Ministerul Educației al Republicii Moldova

Anatol Gremalschi
Iurie Mocanu
Ion Spinei

INFORMATICĂ

Manual pentru clasa a 9-a

Știința

Ministerul Educației al Republicii Moldova

Anatol Gremalschi
Iurie Mocanu
Ion Spinei

INFORMATICĂ

Manual pentru clasa a 9-a

Știința, 2016

CZU 004(075.3)
G 80

Elaborat conform Curriculumului disciplinar în vigoare și aprobat prin Ordinul ministrului educației (nr. 321 din 28 aprilie 2016). Editat din sursele financiare ale *Fondului special pentru manuale*.

Comisia de evaluare: *Ecaterina Adam*, prof. școlar (gr. did. I), Liceul de Creativitate și Inventică „Prometeu", Chișinău; *Mariana Ciobanu*, prof. școlar (gr. did. superior), Liceul Teoretic Român-Francez „Gheorghe Asachi", Chișinău; *Gheorghe Chistruga*, prof. școlar (gr. did. superior), Liceul Teoretic „Mihai Eminescu", Drochia; *Svetlana Golubev-Brînză*, prof. școlar (gr. did. superior), Liceul Teoretic „Nicolae Milescu-Spătaru", Chișinău; *Andrei Sacara*, prof. școlar (gr. did. superior), Liceul Teoretic „Miguel de Cervantes Saavedra", Chișinău

Recenzenți: *Tatiana Cartaleanu*, doctor în filologie, conferențiar universitar, Universitatea Pedagogică de Stat „Ion Creangă"; *Tatiana Baciu*, doctor în psihologie, conferențiar universitar, Universitatea Pedagogică de Stat „Ion Creangă"; *Alexei Colîbneac*, profesor universitar, Academia de Muzică, Teatru și Arte Plastice, maestru în arte

Redactor: *Mariana Belenciuc*
Corectori: *Maria Cornesco, Tatiana Darii*
Redactor tehnic: *Nina Duduciuc*
Machetare computerizată: *Anatol Andrițchi*
Copertă: *Vitalie Ichim*

Întreprinderea Editorial-Poligrafică Știința,
str. Academiei, nr. 3; MD-2028, Chișinău, Republica Moldova;
tel.: (+373 22) 73-96-16; fax: (+373 22) 73-96-27;
e-mail: prini@stiinta.asm.md; prini_stiinta@yahoo.com;
www.editurastiinta.md

DIFUZARE
ÎM Societatea de Distribuție a Cărții *PRO-NOI*,
str. Alba-Iulia, nr. 75; MD-2051, Chișinău;
tel.: (+373 22) 51-68-17, 71-96-74; fax: (+373 22) 58-02-68;
e-mail: info@pronoi.md; www.pronoi.md

Descrierea CIP a Camerei Naționale a Cărții

Gremalschi, Anatol

Informatică: Man. pentru clasa a 9-a/Anatol Gremalschi, Iurie Mocanu, Ion Spinei; comisia de evaluare: Ecaterina Adam [et al.]; Min. Educației al Rep. Moldova. – Ch.: Î.E.P. *Știința*, 2016 (Tipogr. „BA-LACRON" SRL). – 144 p.

ISBN 978-9975-85-013-1

004(075.3)

ISBN 978-9975-85-013-1

CUPRINS

Dragi prieteni,

Cunoaștem deja că informatica este un domeniu al științei care studiază metodele de păstrare, transmitere și prelucrare a informației cu ajutorul calculatoarelor.

În clasele precedente de gimnaziu am examinat noțiunile de bază ale informaticii – date, informație, calculator, executant, algoritm – și ne-am format deprinderile practice de lucru la calculator. Datorită editoarelor de texte putem crea și prelucra cele mai diverse documente, iar cu ajutorul procesoarelor de calcul tabelar putem sistematiza și prelucra datele numerice și cele textuale. Am creat mai multe programe pentru comanda executanților și ne-am convins că funcționarea calculatoarelor moderne este guvernată de algoritmi.

