:word;sema:boolean;
begin
readln(n); sema:=true;
repeat
a:=0; b:=1; c:=a+b;
while c<n do begin a:=b;b:=c;c:=a+b;end;
if c=n then begin if sema then write(n,'  Fibonacci')
else write(c);
n:=0;
end;
else begin
write(b,'+'); sema:=false;
n:=n-b;
end;
until n=0;
readln;end.
```

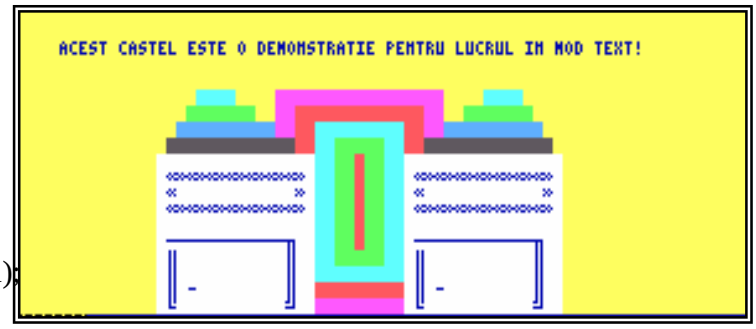
PROBLEME DIVERSE

1. Construiti un Program demonstrativ pentru lucrul in mod text, utilizand structurile studiate.

Solutie : (in plus, pentru design, sunt utilizate si procedurile standard de stergere, colorare, pozitionare in ecran, sunet, intarziere, desenarea de ferestre si caractere grafice)

----------A: \BAC\INTRO\CASLE.PAS ----------

Acest program este o aplicatie a modului de lucru text}




```
¦Program castel;
¦uses crt;
¦var i,j,k:integer;ch:char;
begin
¦clrscr;textbackground(1);
¦for i:=1 to 10 do begin clrscr;
¦textbackground(i+1);
¦window(30+i*2,5+i,50-i*2,20-i);
end;
¦for i:=1 to 5 do begin  clrscr;
¦textbackground(i+3);window(25-i,5+i,25+i*2,19);end;
¦for i:=1 to 5 do begin
clrscr;
¦textbackground(i+3);window(55-2*i,5+i,55+i,19);end;
¦clrscr;textbackground(1);
¦window(1,1,79,24);textcolor(14);
¦gotoxy(20,3);write('ACEST CASTEL ESTE O DEMONSTRATIE LA MODUL TEXT!');
¦textbackground(0);
gotoxy(21,11);write('«»«»«»«»«»«»«»«»');
¦gotoxy(21,12);write('«           »');gotoxy(21,13);write('«»«»«»«»«»«»«»«»');
¦gotoxy(46,11);write('«»«»«»«»«»«»«»«»');gotoxy(46,12);write('«           »');
¦gotoxy(46,13);write('«»«»«»«»«»«»«»«»');
¦gotoxy(21,15);write('------------+');
¦gotoxy(21,16);write('¦           ¦');
¦gotoxy(21,17);write('¦           ¦');
¦gotoxy(21,18);write('¦ -         ¦');
¦gotoxy(21,19);write('+           +');
¦gotoxy(46,15);write('------------+');
gotoxy(46,16);write('¦           ¦');
¦gotoxy(46,17);write('¦           ¦');
¦gotoxy(46,18);write('¦ -         ¦');
¦gotoxy(46,19);write('+           +');
¦for i:=1 to 30 do begin sound(50*i);delay(100);end;
¦for i:=1 to 30 do begin sound(50*i);delay(100);end;
¦for i:=1 to 30 do begin sound(1000 div 100*i);delay(100);end;
¦for i:=1 to 30 do begin sound(1000 div 100*i);delay(100);end;
¦for i:=1 to 30 do begin sound(50*i);delay(100);end;
¦for i:=1 to 30 do begin sound(1000 div 100*i);delay(100);end;
¦for i:=1 to 30 do begin sound(50*i);delay(100);end;
¦for i:=1 to 30 do begin sound(1000 div 100*i);delay(100);end;
¦for i:=1 to 30 do begin sound(50*i);delay(100);end;
¦for i:=1 to 30 do begin sound(1000 div 100*i);delay(100);end;
¦nosound;readln(ch);
¦ch:=readkey
```

¦end.

2. Sa se calculeze suma : $\sum_{i=1}^{n} x^{2i-1}/(2i-1)!$

Solutie (cod Pascal) :

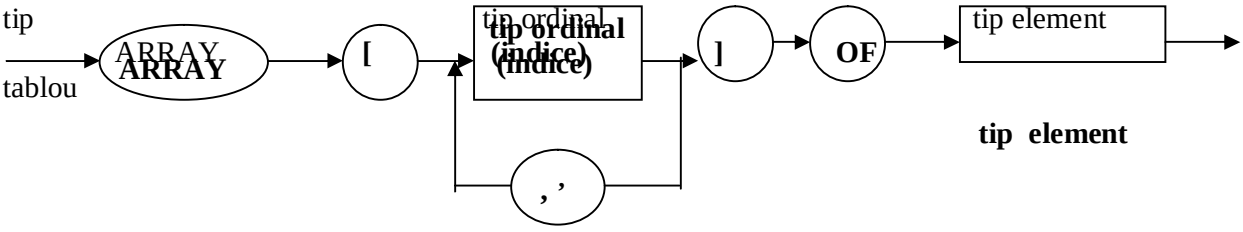
```
program calcul_sume_diverse;
 var x,n,p1,p2,i,k:integer;
    s:real;
 begin
 readln(n,x);
 s:=0;
 for i:=1 to n do
  begin
  p1:=1;
  for k:=1 to 2*i-1 do p1:=p1*x;{calcul putere}
  p2:=1;
  for k:=1 to 2*i-1 do p2:=p2*k; {calcul factorial}
   s:=s+p1/p2;
  end;
 writeln('s=',s:4:2);
 readln;
 end.
```

CAPITOLUL II

TIPURI STRUCTURATE DE DATE

I. TIPUL TABLOU

Un tablou este o structura cu numar fix de componente, toate de acelasi tip. Tipul componentelor este tipul de baza al tabloului. Tipul inducelui fixeaza numarul ma-xim de elemente ale tabloului. Sunt permise mai multe tipuri index,separate prin virgule. Pentru tipul elementelor se poate alege orice tip standard sau utilizator, simplu sau structurat. Se poate defini un tablou in sectiunile :type, const, sau var ale unui Program. Diagrama de sintaxa:



OBS :Tipul ordinal mentionat la indice va fi de maxim 256 valori, deci : byte,char,boolean sau subdomenii ale tipurilor ordinale cu mai mult de 256 elemente : word, integer, longint.
Tipul elementului unui tablou poate fi predefinit in Pascal sau definit de utilizator.

Exemplul 1:
```
type matrice=array[1..5,1..5] of integer;
 var a,b:matrice;
```
Exemplul 2:
```
type vector=array[1..5] of byte;
const a:vector=(2,3,7,250,8);
```
sau, pentru bidimensional:
```
type vect=array[1..2,1..3] of integer;
const b:vect=((-2,7,8),(4,0,-20));
```
Cu elementele unui tablou se pot face toate operatiile admise de tipul de baza al tabloului.
Selectarea elementului se face utilizand indicele : (pentru exemplele de mai sus)

a[i], b[i,j], unde i=indicele de linie, j=indicele de coloana

A. ALGORITMII SIMPLI SELECTATI DE PROGRAMA(care utilizeaza tipul structurat tablou)

A1. Aflarea maximului si minimului dintr-o lista de elemente.

A2. Sortari (ordonari):

- sortarea prin selectie directa
- sortarea dupa minim sau maxim
- sortarea prin numarare
- metoda bulelor(Bubble -Sort)
- metoda sortarii Merge-Sort
- metoda sortarii Quick-Sort
- metoda sortarii matematice(Donald Shell)
- metoda sortarii prin interclasare

A3. Interclasarea a doua siruri

A4. Cautarea simpla si binara

In cele ce urmeaza vom considera drept “complexitate”, numarul de operatii efectuate de algoritm, si o vom nota O(e(n)) , la argument fiind o expresie e(n) unde n este un numar natural, de obicei un numar legat de o operatie care se contorizeaza in algoritm (uzual o comparatie).In marea majoritatea cazurilor n se refera la numarul de date, dar nu cele initiale. Expresia e(n) poate fi : un polinom in variabila n(complexitate polinomiala), o expresie exponentiala de argument n si baza naturala(complexitate exponentiala), o expresie logaritmica de argument n si baza naturala(complexitate logaritmica). In cazul unui polinom se mai analizeaza si gradul in raport cu variabila argument: daca gradul este 1, complexitatea va fi polinomiala O(n), daca este 2, complexitatea va fi $O(n^2)$, etc.

A1. Aflarea maximului si minimului dintr-o lista de elemente

max←a1

pentru i=1,n,1 executa

daca max<ai atunci max←ai

min←a1

pentru i=1,n,1 executa

daca min>ai atunci min←ai

```
PrProgram max_min;
Var a:array[1..10] of integer;
      n,i:byte;  max,min:integer;
begin
write('Dati dimensiunea vectorului:');
readln(n);
writeln('Dati elementele vectorului:');
for i:=1 to n do readln(a[i]);
max:=a1;
for i:=1 to n do if max<a[i] then
max:=a[i];
min:=a1;
for i:=1 to n do if min>a[i] then
min:=a[i];
writeln('Maximul=',max:4,'Minimul=',
min:4);
readln;end.
```

A2.Algoritmii de sortare(ordonare), asa cum sugereaza si denumirea lor, realizeaza asezarea elementelor unui vector (multime) in ordine crescatoare sau descrescatoare si sunt diversi. Cei prezentati in cele ce urmeaza reprezinta numai o parte dintre acestia .
Sortarea prin selectie directa

```
pentru i=1, n-1,1 executa
    pentru j=i+1, n,1 executa
        daca ai>aj atunci schimba(ai,aj)
```

Schimbarea lui ai cu aj se realizeaza intr-o procedura al carei continut este reprezentat de algoritmul urmator:

```
procedure schimba(ai,aj)
aux=   variabila auxiliara, de acelasi tip cu ai, aj
        aux←ai
        ai←aj
        aj←aux
```

Complexitatea este de ordin “polinomial”, adica $O(n^2)$, deoarece exista doua iteratii, imbricate (una in interiorul celeilalte) de ordin n. Analog pentru algoritmii de sortare urmatori .

Sortarea dupa minim –varianta cu multimi

Fie vectorul a de n componente distincte; consideram multimea y a componentelor sale si contorul k, initializat nul, care numara elementele noului vector b, obtinut din a dupa ordonare.
O variabila booleana vb, semnaleaza obtinerea fiecarui element al vectorului b.
De retinut : aceast algoritm nu poate fi aplicat decat unui sir de elemente distincte, deoarece utilizeaza notiunea de multime, in care unul, doua sau mai multe elemente egale sunt inscrise o singura data.

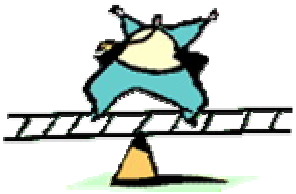
```
k←0(* y contine elementele vectorului a care
                stim ca sunt distincte*)
        y←{a1,a2,..,an}
        repeta
        vb←true
  min← maxval (minimul se initializa si cu un nr. f. mare)
        pentru i=1,n,1 executa
                daca min>ai si ai ∈ y atunci:
                        min←ai
                        vb←false
        y=y-{min}  k←k+1;  bk←min
        pana cand vb
```

Exercitiu:
1.Rescrieti algoritmul dat mai sus, urmarind maximul in loc de minim

Sortarea prin numarare

Se numara cate elemente din sirul dat sunt mai mici decat elementul comparat apoi se reaseaza in ordinea crescatoare a predecesorilor fiecarui element. Un vector auxiliar k, contine numarul predecesorilor .

```
citeste ai,  cu i=1,n          pentru i=1,n,1 executa ki←0
  (ordonarea vectorului a)
  pentru i=1,n-1,1 executa
                        pentru j=i+1,n,1 executa
                            daca ai<aj atunci kj←kj+1
                            altfel ki←ki+1
     pentru i=1,n,1 executa bki +1←ai
```



Sortarea cu bule (Bubble-Sort)

```
citeste ai, cu i=1..n si utilizeaza o variabila semafor vb
     repeta
     vb←true
pentru i=1,n-1,1 executa:
                  daca ai>ai+1 atunci
                                schimba(ai, ai+1)
                                vb←false
pana cand vb
```

Sortarea Merge-Sort

s=marginea stanga(primul indice din sir) d=marginea dreapta(ultimul indice din sir)

Algoritmul este recursiv, varianta iterativa ramane exercitiu.

```
                procedura Merge(s,d)
   vb←true
   pentru i=1,n-1,1 executa
            daca ai>ai+1 atunci :
                                schimba(ai,ai+1)
                                vb←false  j←i+1
   daca vb=false (sirul nu este inca ordonat) atunci:
                                k←[(s+d)/2]
                                daca j<k atunci merge(s,k)
                                altfel merge(s,k+1)
```

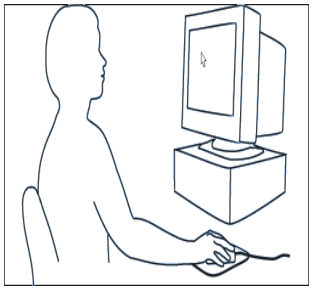
Complexitatea scade prin injumatatirea vectorului si utilizarea variabilei booleene, ramanand de ordin polinomial . Analog pentru algoritmul :

Sortarea Quick-Sort

```
citeste ai, cu i=1..n
     pentru i=1,n-1,1 executa
pentru j=i+1,n,1 executa
                  daca ai>aj atunci  schimba(ai, aj)
```

Complexitate polinomiala. Varianta cu semafor(variabila booleana) a acestui algoritm va fi analoga sortarii cu bule, cu deosebirea ca sunt utilizati doi indici i si j in loc de unul singur :

```
 citeste ai, cu i=1..n
     repeta
  semafor←true
    pentru i=1,n-1,1 executa
    pentru j=i+1,n,1 executa  daca ai>aj atunci
                                schimba(ai, aj)
```



```
                    semafor←false
  pana cand semafor=true
```

Sortarea matematica (Donald Shell)
Se utilizeaza vectorul a, numarul n si indicele e (indice de lucru)o variabila auxiliara aux si una booleana vb.

```
e←n
repeta
s←e div 2
repeta
    vb←true
    pentru i=1,n-e,1 executa
                    j←i+e
                    daca aj<ai atunci
                            ai←aj
                            vb←false
pana cand vb
pana cand e=1
```

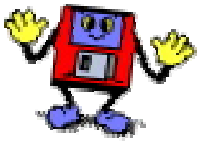
Sortarea prin interclasare

Se utilizeaza s si d, cu aceleasi semnificatii ca si la sortarea rapida, doi vectori a si b, a neordonat iar b ordonat, de aceasi dimensiune, variabila m pentru indicele de mijloc, si contorii k,i,j. Complexitatea,desi ramane de ordin polinomial, se micsoreaza prin injumatatirea numarului de operatii (sortarea pe jumatati, realizata recursiv)

procedura sortinc(s,d) (varianta recursiva a metodei <u>divide et impera</u>))

```
daca s<d atunci
        m←[(s+d)/2]
        sortinc(s,m)
        sortinc(m+1,d)
        interclasare
```

procedura interclasare (varianta iterativa)
(se va lucra cu 3 contori, i, j, k si se va calcula al patrulea – indicele m) m←(s+d)/2 i←s j←k+1 k←s-1
(se da vectorul a si se construieste vectorul b)
s= indice stanga; i retine indicele s ; k retine mai putin cu o unitate decat i ; j retine mai mult cu o unitate decat k; m retine indicele elementului din mijlocul vectorului; contorii i si j urmaresc elementele vectorului dat a, iar k pe cele ale vectorului care se va obtine b.



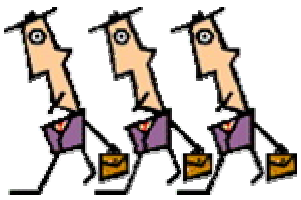
```
repeta
    k←k+1
    daca ai<aj atunci
            bk←ai
            i←i+1
        altfel
            bk←aj
            j←j+1
pana cand (i>s) sau (j>d)
```

```
daca i>s atunci
                pentru i=j,d,1 executa
                        k←k+1
                        bk←ai
        altfel
                pentru j=i,s,1 executa
                        k←k+1
                        bk←aj

pentru i=s,d,1 executa ai←bi
```

A3. Interclasarea a doua liste

Vom considera vectorii a si b, ordonati, care nu au obligatoriu aceeasi dimensiune. Fie m si n dimensiunea primului, repectiv a celui de-al doilea vector. Vectorul c va contine componentele celor doi vectori a si b, scrise o singura data.

```
i←1  j←1 k←1
repeta
        daca ai<=bj atunci :
                ck←ai
                i←i+1
        altfel
                ck←bj
                j<---j+1
k←k+1
pana cand (i>m) sau (j>n)
```



(Daca unul dintre vectorii a sau b si-a epuizat elementele, atunci vectorul c se va completa cu elementele ramase.din celalalt vector.)

```
atat timp cat (i<=m) executa :
                ck←ai
                i←i+1
                k←k+1

atat timp cat (j<=n) executa :
                ck← bj
                j←j+1
                k←k+1
pentru i=1,k-1,1 executa scrie ci
```

A4. Cautarea simpla, cautarea binara

Algoritmii de cautare, verifica, dupa cum sugereaza si denumirea, existenta unui element intr-o multime(sir, lista, vector dat) de elemente.

a) Cautarea simpla

Complexitatea este de ordin polinomial, si deoarece se executa numai maxim n cautari, notam O(n)

a=vector de elemente date , x=element cautat

```
vb← false
pentru i=1,n,1 executa
                    daca ai=x atunci vb←true
daca vb atunci scrie ‘Gasit’
          altfel scrie  ‘Negasit’
```

b) Cautarea binara

Acest mod de cautare presupune impartirea consecutiva a vectorului dat in doi vectori de lungimi egale, compararea valorii cautate cu elementul din mijloc si continuarea cautarii numai pe unul dintre cei doi vectori-jumatati, in functie de rezultatul acestei comparari. Ceea ce se utilizeaza, deci variabilele sunt prezentate anterior algoritmului:

a, x, s,d, k(s=stanga, d=dreapta, x=element cautat, k=contor de mijloc, a=vector)

variante recursive

```
procedura caut_bin1(s,d)
daca a(s)=x atunci scrie ‘gasit’
altfel
daca s<d atunci
          k←[(s+d)/2]
          caut_bin1(s,k)
          caut_bin1(k+1,d)
```

Aceasta varianta are avantajul ca sirul nu este ordonat dar si neajunsul unui numar de operatii relativ mai mare .Cea care se utilizeaza cel mai des este urmatoarea :

```
procedura caut_bin2(s,d)
daca s<=d atunci
          k←[(s+d)/2]
          daca x=ak atunci scrie ‘gasit’
          altfel daca x<ak atunci caut_bin2(s,k)
          altfel caut_bin2(k+1,d);
```

Complexitatea este in acest caz de ordinul O(log(n)) deci algoritmul este mai bun dar dezavantajul este necesitatea sortarii vectorului a in prealabil.

(log(n) < n, log(n) < n^2, etc.)

Ca varianta iterativa este prezentata alternativa celei precedente, cu aceeasi mentiune a necesitatii ordonarii prealabile a sirului de elemente.

varianta iterativa procedura caut_bin3

```
s←1  d←n
 repeta
 daca s<d atunci
                    k←[(s+d)/2]
                    daca x=ak atunci
                                        scrie ‘gasit’
                                        exit
                              altfel
                                        daca x<ak atunci d←k
                                         altfel s←k+1
```

pana cand k=0

Exercitii (solutii in pseudocod)

3. Ordonati crescator o matrice bidimensionala, in raport cu elementele de pe diagonala principala.
4. Ordonati crescator o matrice bidimensionala, in raport cu elementele de pe diagonala secundara.
5. Ordonati crescator o matrice bidimensionala, in raport cu elementele de pe linii.

SOLUTIILE EXERCITIILOR DE LA CAPITOLUL II, PRAGRAFUL I

Exercitiul 1, de la paragraful &1(sectiunea Algoritmii simpli din Programa)
Cerinta era: transformarea algoritmului de o maniera in care sortarea sa se realizeze prin insertie dupa maxim, nu dupa minim :

```
                k←0(* y contine elementele vectorului a care
                      stim ca sunt distincte*)
                y←{a1,a2,..,an}
                repeta
                vb←true
        max← -maxint (maximul se initializa si cu un nr. f. mic)
                pentru i=1,n,1 executa
                          daca max<ai si ai ∈ y atunci:
                                                max←ai
                                                vb←false
                y=y-{max}   k←k+1;   bk←max
                pana cand vb
```

Exercitiul 2, acelasi paragraf, cerea transformarea structurata a algoritmului prezentat la sortarea rapida , primul fiind realizat in maniera nestructurata.

```
{Procedure sortare_rapida}
        repeta
    semafor←true
      pentru i=1,n-1,1 executa
      pentru j=i+1,n,1 executa  daca ai>aj atunci
                                      schimba(ai, aj)
                                      semafor←false
    pana cand semafor=true
   {          Program principal          }

      citeste ai, cu i=1..n
      sortare_rapida;
```

Exercitiul 4. Este asemanator cu cel precedent, in varianta cu contor sau fara, sortarea in raport cu elementele diagonalei secundare va tine cont, pentru interschimbarea a doua elemente de conditia :

a[i,n-i]>a[i+1,n-i-1], unde I este indicele de linie.

```
program sortare_pe_diagonala_secundara;
type indice=1..10;
     vect=array[indice] of integer;
var a:array[indice] of vect;
    sema:boolean;i,j,k,n:byte;aux:integer;b:vect;
  begin
    readln(n);for i:=1 to n do for j:=1 to n do readln(a[i,j]);
    repeat
    sema:=true;
    for i:=1 to n-1 do
                  if a[i,n-i]>a[i+1,n-i-1] then begin
                     b:=a[i];a[i]:=a[i+1];a[i+1]:=b;
                     for j:=1 to n do begin
                     aux:=a[j,n-i];a[j,n-i]:=a[j,n-i-1];a[j,n-i-1]:=aux;
                     end;
                     sema:=false;
                     end;
    until sema=true;
    for i:=1 to n do begin
         for j:=1 to n do write(a[i,j],' ');
         writeln;
         end;

         readln;
         end.
```

In continuare este prezentata solutia pentru ordonarea tuturor elementelor matricei, pe linii crescator**; conditia de schimbare: a[i,j]>a[i+1,j]**

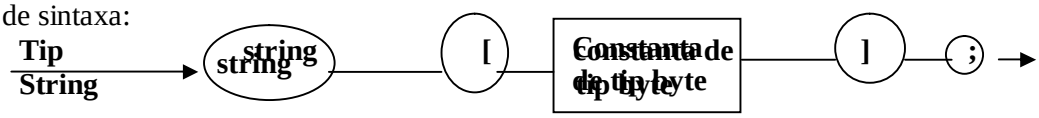
```
var a:array[1..10,1..10] of integer;
  aux,i,j,n:integer;semafor:boolean;
     begin
    read(n);
    for i:=1 to n do
    for j:=1 to n do
    read (a[i,j]);
    for j:=1 to n do begin
    repeat
    semafor:=true;
    for i:=1 to n-1 do
    if (a[i,j]>a[i+1,j]) then
    begin
    aux:=a[i,j];
    a[i,j]:=a[i+1,j];
    a[i+1,j]:=aux;
    semafor:=false;
    end;
    until semafor;
    end;
    for i:=1 to n do begin
    for j:=1 to n do write(a[i,j],' ');
    writeln;
    end;
    end.{}
```

ALTE TIPURI STRUCTURATE DE DATE

II. TIPUL STRING

Tipul string este un tip special de tablou cu elemente de tip char. El memoreaza in afara caracterelor din sir si numarul acestora (in pozitia 0 a sirului). Numarul de caractere al unei date de tip string poate varia in timpul executiei programului intre 0(sirul vid: ‘’) si o valoare maxima specificata in declaratia sirului respectiv.

Diagrama de sintaxa:



Obs: Specificarea dimensiunii dintre parantezele patrate poate lipsi, caz in care se va considera dimensiunea maxima adica 256 elemente. Modalitati de declarare :

Exemplul 1:

```
const sir:string[5]=‘Maria’;
```

Exemplul 2 :

```
type sir= string[10];
var a:sir;
```

Exemplul 3:

```
var sir, sir1, sir2: string[10]
```

Stringul este un un vector care ocupa in memorie un numar de octeti egal cu valoarea constantei dintre paranteze, plus 1, intrucat se memoreaza si lungimea efectiva a sirului.

a) Atribuirea unei variabile de tip string se realizeaza cu o expresie de tip string, dar de o lungime compatibila cu lungimea declarata(daca este mai mare, caracterele in plus se vor neglija)
b) Selectia este analoga tipului tablou : a[i], este caracterul de pe pozitia i
c) Intre doua stringuri se stabileste o relatie de ordonare(lexicografica), fie pe baza lungimii, iar daca sunt egale ca lungime, pe baza numarului de ordine in codul ASCII al componentelor de pe aceeasi pozitie.Reamintim ca tipul caracter, fiind un tip ordinal, admite operatori relationali, astfel ca un caracter este mai mare dacat altul daca numarul de ordine in codul ASCII asociat primului este mai mare decat al celui de-al doilea: ord(‘A’) < ord(‘B’)

De exemplu vom avea: ‘abc’ < ‘abcd’ dar si : ‘abc’ < ‘bcd’

c) Citirea si scrierea se realizeaza cu procedurile standard, cu observatia ca, in cazul cand variabilele nu au toate lungimea declarata se va folosi numai procedura READLN.
d) Functiile si procedurile specifice sunt cele care opereaza numai cu siruri de caractere, disponibile in biblioteca limbajului, in unitul SYSTEM. Pentru cod Pascal, de exemplu: n=length(sir) ; sir1:= copy(sir,a,b); delete(sir,a,b); insert(sir1,sir2,a); n=pos(sir1,sir2); sir=concat(sir1,sir2) etc.

Ce semnificatie au aceste functii si proceduri specifice ?

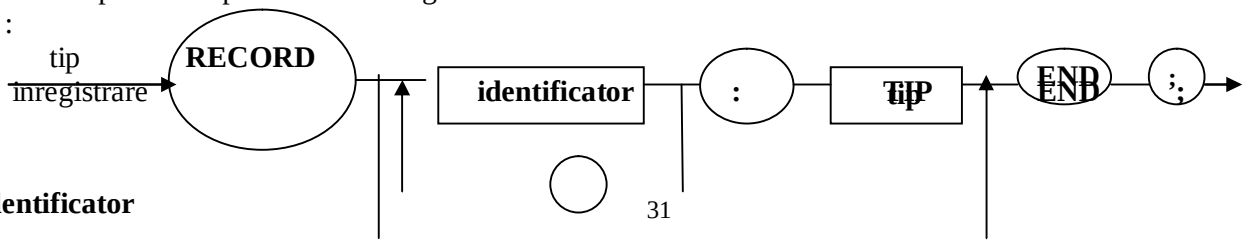
n= length(sir) este lungimea sirului *sir ;* n=pos(sir1,sir2) este indicele incepand cu care subsirul *sir1* se regaseste in sirul *sir2,* daca nu se gaseste, valoarea intoarsa este 0; sir1=copy(sir,i,j) este sirul de caractere obtinut copiind din sirul *sir,* din pozitia i pe lungimea j toate caracterele;delete(sir,i,j) efec-tueaza stergerea caracterelor sirului *sir,* incepand cu pozitia i, pe lungimea j(deci sterge j caractere incepand cu cel aflat pe pozitia i); insert(sir1,sir2,i) intercaleaza in sirul *sir1*, toate caracterele sirului *sir2,* incepand cu pozitia i, a sirului *sir1*.

e) Concatenarea se face si cu ajutorul operatorului “+”. Exemplu: ‘abc’+’def’=’abcdef’

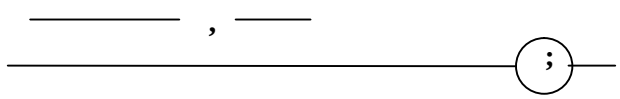
II.3 TIPUL INREGISTRARE FIXA(ARTICOL)

Tipul inregistrare fixa (ARTICOL) cuprinde un numar fix de componente numite CAMPURI, care pot avea tipuri diferite. Diagrama de sintaxa:

:



identificator



Campurile pot fi si ele de tip inregistrare; “tip” poate fi orice tip de date Pascal, predefinit sau utilizator, cu exceptia tipului FILE. Identificatorii campurilor sunt locali definitiei de tip inregistrare si invizibili in exterior. Modalitati de declarare:

Exemplul 1:
```
type  elev= record
                nume: string;
                varsta:byte;
                end;
var a:elev;
```
Exemplul 2:

Un vector cu astfel de elemente se poate declara: var tab:array[1..10] of elev;

In primul caz, accesul la camp se realizeaza cu ajutorul operatorului “.” , numit “de selectie”.

a.nume

In al doilea caz, se tine cont de faptul ca o inregistrare este un element de tablou. Ca urmare:

i----------> tab[i].nume

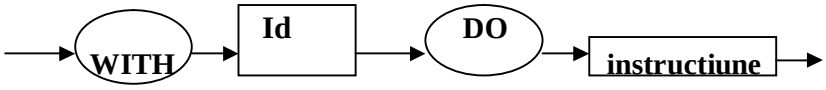
Operatiile permise sunt cele pe care le admite tipul de baza .

Instructiunea WITH : simplifica accesul la campurile unei inregistrari. Presupunand ca ID este identificatorul unei variabile de tip inregistrare, vom obtine de forma:

WITH Id DO instructiune

Sau :

WITH



instructiune

Exemplu : i:=1;

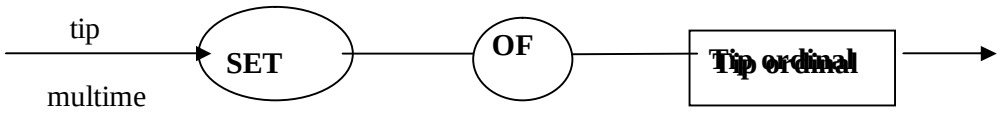
with a[i] do begin readln(nume); i:=i+1; readln(varsta); end;

II.4 TIPUL MULTIME

Denumirea acestui tip ne apropie de domeniul matematicii, fiind prezent numai in codul Pascal, nu si in C++.O multime este o colectie de date de acelasi tip, numit de baza. Ordinea elementelor nu are importanta. O valoare nu apare de mai multe ori in multime.

Diagrama de sintaxa:

SET



Tipul ordinal este de maxim 256 elemente, deci poate fi unul din cele predefinite: char, byte, boolean, sau un subdomeniu al celorlalte tipuri ordinale. In memorie sunt pozitionati pe 1 numai bitii corespunzatori, ca numar de ordine, valorilor din multime, ceilalti fiind pozionati pe 0. O multime poate fi construita cu ajutorul elementelor sale, folosind “constructorii” de multimi: [ ], astfel:

[val1..val2] sau : [val1, val2, val3] daca nu sunt valori consecutive.

Declarari de multimi:

Exemple:

a) const a: set of byte=[0..10];

b) var b: set of ‘A’..’Z’; Notatie pentru multimea vida: [ ]
c: set of 0..10;
d: set of char; Atribuiri: a:=[0..10];
e: set of byte; b:=[‘A’.. ‘C’];
f: set of boolean; Relatii : a>=b;

(a include b, evident fals in exemplul de mai sus)

Operatiile posibile cu datele de acest tip sunt specific matematice: reuniunea(+), intersectia(*), diferenta(-); relatiile sunt : de apartenenta (in), incluziune nestricta (<=, >=), egalitate(=), respectiv incluziune stricta (<, >).

Observatie: In paranteze sunt trecuti operatorii corespunzatori *in cod Pascal* . In pseudocod vom utiliza notatiile de la matematica($\cup$ este reuniune,$\cap$ este intersectie, $\subset$ este incluziune, $\in$ este apartenenta, $\not\subset$ este neinclus in , $\notin$ este nu apartine)

Exercitii: Transcrieti in Pascal, pseudocodul asociat urmatorilor algoritmi:

1. Se da un sir de numere intregi (in intervalul [-256;255]); selectati intregii pozitivi intr-o multime A si afisati elementele acesteia.
2. Se considera 2 vectori de intregi (in intervalul [-256;255]). Afisati elementele lor comune utilizand intersectia de multimi.
3. Doua cuvinte au caractere diferite si caractere comune. Alcatuiti, utilizand multimile, un al treilea cuvant continand elementele (caracterele) care se repeta in cele doua cuvinte date.
4. Ordonati alfabetic o lista de 10 elevi, avand inregistrate numele si nota, cu un cap de tabel continand doua coloane: NUME, NOTA.

Solutiile exercitiilor propuse (in pseudocod) :

1. Notam : v= sirul de numere intregi ; a=multimea elementelor lui ; i=contor; x=element al multimii a; n=numar de componente ale lui v; p= numar de elemente ale multimii a

```
citeste vi, i=1..n
a←{} (*multimea vida*)
pentru i=1,n,1 executa
        daca v(i) > 0 atunci a←a∪{v(i)}
pentru x=0,255,1 executa
        daca x∈a atunci scrie x
```

2. Notam : u,v=vectorii de intregi; a, b=multimile elementelor lor; x=element de acelasi tip cu componentele vectorilor; c=multimea intersectie ; j,i= contori

```
citeste ui,vi, i=1..n; j=1..m
a← {}
b←{} (*multimile sunt initial vide*)
pentru i=1,n,1 executa   a←a∪{u(i)}
pentru j=1,m,1 executa   b←b∪{v(j)}
c= a∩b
pentru x=0,255,1 executa
        daca x∈c atunci scrie x
```

3. Notam : x,y= date de tip string; a,b,c= multimi de caractere; i,j= contori

Singura diferenta raportat la algoritmul precedent este introducerea elementelor.

```
citeste x, y     a← {} b←{}
(*multimile sunt initial vide*)
n←length(x)
m←length(y)              (* n si m retin numarul de componente ale celor *)
pentru i=1,n,1 executa   a←a∪{x(i)}          (*doua stringuri*)
pentru j=1,m,1 executa   b←b∪{y(j)}
c= a∩b
pentru ord(z)=0,255,1 executa
```

```
                daca z∈c atunci scrie z
```

4. Notam: elev variabila de tip articol, avand campul nume (string) si nota(subdomeniu) tab este vector de elemente de tip elev; aux1 si aux2 sunt variabile auxiliare de tipul cam-purilor tipului articol definit; n este numarul de componente ale vectorului; i este contor ; denumirile celor doua campuri : nume, nota; semafor este variabila de tip boolean

```
citeste n
pentru i=1,n,1 executa
            ┌ citeste tab(i).nume
            └ citeste tab(i).nota
┌ repeta
│ semafor←true
│ pentru i=1,n-1,1 executa
│             daca tab(i).nume>tab(i)+1.nume atunci
│                     ┌schimba(tab(i).nume,tab(i+1).nume)
│                     │ schimba(tab(i).nota,tab(i+1).nota)
│                     └semafor←false
└─    pana cand semafor
  pentru i=1,n,1 executa
            ┌── scrie tab(i).nume
            └── scrie tab(i).nota
```

II.5. FISIERE

A. FISIERE CU TIP
B. FISIERE TEXT

FISIERUL este o structura de date de acelasi tip, dar cu un numar nefixat de componente. El se identifica printr-o variabila de tip “fisier”. Componentele fisierului se numesc articole. Colectia de date de tip fisier se memoreaza pe suport extern.

Limbajul Pascal clasifica fisierele, din punct de vedere al modului de reprezentare a datelor pe suportul extern, in fisiere text (datele sunt in format ASCII) si binare(datele sunt memorate in format identic celui din memoria interna). Fisierele binare se impart la randul lor in fisiere cu tip si fisiere fara tip. In primul caz prelucarile de intrare/iesire sunt pe articole de structura si lungime fixa, in al doilea, pe blocuri binare de structura si lungime fixa. Exista doua fisiere standard de intrare si iesire: INPUT,(asig-nat tastaturii) OUTPUT(asignat monitorului), de tip text, definite in unitul SYSTEM. Aceste fisiere se numesc standard si se deschid, respectiv inchid automat la inceputul si sfarsitul executiei fiecarui Program . Fisierele nestandard trebuie sa fie identificate printr-o variabila de tip fisier, declarata inainte de folosire.

Modalitati de declarare:

```
var   f: text; { defineste un fisier de tip text}
      g: file of tip;(defineste un fisier de articole de tipul tip)
```

In Pascal exista proceduri si functii standard de prelucrare a fisierelor, incluse in unitul SYSTEM. Pentru a-l defini “fizic” (asocierea variabilei de tip fisier cu un fisier fizic existent pe disc) se foloseste procedura ASSIGN( ) astfel:

```
Assign(f, ‘text.txt’); Assign(g, ‘Input.dat’);
```

Prima variabila este cea de tip fisier, a doua este numele dat de utilizator fisierului de pe disc eventual cu “calea” corespunzatoare. Fisierele de tip text se pot vizualiza si parcurge la fel ca orice fisier neexecutabil, cu comanda “view”, respectiv “edit”, echivalente tastelor “F3”, respectiv “F4”, din utili-zatorul NC (Norton Commander)

In continuare, fisierele se pregatesc pentru scriere cu comenzile : REWRITE(f); REWRITE(g); iar de citire cu comenzile: RESET(f); RESET(g); Comenzile sunt proceduri standard ale limbajului, specifice fisierelor. Exista unele deosebiri intre cele doua tipuri de fisiere in momentul scrierii sau citirii de date.

Exemple :
```
type elev= record
              nume: string; nota: 1..10; end;
var f: text; g:file of elev; a: elev; b:string; c:char;
```
Vom putea utiliza instructiunile: read(f,b); readln(f);
write(f,b); writeln(f);
sau pe scurt : readln(f,b); writeln(f,b);

in primul caz, insemnand citirea/scrierea unui sir de caractere in "f", urmata de citirea/scrierea unui rand liber, deci salt la randul urmator, in al doilea caz o singura instructiune continand aceste doua actiuni. Daca datele sunt vectori de tip numeric, spre exemplu:

```
var t: array[1..10] of integer;
```
Secventa urmatoare este corecta , permitand inscrierea elementelor unui vector in fisierul f :

```
for i:=1 to 10 do readln(t[i]); for i:=1 to 10 do write(f,t[i]: 5);
```
Tipul "fisier text" admite afisarea cu format, pentru toate tipurile de date. Fisierul de articole permite scrierea si citirea unei inregistrari intregi. Sa presupunem ca exista declaratia:

```
var tab: array[1..10] of elev;
```
Citirea din fisier si scrierea in fisier, a valorilor unei astfel de variabile :

```
for i:=1 to 10 do read(g, tab[i])
for i:=1 to 10 do write(g,tab[i]);
```
Alte operatii sunt cea de adaugare, respectiv cautare, directa sau secventiala in fisierul de inregistrari

Pentru fisierele de tip text, procedura(comanda, instructiunea) standard APPEND(f) pozitioneaza cursorul la sfarsit de fisier, dupa care se pot scrie noi date. Cautarea in fisierul text se poate face numai secvential, caracter cu caracter, secventa cu secventa sau linie cu linie.

Alaturi de cautarea secventiala, fisierul de inregistrari admite si cautarea directa cu ajutorul procedurii SEEK(g,n); unde n este numarul de ordine al articolului pe care se va pozitiona cursorul, urmand a se face actualizarea campurilor. Inchiderea ambelor tipuri de fisiere se va face cu ajutorul procedurii CLOSE(), unde la argument este identificatorul de fisier. (close(f), close(g))

Dupa inchidere, fisierele se pot redenumi cu ajutorul comenzii(procedura): RENAME(f, nume_nou)
de exemplu, pentru fisierul "text.txt" vom scrie rename(f, 'nume.txt'); unde primul argument este variabila de tip fisier, iar al doilea argument este noua denumire a aceluiasi fisier.
De asemenea, pentru ambele fisiere exista in biblioteca Pascal :

eof(f), eof(g)	-procedura care testeaza sfarsitul de fisier
sizeof(f), sizeof(g)	-procedura care intoarce dimensiunea fiserului
erase(f), erase(g)	-procedura care sterge fizic fisierul de pe disc

numai pentru fisierele text :

eoln(f)	- returneaza true daca pointerul indica sfarsit de linie sau fisier
seekeoln(f)	- sare peste TAB sau blanc, si returneaza idem ca precedenta.
seekeof(f)	- sare peste TAB, blanc, CR+LF, si returneaza true dace se detecteza sfarsit de fisier
truncate)f)	- scrie marcajul de sfarsit de fisier pe locul indicat de pointer.

II.6 SUBPROGRAME - PROGRAMAREA STRUCTURATA

Subprogramele sunt parti identificabile prin nume ale unui program si care se pot activa (lansa in executie) la cerere prin intermediul acestor nume. Limbajul Pascal are doua tipuri de subprograme :

1) Functiile returneaza o valoare, de aceea lor li se asociaza si un tip
2) Procedurile efectueaza numai prelucrari de date, fara a returna in mod necesar o valoare (putand returna eventual mai multe valori).

Limbajele dispun de functii si proceduri standard, dar utilizatorul poate crea la randul sau subprograme si apelandu-le din programul principal, sa realizeze asa numita structurare.

La declararea procedurii sau a functiei pot fi specificati asa numitii parametrii formali intr-o lista aflata in antet.

Diagrama de sintaxa a unei proceduri este:

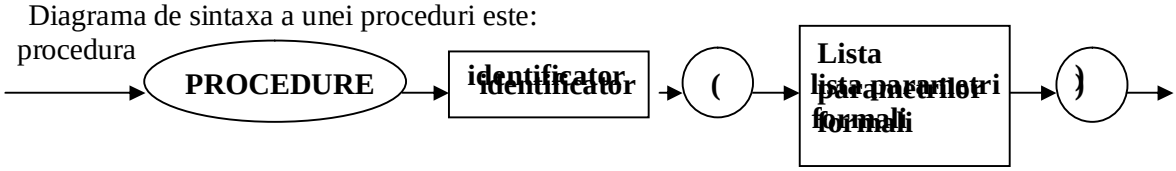
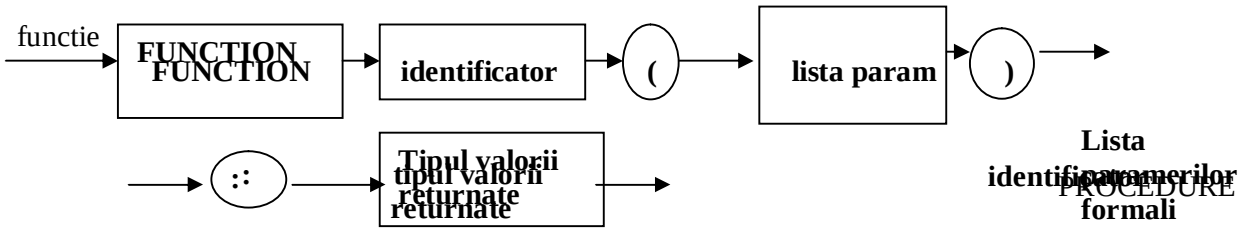
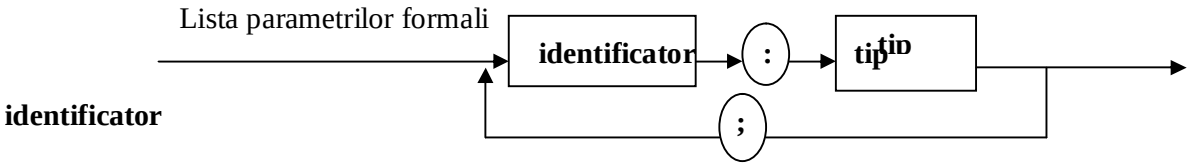


Diagrama de sintaxa a unei functii este:





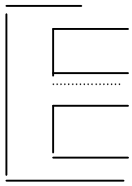
Structura unui subprogram este aceeasi cu cea a unui program principal in sensul ca are drept componenete:

-antetul
-sectiunea declarativa
-sectiunea executabila

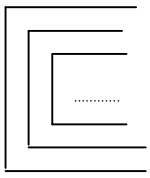
Dezvoltarea unui subprogram in cadrul programului principal se poate face dupa modele:

A. secvential B. nested (imbricat)

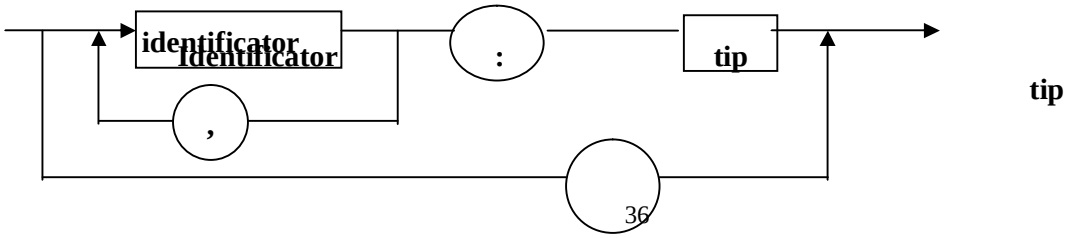
SECVENTIAL:



NESTED:



Obs: Pentru un singur subprogram, cele doua modele se confunda. Daca lista prametrilor for-mali cuprinde mai multi identificatori, acestia se vor separa prin ; si se vor declara cu tip. Daca sunt mai multi identificatori de acelasi tip in lista, atunci acestia se vor separa prin virgula, tipul declarandu-se o singura data.



;

:

La apelare, parametrii efectivi se separa numai prin virgula. Apelul acestor parametri se poate face in doua moduri:

A. prin valoare

B. prin adresa (referinta)- in primul rand pentru tablouri

Acesti parametri sunt precedati de cuvantul cheie -VAR

MASIVUL este orice tablou identificat printr-un nume care reprezinta insasi adresa zonei de memorie in care s-au inregistrat valorile componentelor sale In fata acestui tip de parametru formal se scrie cuvantul cheie VAR.

Domeniul de vizibilitate al unui identificator este zona de Program in care este valabila declaratia sau definitia acelui identificator.

In cadrul unui bloc(program, functie, sau procedura) un identificator poate fi declarat o singura data, si va fi recunoscut doar in interiorul blocului respectiv, motiv pentru care se numesc si entitati locale. Ele apar la lansarea in executie a blocului , spatiul lor fiind rezervat in stiva de executie a blocului, iar memoria rezervata este dealocata la incheierea executiei.

Daca un bloc nu are parametri formali(procedura sau functie) atunci nu exista schimb de informatii intre acest bloc si programul principal sau acest schimb se face prin variabile globale.

Observatie: Dezvoltarea mentionata poate fi schimbata cu ajutorul directivei (cuvant cheie) FORWARD.

Exercitii la tipurile structurate de date

1.

Scrieti in cod Pascal un program care sa exemplifice modelul de prgramare structurata. De exemplu, citind un numar natural n, daca acesta este par, sa calculeze suma a n numere introduse de la tastatura, in caz contrar, sa se calculeze produsul lor.

Solutie: s= variabila suma, locala functiei, p=variabila produs, locala procedurii

N= numar natural , variabila globala.

```
Program sintaxa_program;
label 9;
type interval=1..10;
var i:interval; n:integer;
function adunare(n:integer):integer;
var s,i:integer;
begin
  i:=1;s:=0;
  while(i<=n) do begin
              readln(x);
              s:=s+x;
              inc(i);
              end;
 adunare:=s;
end;
procedure produs(n:integer);
var p,x:integer;
begin
  i:=1;p:=1;
  while(i<=n) do begin
              readln(x);
              p:=p*x;
              inc(i);
              end;
 write('produsul este:',p);
 end;
```

```
begin
 9: begin writeln(^A);
        writeln(#13);
        end;
 read(n);
 if(n mod 2=0) then begin
 write('suma este:',adunare(n));
 produs(n);
 end
 else goto 9;
 readln;
 end.
```

2. Se citesc doua siruri de caractere s1 si s2. Se scrie un sir nou format din caracterele celor doua siruri,folosind o singura data fiecare caracter. (cod Pascal)

```
 Program familie_de_cuvinte;
var a,b,c:set of 'a'..'z';
   s1,s2:string[10];
   ch:'a'..'z';
   i:byte;
begin
   a:=[];
   b:=[];
   c:=[];
   writeln('dati cuvintele :');
   readln(s1);
   readln(s2);
   for i:=1 to length(s1) do if s1[i] in ['a'..'z'] then a:=a+[s1[i]];
   for i:=1 to length(s2) do if s2[i] in ['a'..'z'] then b:=b+[s2[i]];
   c:=a*b;
   for ch:='a' to 'z' do if ch in c then write(ch);
   end.
```

3.

Scrieti un program care sa afiseze un mesaj in cazul in care, parcurgand un text, regaseste un anumit sir de caractere, „virus”.

(cod Pascal)

Solutie:

Textul este un tablou de linii, iar virusul constanta ‚abc’

```
Program scan_virus;
 const virus='abc';
 type linie=string;
 var  n,i:byte;
      text:array[1..10] of linie;
 begin
    writeln('nr de linii ale textului');
    readln(n);
    writeln('dati textul');
    for i:=1 to n do readln(text[i]);
    for i:=1 to n do if pos(virus,text[i])<>0
    then write('virus gasit');
    readln;
    end.
```

4. Scrieti un program care depisteaza si inlatura un sir de caractere dintr-un fisier text.

(cod Pascal)

```
Program devirusare_text;
 const virus='abc';
 type linie=string;
 var  n,a,i:byte;
      ch:char;
      text:array[1..10] of linie;
      begin
     writeln('nr de linii ale textului');
```

```
readln(n);
writeln('dati textul');
for i:=1 to n do readln(text[i]);
for i:=1 to n do if pos(virus,text[i])<>0 then write('virus gasit');
writeln(' doriti devirusare ?(D/N)');
readln(ch);
if upcase(ch)='D' then if pos(virus,text[i])<>0 then
      begin
      a:= pos(virus,text[i]);
      delete(text[i],a,length(virus));
                  end;
for i:=1 to n do writeln(text[i]);
end.
```

5. Ordonati alfabetic o lista de elevi, continand numele si nota fiecaruia.