Manualul de față are drept scop însușirea de către elevi a cunoștințelor necesare pentru dezvoltarea gîndirii algoritmice și formarea culturii informaționale. Realizarea acestui obiectiv presupune extinderea capacităților fiecărei persoane de a elabora algoritmi pentru rezolvarea problemelor pe care le întîmpină în viața cotidiană.

Este cunoscut faptul că algoritmii descriu foarte exact ordinea și componența operațiilor necesare pentru prelucrarea informației. De obicei, în procesul elaborării utilizatorii schițează algoritmii într-un limbaj de comunicare între oameni, de exemplu, în limba română, rusă sau engleză. Pe parcurs, pentru o descriere mai sugestivă a prelucrărilor preconizate, pot fi utilizate schemele logice. Însă pentru a fi înțeles de calculator, orice algoritm, în versiunea finală, trebuie scris într-un limbaj de programare.

În manual, în scopuri didactice, este utilizat limbajul de programare PASCAL. Numit astfel în cinstea matematicianului și filosofului francez Blaise Pascal, acest limbaj a cunoscut o răspîndire mondială, fiind cel mai potrivit pentru predarea și învățarea informaticii. Reprezentînd un instrument destinat formării și dezvoltării gîndirii algoritmice, limbajul PASCAL include toate conceptele de bază utilizate în sistemele informatice moderne: variabile, constante, tipuri de date, instrucțiuni simple și instrucțiuni compuse.

Fiecare porțiune de materie teoretică din manual este urmată de un șir de exemple, exerciții și întrebări de control. Se consideră că elevul va introduce și va lansa în execuție programele incluse în manual, va răspunde la întrebări și, cînd e cazul, el însuși va scrie și va depana programele PASCAL, necesare pentru rezolvarea problemelor propuse.

Toate programele incluse în manual au fost testate și depanate în mediul de programare Turbo PASCAL 7.0. Evident, elevii și profesorii pot utiliza și alte produse program, destinate instruirii asistate de calculator.

Fiind o parte indispensabilă a culturii moderne, informatica are un rol decisiv în dezvoltarea umană și deschide perspective promițătoare pentru fiecare din noi, cei care trăim și activăm într-o societate, denumită foarte sugestiv – societate informațională. Valorificarea acestor perspective depinde, în mare măsură, de nivelul de cunoaștere a principiilor fundamentale ale limbajelor de programare și de capacitatea de a le aplica în practică.

Vă dorim succese,

Autorii

ALGORITMI, PROGRAME ȘI EXECUTANȚI

Este cunoscut faptul că *algoritmul* reprezintă o mulțime finită de instrucțiuni care, fiind executate într-o ordine bine stabilită, produc în timp finit un rezultat. Procesul de elaborare a algoritmilor se numește *algoritmizare*.

În informatică noțiunea de algoritm se asociază în mod obligatoriu cu noțiunea de executant. *Executantul* reprezintă un obiect care poate îndeplini anumite comenzi. Mulțimea acestor comenzi formează repertoriul executantului.

În clasa a 8-a am studiat executanții **Cangurul** și **Furnica**, elaborați în scopuri didactice pentru școlile din țara noastră. Amintim că executantul **Cangurul** poate îndeplini comenzile `PAS`, `ROTIRE`, `SALT`, iar executantul **Furnica** – comenzile `SUS`, `JOS`, `DREAPTA`, `STÎNGA`.

Deosebim două moduri de comandă a executanților: comandă manuală și comandă prin program. Modul de *comandă manuală* presupune introducerea separată a fiecărei comenzi și îndeplinirea ei imediată de către executant. Modul de *comandă prin program* presupune scrierea în prealabil a unei secvențe de comenzi, introducerea acestei secvențe în memoria centrului de comandă a executantului și îndeplinirea comenzilor respective în regim automat, fără intervenția utilizatorului.

Secvențele de comenzi destinate îndeplinirii automate de către executanți se numesc *programe*. Evident, fiecare program reprezintă un algoritm scris în limbajul executantului respectiv. Procesul de elaborare a programelor se numește *programare*.