```
Program ordonare_alfabetica;
type elev=record
nume:string;
nota:integer;
end;
var tab:array[1..10] of elev;
   aux:elev;
   sema:boolean;
   i,n:byte;
procedure pahare;
begin
aux:=tab[i];
tab[i]:=tab[i+1];
tab[i+1]:=aux;
sema:=false;end;

procedure ordonare;
begin
repeat
sema:=true;
for i:=1 to n-1 do  if tab[i].nume>tab[i+1].nume
then pahare;
until sema=true;
    end;
begin
write('dati numarul de elevi');
readln(n);
for i:=1 to n do begin
write ('numele=');readln (tab[i].nume);
write ('nota=');readln (tab[i].nota);
end;
ordonare;
for i:=1 to n do begin
write ('numele=');writeln (tab[i].nume);
write ('nota=');writeln (tab[i].nota);
end;
END.
```

5. Se citesc 6 numere de la tastatura si se inscriu cu format intr-un fisier text. Se afisaza conti-nutul fisierului pe ecran, apoi se ordoneaza numerele din fisier si se scriu in continuarea celor deja existente. Care va fi continutul noului fisier? (cod Pascal)

```
Program numere_fisier;
   uses crt;
   var f1,f2,f3:text;
      aux,i,n:integer;
      vb:boolean;
      ch:char;
      linie:string;
      tab:array [1..10] of integer;
      begin
      clrscr;
   assign (f1,'in_out.txt');
   rewrite (f1);
   writeln(f1,'numerele sunt:');
   readln (n);
```

```
for i:=1 to n do begin readln (tab[i]); write (f1,tab[i]:7);
end;
close(f1);
assign (f1,'in_out.txt');
reset (f1);
i:=0;
while not eof(f1) do begin
read (f1,ch);
write (ch);
end;
close(f1);
writeln;
assign (f1,'in_out.txt');
reset (f1);
while not eof (f1) do begin
readln(f1,linie);writeln (linie);end;
close(f1);
assign (f1,'in_out.txt');
reset (f1);
readln (f1);
while not eof (f1) do
read(f1,tab[i]);
repeat
vb:=true;
for i:=1 to n-1 do if tab[i]>tab[i+1] then begin
aux:=tab[i];
tab[i]:=tab[i+1];
tab[i+1]:=aux;
vb:=false end;
until vb;
append(f1);
writeln (f1);
for i:= 1 to n do write(f1,tab[i],' '); close(f1); end.
```

```
FISIERUL IN_OUT.TXT

numerele sunt:
      6      3      7      8      9
3 6 7 8 9
```

6. Se citeste un sir de caractere de la tastatura (standard INPUT), reprezentand o expresie algebrica Se cere sa se scrie sub forma poloneza postfixata (intai operanzii, apoi operatorii, in ordinea efectuarii calculelor, de la stanga la dreapta) (cod Pascal)

Solutia utilizeaza Programarea structurata, variabila **e** retine sirul de caractere, **p** retine forma polonea postfixata, iar procedurile **s, p, t, f** se pot interapela inainte de a fi explicitate, prin intermediul directivei FORWARD, cuvant-cheie, scris in dreptul antetului subProgramului :

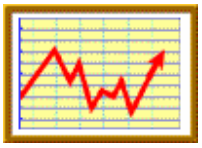
```
Program forma poloneza_ static;
uses crt;
var p,e:string;
   i,l:byte;
procedure citire;
begin
write('exp:');
readln(e);
l:=length(e);
p:='';
writeln;
end;
procedure s;    forward;
procedure f;
begin
   if e[i]='(' then begin
      inc(i);
      s;
      inc(i);
      end
      else
        begin
        p:=p+e[i];
        inc(i);
        end;
 end;
procedure t;
begin
 f;
 while (e[i] ='*')and(i<=l) do begin
   i:=i+1;
```

```
    f;
    p:=p+'*';
 end;
 while (e[i] ='/')and(i<l) do begin
    i:=i+1;
    f;
    p:=p+'/'; end;

end;
procedure s;
begin
 t;
 while (e[i]='+')and(i<l) do begin
    i:=i+1;
    t;
    p:=p+'+';
    end;
 while (e[i]='-')and(i<l) do begin
    i:=i+1;
    t;
    p:=p+'-';
    end;
 end;
begin
clrscr;
citire;
i:=1;
s;
writeln('expresie in forma poloneza  : ',p:20);
readln;
end.
```

7. Realizati un program care inlocuiasca comenzile COPY NUME_1.TXT NUME_2.TXT, si TYPE NUME_2.TXT

```
Program copy_type;
 var  f1,f2:text;
      i:integer;
      ch:char; a:string;
procedure scrie;
begin
rewrite(f1);
for i:=1 to 2 do
begin
readln(a);
writeln(f1,a);
end;end;
BEGIN
assign(f1,'NUME_1.txt');
scrie;
assign(f2,'NUME_2.txt');
reset(f1);
rewrite(f2);
while not eof(f1) do
begin
 read(f1,ch);
 write(f2,ch);
end;
close(f1);
reset(f2);
while not eof(f2) do
 begin
  read(f2,ch);
  write(ch);
end;
close(f2);
readln;
end.
```

Fisierele NUME_1 si NUME_2 sunt de tip text, create prin Program, iar procedura **scrie** realizeaza scrierea in primul fisier, cel care va fi copiat.



8. Scrieti un program care sa inlocuiasca comenzile EDIT si RENAME . Utilizati fisierul text IN.TXT Modificati continutul acestui fisier si redenumiti-l OUT.TXT.

```
var f,g:text;a:string;
begin
assign(g,'IN.TXT');
append(g);
write(g,'    BYE BYE !    ');
assign(f,'OUT.TXT');
close(g);
erase(f);
Rename(g,'IN.TXT');
end.
```

8. Dintr-un fisier cu tip utilizator, avand inregistrarile cu numele si varsta unor elevi, sa extraga al doilea si sa se afiseze pe ecran.
Solutia care este prezentata mai jos va utiliza procedura standard de acces direct: **seek**

```
type elev=record
            nume:string;
            varsta:byte;
             end;
             var  tab:array[1..10] of elev;
                  f:file of elev;
                 a:elev;i,n:integer;
             begin
             assign(f,'elevi.dat');
             readln(n);rewrite(f);
             for i:=1 to n do eadln(tab[i].nume);
             readln(tab[i].varsta);
             write(f,tab[i]);
               end;
             reset(f);
             seek(f,2);
              read(f,a);
              writeln(a.nume);
              writeln(a.varsta);
             close(f); end.
```

9.
Se da arborescenta de directoare si fisiere a unei unitati de disc.Cunoscand numele unui fisier sa se determine subdirectoarele din care face parte.

```
Program arbore_directoare;
type nod=record
     nume:string[20];
     tata:integer;
     end;
var v:array[1..20] of nod;
    n,ind:integer;
    x:string[20];
function cauta(x:string):integer;
var i:integer;
begin
  cauta:=0;
  for i:=1 to n do
      if x=v[i].nume then cauta:=i;
end;
procedure citire;
var i,j:integer;
    x:string[20];
begin
write('Nr. de elemente din structura arborescenta');
readln(n);
for i:=1 to n do  readln(v[i].nume);
for i:=1 to n do begin
    writeln('Dati tatal lui ',v[i].nume);
    readln(x);
    v[i].tata:=cauta(x);
end;
end;
begin
citire;
writeln('Dati un nume:');
readln(x);
ind:=cauta(x);
if ind=0 then writeln('Nu exista') else
 begin
 ind:=v[ind].tata;
 while ind<>0 do begin
                writeln(v[ind].nume);readln;
                dec(ind);
end;end;end.
```

CAPITOLUL III

BACK ELEMENTAR SAU ITERATIV

Metoda backtracking desi neperformanta din punct de vedere al timpului de lucru(exponential) este potrivita pentru rezolvarea problemelor in care solutia se prezinta sub forma de tablou(uni- sau multidimensional) Numarul de elemente este dat si finit. Exista si cateva probleme clasice, considerate ca modele de rezolvare pentru o arie mai extinsa.

Ideea principala este testarea solutiei obtinute la fiecare pas, numeric, in raport cu numarul posibil de elemente, si indeplinirea unui criteriu, conditie, propozitie, iar daca aceste conditii sunt indeplinite, se poate continua pana la epuizarea tuturor elementelor disponibile, in caz contrar, “efectuan du-se un pas inapoi”, si reluandu-se cautarea in restul multimii de valori posibile.

Vectorul solutie apare ca o stiva de elemente, care se construieste, se valideaza, se testeaza, se afisaza, prin adaugarea consecutiva a cate unui element, iar in caz de nereusita se devanseaza, se trage cu un pas inapoi, pentru a relua procedeul pana la epuizarea tuturor elementelor date. In cod Pascal, vom considera exemplul cel mai simplu, “problema permutarilor” , care se enunta astfel:
Se considera primele n numere naturale consecutive. In cate moduri se pot aseza aceste numere astfel incat fiecare grupare sa difere numai prin modul de asezare al elementelor. Pasii necesari pentru obtinerea solutiilor posibile, prin procedeul de mai sus sunt :

a) **Initializarea** stivei pe toate componentele cu o valoare minimala :

```
procedure init( k:integer; var st:stiva);
begin st[k]:=0; end;
```

b) Obtinerea **elementului urmator** al stivei posibilei solutii:

```
procedure succesor(var as:boolean; var st:stiva; k:integer);
begin if st[k]<n then begin
                        st[k]:=st[k]+1;
                        as:=true;
                        end else as:= false;
end;
```

In acest exemplu clasic, cel al calculului de permutari, elementul st[k], pentru a avea succesor, deci semaforul “as” sa fie “pe verde”, se va stabili conditia de a nu fi depasit numarul n total de elemente date, in caz contrar semaforul este “pe rosu”, deci as=false.

c) La fiecare moment de timp, deci pentru fiecare k se verifica **validitatea** stivei de elemente obtinute:

```
procedure valid(var ev: boolean; var st:stiva; k:integer);
begin ev:=true;
for i:=1 to k-1 do if st[k]=st[I] then ev:=false;
end;
```

Aici, de exemplu, nu sunt permise numerele care se repeta, deci egale.(elementele permutarii sunt distincte)

d) O functie **test** va raspunde daca am exact numarul de obiecte cerut pentru ca stiva obtinuta sa fie sau nu o solutie posibila:

```
function solutie(k:integer): boolean;
begin solutie:=(k=n); end;
```

e) In final o procedura fara parametri va tipari solutia:

```
procedure tipar;
begin for i:=1 to n do write(st[i]); end;
```

Mecanismul principal va apela conform ideii prezentate aceste subprograme in mod aproape identic, indiferent de enuntul problemei care se rezolva prin aceasta metoda, singurele care pot suferi modificari fiind subprogramele.

Algoritmul in pseudocod:

```
k←1  init(st,k)
atat timp cat (k>0) executa
    repeta
        succesor(as,st,k)
    daca as atunci valid(ev,st,k)
    pana cand (not as) sau (as si ev)
    daca as atunci
        daca solutie(k) atunci tipar
        altfel
            k←k+1
            init(st,k)
    altfel k←k-1
```

Comentarii: Repetitia se opreste daca nu exista succesor sau, chiar daca exista succesor si stiva este validata, urmand sa se testeze obtinerea unei solutii reale, cu numarul complet de elemente cerute.

Daca nu s-a obtinut as=true deci nu exista succesor, se face un pas inapoi: k←k-1. Desigur pasul inapoi nu trebuie sa ne dea un contor negativ, ceea ce impune conditia: k>0. In cazul cand as=true si ev=true, deci exista succesor si stiva obtinuta este validata , se trece la pasul urmator, prin initializarea unui nou nivel: k=k+1. Procedeul va fi exemplificat pe trei exemple clasice:

1. Problema reginelor
2. Problema partitiilor unui numar natural
3. Problema platii in bancnote de valori date

1. Problema reginelor

N regine se asaza pe o tabla n×n, astfel ca sa nu se atace reciproc. Se cer toate posibilitatile de aranjare.

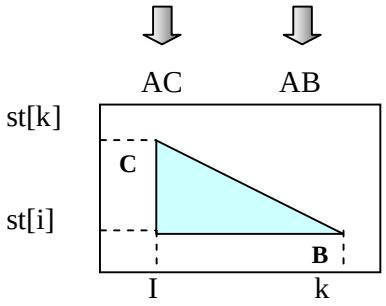
Particularizarea subprogramelor:

Apare o noutate in validarea solutiei, deoarece intervine specificul situatiei ca doua regine sa nu se atace:

```
procedure valid(var ev: boolean; var st:stiva; k:integer);
begin ev:=true;
for i:=1 to k-1 do if ((st[k]=st[i]) or (abs(st[k]-st[i])=abs(k-i)
then ev:=false;
end;
```

Reginele se ataca pe verticala si orizontala, precum tura, iar pe diagonala precum nebunul. Verticala se va exclude prin alegerea valorii k-1 pentru contorul maximal, ramanand numai conditia de pe orizontala : st[k] ≠st[i] iar cand Δ ABC este dreptunghic isoscel adaugand conditia de pe diagonala :

AC≡AB→abs(st[k]-st[i])=abs(k-i)



2. Problema partitiilor unui numar natural

Sa se descompuna numarul n in suma de numere naturale mai mici decat el .Sa se afiseze toate si-

tuatiile posibile. (exemplu: 5=2+3=1+4=..)
Particularizarea subprogramelor:
La orice nivel k in stiva, valoarea minima a componentei este 1 iar cea maxima este :
n-(k-1) = n-k+1, daca toti dinaintea lui au fost egali cu 1.
Deci conditia din procedura succesor devine: st[k]< n-k+1.
Problema cerea impartirea numarului intr-o suma de numere care sa fie egala cu n ca urmare apare variabila s necesara pentru calculul sumei elementelor din stiva:

```
s:=0;
for i:=1 to k do s:=s+st[i];
```

Un vector valid la un moment dat, presupune ca acest s sa nu fie depasit de n :

```
if s<=n then ev:=true else ev:=false;
```

Deci, daca s=n , atunci s-a obtinut chiar o solutie : solutie:=(s=n); Numarul elementelor stivei nu este necesar n, ci k, deci se va tipari un vector de k elemente : for i:=1 to k do write(st[i]);
3. Problema platii in bancnote de valori date
Sa se achite o suma s, data, in bancnote de valori ai, i=1..n, date, unde n este un numar natuaral.
Particularizarea subprogramelor:
Pentru initializarea stivei se utilizeaza valoarea -1 in loc de valoarea 0, deoarece 0 este valoare admisa in solutie, unele monede putand lipsi, deci vor fi luate de 0 ori.

```
st[k]:=-1; ( init(st,k))
```

In momentul in care se citesc valorile admise pentru monezi ai, i=1..n, se vor calcula si numerele maxim posibile de monezi, pentru fiecare valoare data .

```
bi= s div ai, unde s este data.
```

Pentru a verifica daca exista element succesor in stiva se calculeaza suma partiala obtinuta cu numarul de monezi deja gasite pana la pasul k, utilizand o variabila auxiliara s1:

```
s1:=0;
for i:=1 to k do s1:=s1+st[i]*a[i];
```

iar conditia pentru element este :

```
st[k]< b[k];
```

Procedura succesor devine:

```
procedure succesor(var as:boolean; var st:stiva; k:integer);
  begin
    if st[k]<b[k] then begin
                         s1:=0;
                         for i:=1 to k do s1:=s1+a[i]*st[i];
                         as:=(s1<s); end
                         else a:=false;
    if as then st[k]:=st[k]+1;
  end;
```

Validarea difera de asemenea de cazul permutarilor, deoarece orice stiva astfel obtinuta si testata pana la pasul k este valida: ev:=true;
Solutia propriu-zisa insa presupune recalcularea sumei , cu ajutorul unei variabile auxiliare s1, si compararea ei cu valoarea data s : s1:=0; for i:=1 to k do s1:=s1+a[i]*st[i];
solutie:=(s1=s);
Si in aceasta problema, ca si la partitia numarului natural, stiva solutie va contine k elemente si nu n, desi k poate fi egal eventual cu n:

```
for i:=1 to k do write(st[i], 'de valoare ', a[i]);
```

Ce observam in toate cele trei exemple, este faptul ca pseudocodul algoritmului ramane identic, shimbandu-se numai continutul procedurilor, in raport cu cerintele.
Rezolvarile devin mai greoaie cand este vorba de cod C++, unde este afectata chiar stiva de date a calculatorului, programul fiind mai dificil de finalizat.

PROBLEME LA CAPITOLUL III (APLICATII LA METODA BACKTRACKING)

PROBLEMA 1
SA SE AFISEZE ARANJAMENTELE DE N LUATE CATE P SI NUMARUL LOR

```
PROGRAM ARANJAMENTE;
uses crt;
type stiva=array[1..100] of integer;
var st:stiva;n,k,p,nr_sol:integer;as,ev:boolean;

procedure init(k:integer;var st:stiva);
begin
st[k]:=0;
end;

procedure succesor(var as:boolean;var st:stiva;k:integer);
begin
if st[k]<n then begin
                  st[k]:=st[k]+1;
                  as:=true;
                end
          else as:=false
end;

procedure valid(var ev:boolean;st:stiva;k:integer);
var i:integer;
begin
ev:=true;
for i:=1 to k-1 do
        if st[k]=st[i] then ev:=false
end;

function solutie(k:integer):boolean;
begin
solutie:=(k=p);
end;

procedure tipar;
var i:integer;
begin
for i:=1 to p do write(st[i]:3);
writeln;
end;

begin {pp}
clrscr;
write('n=');readln(n);
write('p=');readln(p);
k:=1;init(k,st);nr_sol:=0;
while k>0 do begin
 repeat
```

```
    succesor(as,st,k);
    if as then valid(ev,st,k);
  until not as or as and ev;
  if as then if solutie(k) then begin
                                  tipar;
                                  nr_sol:=nr_sol+1;
                                  if (nr_sol mod 24)=0 then readln;
                                end
                           else begin
                                  k:=k+1;
                                  init(k,st)
                                end
       else k:=k-1
 end;
   write('nr aranjamentelor de ',n,' luate cate ',p,' este ',nr_sol);
   end.
```

PROBLEMA 2

PE O TABLA DE SAH DE DIMENSIUNE N. O PIESA-CAL SE DEPLASEAZA IN TOATE POZITIILE; SA SE AFISEZE TRASEELE POSIBILE.

```
PROGRAM CAI;
uses crt;
type stiva=array[1..100] of integer;
var st:stiva;n,k,nr_sol:integer;as,ev:boolean;
procedure init(k:integer;var st:stiva);
begin
t[k]:=0;
end;
procedure succesor(var as:boolean;var st:stiva;k:integer);
begin
if st[k]<n then begin
                  st[k]:=st[k]+1;
                  as:=true;
end
          else as:=false
end;
procedure valid(var ev:boolean;st:stiva;k:integer);
var i:integer;
begin
if k=1 then
         ev:=true
       else
         if k=2 then
               ev:=abs(st[k]-st[k-1])<>2
else
ev:=(abs(st[k]-st[k-2])<>1) and (abs(st[k]-st[k-1])<>2)
end;

function solutie(k:integer):boolean;
```

```
begin
solutie:=(k=n);
end;

procedure tipar;
var i:integer;
begin
for i:=1 to n do write(st[i]:3);
writeln;
end;
begin {pp}
clrscr;
write('n=');readln(n);
k:=1;init(k,st);nr_sol:=0;
while k>0 do begin
            repeat
              succesor(as,st,k);
              if as then valid(ev,st,k);
            until not as or as and ev;
            if as then if solutie(k) then begin
          tipar;
          nr_sol:=nr_sol+1;
          if (nr_sol mod 24)=0 then
          readln;
         end
   else begin
         k:=k+1;
         init(k,st)
        end
else k:=k-1
end;
if nr_sol<>0 then write('nr_sol=',nr_sol)
else write('nu exista solutie');
end;
```

PROBLEMA 3

UN COMIS-VOIAJOR ARE DE PARCURS DRUMUL INTRE DOUA LOCALITATI A SI B, INTRE CARE CUNOASTE TOATE ORASELE, SI URMEAZA SA SE INTOARCA INAPOI DE UNDE A PLECAT.AFISATI TOATE TRASEELE POSIBILE.

```
PROGRAM COMIS_VOIAJOR;
uses crt;
type stiva=array[1..100] of integer;
var s:stiva;n,k,i,j,nr_sol:integer;as,ev:boolean;
   a:array[1..50,1..50] of 0..1;

procedure init(k:integer;var s:stiva);
begin
s[k]:=1;
end;
procedure succesor(var as:boolean;var s:stiva;k:integer);
begin
```

```
if s[k]<n then begin
                s[k]:=s[k]+1;
                as:=true;
              end
         else as:=false
end;

procedure valid(var ev:boolean;s:stiva;k:integer);
var i:integer;
begin
ev:=true;
if a[s[k-1],s[k]]=0 then ev:=false
                    else
                      for i:=1 to k-1 do
                        if s[k]=s[i] then ev:=false;
if (k=n) and (a[1,s[k]]=0) then ev:=false;
end;

function solutie(k:integer):boolean;
begin
solutie:=(k=n);
end;

procedure tipar;
var i:integer;
begin
for i:=1 to n do write(s[i]:3);
write('1':3);writeln;
end;

begin {pp}
clrscr;
write('n=');readln(n);
for i:=1 to n do
for j:=1 to i-1 do begin
                  write('a(',i,',',j,')=');
                  read(a[i,j]);
                  a[j,i]:=a[i,j];
                 end;
st[1]:=1;k:=2;init(k,s);nr_sol:=0;
while k>0 do begin
             repeat
               succesor(as,s,k);
               if as then valid(ev,s,k);
             until not as or as and ev;
             if as then if solutie(k) then begin
                                             tipar;
                                             nr_sol:=nr_sol+1;
                                             if (nr_sol mod 24)=0 then readln;
                                           end
                                      else begin
                                             k:=k+1;
                                             init(k,s)
                                           end

                  else k:=k-1
            end;
if nr_sol<>0 then write('nr traseelor este de ',nr_sol)
          else write('nu are solutie');
end.
```

Autor: CAZACU N. NINA LICEUL C. A. ROSETTI,, BUCURESTI

PROBLEMA 4

IONEL TREBUIE SA COLOREZE O HARTA PENTRU A DOUA ZI, LA GEOGRAFIE. EL NU ARE VOIE SA REPETE CULORILE SI NU ARE LA INDEMANA DECAT UN NUMAR DE CREIOANE. DE ASEMENEA HARTA ARE UN NUMAR DAT DE TARI. AJUTATI-L PE COPIL SA-SI FAC TEMA SCRIIND TOATE POSIBILITATILE DE COLORARE.