În clasa a 8-a am creat mai multe programe pentru comanda executanților **Cangurul** și **Furnica**: desenarea pătratelor, ornamentelor și a spiralelor, aranjarea caracterelor imprimabile într-o anumită ordine etc. Programele respective au fost scrise într-un limbaj de programare ce conține instrucțiunile simple `PAS`, `SALT`, `ROTIRE`, `SUS`, `JOS`, `DREAPTA`, `STÎNGA`, apel de procedură și instrucțiunile compuse `REPETĂ`, `CÎT`, `DACĂ`.

Fiind elaborați în scopuri pur didactice, executanții **Cangurul** și **Furnica** nu permit efectuarea unor prelucrări complexe ale informației. În cazul problemelor de o reală importanță practică, prelucrarea informației poate fi realizată cu ajutorul unor executanți mult mai puternici, și anume: cu ajutorul calculatoarelor moderne.

În general, orice calculator modern reprezintă un executant care îndeplinește în mod automat programele încărcate în memoria lui internă. Să reținem că orice program din memoria internă a calculatorului reprezintă, de fapt, o succesiune de cuvinte binare care indică ordinea (secvența) calculelor.

Pentru exemplificare, prezentăm în continuare un fragment de program scris în limbajul calculatorului:

```
10010101 10000011 00110100 01000100
01010010 01011101 00010010 10010101
11010010 01001100 00101001 01110100
00010101 01010100 11111010 10100011
```

Întrucît alcătuirea programelor reprezentate în formă de cuvinte binare este un lucru extenuant și ineficient, s-a convenit ca algoritmii destinați soluționării problemelor cu ajutorul calculatorului să fie scriși în limbaje speciale, denumite limbaje de programare de nivel înalt. Cele mai răspîndite limbaje de programare de nivel înalt sînt ALGOL, FORTRAN, BASIC, PASCAL, C, JAVA etc. Toate aceste limbaje conțin atît mijloace pentru descrierea și apelarea subalgoritmilor, cît și instrucțiuni pentru programarea algoritmilor liniari, algoritmilor repetitivi și a celor cu ramificări. În școlile din țara noastră, în scopuri didactice, este utilizat limbajul PASCAL, care este cel mai potrivit pentru predarea și învățarea informaticii.

Pentru exemplificare, prezentăm în continuare un program PASCAL care calculează rădăcinile ecuațiilor de gradul I:

```
Program Exemplu;
var a, b, x : real;
begin
  readln(a, b);
  if a<>0 then
    begin
       x:=-b/a;
       writeln('Ecuatia are o singura radacina');
       writeln(x);
    end;
  if (a=0) and (b=0) then
      writeln('Ecuatia are o multime infinita de radacini');
  if (a=0) and (b<>0) then
     writeln('Ecuatia nu are sens');
end.
```

Comparînd fragmentul de program scris în limbajul calculatorului cu programul `Exemplu`, ne convingem că aplicarea limbajului PASCAL simplifică foarte mult procesele de elaborare a programelor.

În general, în cazul utilizării unui limbaj de programare de nivel înalt, procesul de rezolvare a unei probleme prin intermediul calculatorului include următoarele etape:

1) schițarea algoritmului într-un limbaj de comunicare între oameni, de exemplu în limba română, rusă sau engleză;

2) descrierea mai sugestivă, dacă este cazul, a prelucrărilor preconizate recurgînd la schemele logice;

3) scrierea algoritmului într-un limbaj de programare de nivel înalt, de exemplu, în PASCAL;

4) traducerea programului din limbajul de nivel înalt în limbajul calculatorului (în secvențe de cuvinte binare);

5) derularea programului pe calculator, depistarea și corectarea eventualelor erori.

Traducerea programelor din limbajul de nivel înalt în limbajul calculatorului se numește *compilare* și se efectuează în mod automat cu ajutorul unor programe speciale, denumite *compilatoare*.