```
PROGRAM HARTI;
uses crt;
type stiva=array[1..100] of integer;
var s:stiva;n,k,i,j,nr_sol:integer;as,ev:boolean;
   a:array[1..50,1..50] of 0..1;

procedure init(k:integer;var st:stiva);
begin
s[k]:=0;
end;

procedure succesor(var as:boolean;var s:stiva;k:integer);
begin
if s[k]<n then begin
                s[k]:=s[k]+1;
                as:=true;
              end
         else as:=false
end;

procedure valid(var ev:boolean;s:stiva;k:integer);
var i:integer;
begin
ev:=true;
for i:=1 to k-1 do
if (s[i]=s[k]) and (a[i,k]=1) then ev:=false;
end;

function solutie(k:integer):boolean;
begin
solutie:=(k=n);
end;

procedure tipar;
var i:integer;
begin
for i:=1 to n do writeln('tara ',i,' culoarea ',s[i]);
readln;
end;


              begin {pp}
              clrscr;
              write('n=');readln(n);
              for i:=1 to n do
              for j:=1 to i-1 do begin
                              write('a(',i,',',j,')=');
                              read(a[i,j]);
                              a[j,i]:=a[i,j];
                             end;
              k:=1;init(k,s);nr_sol:=0;
              while k>0 do begin
```

```
repeat
  succesor(as,s,k);
  if as then valid(ev,s,k);
until not as or as and ev;
if as then if solutie(k) then begin
                              tipar;
                              nr_sol:=nr_sol+1;
   if (nr_sol mod 24)=0 then readln;
                              end
                         else begin
                              k:=k+1;
                              init(k,s)
                              end
                         else k:=k-1
                              end;
if nr_sol<>0 then write('nr_sol=',nr_sol)
else write('nu exista solutie');
end.
```

PROBLEMA 5
PRODUSUL CARTEZIAN A N MULTIMI
SE DAU MULTIMILE A1={1,2,...,K1}
A2={1,2,...,K2}...
AN={1,2,...,KN}
SE CERE SA SE AFISEZE PRODUSUL LOR CARTEZIAN
OBS- VECTORUL A RETINE VALORILE K1,K2,...,KN

```
PROGRAM PRODUS_CARTEZIAN;
uses crt;
type stiva=array[1..100] of integer;
var st:stiva;n,k,nr_sol,i:integer;as,ev:boolean;a:array[1..100] of integer;



procedure init(k:integer;var st:stiva);
begin
st[k]:=0;
end;

procedure succesor(var as:boolean;var st:stiva;k:integer);
begin
if st[k]<a[k] then begin
                   st[k]:=st[k]+1;
                   as:=true;
                  end
              else as:=false
end;

procedure valid(var ev:boolean;st:stiva;k:integer);
var i:integer;
begin
ev:=true;
end;

function solutie(k:integer):boolean;
begin
solutie:=(k=n);
end;
```

```
procedure tipar;
var i:integer;
begin
for i:=1 to n do write(st[i]:3);
writeln;
end;

begin {pp}
clrscr;
write('numarul de multimi=');readln(n);
for i:=1 to n do begin
                  write('a[',i,']=');
                  readln(a[i]);
                 end;
k:=1;init(k,st);nr_sol:=0;
while k>0 do begin
              repeat
                succesor(as,st,k);
                if as then valid(ev,st,k);
              until not as or as and ev;
              if as then if solutie(k) then begin
                                              tipar;
                                              nr_sol:=nr_sol+1;
                                              if (nr_sol mod 24)=0 then readln;
                                             end
                                       else begin
                                              k:=k+1;
                                              init(k,st)
                                             end
                   else k:=k-1
            end;
write('nr_sol=',nr_sol);
end.
```

Observatie :

Din punct de vedere matematic, algoritmul se desfasoara de maniera urmatoare :

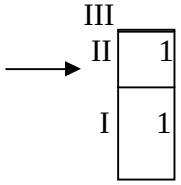
Se dau multimile : A1={1..k1} A2={1..k2} .. An={1..kn}
Se cere produsul cartezian : A1×A2×..×An={ (x1,..,xn)| xi∈Ai, i=1..n}

Exemplu : A1={1,2},A2={1,2,3}, A3={1,2,3}
A1×A2×A3= {(1,1,1),(1,1,2),(1,1,3),(1,2,1),(1,2,2),(1,2,3),(1,3,1),(1,3,2),(1,3,3),
(2,1,1),(2,1,2),(2,1,3),(2,2,1),(2,2,2), (2,2,3),(2,3,1),(2,3,2),(2,3,3)}
Pentru rezolvare se foloseste stiva **st** si un vector **a,** care retine numerele k1,k2,..,kn
Particularizarea procedurilor:

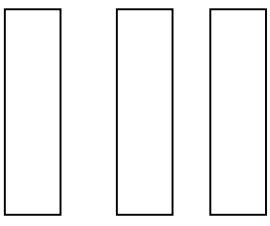
a) La nuvelul k al stivei, validarea este de prisos deoarece orice element aflat pe acest nuvel va fi valid, caurmare: ev:=true;
b) Limita superioara la nivelul k este a[k] nu n.

1 2 → a[1]<=2 →st[1]<=2 (in procedura succesor)
1 2 3→ a[2]<=3→st[2]<=3
1 2 3→ a[3]<=3→st[3]<=3

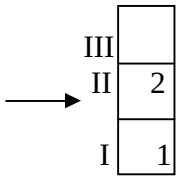
Construirea solutiilor:



Pe nivelul I scriem 1: se scriu toate tripletele cu prima componenta 1 si a doua tot 1. Acestea sunt: (1,1,2), (1,1,2), (1,1,3), in ordinea specificata, pe nivele.

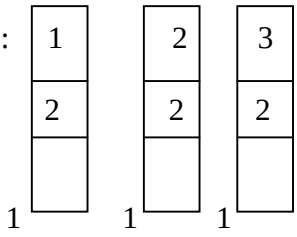


Am obtinut solutiile : 1 2 3 Pe nivelul 3 nu mai avem succesor,
1 1 1 deci se coboara in stiva pe nivelul 2
1 1 1

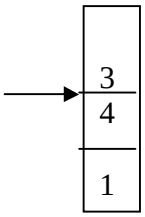


Se scriu toate tripletele cu prima componenta 1 si a doua 2. Acestea sunt: (1,2,1), (1,2,2), (1,2,3)

Am obtinut solutiile :



Pe nivelul 3 nu mai avem succesor, deci se coboara in stiva pe nivelul 2



Procedeul continua pana cand nu mai exita succesor pe nuvelul 1
Solutiile obtinute se tiparesc.

Modificarile se vor gasi numai in procedurile succesor si valid

```
procedure succesor(var st:stiva;var as:boolean; k:integer);
begin
if st[k]<a[k] then begin
                    st[k]:=st[k]+1;
                    as:=true;
                     end else
                    as:=false; end;
procedure valid(var st:stiva;var as:boolean; k:integer);
begin ev:=true; end;
```

Celelalte proceduri si implementarea sunt identice.

PROBLEMA 6

REZOLVATI PROBLEMA RUCSACULUI, UTILIZAND METODA BACK, ELEMENTAR NERECURSIV:

```
PROGRAM RUCSAC;
uses crt;
type stiva=array[1..100] of integer;
var st,c,g,iau:stiva;gg,cmax,cc,i,n,k:integer;as,ev:boolean;
{n-obiecte,c-castigurile asociate,g-greutatile asociate
gg-greutatea maxima admisibila,cc-castig curent,cmax-castig maxim}

    procedure init(k:integer;var st:stiva);
    begin
    st[k]:=-1;
    end;

    procedure succesor(var as:boolean;var st:stiva;k:integer);
    begin
    if st[k]<1 then begin
```

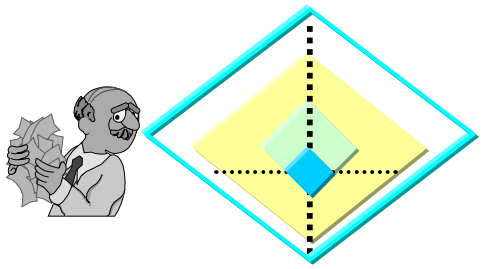
```
                st[k]:=st[k]+1;
                as:=true;
              end
          else as:=false;
end;

procedure valid(var ev:boolean;st:stiva;k:integer);
var greutate,i:integer;
begin
greutate:=0;
for i:=1 to k do
  if st[i]=1 then greutate:=greutate+g[i];
ev:=greutate<=gg;
end;

procedure tipar;
var i:integer;
begin
writeln('o solutie cu castig maxim: ');
for i:=1 to n do if iau[i]<>0 then write(i:3);
writeln;
writeln('castig maxim=',cmax);
readln;
end;

begin {pp}
clrscr;
write('cate tipuri de obiecte avem ?');readln(n);
write('greutatea admisa ');readln(gg);
for i:=1 to n do begin
              writeln('castigul obtinut prin obiectul ',i,' ');
              readln(c[i]);
              writeln('greutatea  obiectului ',i,' ');
              readln(g[i]);
              end;
k:=1;init(k,st);cmax:=0;
while k>0 do begin
             repeat
               succesor(as,st,k);

  if as then valid(ev,st,k);
           until   not as or as and ev;
           if as then if k=n then

          begin
          cc:=0;
          for i:=1 to n do
     if st[i]=1 then cc:=cc+c[i];
    if cc>=cmax then
                begin
                 cmax:=cc;
                 for i:=1 to n do
                   iau[i]:=st[i];
                end;
               end
              else begin
                   k:=k+1;
                   init(k,st)
                  end
      else k:=k-1
```

```
end;
if cmax<>0 then tipar
        else write('nu are solutie');
end.
```

CAPITOLUL IV

RECURSIVITATEA

Gandirea specific recursiva din matematica, o regasim in fenomenul de autoapelare a functiilor sau procedurilor din programarea structurata recursiva.



1. FUNCTII RECURSIVE

1.1. Calculul lui n factorial (n!)

pseudocod :

```
functia fact(n)
        daca n=0 atunci fact←1
        altfel fact← n*fact(n-1)
```

cod Pascal :

```
var n: integer;
 function fact(n:integer):longint;
 begin if n=0 then fact:=1 else fact:=n*fac(n-1); end;
```

```
begin write('n='); readln(n); writeln('Factorial de ',n,' este: ',fact(n)); end.
```

1.2 Sirul lui Fibonacci : a(0)=0, a(1)=1, a(n)= a(n-1)+a(n-2), n>=2

pseudocod :

```
functia fibo(n)
        daca n=0 atunci fibo←0
        altfel daca n=1 atunci fibo←1
        altfel fibo←fibo(n-1)+fibo(n-2)
```

cod Pascal :

```
var n:integer;
function fibo(n:integer):integer;
begin if n=0 then fibo:=0
        else if n=1 then fibo:=1
        else fibo:=fib(n-1)+fibo(n-2);
end;
begin write('n='); readln(n);
        writeln('Valoarea termenului al ',n,' -lea',' din
        sirul Fibonacci este: ',fibo(n));
end.
```

1.3 Cmmdc si cmmmc a doua numere naturale a si b

Pentru rezolvarea acestei probleme se pot folosi atat algoritmul lui Euclid cat si metoda diferentelor succesive . Deoarece algoritmul lui Euclid l-am prezentat in capitolele precedente, iterativ, el ramane ca exercitiu, sa fie transformat recursiv. Aici va fi prezentata metoda diferentelor, recursiv, si care ni se pare mult mai simpla:

pseudocod :

```
functia cmmdc(a,b)
  daca a=b atunci cmmdc←a
  altfel
      daca a>b atunci cmmdc←cmmdc(a-b,b)
      altfel cmmdc←cmmdc(a,b-a)
```

cod Pascal :

```
var a,b:integer;
    function cmmdc(a,b:integer):integer;
 begin  if a=b then cmmdc:=a
                  else
                   if a>b then cmmdc:=cmmdc(a-b,b)
              else cmmdc:=cmmdc(a,b-a); end;
      begin write('a=');readln(a);write('b='); readln(b);
   writeln('cmmdc al numerelor ',a,' si ',b,' este : ', cmmdc(a,b));end.
```

2. PROCEDURI RECURSIVE

2.1. Inversarea elementelor unui sir fara a utiliza vectori

pseudocod :

```
procedura inv;
  citeste c
  daca c<>'0' atunci inv
  scrie c
```

cod Pascal :

```
var c: char; a:string;
 procedure inv(i:byte);
 begin
 read(a[i]);c:=a[i];
 if c<>'0' then begin
                    inc(i); inv(i);end;
 if a[i]<>'0' and i>0 then begin write(a[i]); dec(i);end;
 end;

begin
writeln('Apasati Enter, apoi introduceti sirul de caractere:');
c:=readkey; inv(0); end.
```

2.2 Varianta recursiva a metodei backtracking

In varianta recursiva subprogramele raman identice cu exceptia procedurii succesor care se transforma in functie de testare a existentei unui element succesiv, intorcand chiar valoarea variabilei “as”)

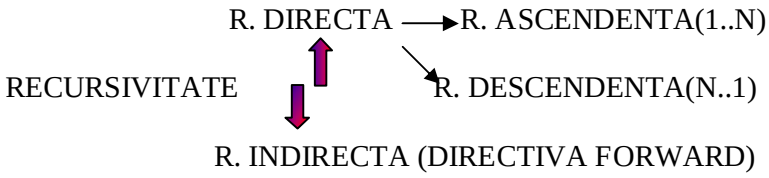
pseudocod :

```
procedura back(k)                    Apel din Programul principal: back(1)
daca solutie(k) atunci tipar
altfel
    init(st,k)
atat timp cat succesor(st,k) executa :
                                     valid(ev,st,k)
                                     daca ev atunci back(k+1)
```

cod Pascal:

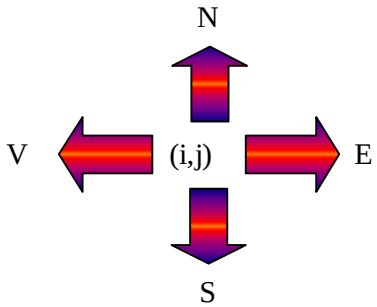
```
var  n: integer;
procedure back(k:integer);
begin if solutie(k) then tipar
        else  begin
              init(st,k);
              while succesor(st,k) do begin
                                      valid(ev,st,k);
                                      if ev then back(k+1);
                                      end;
end;
begin write('n='); readln(n); back(1); end.
```

Acest tip de recursivitate se numeste *ascendenta*, cand in primul apel, parametrul are valoarea minima, in caz contrar, se numeste r*ecursivitate descendenta* .



2.2.1 LABIRINTUL

O matrice n x n poate fi completata cu numere de la 0 la n x n-1. Fiecare numar se poate descrie in binar intr-un sir de 0 si 1.Consideram patru posibilitati de iesire dintr-o pozitie (i,j) si anume:



Deoarece admitem ca miscarea se poate face numai orizontal sau vertical. In acest mod, o matrice poate fi analogata cu un labirint. Din fiecare pozitie se va putea iesi intr-un mod descris mai sus, astfel incat valorile coordonatele unui traseu sa formeze o stiva bidimensionala: st[1,i]=x(linia) si st[2,i]= y(coloana). Pentru a testa iesirea de pe tabla matriciala vom borda tabloul de numere date cu valoarea n x

n, care nu poate aparea in nici o casuta. Cele 4 directii de miscare pot fi obtinute prin intersectia numarului din casuta (i,j) cu unul din numerele:

$2^0, 2, 2^2, 2^3$

care se scriu in baza 2(binar) astfel:

2^0 =1=0001
2 =2=0010
2^2 =4=0100
2^3 =8=1000

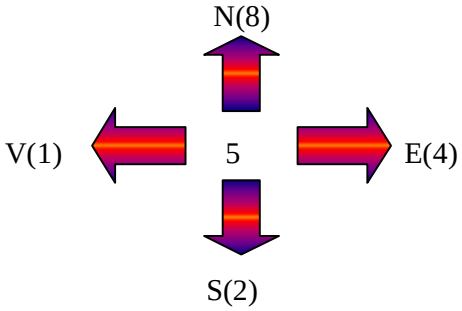
Intersectia cu numarul din casuta se face prin operatorul AND, intre continutul casutei si numerele mai sus mentionate, asimilate celor 4 directii.

Notam matricea a (i,j), i,j=1..n si a(i,j)=5= 0101(in binar). Ca urmare:

Care vor fi posibilitatile de a parasi aceasta casuta?

Intersectam pe rand cu cele patru "directii":

0101 and 0001= 0001<>0 (spre V)
0101 and 0010= 0000= 0
0101 and 0100= 0100= 01000<>0 (spre E)
0101 and 1000= 0000= 0



Asadar, intr-o casuta (i,j) cu continutul 5, se poate pleca in directiile: E si V.

In prima etapa vom citi deci dimensiunea matricei si continutul casutelor care imagineaza labirintul. De asemenea, pentru a descrie solutia se va folosi o stiva bidimensionala care va contine coordonatele punctelor (pozitiilor) parcurse. Initializarea ei se face cu pozitia de plecare initiala, si care este cunoscuta (x,y). Pentru a masura lungimea drumului- solutie se va utiliza o variabila transmisa prin adresa: k.. Acest contor va fi parametrul procedurii principale, valoarea initiala fiind k 0 (recursivitate ascendenta). Lista de parametrii formali a procedurii principale mai contine: (x,y)=coordonatele pozitiei curente; matricea a si vectorul st, variabile transmise prin nume(adresa) Incadrarea in schema presupune utilizarea unor proceduri de forma :

```
procedure init(x,y:integer;var st:stiva;var k:integer);
begin st[1,k]:=x; st[2,k]:=y; end;

procedure valid(var ev:boolean;var st:stiva;var k:integer);
  var i:byte;
begin ev:=true;
      for i:= 1 to k-1 do if (st[1,i]=st[1,k]) and (st[2,i]=st[2,k])
      then ev:=false;
end;

 function solutie(var a:matrice;x,y:integer):boolean;
begin solutie:=(a[x,y]=n*n); end;
 procedure tipar(var st:stiva;var k:integer);
 var i:byte;
begin writeln;
 for i:=1 to k do write(st[1,i],' ',st[2,i]);
 end
```

Procedura principala, care le utilizeaza va fi analoga schemei unidimensionale:

```
procedure back(var a:matrice;x,y:integer;var st:stiva;var k:integer);
  var i:byte;
 begin if solutie(a,x,y) then tipar(st,k) else begin
                                                k:=k+1;
                                                init(x,y,st,k);
                                                valid(ev,st,k);
 if ev then for i:=1 to 4 do
                                  case i of
( while succesor do                1:if(a[x,y] and 8)<>0 then back(a,x-1,y,st,k);
 back(a,x1,y1,st,k))               2:if(a[x,y] and 4)<>0 then back(a,x,y+1,st,k);
                                    3:if(a[x,y] and 2)<>0 then back(a,x+1,y,st,k);
                                    4:if(a[x,y] and 1)<>0 then back(a,x,y-1,st,k);
                                   end; k:=k-1;
                                 end;
  end;
```

Programul va citi n=dimensiunea labirintului(matricei), apoi continutul casutelor, pozitia (x,y) de plecare, si va borda suprafata cu “casute imposibile”, incarcate cu valoarea n*n:

```
for i:=1 to n do
        begin a[0,i]:=n*n; a[n+1,i]:=n*n; a[i,0]:=n*n; a[i,n+1]:=n*n;end;
```

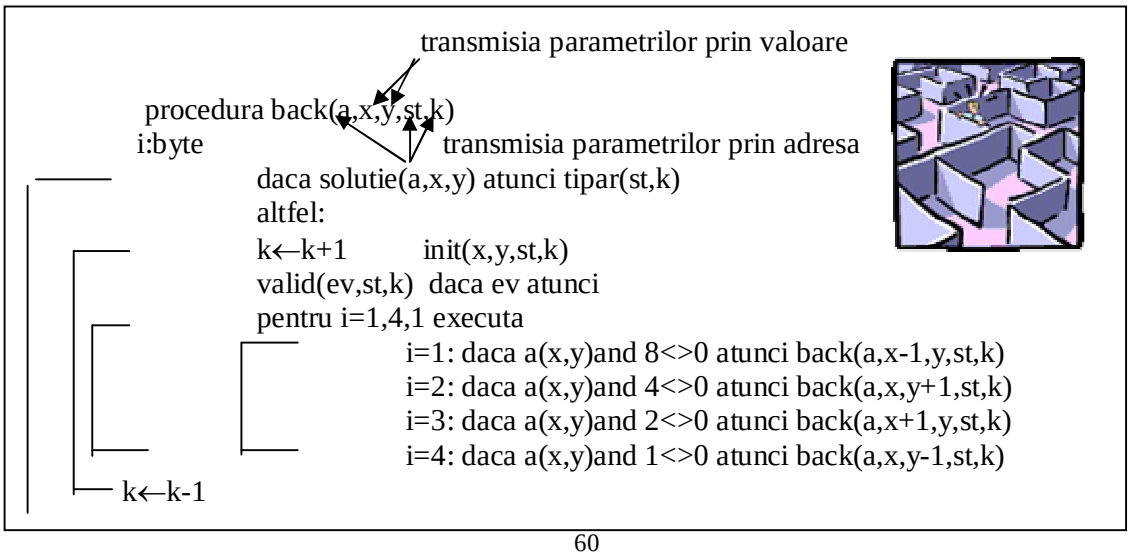
Apelul initial al procedurii principale se va face cu valoarea:

```
k:=0;  back(a,x,y,st,k);
```

Observatie: Asemanarea cu procedura standard schematizata este evidenta, fiind schimbata variabila de ascendenta: din k, cum este in standard, in pozitia (x,y) curenta.

Singura procedura care lipseste este succesorul, locul lui fiind luat de cele 4 optiuni posibile de iesire din pozitia curenta (x,y). Daca este posibila macar o optiune atunci stiva se va continua pe directia respectiva, urmand verificarea obtinerii iesirii din labirint cu testul solutie(k).Validarea se face inainte de trecerea la succesor. Daca stiva(traseul) nu este valida(corect) atunci se face descresterea k:=k-1 a nivelului din stiva, urmand reluarea cautarii din alta casuta, pe alte directii. Daca s-a obtinut solutia completa, se va tipari, in caz contrar se va continua cu initializarea unui nou nivel in stiva si cautarea va continua (k:=k+1) procedeul reluandu-se. Pseudocodul algoritmului recursiv, dupa schema, asa cum am incercat sa-l prezentam este urmatorul(comparati cu pseudocodul iterativ dupa schema):

Labirint



2.2.2 . SARITURA CALULUI

Principiul de rezolvare este analog, doar ca saritura calului se va face in 8 pozitii posibile, din centru, radial, ocupand 8 locuri ale caror coordonate se obtin din cele initiale prin adunarea cu componentele unui bivector constant :
u(i,j), cu i=1..2 si j=1..8
reprezentand cele 8 mutari posibile ale unui cal, plecand din pozitia initiala (x,y)

1: (x-2,y+1) (linia, coloana)
2: (x-1,y+2)
3: (x+2,y+1)
4: (x+1, y+2)
5: (x-1, y+2)
6: (x-2, y+1)
7: (x-2, y-1)
8: (x-1 ,y-2)

$$u=\begin{Vmatrix} -2 & 1 \\ -1 & 2 \\ 2 & 1 \\ 1 & 2 \\ -1 & 2 \\ -2 & 1 \\ -2 & -1 \\ -1 & -2 \end{Vmatrix}$$

Conditia ca o casuta sa fie un pas valid(mutare) este ca ea sa fie libera (=0) si coordonatele ei sa se afle in perimetrul permis. Solutia intreaga presupune parcurgerea(acoperirea) intregii table, ca urmare k= n*n, unde n= dimensiunea tabloului sik este contorul care urmareste traseul, si numara pasii. Notam :

st: stiva (stiva=array[1..2,1..100] of tip) solutia
a:matrice (matrice=array[1..10,1..10] of tip)

si le initializam cu “0”. Procedurile si functiile utilizate, dupa schema sunt:

```
procedure  init(xo,yo,k:integer);
begin a[xo,yo]:=k; st[1,k]:=xo; st[2,k]:=yo; end;
procedure valid( var ev:boolean;xo,yo:integer);
begin ev:=((xo<=n) and (yo<=n) and (xo>=1) and (yo>=1)
          and (a[xo,yo]=0));
end; { conditiile pentru ca piesa sa nu iasa din tabla de sah}
```

```
function solutie(k:integer):boolean;
begin solutie:=(k=n*n); end;
```

```
procedure tipar;
 var i:integer;
 begin for i:=1 to n*n do writeln('linia=',st[1,i]:4,
                                  'coloana=',st[2,i]:4); end;
```

Procedura apelanta a celor descrise anterior se incadreaza, cu mici modificari in back-ul recursiv, cu schema putin inversata, testul final si afisarea avand pozitionarea asemanatoare cu cea din back-ul elementar. Procedura succesor() lipseste, inlocuita de cele 8 miscari posibile ale calului, asemanator cu problema labirintului, unde se putea realiza deplasarea in 4 directii. Cele 8 pozitii posibile sunt validate.

SARITURA CALULUI

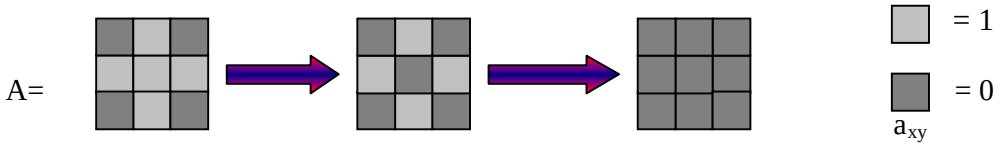
```
procedure back (x,y,k:integer);
var for I:=1 to 8 do begin
                                    xo:= x+u[1,i]; yo:=y+[2,i];
                                    valid (ev,xo,yo);
                        if ev then if solutie (k) then tipar
                        else begin
                                init (xo,yo,k);
                                back(xo, yo,k+1);
                                a[xo,yo]:=0; end;
                        end;
end;
```

Apelul initial este back(x,y,1), deci pentru k=1 iar x si y sunt date si reprezinta coordonatele punctului de plecare pe tabla de sah. Datele initiale mai contin dimensiunea tablei n. De exemplu: n=5, x=3, y=2. Matricea tablei de sah se initializeaza cu “0” pe toate componentele, la fel si stiva bidimensionala:

```
for i:=1 to n do for j:=1 to n do a[i,j]:=0;
for i:=1 to n*n do begin
                        st[1,i]:=0; st[2,i]:=0;
                        end;
```

2.2.3 ALGORITMUL DE FILL

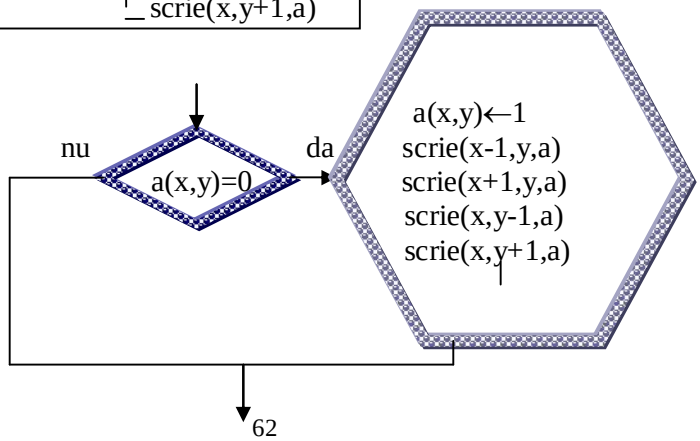
Intr-o matrice binara A(cu elementele 0 si 1) se considera vecinul unui punct (x,y) orice punct stanga, dreapta, sus, jos, in raport cu pozitia data.Daca elementul dat este 0 se transforma in 1 iar vecinii se umplu tot cu 1, continuandu-se pana in momentul intalnirii unei pozitii deja completate cu 1, caz in care executia se incheie. Pentru a nu mai testa iesirea din matrice, acesta este bordata numai cu elelmente de 1.



pseudocod:

```
procedura scrie(x,y,a)
daca a(x,y)=0 atunci  a(x,y)←1
                      scrie(x-1,y,a)
                      scrie(x+1,y,a)
                      scrie(x,y-1,a)
                      scrie(x,y+1,a)
```

schema logica:



cod Pascal

```
type matrice=array[0..10,0..10] of integer;
 var n,x,y,i,j:byte; a:matrice;
procedure scrie (x,y :byte ; a: matrice );
write('n='); readln(n); {dimensiune matrice}
writeln('dati elementele matricei(0 sau 1) :');
for i:=1 to n do for j:=1 to n do readln(a[i,j]);
{bordare cu elemente de 1}
for i:=1 to n do begin
a[0,i]:=1;a[i,0]:=1;a[n+1,i]:=1;a[i,n+1]:=1;end;
writeln('dati pozitia initiala: ');readln(x,y);
scrie(x,y,a); {apel procedura umplere}
for i:=1 to n do
begin  {afisare dupa umplere}
writeln;for j:=1 to n do write(a[i,j],' ');
end;
```

3. DIVIDE ET IMPERA

3.1 MAXIMUL SI MINIMUL DINTR-UN SIR DE ELEMENTE

Ideea este urmatoarea: sirul de elemente, asimilat unei structuri vectoriale, se va imparti mereu in doi subvectori de lungimi egale iar in momentul in care impartirea nu mai poate avea loc (capetele sunt egale) atunci maximul cautat este chiar elementul ramas in sir.

Cautarea maximului(minimului) presupune aflarea celui mai mare element din fiecare subsir si compararea acestor doua rezultate. Maximul (minimul) va fi cel mai mare (mic) dintre cele doua maxime (minime), de aceea se va folosi o functie nu o procedura.

pseudocod : (a= vectorul dat,componente ai, cu: i=1..n)

```
    functia max(i,j) de tip intreg
    daca i=j atunci max←ai
    altfel:
        max1←max(i,(i+j) div 2)
        max2←max((i+j) div 2 + 1,j)
        daca max1> max2 atunci max←max1
        altfel max←max2
```

Apelul initial: max(1,n) deci : i=1, j=n unde n=lungimea vectorului dat(numarul de elemente)

cod Pascal :

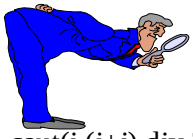
```
type vector=array[1..10] of integer;
var n,i;byte;a:vector;
        function max(i,j;integer;var a: vector):integer;
         var max1,max2:integer;
        begin
        if (i=j) then max:=a[i]
        else if (i=j-1) then
          begin
         if a[i]<a[j] then max:=a[j]
        else max:=a[i];
        end
        else
```

```
 begin
max1:=max(i,(i+j) div 2,a);
max2:=max((i+j) div 2,j,a);
if max1>max2 then max:=max1
      else max:=max2;
end; end;

        begin write('n='); readln(n);
            writeln('Elementele vectorului: ');
            for i:=1 to n do readln(a[i]);
            writeln('Maximul este: ',max(1,n));
            readln;
            end.
```

3.2 CAUTARE BINARA

Pentru gasirea unei valori intr-un sir dat, algoritmul "cautarii binare" utilizeaza metoda divide et impera, conform cu care, cautarea se va face succesiv, pe jumatati in descrestere, iar decizia de selectare a unei astfel de diviziuni depinde de rezultatul compararii cu "mijlocul" la un moment dat.
pseudocod:

```
functie caut(i,j) :boolean
daca i=j atunci
        daca a(i)=x atunci caut←'da'  (caut=adevarat)
                    altfel caut←'nu'   (caut=fals)
altfel              daca x< a((i+j) div 2) atunci caut←caut(i,(i+j) div 2)
                    altfel caut←caut((i+j) div 2 + 1, j)
```



Se poate utiliza procedura, apeland la o variabila logica auxiliara:

```
gasit:boolean ( declaratie globala)
procedure caut(i,j);
daca i=j atunci     daca a(i)=x  atunci gasit ←'da'  (gasit=adevarat)
                    altfel gasit ='nu'   (gasit=fals)
altfel              daca x< a((i+j) div 2) atunci caut(i,(i+j) div 2)
                    altfel caut((i+j) div 2 + 1, j)
```

Valorile parametrilor la primul apel si, atat pentru functie cat pentru procedura sunt legate de lungimea sirului dat, iar elementul x cautat este declarat si initializat global:

valorile parametrilor la primul apel: i=1 si j=n

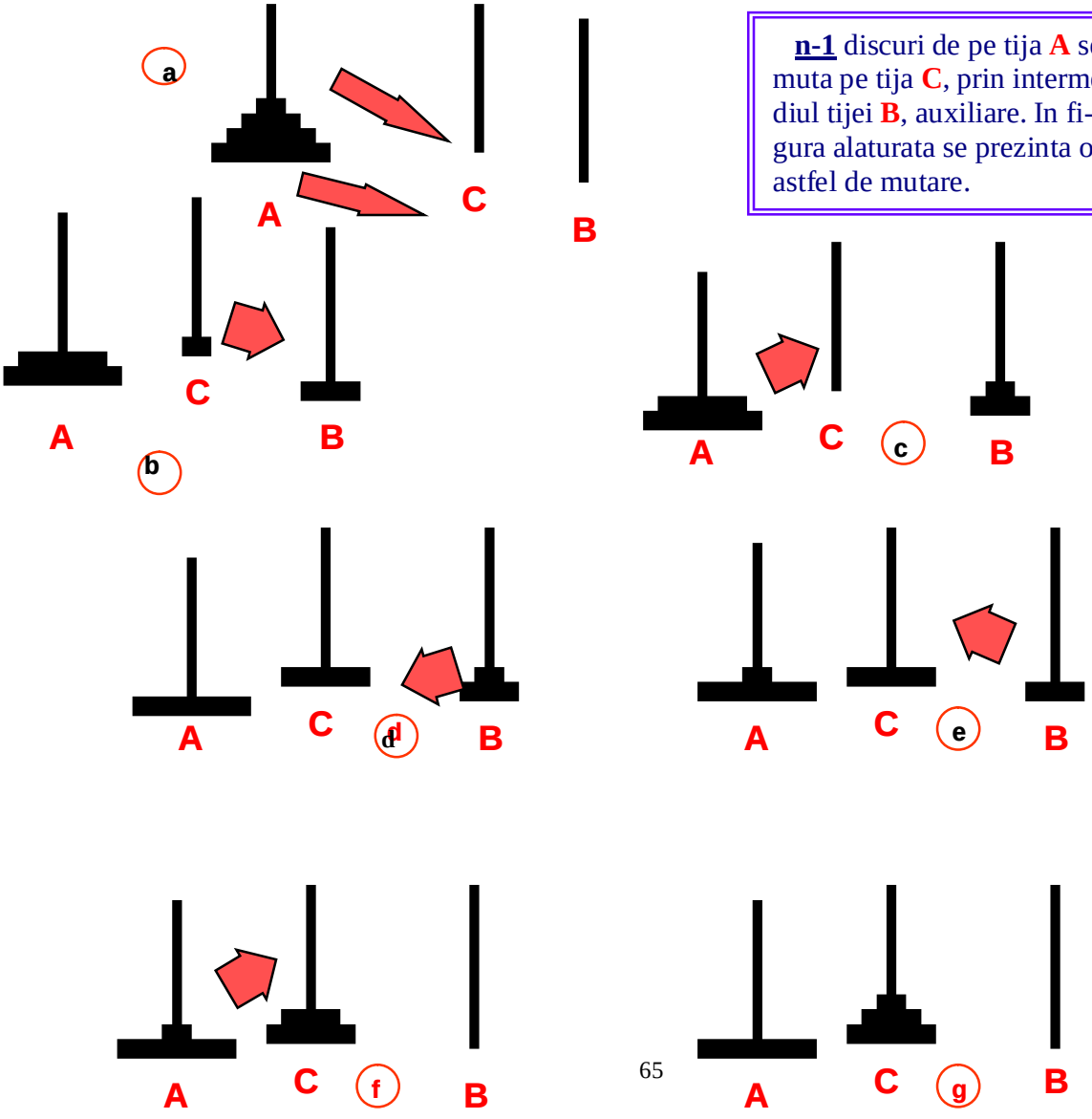
In continuare prezentam codul Pascal asociat utilizarii procedurii, lasand ca exercitiu adaptarea acestuia pentru utilizarea functiei de cautare analoge, precum si translatia in cod C++.

```
procedure caut(i,j:integer);
                  begin
                  if (i=j) then  begin  if a[i]=x then gasit:=true
                                        else gasit:=false;
if gasit then writeln('da') else writeln('nu');
                                  end
                  else  begin if x<a[(i+j) div 2] then caut(i,(i+j) div 2)
                                  else caut((i+j) div 2 + 1,j);
                                        end;
```

```
        end;
        begin  write('numar elemente sir dat, n='); readln(n);
        writeln('Elementele sirului dat: ');
        for i:=1 to n do readln(a[i]);
write('Valoarea cautata: '); readln(x);
        writeln('Este valoarea in sirul dat ?');write('Raspuns: ');
          caut(1,n);
              readln;end.
```

3.3 TURNURILE DIN HANOI

Aceeasi metoda "Divide et Impera" , rezolva si problema ce o vom prezentata in continuare , cunoscuta sub numele anuntat : problema se va descompune in doua subprobleme ale caror rezultate se vor compara, si combina pana la obtinerea rezultat cerut.



n-1 discuri de pe tija **A** se muta pe tija **C**, prin intermediul tijei **B**, auxiliare. In figura alaturata se prezinta o astfel de mutare.

f **g**

Text: Pe o tija A se afla n discuri asezate de jos in sus in ordinea descendendenta a dimensiunii. Se cere sa se mute aceste discuri pe o alta tija B, unul cate unul, respectand regula, deci deasupra unui disc dat se va aseza numai unul mai mic. Se va utiliza o tija intermediara C.
In acest context, vom apela la metoda Divide et Impera, deci vom descompune problema succesiv, recursiv in raport cu n, in doua subprobleme dupa cum urmeaza:

Subproblema 1
Se muta n-1 discuri de pe A pe C utilizand ca tija intermediara B
Subproblema 2
Se muta n-1 discuri de pe C pe B utilizand ca tija intermediara A (n>=2)
Pentru un singur disc: (A,B)

Notam functia rezolvanta cu : H(n,a,b,c) unde a,b,c vor fi de acum tijele date. Asadar pentru n=1 : H(1,a,b,c)=(a,b)

In algoritm vom asocia acestor tije chiar literele corespunzatoare, pentru a descrie mutarile pe tije. Functia va fi de tip recursiv descendent :
(Tija intermediara se va situa permanent dupa tijele sursa si destinatie- ultima)

$$H(n,a,b,c) = \begin{cases} (a,b) & \text{daca } n=1 \\ H(n-1,a,c,b),\ ab,\ H(n-1,c,b,a) & \text{daca } n>=2 \end{cases}$$

Forma de procedura, cu afisarea mutarilor prin intermediul caracterelor ‘a’, ‘b’, ‘c’ va avea forma:
pseudocod :

```
procedura H(n,a,b,c)
daca n=1 atunci scrie (a,b)
altfel
    H(n-1, a, c b)
    scrie ( a,b )
    H(n-1, c, b, a)
```

cod Pascal: (Valorile initiale n,a,b,c se vor da de catre utilizator)

```
var a,b,c: char; n:byte;
      procedure H(n,a,b,c);
      begin
      if n=1 then writeln(a,b)
                  else begin
                    H(n-1,a,c,b);
                    writeln(a,b);
                   H(n-1,c,b,a);
                    end;
       end;
begin write(‘numar de discuri= ‘); readln(n);
                  a:=‘a’; b:=‘b’; c:=‘c’;
                  H(n,a,b,c);
                   readln;
    end.
```

PROBLEME LA CAPITOLUL III

1. CALCULAREA MAXIMULUI PRIN RECURSIVITATE

Functia MAXIM(I) cu valori de tipul elementelor date, este construita pe principiul compararii consecutive a valorii globale preluate din principal si refacerea acesteia daca este intalnit un element superior . (initializarea maximului se va face cu primul element :max:=v[1])

```
type  tablou= array[1..10] of integer;
var n, i, max, j, aux : integer; v:tablou;
function maxim(i:integer):integer;
begin
if (max<v[i]) then max:=maxim(i+1); maxim:=max;
end;
begin
readln(n); for i:=1 to n do readln(v[i]); max:=v[1]; write(maxim(2):5);
end.
```

2. CALCULUL RECURSIV AL COMBINARILOR

Se utilizeaza formula de recurenta din matematica :

```
var k,n:byte;
function combinari(n:byte;k:byte):real;
begin
if ((k=0) or ((k=1) and (n=1))) then combinari:=1
else
if (k>=1) and (n>1) then combinari:=combinari(n-1,k)+combinari(n-1,k-1);
end;
begin
readln(n,k);
write(combinari(n,k):7:2);
end.
```

3. CALCULUL RECURSIV AL FUNCTIEI PUTERE

Si aici este utilizate recurenta matematica specifica, functiei putere, de data aceasta.

```
Program putere_recursiva;
var x,n:byte;
function putere(k:byte):longint;
begin
if k=0 then putere:=1 else
putere:=x*putere(k-1);
end;
begin
readln(x,n);
writeln(putere(n):7);
readln;
end.
```

4. ORDONAREA UNUI SIR DE CARACTERE, UTILIZAND METODA DIVIDE ET IMPERA

Transmiterea sirului se realizeaza prin adresa deoarece el urmeaza sa fie modificat ulterior.

```
Program sir_ordonat_binar;
var a:string;
function divide(l,r:byte;v:string):string;
```

```
var k,n:byte;aux:char;
begin
n:=length(v);
if l=r then divide:=v[l] else
if r=l+1 then begin
aux:=v[l];v[l]:=v[r];v[r]:=aux;
divide:=v;
end
else begin
k:=trunc((l+r)/2);
divide:=divide(l,k,v)+divide(k+1,r,v);
end;
end;
begin
readln(a);
writeln(divide(1,length(a),a));
readln; end.
```

5. SE DAU TREI BILE DE CULORI DIFERITE, ASEZATE PE O TIJA. SE CERE SA SE MUTE PE O ALTA TIJE, UTILIZAND O TIJA INTERMEDIARA

Se va utiliza metoda din “problema tunurilor din Hanoi”, deci metoda divide et impera, adaptata la cerintele specifice.

```
Program tije;
 Const nrtije=3; { 1+2+3=6}
 Type tija=1..nrtije;
 Var n:byte;
Procedure muta_bile(x:byte; I,j:tija);
  { se muta o bila de pe tija I pe tija j}
  begin
 if x>1 then { daca ramane macar o bila}
begin
 muta_bile(x-1, i,6-i-j); {tija 6-i-j este de manevra}
writeln(I, ‘->’, j);
muta_bile(x-1, 6-i-j, j);
end
else
writeln(i, ‘->’, j);
end;
begin
      write(‘Numarul de bile n=’); readln(n);
      muta_bile(n,1,2);
      end.
```

CAPITOLUL V

ANALIZA COMBINATORICA

I. Produs cartezian

Metoda aplicata se numeste “lexicografica” deoarece se pleaca de la cel mai mic element in sens lexicografic si anume : v=(1,1,..,1) pentru un produs de “m” multimi avand fiecare n[m] elemente. Fie concret, un exemplu:

m=2 (doua multimi)
A={1,2,3}, n[1]=3 (card A)
B={1,2,3,4}, n[2]=4 (card B)

Atunci metoda va urmari urmatoarele etape:

			SOLUTIE
j=1 k=1	v(1)=1	v(2)=1 k=2	scrie (v(1), v(2))= (1,1) k=3
j=1 k=1	v(1)=1	v(2)=2 k=2	scrie (v(1),v(2))= (1,2) k=3
⋮			⋮
j=3 k=1	v(1)=3	v(2)=4 k=2	scrie (v(1),v(2))=(3,4) k=3

Pseudocod :

```
pentru j=1, n[k], 1 executa
    v(k) = j
    k←k+1
```

J este indicator pentru elementele din fiecare multime; el se va modifica numai dupa ce vor fi cuplate toate elementele acesteia.(in acest exemplu elementele unei multimi coincid cu pozitiile lor in multime) K care mentioneaza multimea, mai precis pozitia ei in vectorul de multimi dat.(a cata este)

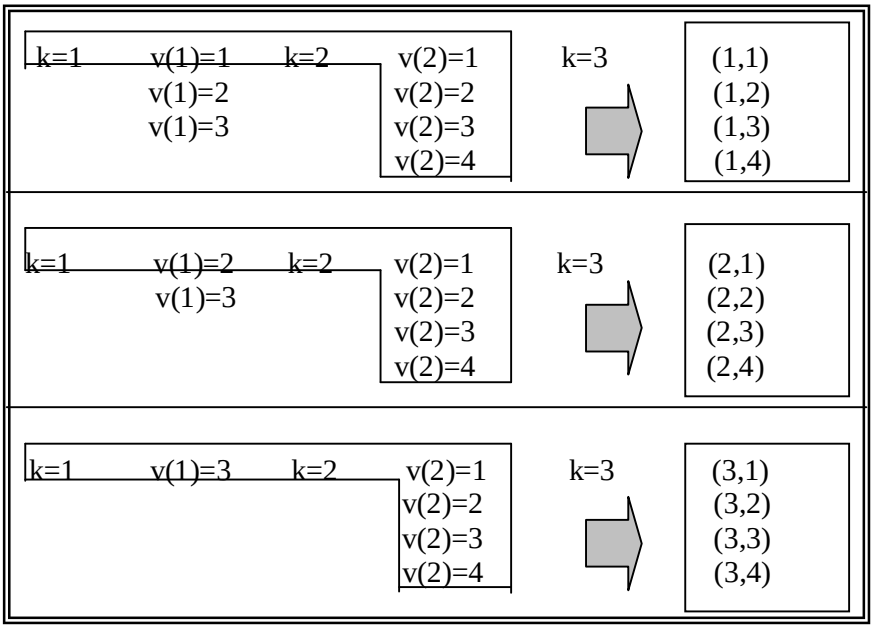
Afisarea intervine la k = m+1, unde m=numarul de multimi, n(i), i=1..m este vectorul numarului de elemente pentru fiecare multime iar elemenetele sunt numere naturale incepand cu 1 si terminand cu cardinalul acelei multimi. Solutia este pastrata de vectorul de perechi :

V xy={(x,y)|x∈Ai,y∈Aj, i=1,m; j=i+1,m}

m=1	n[1]=3 1,2,3 j=1,…,n[1]	A={1,2,3}
m=2	n[2]=4 1,2,3,4 j=1,…,n[2]	B={1,2,3,4}

Autor: CAZACU N. NINA LICEUL C. A. ROSETTI,, BUCURESTI

O schema restransa a algoritmului este desenata in figura ce urmeaza. Procedura in pseudocod va fi recursiva dupa k=indicele numarator al multimilor(la noi k<=2) Recursivitatea este in raport cu k deoarece fiecare element al produsului cartezian va avea exact m elemente, afisarea lui intervenind pentru k=m+1. SOLUTIE



Conform schemei precedente, obtinerea elementelor in ordine lexicografica este evidenta. Presupunand ca avem o solutie v(i), i=1..m , cu v(i)< n[i], i=1..m atunci exista o solutie urmatoare (succesoare) doar daca unul din elementele vectorului (v1,…. vm) poate fi marit. In caz contrar, s-a obtinut pentru fiecare componeneta, cardinalul fiecarei multimi date:(n[1], n[2], …, n[m]).
In cazul nostru, ultimul element este perechea (3,4), unde : n[1]=3, n[2]=4

Procedura in pseudocod :

```
procedura prod_cart(k)
j:byte
daca (k=m+1) atunci scrie_solutia
        altfel   pentru j=1,n(k),1 executa :
                     v(k)=j ; prod_cart(k+1)
```

Apelul initial este pentru valoarea k=1, recursivitatea fiind ascendenta ⇨ prod_cart(1)
Codul Pascal corespunzator va fi :

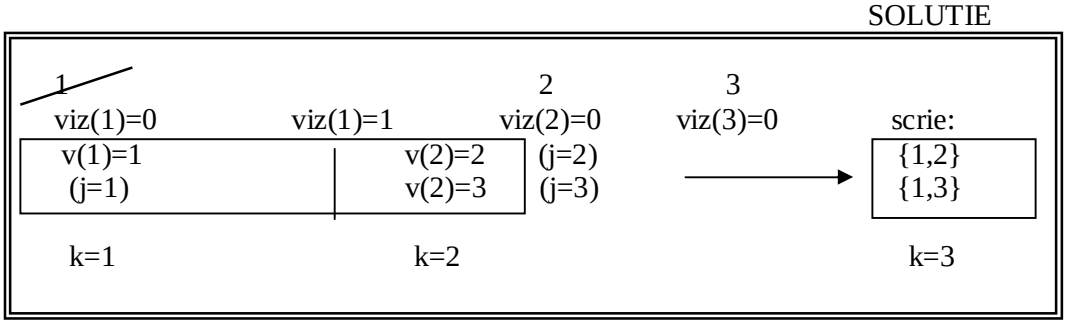
```
var m,j:byte;  v,n:array[1..100] of byte;
        procedure prod_card(k:byte);
        var j:byte;
        begin if k=m+1 then scrie else
                for j:=1 to n[k] do begin
                                v[k]:=j;
                                prod_cart(k+1);
                                end;
        end;
begin write (‘m=’); readln(m); writeln(‘cardinalele celor’,m,’multimi:’);
```

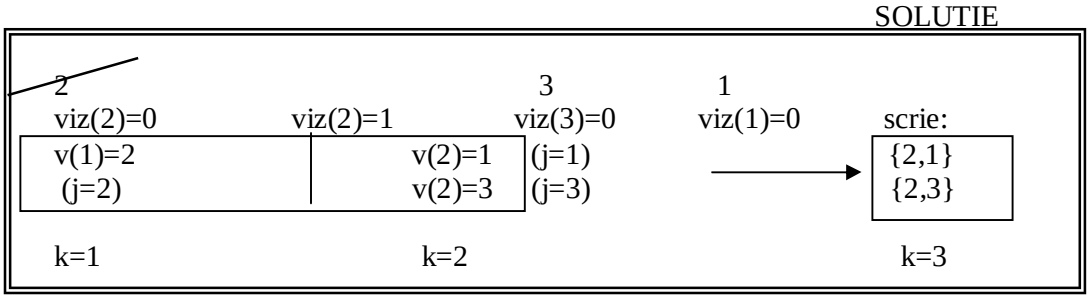
for i:=1 to m do readln (n[i]); {apel procedura} post_card(1); end.

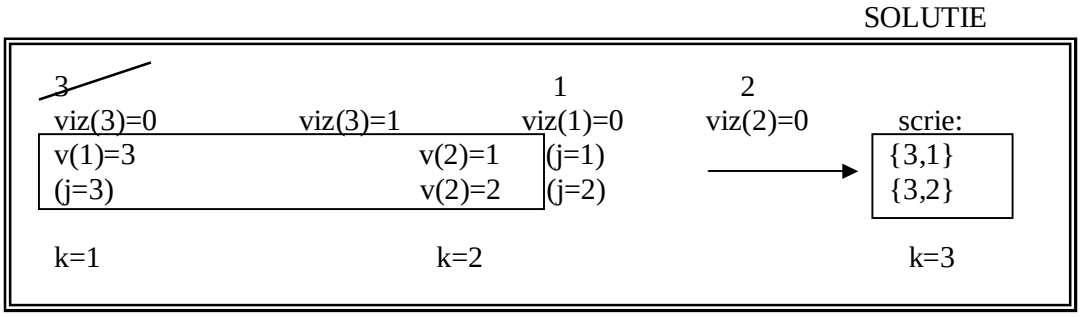
Complexitatea este de ordinul O(m* n^m) *deci nu foarte performanta*.

II. Aranjamente (de n luate cate m, m<=n)

In rezolvarea care urmeaza se va folosi tot metoda lexicografica deci ordonarea crescatoare a elementelor si recursivitatea ascendenta. Pentru ca elementele vectorului de m componente sa nu se repete, ele fiind in aceasi submultime, se utilizeaza si un vector de marcare : viz(i), i=1….n , initializat cu 0 peste tot. Daca un numar din multimea data este scris intr-o solutie de m elemente, el va fi "marcat", adica viz(i)=1, pentru a nu se repeta. Fie : m=2, n=3 si multimea A = {1,2,3} din care ne propunem obtinerea submultimilor (aranjamentelor) de 2 elemente. Se stie ca aranjamentele sunt submultimi ale unei multimi date, care difera intre ele atat prin natura obiectelor cat si prin ordinea din interior , intre elementele submultimii.







pseudocod :