Întrebări și exerciții

❶ Amintiți-vă cel puțin trei algoritmi pe care i-ați studiat la lecțiile de matematică și informatică.
❷ Ce fel de informații trebuie să conțină o descriere completă a unui executant?
❸ Care sînt mijloacele principale de reprezentare a algoritmilor?
❹ Prin ce se deosebește modul de comandă prin program de modul de comandă manuală?
❺ Care este diferența între algoritmi și programe? Argumentați răspunsul dvs.
❻ Numiți etapele principale de rezolvare a unei probleme cu ajutorul calculatorului.
❼ Care este diferența între un limbaj de programare de nivel înalt și limbajul calculatorului?

Capitolul 1

VOCABULARUL ȘI SINTAXA LIMBAJULUI PASCAL

1.1. Inițiere în limbajul PASCAL

Vom examina următorul program:

```
1  Program P1;
2   { Suma numerelor intregi x, y, z }
3  var x, y, z, s : integer;
4  begin
5    writeln('Introduceti numerele intregi x, y, z:');
6    readln(x, y, z);
7    s:=x+y+z;
8    writeln('Suma numerelor introduse:');
9    writeln(s);
10 end.
```

Numerele *1, 2, 3, ..., 10* din partea stîngă a paginii nu fac parte din programul PASCAL. Ele servesc doar pentru referirea liniilor în explicațiile ce urmează.

Linia 1. Cuvîntul **program** este un cuvînt rezervat al limbajului, iar `P1` este un cuvînt-utilizator. Cuvintele rezervate servesc pentru perfectarea programelor, iar cuvintele-utilizator – pentru denumirea variabilelor, subalgoritmilor, programelor, constantelor etc.

Linia 2. Este un text explicativ, un comentariu. Comentariul începe cu simbolul "{" și se termină cu "}". Comentariul nu influențează în niciun fel derularea programului și este destinat exclusiv utilizatorului.

Linia 3. Cuvîntul rezervat **var** (*variable* – variabilă) descrie variabilele `x`, `y`, `z` și `s`, aplicate în program. Cuvîntul `integer` (*întreg*) indică tipul variabilelor respective. Prin urmare `x`, `y`, `z` și `s` pot avea ca valori numai numere întregi. Linia respectivă formează partea declarativă a programului.

Linia 4. Cuvîntul rezervat **begin** (început) indică începutul părții executabile a programului.

Linia 5. Această linie indică afișarea unui mesaj la dispozitivul-standard de ieșire, în mod obișnuit pe ecran. Cuvîntul `writeln` (*write line* – scrie și trece la linie nouă) reprezintă apelul unui subalgoritm-standard, argumentul fiind textul mesajului ce se afișează:

```
Introduceti numerele intregi x, y, z:
```

Menționăm că apostrofurile nu fac parte din textul care va fi afișat.

Linia 6. Citirea a trei numere de la dispozitivul-standard de intrare, în mod obișnuit – tastatura. Numerele sînt tastate în aceeași linie și sînt despărțite de unul sau mai multe spații. După tastarea ultimului număr se acționează tasta <ENTER>. Numerele citite sînt depuse în variabilele `x`, `y`, `z`. Cuvîntul `readln` (*read line* – citire și trecere la linie nouă) reprezintă apelul unui subalgoritm-standard. Argumentele subalgoritmului sînt numele variabilelor în care sînt memorate numerele întregi introduse.

Linia 7. Instrucțiunea de atribuire. Variabila `s` primește valoarea `x+y+z`.

Linia 8. Afișarea mesajului

```
Suma numerelor introduse:
```

la dispozitivul-standard de ieșire.

Linia 9. Afișarea valorii variabilei `s` la dispozitivul-standard de ieșire.

Linia 10. Cuvîntul rezervat **`end`** indică sfîrșitul părții executabile, iar punctul – sfîrșitul programului.

Prin urmare un program în limbajul PASCAL este alcătuit din următoarele componente:

- **antetul**, în care se specifică denumirea programului;
- **partea declarativă**, în care se descriu variabilele, funcțiile, subalgoritmii etc. folosiți în program;
- **partea executabilă**, care include instrucțiunile ce urmează să fie executate într-o anumită ordine de calculator.

Pentru editarea, compilarea și lansarea în execuție a programelor PASCAL, au fost elaborate aplicații speciale, denumite medii de dezvoltare a programelor. De obicei, în laboratoarele școlare de informatică este instalată aplicația Turbo PASCAL 7.0. Interfața grafică a acestei aplicații conține meniuri ce permit efectuarea următoarelor operații:

- introducerea și editarea programelor PASCAL;
- păstrarea programelor PASCAL în fișiere distincte;
- deschiderea, editarea și salvarea fișierelor-text și a fișierelor ce conțin programe PASCAL;
- depistarea erorilor în programele PASCAL;
- compilarea și lansarea în execuție a programelor PASCAL.

Întrebări și exerciții

❶ Introduceți și lansați în execuție programul `P1`.

❷ Care sînt părțile componente ale unui program PASCAL?

❸ Introduceți și lansați în execuție următorul program:

```
Program P2;
  { Afisarea constantei predefinite MaxInt }
begin
```

```
  writeln('MaxInt=', MaxInt);
end.
```

❹ Indicați antetul, partea declarativă și partea executabilă ale programului `P2`. Explicați destinația fiecărei linii a programului în studiu.

❺ Modificați programul `P1` în așa mod ca el să calculeze suma numerelor *x*, *y* introduse de la tastatură.

1.2. Metalimbajul BNF

Un limbaj de programare se definește prin sintaxa și semantica lui. E cunoscut faptul că sintaxa este un set de reguli care guvernează alcătuirea propozițiilor, iar semantica este un set de reguli care determină înțelesul, semnificația propozițiilor respective. În cazul limbajelor de programare, echivalentul *propoziției* este *programul*.

Sintaxa oricărui limbaj de programare poate fi descrisă cu ajutorul unui limbaj întrebuințat în comunicarea dintre oameni, cum ar fi, de exemplu, româna, engleza, franceza etc. S-a considerat însă că o astfel de descriere este voluminoasă și neunivocă. Pentru o descriere concisă și exactă a sintaxei limbajelor de programare, s-au elaborat limbajele speciale, denumite *metalimbaje*. Cel mai răspîndit metalimbaj este cunoscut sub denumirea de *BNF – Forma Normală a lui Backus*.

Metalimbajul BNF utilizează următoarele simboluri:

• **simbolurile terminale**, adică simbolurile care apar exact la fel și în programele PASCAL;

• **simbolurile neterminale**, care desemnează unitățile (construcțiile) gramaticale ale limbajului.

Simbolurile neterminale se înscriu între semnele "<" și ">".

De exemplu, cifrele `0`, `1`, `2`, ..., `9`, literele `A`, `B`, `C`, ..., `Z` sînt simboluri terminale, iar <*Cifră*>, <*Literă*> sînt simboluri neterminale.

Descrierea sintaxei limbajului PASCAL constă dintr-un set de formule metalingvistice.

Prin **formulă metalingvistică** vom înțelege o construcție formată din două părți, stînga și dreapta, separate prin simbolurile "::=" ce au semnificația de *"egal prin definiție"*. În partea stîngă a formulei se găsește un simbol neterminal.

O formulă metalingvistică permite descrierea, în partea ei dreaptă, a tuturor alternativelor posibile de definire a simbolului neterminal, prin folosirea caracterului "|" cu semnificația *"sau"*.

De exemplu, formula

<*Cifră*> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

definește unitatea gramaticală <*Cifră*> ca fiind unul din caracterele (simbolurile terminale) `0`, `1`, ..., `9`.

La fel se interpretează și formula metalingvistică:

<*Literă*> ::= `a | b | c | d | e | f | g | h | i | j | k | l |m |`
`n | o | p | q | r | s | t | u | v | w | x | y | z`

În partea dreaptă a unei formule metalingvistice pot apărea două și mai multe simboluri consecutive. Situația corespunde operației de **concatenare** (alipire) a lor.

Astfel:

<Id> ::= <Literă><Cifră>

definește construcția gramaticală *<Id>* ca fiind o literă urmată de o cifră.

Exemple:

1) `a9`

2) `a1`

3) `c3`

4) `x4`

5) `d0`

6) `e8`

În unele situații alternativele de definire a unui simbol neterminal se pot repeta de un număr oarecare de ori (chiar de zero ori), fapt ce va fi marcat prin încadrarea lor în acoladele {, }.