```
procedura aranjam(k)
daca k=m+1 atunci scrie_solutia
        altfel : pentru j=1,n,1 executa
                    daca viz(j)=0 atunci   v(k)=j
                                           viz(j)=1
                                           aranjam(k+1)
                                           viz(j)=0
```

Cod Pascal :

```
var v:array[1..100] of byte; viz:array[1..100] of 0..1; m,n,j,k:byte;
                    procedura aranjam (k:byte);
        begin if k=m+1 then scrie_solutie else
                    for j:=1 to n do
                        if viz[j]=0 then begin
                            v[k]:=j; viz[j]:=1;
                            aranjam(k+1);viz[j]:=0;
                            end;
        end;
{Program principal}
        begin {citeste m si n}
            write ('n=');readln(n); write('m='); readln(m);
            {initializeaza vectorul marcajelor}
            for j:=1 to n do viz[j]:=0;
            {apeleaza recursiv ascendent procedura aranjam (k)};
            aranjam(1);
    end.
```

Complexitatea este de ordinul : O(n!/(n-m)!)
Analog, pentru m=n se obtin permutarile a n obiecte ; se transforma corespunzator codul precedent:

```
var v:array[1..100] of byte; viz:array[1..100] of 0..1; n,j,k:byte;
procedure perm(k:byte);
begin if k=n+1 then scrie solutie else
                    for j:=1 to n do
                        if viz[j]=0 then begin
                            v[k]:=j; viz[j]:=1;
                            perm(k+1);
                            viz[j]:=0; end; end;
```

```
        begin
        write('n='); readln(n);
        {initializeaza vectorul marcajelor}
        for j:=1 to n do viz[j]:=0;
{apeleaza recursiv ascendent procedura perm(k) pentru k=1
        perm(1);
        end.
```

Complexitatea algoritmului se obtine prin particularizare pentru m=n, ordinul O(n!).(0!=1)

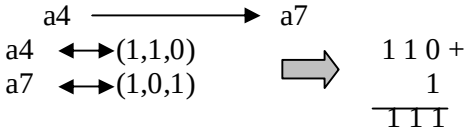
III. Submultimile unei multimi

Metoda se numeste "metoda vectorilor caracteristici", deosebind rezolvarea care urmeaza de cele doua precedente. Sa luam pentru exemplificare multimea:

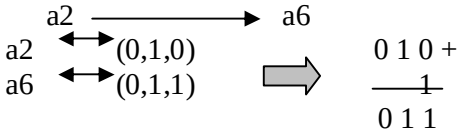
A= {1,2,3} avand submultimile:

a1={1}	(1,0,0)	a4={1,2}	(1,1,0)	a7={1,2,3}	(1,1,1)
a2={2}	(0,1,0)	a5={1,3}	(1,0,1)	ϕ	(0,0,0)
a3={3}	(0,0,1)	a6={2,3}	(0,1,1)		

Se observa ca nu s-au repetat multimile prin permutarea obiectelor, stiut fiind ca nu se vor obtine multimi diferite. Deci obtinerea unei submultimi a(j) din a(i) s-ar putea face conform adunarii binare: (sistem de numeratie cu baza 2)

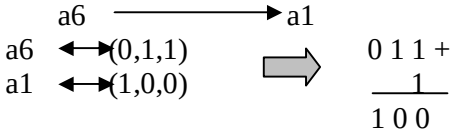


sau :



In ambele cazuri : 0 → 1

In alte cazuri, de exemplu :



Deci in acest caz : 1 → 0

1 semnaleaza existenta elementului de pe pozitia respectiva in submultime,iar 0 absenta lui.
Pseudocodul se reduce la urmatorul rationament (algoritm):

```
daca vj←1 atunci :
        vj←0
        j←j - 1
altfel :  vi←1    scrie solutia
```

Operatiile se efectueaza de la dreapta la stanga iar indicele j trebuie sa fie strict pozitiv. Mai utilizam ca urmare o variabila semafor (posibil) Pseudocodul devine:

```
posibil←true
atat timp cat posibil executa:
        j←n
        atat timp cat (vj=1) executa:
                vj=0
                j←j - 1
        daca j=0 atunci posibil← false
        altfel :  vj←1
                  pentru i=1,n,1 executa:
                          daca vi=1 atunci scrie(i)
```

codul Pascal :

var v: array[1..100] of 0..1; i,j,n:byte; posibil: boolean;

```
procedure submultimi;
begin posibil:=true; for i:=1 to n do v[i]:=0;
while posibil do begin
                 j:=n;
                  while (v[j]=1) do begin
                                        v[j]:=0;
                                         j:=j-1;
                                         end;
If j=0 then posibil:=false else
                                    begin
                                    v[j]:=1;
                                    for i:=1 to n do
                                    if v[i]=1 then write(j:4);
                                    end;
                                   end;
```

```
end;
```

```
begin
        write('n='); readln(n);
        { numarul de elemente ale multimii date, considerate de la 1 la n}
        submultimi; {apel procedura}
        end.
```

OBS: Ordinul de complexitate, este analog celui de la produsul cartezian

III. Metodele clasice de programare in comparatie cu cele ale analizei combinatorice

Dupa cum am vazut, problemele abordate in acest capitol nu sunt noi. Subiectele lor le-am intalnit si la capitolele precedente. Difera metodele de rezolvare. Analiza combinatorica foloseste vectorii caracteristici si ordonarea lexicografica.

Sa consideram problema aranjamentelor de n luate cate m(m<=n)

A) metoda backtracking

```
program aranjamente;
uses crt;
type stiva=array[1..100] of integer;
var st:stiva;
n,m,k:integer;
as,ev:boolean;

procedure init(var st:stiva;k:integer);
begin
st[k]:=0;
end;

procedure succesor(var st:stiva;var
as:boolean;k:integer);
begin
if st[k]<n then
  begin
    st[k]:=st[k]+1;
    as:=true;
  end  else as:=false;
end;

procedure valid(var st:stiva;var ev:boolean;k:integer);
var i:integer;
 begin
  ev:=true;
  for i:=1 to k-1 do if (st[k]=st[i]) then ev:=false;
 end;

function solutie(k:integer):boolean;
begin
 solutie:=(k=m);
end;

procedure tipar;
var i:integer;
 begin
 for i:=1 to m do write(st[i],' ');
 writeln;
 end;
{======================}
 begin
 clrscr;
 write('n= ');readln(n);
                write('m= ');readln(m);
                k:=1;
                init(st,k);
                while k>0 do
                begin
                 repeat
                succesor(st,as,k);
                if as then valid(st,ev,k);
                until (not as) or (as and ev);
   if as then if solutie(k) then tipar
     else
       begin
         k:=k+1;
         init(st,k);
       end else k:=k-1;
  end;
readln;
end.
```

Se pleaca de la un vector nul, care se completeaza cu m elemente, punand conditia ca acestea sa nu se repete in interiorul unei submultimi(procedura valid) si cautand succesori pana la cel maxim, care este n (procedura succesor)

B) Metoda vectorilor caracteristici-analiza combinatorica

```
Program
generare_submultimi_cu_vectori_caracteristici;
Var n:byte;
Procedure generare(n:byte);
```

Autor: CAZACU N. NINA LICEUL C. A. ROSETTI,, BUCURESTI

```
var v:array[1..10] of byte;f:text;
    e_posibil:boolean;i,j:byte;
  begin
  assign (f,'vectori.txt');
  Rewrite(f);
  Writeln(f,'Submultimile multimii {1,2,..,.,n}');
  Writeln(f,'multimea vida');
  for i:=1 to n do v[i]:=0;
  e_posibil:=true;
  While e_posibil do begin
      j:=n;
      While (j>0) and (v[j]=1) do begin
                  v[j]:=0;
                  j:=j-1;
            end;
   if j=0 then e_posibil:=false
   else begin
    v[j]:=1;
    write(f,'{');
    for j:=1 to n do
        if v[j]=1 then write(f,j,' ');
    writeln(f,'}');
    end;
  end;
  close(f);
end;
begin
Write('n=');
readln(n);
generare(n);
end.
```

Rezultatele executiei, sub forma unor multimi, incadrate de acolade, vor fi trecute intr-un fisier de tip text. Restul este analog cu prezentarea de la paragraful &1, acest capitol.

In varianta rezolvarii cu “obiecte”, procedurile sunt campuri ale unor inregistrari care sunt chiar o-biectele utilizator :

C) Se considera o multime de n copii, din care nb sunt baieti si restul sunt fete. Sa se formeze grupuri copii, in care diferenta intre numarul de baieti si cel de fete sa nu fie mai mica decat o diferenta data dif.

```
program grupuri;
uses crt;
const nmax=25;
type vect=array[0..nmax] of integer;
submul=object
    v:vect;
    e_posibil :0..1;
    n:1 ..nmax;
procedure init;
procedure generare;
end;
  var s:submul;
  nb,nf,n,i,dif :integer;
procedure submul.init;
var i:integer;
begin for i:=0 to n do v[i]:=0;
e_posibil:=1
end;
procedure submul.generare;
var i:integer;
begin i:=n;
  while v[i]=1 do begin v[i]:=0;
          i:=i-1;
        end;
 if i=0 then e_posibil:=0
 else v[i]:=1;
end;
procedure prelucrare(n,nb,nf,dif:integer; v:vect);
var  n1,n2,i:integer;
begin
    n1:=0;
    n2:=0;
    for i:=1 to nb do if v[i]=1 then n1:=n1+1;
    for i:=nb+1 to n do if v[i]=1 then n2:=n2+1;
    for i:=1 to n do if v[i]=1 then write (i, ' ');
    if (n1-n2>=0) and (n1>=n2+dif) then
    writeln (' este solutie')
    else writeln (' nu este solutie ');
    readln;
    end;
{program principal}
begin clrscr;
write ('introduceti numarul de copii: ');readln (s.n);
write ('introduceti numarul de baieti: ');readln (nb);
write ('introduceti diferenta: '); readln (dif);
with s do begin init;
while e_posibil<>0 do begin
          prelucrare(n,nb,nf,dif,v);
          generare;
     end
  end
end.
```

Definirea obiectelor se realizeaza utilizand cuvantul cheie OBJECT in modul urmator:

```
Type submul=object
     v:vect;
     e_posibil :0..1;
     n:1 ..nmax;
     procedure init;
     procedure generare;
```

```
                    end;
            var s:submul;
```

Campurile obiectului "submul" (submultimi) sunt : v-de tip vector, e_posibil, cu rol de semafor numeric, n-de tip numeric, precum si procedurile **init, generare.**
Declararea obiectului "submul", se realizeaza in sectiunea VAR, dupa ce a fost definit ca tip in Sectiunea TYPE. Variabile de tip obiect cu care se lucreaza este **s** (:submul)
Apelul la campuri se realizeaza ca in orice inregistrare, iar operatorul WITH are acelasi rol.

CAPITOLUL VI

STRUCTURI DINAMICE DE DATE

Pentru inceput vom face o clasificare a structurilor inlantuite care se studiaza cu operatiile specifice:

1. Structurile de date inlantuite: stiva, coada, lista dubla, lista circulara
2. Operatii specifice cu aceste structuri: creare, inserare, stergere, cautare.
3. Arbori binari (un caz aparte de structuri dinamice)
   Operatii cu arbori binari: creare, traversare, inserare, stergere, cautare
4. Arborii de cautare (un caz particular de arbori)
   Operatii specifice: creare, cautare.

Pe parcursul acestui capitol nu vom insista asupra pseudocodului, dat fiind ca nu se folosesc algoritmi noi ci se creaza proceduri specifice, verificate, in cod, ce permit studiul acestor tipuri de date aparte. Procedurile standard de alocare (**new**()) si de dealocare (**dispose**()) a memoriei sunt specifice limbajului Pascal.

Memoria unui calculator se prezinta sub forma unei succesiuni de octeti, fiecare octet avand o adresa binara. Prin conventie, adresa este numarul de ordine al unui octet. Un octet este format din 8 biti, fiecare bit putand memora 0 sau 1. Diversele tipuri de date ocupa doi sau mai multi octeti si au un nume, o adresa si o valoare. Daca o variabila este retinuta pe o lungime de p octeti, atunci adresa ei este memorata pe primul octet din cei p consecutivi. Cuvantul **nil** reprezinta adresa nula.

Alocarea in Pascal se poate face in doua moduri : static si dnamic.

A) In primul caz, alocarea se face o singura data la inceputul programului, in sectiunea VAR.
B) In al doilea caz, alocarea se face pe parcursul programului si numai daca este nevoie, renuntandu-se in caz contrar, strategie care conduce la economie de memorie.

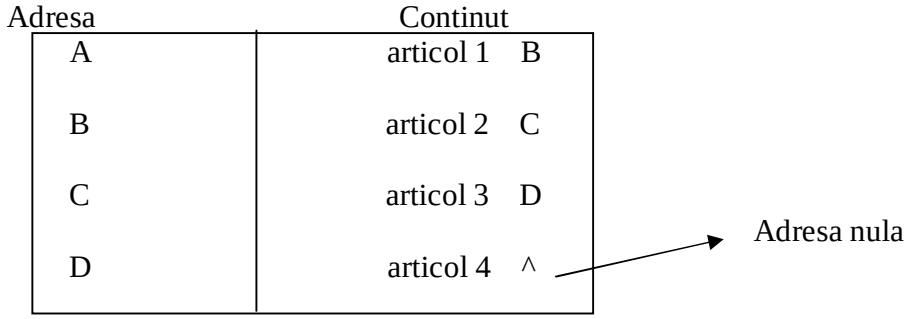
Alocarea inlantuita

Sa alcatuim urmatoarea schema :

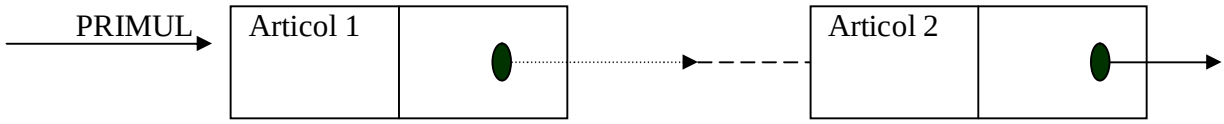
L0+C	ARTI 1
L0+2C	ARTI2
L0+3C	ARTI3
L0+4C	ARTI4

(ALOCARE SECVENTIALA)

Sa presupunem ca fiecare mod retine in el si adresa nodului urmator. Se va obtine : alocarea "dinamica"



O alta reprezentare va folosi pentru adrese sagetile :



Un nod ocupa un cuvant care este format din doua campuri : INFO, LEG
Alocarea dinamica are loc intr-o secventa de tipul:

```
type ref=^regi;
        regi=record
            info:integer; leg: ref;
            end;
    var p:ref;
```

Aici, variabila p de tip referinta retine adrese de inregistrari, fiecare inregistrare avand doua campuri: informatia utila si adresa unei alte inregistrari. Procedura new(p) retine spatiu pentru o inregistrare, in numar de octeti consecutivi, primul octet avand adresa depusa in p; daca p a retinut adresa unei inregstrari, atunci procedura dispose(p) elibereaza acest spatiu. Accesul la camp :

a) acces la campul informational: p^. info;
b) acces la campul adresa: p^. info;
c) acces la campul informational al carui prim octet este retinut in p^.leg: p^.leg^.info;

Vom incepe prin prezentarea pe scurt a listei liniare simplu inlantuite (l.l.s.i.) (Vezi Anexa_2) Cele mai utilizate structuri sunt listele simple, fie de tip stiva, fie de tip coada. Lista simpla de tip general, este prezentata mai jos, prin procedura de creare iterativa:

```
type  lista=^ nod;
        nod=record
      info:integer; leg:lista;
      end;
```

(Ca exercitiu, propunem crearea l.l.s.i. de tip recursiv)

```
var p,b:lista;
```

{variabila “p” retine adresa primei inregistrari , iar alta variabila “b” va retine adresa ultimei inregistrari, create }
{ in sirul de n-1 elemente }

```
procedure creare;
var nou:stiva; j,n:byte;
begin {creare nod de inceput}
      new(nou); readln(nou^.info); p:=nou;b:=nou;
      {creare celelalte noduri}
      readln(n);
      for j:=2 to n do begin
                  new(nou); readln(nou^.info);
                  b^.leg:=nou; b:=nou;
```

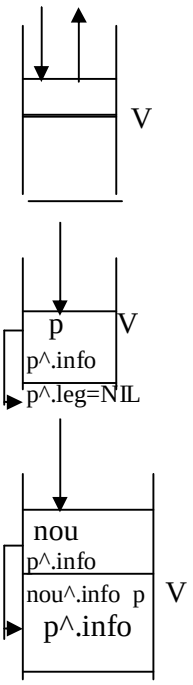
```
end;      end;
```

Listarea este aceeasi cu cea a cozii(vezi II).

I. STIVA

lifo ⟷ last in, first out

I.1 Creare stiva dinamica




```
type  stiva=^ nod;
              nod=record
              info:integer; leg:stiva;
              end;
```

```
var v:stiva;
 procedure creare;
 var nou, p:stiva; j,n:byte;
  begin {creare nod de inceput}
              new(p); readln(p^.info); p^.leg:=nil;
              {creare celelalte noduri} v:=p;
              readln(n);
 for j:=2 to n do begin
              new(nou); readln(nou^.info);
               nou^.leg:=v; v:=nou;
              end;
end;
```

I.2.1 Stergerea unui nod

```
var aux:stiva;
begin if v=nil then write('stiva vida')
                    else
                    begin
                    aux:=v;
                    v =v^.leg;
                    dispose (aux);
                    end;
end; {singurul nod care se poate sterge este v=varful}
```

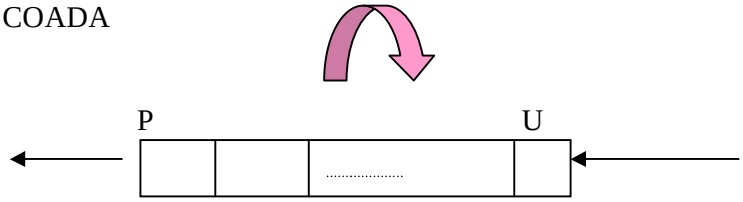
I.2.2 Inserarea unui nod

```
var nou:stiva;
begin new(nou); readln (nou^.info);
        nou^. leg:=v; v:=nou;
end; {inserarea se poate face doar la varf, deci peste v}
```

I.2.3 Cautarea unei valori date x (secventa nod cu nod)

```
var p:stiva;
begin p:=v; while(p<>nil) do begin
                if x=p^.info then write('gasit');
                p:=p^.leg;
                end;
end;
{accesul este numai secvental, deci cautarea se va face nod cu nod}
```

II COADA



fifo ⟷ first in, first out

Intrarile (iesirile) se vor efectua numai la sfarsit iar iesirile numai la inceput (stergerile). Spre deosebire de stiva care avea numai pointerul de varf v, coada are doi pointeri: unul de inceput (p-primul) si unul de sfarsit (u-ultimul)

II.1 Crearea cozii

```
var p:coada; {variabila globala}
procedure creare;
var nou:coada; i,n: byte; {variabile locale}
begin
new(p); readln(p^.info); p^.leg:=nil; u:=p;
{primul nod inserat devine ultimul, daca ele se vor efectua}
{urmatoarele inserari}
write(' dati numarul total de noduri:'); readln(n);
for i:=2 to n do  begin
    new(nou); readln(nou^.info);
    nou^.leg:=nou;
    u:=nou;
    end;
end;
```

II.2 Operatii in coada

II.2.1 Adaugare (dupa ultimul element)

```
var nou: coada;
begin new(nou); readln(nou^.info);
      nou^.leg:=nil;
      u^.leg:=nou;
      u:=nou;
end;
```

II.2.2 Stergere (primul)

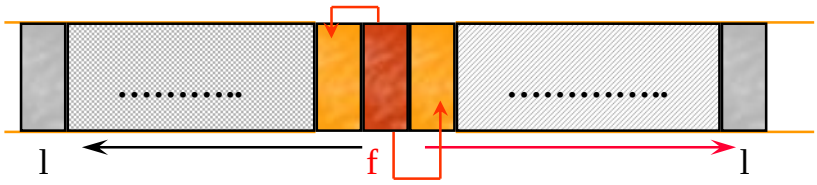
```
                                    var p, aux: coada;
        aux:=p; p:=p^.leg;{ inainteaza pointerul} dispose(aux);
        end;
```

II.2.3 Listare (afisare secventiala)

```
                    procedure listare;
                    var x:coada;
{x este o variabila de lucru de tip coada folosita pentru pozitionare pe pointerul}
{de inceput si apoi parcurgere secventiala- nod cu nod}

                    begin
                    x:=p;
                    while x<>nil do begin
                    write(x^.info,' ');
                    x:=x^.leg;
                    end;
                    end;
```

III Lista dublu inlantuita(lista dubla)

Acest tip de structura este tot un fel de coada in sensul ca prezinta doi pointeri dar numai unul este fix, cel de inceput, celalalt putand sa se aseze fie la stanga, fie la dreapta deoarece acest tip de coada este bidirectional.



Se schimba si tipul care defineste nodul, dupa cum urmeaza:

```
                    type  lista=^nod;
                                        nod=record
                                        info:integer;
                                       prec, urm: lista; end;
```

III.1 Crearea listei duble

```
    var f,l: lista;
procedure creare;
        var  i, n:byte; nou : lista;
         begin write('n=');readln(n);
        {se creaza primul nod}
        writeln('Informatia primului nod:');
        new(f);readln(f^.info);
        f^.prec:=nil; f^.urm:=nil;
        l:=f;
        writeln('Se citesc informatiile celorlalate noduri:');
```

```
for i:=2 to n do begin
                new(nou);readln(nou^.info);
                nou^.urm:=nil;nou^.prec:=l;
                 l^.urm:=nou;
                 l:=nou;
                 end; end;
```

III.2 Adaugarea

Se poate face la inceput(la stanga primului nod) sau la sfarsit (dupa ultimul nod) daca lista a fost creata spre dreapta; daca lista a fost creata spre stanga atunci adaugarea la inceput va fi efectuata la dreapta primului nod, iar adaugarea la sfarsit se va face dupa ultimul din stanga.
Sa presupunem fara a particulariza ca lista se creaza spre dreapta:

```
procedure adaug_la_inceput(x:integer);
            var nou:lista;
           begin
             new(nou);nou^.info:=x;
           nou^.urm:=f;
            f^.prec:=nil;
           f:=nou;
           end;
```

```
    procedure adaug_la_sfarsit(x:integer);
                var nou:lista;
                 begin
                new(nou);nou^.info:=x;
                nou^.prec:=l;
                nou^.urm:=nil;
                l^.urm:=nou;
                l:=nou;
                end;
```

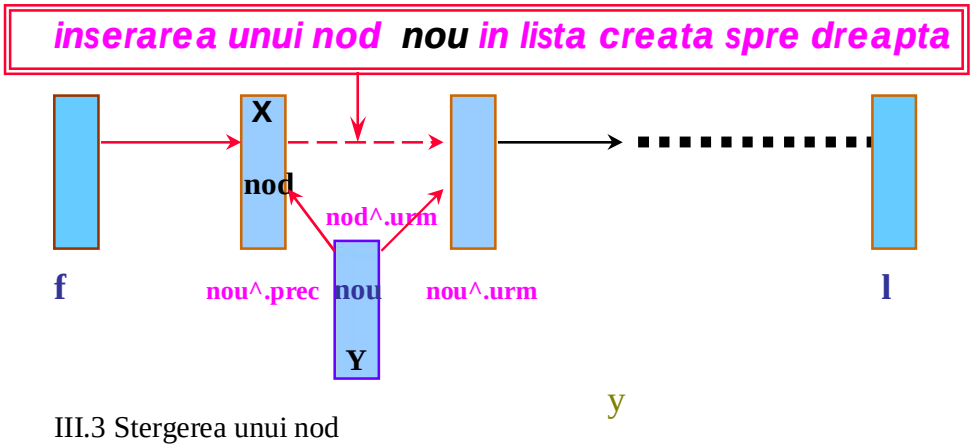
Adaugarea se poate face si dupa un anume nod, sa spunem avand informatia x iar nodul care se va adauga va avea informatia y :

```
    procedure adaug_dupa_un_nod(x,y:integer);
                var nou, nod : lista;
                begin
                nod :=f;
                 while (nod^.info<>x) do nod :=nod^.urm;
                new(nou); nou^.info:=y;
                nou^.prec:=nod; nou^.urm:=nod^.urm;
                nod^.urm:=nou;
                end;
          new(nou);nou^.info:=x;
         nou^.urm:=f;
          f^.prec:=nil; f:=nou; end; dispose(q);  end;
```

```
procedure adaug_la_sfarsit(x:integer);
                    var nou:lista;
          begin
          new(nou);nou^.info:=x;
          nou^.prec:=l;
          nou^.urm:=nil;
          l^.urm:=nou;
          l:=nou;
          end;
```

Adaugarea se poate face si dupa un anume nod, sa spunem avand informatia x iar nodul care se va adauga va avea informatia y :

```
procedure adaug_dupa_un_nod(x,y:integer);
          var nou, nod : lista;
          begin
          nod :=f;
           while (nod^.info<>x) do nod :=nod^.urm;
          new(nou); nou^.info:=y;
          nou^.prec:=nod; nou^.urm:=nod^.urm;
          nod^.urm:=nou; end;
```

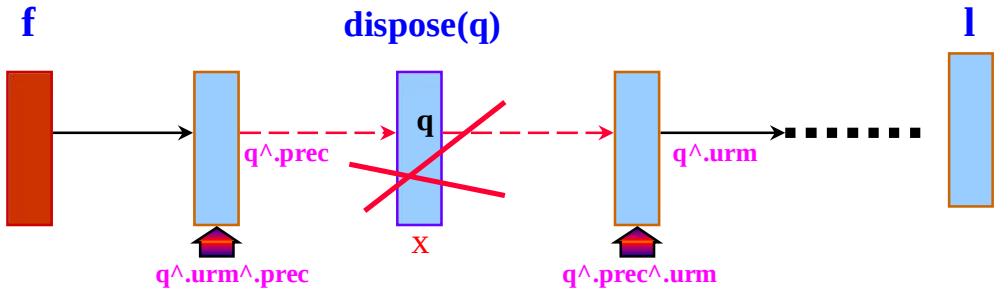


III.3 Stergerea unui nod

Aceasta operatie depinde de modul cum a fost creata lista, in sensul ca parcurgerea pentru cautarea informatiei nodului de sters va urma o anumita directie(dreapta sau stanga) Acelasi lucru intervine si in listare, aceasta putand fi insa efectuata atat intr-o directie cat si in cealalta, deoarece se utilizeaza doi pointeri :

1. f = pointer fix la stanga, l = pointer mobil la dreapta
   (daca lista a fost creata de la stanga la dreapta)
2. f = pointer fix la dreapta, l = pointer mobil la stanga
   (daca lista a fost creata de la dreapta la stanga)

```
procedure stergere_nod_info(x:integer);
                    var q:lista;
                    begin
                   q:=f;
                   while q^.info<>x do q:=q^.urm;
                    q^.urm^.prec:=q^.prec;
                   q^.prec^.urm:=q^.urm;
```



III.4 Parcurgerea

Se va considera ca f si l sunt ca si pana acum variabile globale.De asemenea se presupune ca lista a fost creata spre dreapta.

```
procedure parcurgere_st_dr;
        var x:lista;
        begin
        x:=f;
        while (x<>nil) do
        begin
  write(x^.info,' ');x:=x^.urm; end;
        writeln;end;
```

Analog :

```
procedure parcurgere_dr_st;
    var x:lista;
    begin
    x:=l;
    while (x<>nil) do begin
                write(x^.info,' ');
                x:=x^.prec;
                end;writeln; end.
```

IV Lista circulara

Lista circulara este o l.l.s.i (lista simpla) de tip coada in care ultimul elelment se confunda cu primul. Crearea este identica cu cea a cozii pentru toate elementele intermediare, cu exceptia ultimului, al n-lea, care se leaga de primul, astfel permitand inchiderea listei circulare.(legatura ultimului nod fiind primul)

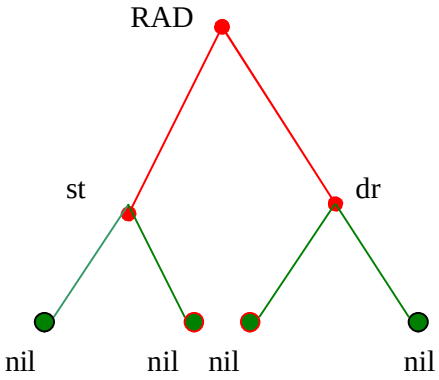
```
var  f,l :lista; n:byte;
 procedure creare;          unde :  lista=^nod;
 var i:byte; nou:lista;                nod=record
 begin                                 info:integer;
 write('n='); readln(n);               leg: lista; end;
 new(f); readln(f^.info);
 f^.leg:=nil;
 l:=f;
for i:=2 to n-1 do begin
 new(nou); readln(nou^.info);
 nou^.leg:=nil;
 l^.leg:=nou;
 l:=nou;
 end;              { adaugarea ultimului si legarea de primul}
 new(nou); readln(nou^.info); nou^.leg:=f; l^.leg:=nou; end;
```

V. Arbori binari

Crearea, in concordanta cu natura structurii de arbore binar va fi de preferinta recursiva, avand o conditie de oprire, de exemplu legata de informatia care se citeste. Tipul unui element este acelasi cu cel prezentat la lista dubla, cu deosebirea ca cele doua campuri adresa se vor numi “descendenti” (stang si drept) sau chiar “fiii” tatalui (nodului curent) Primul tata se mai numeste si radacina (rad).

Deosebirea esentiala este ca, in timp ce lista dubla “crestea” fie intr-o directie fie in alta, acest tip de structura creste concomitent in ambele directii, asezandu-se in jos, pe nivele de descendenta.

```
tipul nodului:
        type lista=^nod
        nod= record
                info:integer;
                st,dr : lista;
                end;
```



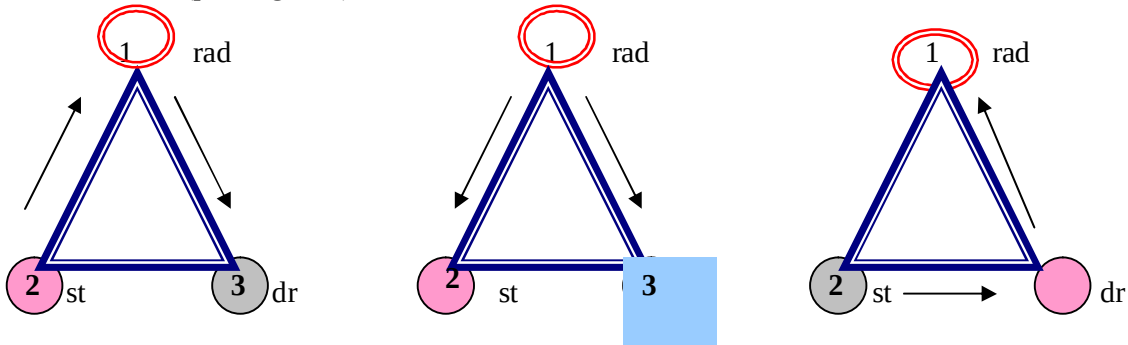
Se observa ca numarul de noduri pline este egal cu numarul de noduri nule(nil). Deci, la citirea datelor va trebui sa introducem informatii semnificative, apoi informatii nesemnificative (nule) in numar egal cu cele precedente, pentru fiii nuli (terminale, sau frunze), plus inca una pentru un prim tata nul.

Deci : informatii utile → nr informatii nule → nr+1 (zerouri)

V.1 Crearea arborelui binar

```
var f:lista;n:integer;
    procedure creare(var f:lista);
       {f= parametru transmis prin adresa}
    begin  {pentru a initializa cu valoarea}
    readln(n);          {pe care o primeste f}
    if n<>0 then begin
                new(f);
                f^.info:=n;
                creare(f^.st);
                creare(f^.dr);
                end
                else    f:=nil;  end;
```

V.2 Traversarea (parcurgerea) arborelui binar



inordine(ST, RAD, DR)	preordine(RAD, ST, DR)	postordine(ST,DR,RAD)

RAD semnifica NODUL RADACINA, sau TATA, iar ST, DR sunt fiii lui, deci subarborii stang si drept
Cele trei proceduri sunt similare, de aceea vom prezenta numai pe prima:

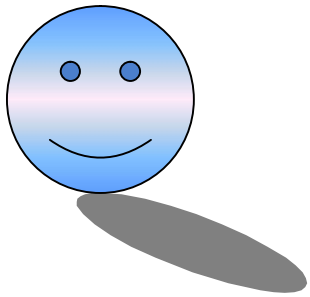
```
PROCEDURE INORDINE(f:lista)
{PARAMETRU TRANSMIS PRIN VALOARE}
    begin
    if f<>nil then begin
            inordine(f^.st);
            write(f^.info:5,' ');
            inordine(f^.dr);
            end; end;
```

Observatie :Este afisata de fiecare data informatia tatalui

V. 3 Operatii cu arborii binari

V.3.1 Cautarea

```
procedure cauta(f:lista; x:integer);
begin
if f<>nil then if f^.info=x then write('gasit)
    else begin
    cauta(f^.st,x);
    cauta(f^.dr,x);    end; end;
```



V.3.2 Stergerea

Nu vom prezenta un procedeu de stergere efectiva ci doar un model de "stergere in afisare" , prin cautare, asadar vor fi afisate elementele partial (mai putin cel pe care dorim sa-l excludem). Acest lucru poate conduce ulterior la reconstituirea listei de elemente, numai pe baza afisarii, deci la crearea unei liste modificate.

```
procedure sterge (f:lista;n:integer);
            begin
            if f<>nil then if f^.info<>n then begin
                                                write(f^.info:5);
                                                sterge(f^.st,n);
                                                sterge(f^.dr,n);
                                                end
                        else begin
                            sterge(f^.st,n);sterge(f^.dr,n);
                            end;  end;
```

Obs: Procedura de adaugare este inclusa in crearea recursiva care realizeaza in afara de radacina, atatea noduri cate se cer. O procedura de adaugare si-ar fi avut sensul numai in cazul cand radacina ar fi fost creata separat, intr-un algoritm de tip iterativ.

VI Arborele binar de cautare

Se numeste astfel deoarece este construit in raport cu o regula de asezare a continuturilor : daca valoarea k, ce urmeaza sa fie informatia noului nod este mai mica decat cea a tatalui existent, nodul va fi un fiu stang, daca este mai mare, nodul va fi un fiu drept. Daca nu exista tata, deci nodul ce urmeaza a fi creat este chiar radacina, atunci descendentii lui sunt doua noduri de tip NIL.

Citirea consecutiva a valorii k conduce la o varianta iterativa, in prima etapa, care este prezentata pentru inceput. Pozitionarea noii valori se realizeaza in functie de informatia tatalui imediat.

In varianta recursiva, transmiterea variabilei f dinamice prin adresa deci pastrarea capului de comparare precum si pozitionarea ulterioara in raport cu informatia acestuia. Programul principal nu se deosebeste esential.

```
var f,p,aux : arbore; j,k,n:integer; (variabile globale)
        procedure creare(k:integer);
                        begin
                        if f=nil then begin
                                        new(f); f^.i:=k; f^.s:=nil;f^.d:=nil;
                                        aux:=f;
                                      end
                                else  begin
                                        new(p); if (aux^.i>k) then begin
                                                        aux^.s:=p;
                                                        p^.i:=k;
                                                        p^.d:=nil;
                                                        p^.s:=nil;
                                                        end;
VARIANTA ITERATIVA                                          else begin
                                                        aux^.d:=p;
                                                        p^.i:=k;
                                                        p^.d:=nil;
                                                        p^.s:=nil;
                                                       end;
begin f:=nil; writeln('Cititi informatiile :'); readln(k);
while (k<>0) do begin creare(k); readln(k); end; readln; end.
```

Procedura de creare a unui arbore binar, in varianta recursiva :

```
procedure creare(var f: arbore; nr:integer);
begin
if f=nil then begin
                new(f); f^.i:=nr; f^.s:=nil; f^.d:=nil;
                end
          else begin
               if (f^.i>nr) then creare(f^.d,nr); end; end;
```

APLICATII LA CAPITOLUL VI

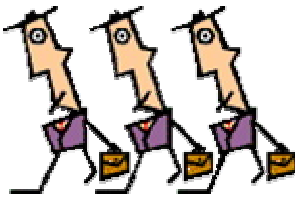
Dintre structurile dinamice de tip L.L.S.I., adica lista simplu inlantuite,cele mai frecvent utilizate sunt stiva si coada, datorita aplicatiilor lor multiple.

1. OPERATII CU LISTA SIMPLA- COADA

```
Program coada;
uses crt;
                type coada=^nod;
                 nod=record
                 info:integer;
                leg:coada;end;
var p,u:coada;
```

In crearea cozii se tine cont de faptul ca ea are doi pointeri- unul de inceput si altul de sfarsit. Adaugarea se va face numai la sfarsit de aceea in loc de l-pointer de loc sau pozitie, sa va folosi u-pointerul de sfarsit.

```
procedure creare;
var u,nou:coada; i,n:integer;
begin
writeln('dati primul numar:');
new(p); readln(p^.info);p^.leg:=nil;
write('Dati numarul total de noduri: ');readln(n);
u:=p;
for i:=2 to n do begin
new(nou);readln(nou^.info);

  nou^.leg:=nil;
  u^.leg:=nou;
  u:=nou;
  end;
  end;

   procedure listare;
        var x:coada;
        begin
        x:=p;
while x<>nil do begin
        write(x^.info,' ');
        x:=x^.leg;
        end;
        end;

      procedure adaugare;
        var x,nou:coada;
        k:integer;
        begin
        writeln('dati  nodul
       de adaugat');
        readln(k);
        x:=p;
        while(x^.leg<>nil)
       do x:=x^.leg;
        new(nou);
 begin
 x:=f;
 while (x<>nil) do x:=x^.urm;
 l:=x;
 while (l<>nil) do begin
 write(x^.info,' ');
 l:=l^.prec;
 end;
 writeln;
 end;

 procedure stergere_nod_info(v:byte);
var x,q:lista;
begin
x:=f;
while x^.info<>v do x:=x^.urm;
x^.urm^.prec:=x^.prec;
x^.prec^.urm:=x^.urm;
dispose(x);
end;
{_____}
```



```
begin
writeln('Creare:');
creare;
parcurgere_st_dr;
writeln('Adaugare la inceput/ se citeste nr. de adaugat:');
readln(v);
adaug_la_inceput(v);
parcurgere_st_dr;
writeln('Adaugare la sfarsit/ se citeste nr. de adaugat:');
readln(v);
adaug_la_sfarsit(v);
parcurgere_st_dr;
writeln('Adaugare dupa un nod/ se citeste informatia nodului');
writeln('dupa care se adauga si numarul de adaugat:');
readln(y,v);
adaug_dupa_un_nod(y,v);
parcurgere_st_dr;
writeln('Stergere / se citeste nr. de sters:');
readln(y);
stergere_nod_info(y);
parcurgere_st_dr;
readln;
end.
```

4.

LISTA CIRCULARA SE DEOSEBESTE DE O LISTA SIMPLA DOAR PENTRU CA ULTIMUL NOD ESTE LEGAT DE PRIMUL;

```
Program L.CIRCULARA;
uses crt;
type lista=^nod;
     nod=record
     info:integer;
     leg:lista;
     end;
var  f:lista;n:byte;
procedure creare;
var i:byte;l,nou:lista;
begin
write('n=');readln(n);
new(f);
readln(f^.info);
f^.leg:=nil;
l:=f;
for i:=2 to n-1 do begin
new(nou);
readln(nou^.info);
nou^.leg:=nil;
l^.leg:=nou;
l:=nou;
end;
```

Aici se face legatura ultimului nod, al n-lea, cu primul, in rest, procedurile sunt asemanatoare cu cele de la lista simpla.

```
new(nou);
  readln(nou^.info);
nou^.leg:=f;
l^.leg:=nou;
end;

procedure afisare;
var x:lista;i:byte;
begin
x:=f; i:=1;
while i<=n do begin
write(x^.info,' ');
x:=x^.leg;inc(i);
end;
write(x^.info,' ');
end;

{_____}
begin
clrscr;
creare;
writeln;
afisare;
readln; end.
```

5.

DIVERSE OPERATII CU LISTELE: ORDONARE,CONCATENARE(LIPIRE), CALCUL FRECVENTA ELEMENTE, ETC.

5.1

ORDONARE LISTA SIMPLA

```
uses crt;
```

```
type lista=^nod;
     nod=record
     info:integer;
     leg:lista;
     end;

 var  f,l,x:lista;
procedure creare;
var n,i:integer;l,nou,x:lista;
begin
new(f);
readln(n);
readln(f^.info);
f^.leg:=nil;
l:=f;
for i:=2 to n do begin
new(nou);
readln(nou^.info);
nou^.leg:=nil;
l^.leg:=nou;
l:=nou;end;
end;
procedure ordonare;
var x:lista;
vb:boolean;aux:integer;
begin
     repeat
     vb:=true;
     x:=f;
     while x<>nil do begin
     if x^.info>x^.leg^.info
     then begin
     aux:=x^.info;
     x^.info:=x^.leg^.info;
     x^.leg^.info:=aux;
     vb:=false;
     end;
     x:=x^.leg;
     end;
     until vb;
     end;

    procedure listare;
    var x:lista;
    begin
    x:=f;
    while x<>nil do begin
    write(x^.info:6,' ');
    x:=x^.leg;end;
    end;

{_____________________________}
    begin
    creare;
    ordonare;
    listare;
    readln;
    end.
```

5.2 CONCATENAREA A DOUA LISTE SIMPLE

```
Program concatenare;
uses crt;
type coada=^nod;
     nod=record
     info:integer;
     leg:coada;end;
var p,u,p1,u1:coada;

procedure creare;
var u,nou:coada; i,n:integer;
begin
writeln('dati primul numar:');
 new(p);
readln(p^.info);p^.leg:=nil;
write('Dati numarul total de
noduri: ');readln(n);
u:=p;
for i:=2 to n do begin
 new(nou);
 readln(nou^.info);
 nou^.leg:=nil;
  u^.leg:=nou;
  u:=nou;
  end;
  end;

 procedure creare1;
 var u1,nou:coada; i,n:integer;
 begin
 writeln('dati primul num†r:');
 new(p1); readln(p1^.info);p1^.leg:=nil;
write('Dati numarul total de noduri:');readln(n);
 u1:=p1;
 for i:=2 to n do begin
 new(nou);readln(nou^.info);
 nou^.leg:=nil;
 u1^.leg:=nou;
  u1:=nou;
  end;
  end;
  procedure adaugare;
  var  y,x,nou:coada;
  k:integer;
  begin
  x:=p;
  while(x^.leg<>nil) do x:=x^.leg;
  y:=p1;
  while y<>nil do begin
  new(nou);
  nou^.info:=y^.info;
  nou^.leg:=nil;
  x^.leg:=nou;
  x:=nou;
  Y:=y^.leg;
  end;
  end;

  procedure listare;
```

```
var x:coada;
begin
x:=p;
while x<>nil do begin
write(x^.info,' ');
x:=x^.leg;
end;
end;
```

```
{_____}
begin
creare;
listare;
creare1;
adaugare;
listare;
readln;end.
```

5.3 FRECVENTA UNUI NUMAR INTR-0 LISTA DE TIP NUMERIC

```
Program frecventa;
var x,f:lista;
m:set of 0..255;
v:array[1..255] of byte;
nr:integer;

procedure creare;
 var n,i:integer;l,nou:lista;
begin
new(f);
readln(n);
readln(f^.info);
f^.leg:=nil;
l:=f;
for i:=2 to n do begin
new(nou);
readln(nou^.info);
nou^.leg:=nil;
l^.leg:=nou;
l:=nou;end;
end;
uses crt;
type lista=^nod;
nod=record
info:byte;
leg:lista;
end;
procedure numar(k:integer);
var x:lista;
begin  x:=f;nr:=0;
while x<>nil do
begin
if x^.info=k then
begin
nr:=nr+1;v[nr]:=k;end;
x:=x^.leg;
end;
end;
{=====================}
```

```
begin
clrscr;
textbackground(1);
textcolor(14);
clrscr;
creare;m:=[];
x:=f;
window(30,10,70,15);
textbackground(15);
textcolor(4);
clrscr;
while x<>nil do   begin
if not(x^.info in m) then
begin
writeln('frecventa lui',x^.info,frecventa(x^.info):6,' ');
m:=m+[x^.info];
readln;
end;
x:=x^.leg;
end;
readln;
end.
```

5. ARBORELE BINAR POATE FI CONSIDERAT UN TIP APARTE DE LISTA , DEOSEBIREA CONSTAND IN ACEEA CA FIECARE NOD ARE DOI DESCENDENTI, UNUL SPRE STANGA SI ALTUL SPRE DREAPTA .

GENERAREA UNUI ARBORE BINAR SI TRAVERSAREA LUI

```
uses crt;
type  lista=^nod;
nod=record
info:integer;
stanga,dreapta:lista;
```

```
    end;
var f:lista;n:integer;
```

CREARE RECURSIVA

```
procedure creare(var f:lista);{parametru transmis prin adresa}
begin           {pentru a initializa cu valoarea}
readln(n);      {pe care o primeste f}
if n<>0 then begin
        new(f);
        f^.info:=n;
        creare(f^.stanga);
        creare(f^.dreapta);
            end
     else
        f:=nil;
        end;
```

CREAREA UNUI ARBORE BINAR. (VARIANTA ITERATIVA)

```
program creare_arbore_binar ;
type ref=^nod;
        nod=record
        info:integer;
        st, dr:ref; end;
var p:ref; x, i, n: integer;
procedure adauga(var p:ref; x:integer);
begin
if p=nil then begin
new(p); p^info:=x; p^.st:=nil; p^.dr:=nil;
end;
else if x<=p^.info then begin listare(p^.st);
write(p^.info, ' ');
listare(p^.dr); end;end;
begin new(p); write('n='); readln(n);
write('dati sirul informatiilor:');
readln(p^.info); p^.st:=nil; p^.dr:=nil;
for i:= 1 to n do begin readln(x); adaug[(p,x); end;
writeln('listare:'); listare(p);
end
```

PARCURGERI DE TOATE TIPURILE: SE VA LISTA IN TREI MODURI

LISTARI RECURSIVE ALE UNUI ARBORE BINAR:

```
procedure inordine(f:lista);
{parametru transmis prin  valoare}
begin
if f<>nil then begin
        inordine(f^.stanga);
        write(f^.info:5,' ');
        inordine(f^.dreapta);
        end;
        end;
procedure preordine(f:lista);
begin
if f<>nil then begin
        write(f^.info:5,' ');
        preordine(f^.stanga);
        inordine(f^.dreapta);
                end;
                end;
procedure postordine(f:lista);
begin
if f<>nil then begin
```

```
postordine(f^.stanga);
postordine(f^.dreapta);
write(f^.info:5,' ');
end;
end;
      begin
      clrscr;
      f:=nil;
      creare(f);
      writeln;
      writeln('listare inordine');inordine(f);
      writeln;
      writeln('listare preordine');preordine(f);
      writeln;
      writeln('listare postordine');postordine(f);
       readln;
       end.
```

7.

INTERCLASAREA A DOUA LISTE NUMERICE

```
uses crt;
type lista=^nod;
    nod=record
    info:integer;
    leg:lista;
    end;

var  l,f,n,f1,f2:lista;
procedure creare1;
var n,i:integer;l,nou:lista;
begin
new(f1);
readln(n);
readln(f1^.info);
f1^.leg:=nil;
l:=f1;
for i:=2 to n do begin
new(nou);
readln(nou^.info);
nou^.leg:=nil;
l^.leg:=nou;
l:=nou;end;
end;
procedure creare2;
var n,i:integer;l,nou:lista;
begin
new(f2);
readln(n);
readln(f2^.info);
f2^.leg:=nil;
l:=f2;
for i:=2 to n do begin
new(nou);
readln(nou^.info);
nou^.leg:=nil;
l^.leg:=nou;
l:=nou;end;
end;
procedure ordonare1;
var x:lista;v:boolean;
aux:integer;
begin
repeat
v:=true;x:=f1;
while x^.leg<>nil do begin
if x^.info>x^.leg^.info then
begin aux:=x^.info;
    x^.info:=x^.leg^.info;
    x^.leg^.info:=aux;
    v:=false;end;
    x:=x^.leg;
    end;

until v;
end;
procedure ordonare2;
var x:lista;v:boolean;
aux:integer;
begin
repeat
v:=true;x:=f2;
while x^.leg<>nil do begin
if x^.info>x^.leg^.info then
begin aux:=x^.info;
    x^.info:=x^.leg^.info;
    x^.leg^.info:=aux;
    v:=false;end;
    x:=x^.leg;
    end;
until v;
end;
procedure interclasare;
var a,b:lista;
begin
a:=f1;b:=f2;
new(f);f^.leg:=nil;
if f1^.info<f2^.info then
        f^.info:=f1^.info
```

```
            else
            f^.info:=f2^.info;
write(f^.info:4,' ');
a:=a^.leg;b:=b^.leg;
l:=f;
while (a<>nil) and (b<>nil) do
begin
new(n);
if a^.info<b^.info then
  begin
  n^.info:=a^.info;
  a:=a^.leg;end
  else
if a^.info>b^.info then
  begin
  n^.info:=b^.info;
  b:=b^.leg;end
 else
  begin
  n^.info:=a^.info;
  a:=a^.leg;b:=b^.leg;
  end;
  write(n^.info:4,' ');
  n^.leg:=nil;
  l^.leg:=n;
  l:=n;
  end;
 if a=nil then while b<>nil do
            begin
            new(n);
            n^.info:=b^.info;
            write(n^.info:4,' ');
            n^.leg:=nil;
            l^.leg:=n;
            l:=n;
            b:=b^.leg;
            end
  else if b=nil then while a<>nil do
            begin
            new(n);
            n^.info:=a^.info;
            write(n^.info:4,' ');
            n^.leg:=nil;
            l^.leg:=n;
            l:=n;
            a:=a^.leg;
                  end;  end;
      procedure afisare;
      var x:lista;
      begin
      x:=f;
      while x<>nil do begin
      write(x^.info);
      x:=x^.leg;end;
      end;
      procedure afisare1;
      var x:lista;
      begin
x:=f1;
while x<>nil do begin
write(x^.info);
x:=x^.leg;end;
end;
procedure afisare2;
var x:lista;
begin
x:=f2;
while x<>nil do begin
write(x^.info);
x:=x^.leg;end;
end;

begin
clrscr;
creare1;creare2;
writeln('Liste ordonate');
ordonare1;ordonare2;
afisare1;
writeln;
afisare2;
writeln;
writeln('Lista interclasata');
interclasare;
readln;
end.
```

8.

ALTERNATIVA DINAMICA A SCRIERII SUB FORMA POLONEZA A UNEI EXPRESII ARITMETICE, INTRODUSE DE UN SIR DE CARACTERE

```
Program forma_poloneza;
 { METODA ESTE DIVIDE ET IMPERA-IN
FUNCTIA F_POLONEZA}
uses crt;
type lista=^nod;
    nod=record
       op:char;
       stanga,dreapta:lista;
       end;
    expresie=array[1..30] of char;
    vector=array[1..30] of integer;
 var e,e_f_p:expresie;p,p_f_p:vector;
 f:lista;a:char;k,i,j,n:byte;
 function f_poloneza(ls,ld:byte;var e_f_p:expresie;var
p_f_p:vector):lista;
 var x:lista;i,j,min:integer;
 begin
 min:=p_f_p[ld];i:=ld;
 for j:=ld downto ls do if p_f_p[j]<min then begin
                                 min:=p_f_p[j];
                                 i:=j;
                                 end;
 new(x);f_poloneza:=x;
 f_poloneza^.op:=E_f_p[i];
 if ls=ld then
   begin
```

```
  f_poloneza^.stanga:=nil;f_poloneza^.dreapta:=nil;

                    end
                    else
                  begin
                  poloneza^.stanga:=f_poloneza(ls,i,
                  ,e_f_p,p_f_p);
                  f_poloneza^.dreapta:=f_poloneza(i+1,ld,
                  e_f_p,p_f_p);
                   end;
                   end;
                   procedure postordine(l:lista;var k:byte);
                   {forma postfixata sau poloneza se
                  obtine prin parcurgere in postordine}
                   begin
                   if L<>nil then begin
                   postordine(l^.stanga,k);
                   postordine(l^.dreapta,k);
                   if (ord(upcase(l^.op))>=65) and
                   (ord(upcase(l^.op))<=90) then
                  write('                ',l^.op:4)
                       else
                  begin   write('nivel',k:2,':',l^.op:2,---->');
                             k:=k+1;
                              end;
                   writeln;readln;
                  end;
                  end;
                  begin
  clrscr;
  j:=0;
  n:=1;
  {citirea expresiei}
  read(a);while a<>'.' do begin
e[n]:=a;n:=n+1;read(a);end;
  n:=n-1;
  {formarea vectorului prioritatilor}
  for i:=1 to n do
              case e[i] of
              ')':j:=j-10;
              '(':j:=j+10;
              '+','-':p[i]:=j+1;
              '*','/':p[i]:=j+10
              else p[i]:=1000;
              end;
{formarea vectorului prioritatilor fara paranteze si a
expresiei fara paranteze}
  j:=1;
for i:=1 to n do if (e[i]<>'(') and (e[i]<>')') then begin
               e_f_p[j]:=e[i];
               p_f_p[j]:=p[i];
               j:=j+1;
               end;
  f:=f_poloneza(1,j-1,e_f_p,p_f_p);
  writeln;textcolor(yellow);
  k:=1;
  postordine(f,k);
  writeln('end!');
  readln;
  end.
```

9.

INVERSAREA LEGATURILOR INTR-O LISTA SIMPLA SI AFISAREA EI

```
Program inversare;
                  uses crt;
                type lista=^nod;
                    nod=record
                    info:integer;
                    leg:lista;
                    end;
                var  f,l,x:lista;
                procedure creare;
                var n,i:integer;l,nou,x:lista;
                begin
                new(f);
                readln(n);
                readln(f^.info);
                f^.leg:=nil;
                l:=f;
                for i:=2 to n do begin
                new(nou);
                readln(nou^.info);
                nou^.leg:=nil;
                l^.leg:=nou;
                l:=nou;end;
                end;
                procedure inversare;
                var aux1,aux2,x:lista;
                begin
                    x:=f;aux2:=nil;
                    while (x<>nil) do

                  begin
                  aux1:=aux2;aux2:=x;
                  x:=x^.leg;
                  aux2^.leg:=aux1;
                  f:=aux2;
                  end;

procedure afisare;
   var x:lista;
   begin
   x:=f;
   while x<>nil do
   begin
   write(x^.info,' ');
   x:=x^.leg;
   end;
   end;
  {_____}
      begin
      clrscr;
      creare;
      writeln;
```

```
      afisare;
      inversare;
      writeln;
```

10.

PARCURGEREA RECURSIVA A UNEI LISTE SIMPLE

```
Program rec;
uses crt;
type lista=^nod;
    nod=record
    info:integer;
    leg:lista;end;
var f,l:lista;
    procedure creare;
    var l,nou:lista; i,n:integer;
    begin
    new(f); readln(f^.info);f^.leg:=nil;


 write('dati numarul total de noduri: ');readln(n);
    l:=f;
    for i:=2 to n do begin
  new(nou);readln(nou^.info);
  nou^.leg:=nil;
  l^.leg:=nou;
  l:=nou;
  end; end;                                    {\\\
procedure parcurgere_rec(x:lista);
begin
write(x^.info,' ');  {afisare informatie utila curenta}
```

11.

PROBLEMA STERGERII NUMERELOR IMPARE DINTR-O LISTA NUMERICA

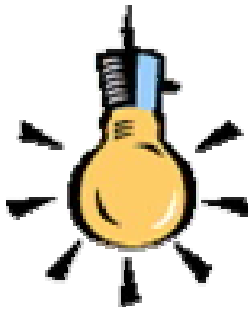
```
uses crt;
type lista=^nod;
    nod=record
    info:integer;
    leg:lista;
    end;
var  p,f,l,x:lista;ch:char;
procedure creare;
var l,nou,x:lista;i,n:integer;
begin
new(f);
readln(f^.info);
f^.leg:=nil;l:=f;
readln(n);
for i:=2 to n do begin
      new(nou);
      readln(nou^.info);
      nou^.leg:=nil;
      l^.leg:=nou;
 l:=nou;
end;
end;
procedure ordonare1;
var x:lista;v:boolean;
aux:integer;
begin
repeat
afisare;
end.

x:=x^.leg ;        {parcurgere: salt la nodul urmator}
if x<>nil then      { atat timp cat lista nu e vida}
parcurgere_rec(x);  {chemare recursiva }      {  de
argument nodul curent}
  end;

{_____________________________________}
                    begin
                    clrscr;
                    creare;
                    parcurgere_rec(f);
                    readln;
                    end.
```

```
v:=true;x:=f;
while x^.leg<>nil do begin
if x^.info>x^.leg^.info then
begin aux:=x^.info;
    x^.info:=x^.leg^.info;


x^.leg^.info:=aux;
    v:=false;end;
    x:=x^.leg;
    end;
until v;
end;
procedure sterg;
var x,l,nou:lista;
 begin
 x:=f;new(p);

if ((f^.info) mod 2=0) then begin
p^.info:=x^.info;p^.leg:=nil;x:=x^.leg;l:=p;
end
else begin p^.info:=0;p^.leg:=nil;x:=x^.leg;l:=p;end;
while (x<>nil) do begin
  if ((x^.info) mod 2=0) then
        begin
        new(nou);
        nou^.info:=x^.info;
        nou^.leg:=nil;
        l^.leg:=nou;
        l:=nou;
```

```
                end;
                x:=x^.leg;
                end;
               end;
procedure afisare;
var l:lista;
begin
l:=p;
while l<>nil do begin
if l^.info<>0 then
writeln('       ',l^.info)
else writeln(l^.info);
l:=l^.leg;end;
end;
procedure afisare1;
var x:lista;
begin
x:=f;
while x<>nil do begin
write(x^.info,' ');
x:=x^.leg;end;
end;
begin
clrscr;
textbackground(1);
textcolor(14);
CLRSCR;
creare;
ordonare1;
WINDOW(50,8,70,15);
 textbackground(15);
 textcolor(4);
 CLRSCR;
writeln('LISTA NUMERELOR DATE:');
                afisare1;
                writeln;
                writeln('Apasati tasta ENTER: ');
                sterg;
                readln(ch);
                if ch=#13 then begin
                 clrscr;
                 textbackground(1);
                 textcolor(14);
                 CLRSCR;
 WINDOW(50,8,70,15);
 textbackground(15);
 textcolor(4);
 CLRSCR;
writeln('LISTA NUMERELOR PARE:');
afisare;
end;
readln;
end.
```

12 : SA SE INSEREZE INTR-O LISTA DE NUMERE, LA LOCUL DE ORDINE CORESPUNZATOR, NUMERELE PRIME MAI MARI DECAT ELEMENTELE DATE. SA SE AFISEZE.

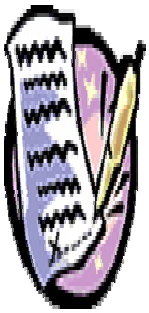
```
uses crt;
type lista=^nod;
    nod=record
    a:word;
    leg:lista;
    end;
var  f:lista;max:word;
procedure creare;
var n,i:integer;nou,x:lista;
begin     { lista trebuie ordonata in prealabil}
new(f);
readln(n);
readln(f^.a);
f^.leg:=nil;
x:=f;
for i:=2 to n do begin
new(nou);
readln(nou^.a);
nou^.leg:=nil;
x^.leg:=nou;
x:=nou;
end;
end;
          procedure ordonare1;
          var x:lista;
             vb:boolean;aux:word;
             begin
           repeat
           vb:=true;
          x:=f;
```



```
        while x^.leg<>nil do begin
           if x^.a>x^.leg^.a then begin
        aux:=x^.a;
        x^.a:=x^.leg^.a;
        x^.leg^.a:=aux;
        vb:=false;
        end;  x:=x^.leg;end;
      until vb;
      end;

function prim(m:word):boolean;
  var i:integer;
  begin
  prim:=true;
  for i:=2 to m-1 do if (m mod i=0) then  prim:=false ;
  end;

procedure prime(m:word);
var i:integer;
begin
     max:=0;
     for i:=1 to m-1 do if prim(i) and (i>max) then max:=i;
end;


  procedure inserare;        {procedura a reusit sa insereze  }
  var x,p:lista;aux:word;     {numarul prim mai mare decat x^.a}
  begin                    {numai dupa acesta.....}
  x:=f;
  while x<>nil do
  begin
  if not( prim(x^.a)) then begin
                        prime(x^.a);
                        new(p);
                        p^.a:=max;
                        p^.leg:=x^.leg;
                        x^.leg:=p;

  end;
  x:=x^.leg;
  end;
  end;
procedure listare;
var x:lista;
begin
x:=f;
while x<>nil do begin
write(x^.a:5,' ');
x:=x^.leg;end;
end;
{>>>>>>>>>>>>>>>>>>>>}
begin
 clrscr;
 creare;
 ordonare1;
 listare;
 writeln;
 inserare;
 listare;
 end.
```

13.

ADUNAREA A DOUA POLINOAME, UTILIZAND LISTELE SIMPLE :C=COEFICIENT, GX, GY= GRADELE VARIABILELELOR X, RESPECTIV Y; POLINOAMELE SE CONSIDERA DE DOUA VARIABILE –X, Y



```
type lista=^nod;
    nod=record
      c,gx,gy:integer;
      nou:lista;
    end;
var f:lista;a,b:integer;

procedure creare;
var l,u:lista;n,i:integer;
begin
readln(n);
new(f);
readln(f^.c,f^.gx,f^.gy);
f^.nou:=nil;
u:=f;
for i:=2 to n do begin
new(l);
read(l^.c,l^.gx,l^.gy);
l^.nou:=nil;
u^.nou:=l;
u:=l;
end;
end;

procedure afisare;
var x:lista;
begin
x:=f;write(x^.c,'x^',x^.gx,'y^',x^.gy);
x:=x^.nou;
while x<>nil do begin
if x^.c<0 then
write(x^.c,'x^',x^.gx,'y^',x^.gy)
      else
write('+',x^.c,'x^',x^.gx,'y^',x^.gy);
x:=x^.nou;end;
end;

begin
creare;
afisare;
end.
```

14.PROBLEME SPECIFICE ARBORILOR

14.1

FIIND DAT UN ARBORE OARECARE, SA SE DETERMINE ARBORELE BINAR ASOCIAT LUI

```
type ref=^nod ;
    vect=array[1..20] of ref;
    nod=record
          info:integer;
          nr_fii:integer;
          fiu:vect;
          end;
    refb=^nodb;
    nodb=record
        info:integer;
        fiu_st, fiu_dr:refb;
```

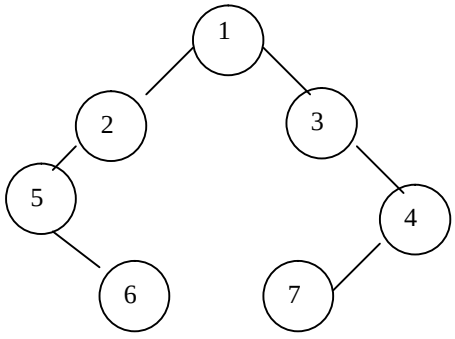
```
                end;
var p: ref; p l :refb;
procedure creare (var p:ref);
var n, i: integer;
begin
  readln(n);
  new(p); p^.info:= n;
  write('nr. fii ai nodului',n); readln(p^. nr_fii);
  for i:=1 to nr_fii do creare(p^.fiu[i]);
  end;
end;
procedure transform(p:ref, varp:info);
var i:integer; a:refb;
begin
 new(p);
 pl^.info:=p^.info;
 pl^.fiu_st:=nil;pl^.fiu_dr:=nil;
 for i:= 1 to nr_fii do
    if pl^.fiu_st=nil then transform(p^.fiu [i], pl^.fiu_st)
                else begin
                     a:=pl^.fiu_st;
                     while a^.fiu_dr<> nil do a := a^.fiu_dr;
                     transform(p^.fiu[i], a^.fiu_dr);
                      end;
end;
procedure listare_arb_bin (pl:refb);
  begin
    if pl <> nil then begin
                write(pl^.info,'');
                listare_arb_bin(pl^.fiu_st);
                listare_arb_bin(pl^.fiu_dr);
                        end;
end;
begin
  creare(p); transform(p, pl);
   writeln('arborele binar liatat in preornine:');
   listare_arb_bin(pl);
end.
```

14.2. <u>REPREZENTAREA PARANTEZATA A UNUI ARBORE</u>

<u>SA SE AFISEZE INFORMATIILE ATASATE NODURILOR UNUI ARBORE BINAR, UTILIZAND PARANTEZELE ROTUNDE PENTRU A MARCA NODURILE SITUATE PE ACELASI NIVEL IN ARBORE.</u>

```
type ref=^nod ;
 nod=record
   info:integer;
   st,dr:ref;
   end;
var p:ref;
procedure creare (var p:ref);
var x:integer;
begin
  readln(x); if x <> 0 then begin
            new(p); p^.info:=x;
            creare(p^.st); creare(p^.dr);
            end
                        else p:=nil;
end;
procedure listare(p:ref);
begin
```

```
begin{program principal}
        creare(p);
        listare(p);
        end.
```



1(2(5(. ,6), .), 3(. , 4(7, .)))

```
        if p=nil then write (‘ ‘);
                                else begin write (p^.info,’ ‘);
                        if (p^.st<>nil) or (p^.dr<>nil)then begin


                    write(‘ ‘); listare(p^.st);
                    write(‘ ‘); listare(p^.dr); write(‘ ‘); end;
                                end;end.
```

14. 3

STERGEREA UNUI NOD INTR-UN ARBORE DE CAUTARE, ASTFEL INCAT SA SE OBTINA TOT UN ARBORE DE CAUTARE.

```
                                type ref=^nod;
                                nod=record
                                info:integer;
                                st, dr:ref; end;
                                var p,q:ref; x, i, n: integer;
  procedure cauta_sterge(varp :ref) ;
  begin if p^.dr<> nil then cauta_sterge(p^.dr)
  else begin q^.info:=p^.info; q:=p; p:=p^.st; dispose(q); end;
  end;
  procedure sterge(var r:ref, x:integer);
  begib if r=nil then writeln(‘numarul nu este in arbore’)
  else if x<r^.info then sterge(r^.st, x)
  else if x>r^.info then sterge(r^.dr, x)
  else begin q:=r;
  if r^.st=nil then begin r:=r^.dr; dispose(q);
  end
  else if r^.dr=nil then begin r:=r^.st;dispose(q);
  end
  else cauta_sterge(r^.st);
  end;end;
  end.
```

14.4.ARBORE ECHILIBRAT (AVL)

Definitia arborelui echilibrat: *Oricare ar fi un arbore ABC (arbore binar de cautare) atunci el se numeste AVL daca si numai daca: i_s-i_d apartine {0,-1,1}; i_s=inaltime subarbore stang, i_d= inaltime subarbore drept, Informatiile Nodurilor=inaltime*

SA SE CONSTRUIASCA UN ARBORE ECHILIBRAT AVAND INFORMATIILE 1,2,3,…,N.

```
                                type ref=^nod;
                                nod=record
                                info:integer;
                                st, dr:ref; end;
                              var p :ref ;n : integer ;
  procedure creare(var x :ref ; s,d :integer) ;
  var m:integer;
  begin new(x);
  m:=(s+d+1)div 2;
  x^.info:=m;
  sif s<=m-1 then creare(x^.st, s, m-1)
  else x^.dr:=nil;
  end;
  procedure listare(x:ref);
  begin if x<> nil then begin listare(x^.st); write(x^.info, ‘’); listare(x^.dr);end;
  end;
  begin
  write (‘n=’); readln(n);
  creare(p,l,n);
  writeln(‘listare:’);
  listare(p); end.
```

Anexa_1

CODUL ASCII

31				41	)	51	3	61	=	71	G	81	Q	91	[						
32	(spatiu)			42	*	52	4	62	>	72	H	82	R	92	\						
33	!			43	+	53	5	63	?	73	I	83	S	93	]						
34	"			44	,	54	6	64	@	74	J	84	T	94	^						
35	#			45	-	55	7	65	A	75	K	85	U	95	_						
36	$			46	.	56	8	66	B	76	L	86	V	96	`						
37	%			47	/	57	9	67	C	77	M	87	W	97	a						
38	&			48	0	58	:	68	D	78	N	88	X	98	b						
39	'			49	1	59	;	69	E	79	O	89	Y	99	c						
40	(			50	2	60	<	70	F	80	P	90	Z	100	d						
101	e	111	o	121	y	131	ƒ	141	□	151	—	161	¡	171	«	181	µ				
		191	¿																		
102	f	112	p	122	z	132	„	142	□	152	˜	162	¢	172	¬	182	¶				
		192	À																		
103	g	113	q	123	{	133	…	143	□	153	™	163	£	173	-	183	·				
		193	Á																		
104	h	114	r	124	|	134	†	144	□	154	š	164	¤	174	®	184	¸				
		194	Â																		
105	i	115	s	125	}	135	‡	145	‘	155	›	165	¥	175	¯	185	¹				
		195	Ã																		
106	j	116	t	126	~	136	ˆ	146	’	156	œ	166	¦	176	°	186	º				
		196	Ä																		
107	k	117	u	127		137	‰	147	“	157	□	167	§	177	±	187	»				
		197	Å																		
108	l	118	v	128	€	138	Š	148	”	158	□	168	¨	178	²	188	¼				
		198	Æ																		
109	m	119	w	129	□	139	‹	149	•	159	Ÿ	169	©	179	³	189	½				
		199	Ç																		
110	n	120	x	130	,	140	Œ	150	–	160		170	ª	180	´	190	¾				
		200	È																		
201	É	211	Ó	221	1	231	ç	241	ñ	251	û										
202	2	212	Ô	222	Þ	232	è	242	ò	252	ü										
203	Ë	213	Õ	223	3	233	é	243	ó	253	ý										
204	Ì	214	Ö	224	à	234	ê	244	ô	254	þ										
205	Í	215	×	225	á	235	ë	245	õ	255	(blanc)ÿ										
206	Î	216	Ø	226	â	236	ì	246	ö												
207	Ï	217	Ù	227	ã	237	í	247	÷												
208	Ð	218	Ú	228	ä	238	î	248	ø												
209	Ñ	219	Û	229	å	239	ï	249	ù												
210	Ò	220	Ü	230	æ	240	ð	250	ú												

Combinatii de taste (cele mai des utilizate)

Alt+18: sare la inceputul paginii
Alt+24: caracterul de dedesubt
Alt+25: sterge tot randul (similar Ctrl+Y)
Alt+27: Save
Alt+127: sterge tot cuvantul din fata (la fel Alt+23)
Alt+257: sare la inceputul cuvantului
Alt+259: sare la sfarsitul paginii
Alt+260: sare la caracterul urmator
Alt+261: un rand mai sus
Alt+262: sare la cuvantul urmator
Alt+263: Delete
Alt+264: Backspace
Alt+265: Tab
Alt+267: sterge tot ce este pe rand, dupa prompter (la fel: ALT+11)
Alt+269: Enter (la fel ALT+13)

CTRL+ALT+DEL : oprire sistem
CTRL+BREAK : intrerupere executie

Bibliografie

1. Cornelia Ivasc, „Bazele informaticii”, ed. Libris, 1994
2. Doina Rancea, „Limbajul Turbo Pascal”, ed. Petrion, 1995
3. Manualele de informatica, aprobate de catre MEC, 2000-2009