De exemplu, formula

<Întreg fără semn> ::= *<Cifră>* {*<Cifră>*}

definește simbolul neterminal *<Întreg fără semn>* ca o secvență nevidă de cifre. Secvențele `0`, `0000`, `001`, `1900`, `35910` sînt conforme acestei definiții, iar secvența `3a5910` – nu.

Formula

<Identificator> ::= *<Literă>* {*<Literă>* | *<Cifră>*}

are următoarea semnificație: un identificator începe cu o literă; după această literă poate urma o secvență finită de litere sau cifre. Astfel, `a`, `a1`, `a1b`, `a23x`, `a14bxz` sînt conforme cu această definiție, dar `2a` – nu.

În cazul în care alternativele de definire a unui neterminal sînt opționale (pot lipsi), ele se încadrează în parantezele drepte [,].

De exemplu, formula

<Factor scală> ::= [+ | –] *<Întreg fără semn>*

definește factorul de scală ca un număr întreg fără semn care poate fi precedat de + sau –. Astfel, `1`, `+1`, `–1`, `20`, `+20`, `–20`, `+003` sînt conforme cu această definiție, dar `3–5` – nu.

Atragem atenția asupra faptului că simbolurile [,], {, } aparțin metalimbajului și nu trebuie confundate cu simbolurile corespunzătoare utilizate în limbajul PASCAL.

Întrebări și exerciții

❶ Explicați termenii *sintaxă* și *semantică*.

❷ Care este destinația unui metalimbaj?

❸ Cum se definește sintaxa unui limbaj cu ajutorul metalimbajului BNF?

❹ Sintaxa unui limbaj foarte simplu este descrisă folosind următoarele formule metalingvistice:

<Cifră> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

<Număr> ::= *<Cifră>* {*<Cifră>*}

<*Semn*> ::= + | –
<*Expresie aritmetică*> ::= <*Număr*> {<*Semn*><*Număr*>}

Care din secvențele ce urmează sînt conforme definiției unității lexicale <*Număr*>?

a) `0`
b) `1`
c) `11100`
d) `00011`
e) `20013`
f) `0+0`
g) `11100+1`
h) `11100-1`
i) `931`
j) `614`
k) `0000`
l) `0001`
m) `-152`
n) `+351`
o) `412`

Care din secvențele ce urmează sînt conforme definiției unității lexicale <*Expresie aritmetică*>?

a) `0+1`
b) `1+0-3`
c) `0+0+4`
d) `1+1-9`
e) `6+6+21`
f) `-13`
g) `21+-16`
h) `-21-16`
i) `68-13`
j) `42+650`
k) `21+00000`
l) `39+00001`
m) `00001-00001`
n) `379-486`
o) `31+12-51+861`

❺ Sintaxa unui limbaj de comunicare utilizator–calculator este definită după cum urmează:

<*Disc*> ::= `A: | B: | C: | D: | E:`
<*Listă parametri* > ::= <*Disc*> {, <*Disc*>}
<*Nume comandă*> ::= `Citire | Copiere | Formatare`
<*Comandă*> ::= <*Nume comandă*><*Listă parametri* >

Care din secvențele ce urmează sînt conforme definiției unității lexicale <*Comandă*>?

a) `Citire`
b) `Citire A:`
c) `Copiere F:`
d) `Copiere A:,`
e) `Formatare D:, E:`
f) `Copiere A: B:`
g) `Citire D`
h) `Formatare D:, F:`
i) `Copiere E:, A:,`
j) `Copiere F:, A:`

1.3. Diagrame sintactice

Diagramele sintactice descriu mai clar sintaxa unui limbaj de programare. Reprezentarea prin diagrame poate fi derivată din notația BNF, după cum urmează.

Fiecărui simbol terminal îi corespunde un cerc sau un oval în care se înscrie simbolul respectiv. Simbolurile neterminale se înscriu în dreptunghiuri. Ovalurile și dreptunghiurile se reunesc conform diagramelor din *figura 1.1*.

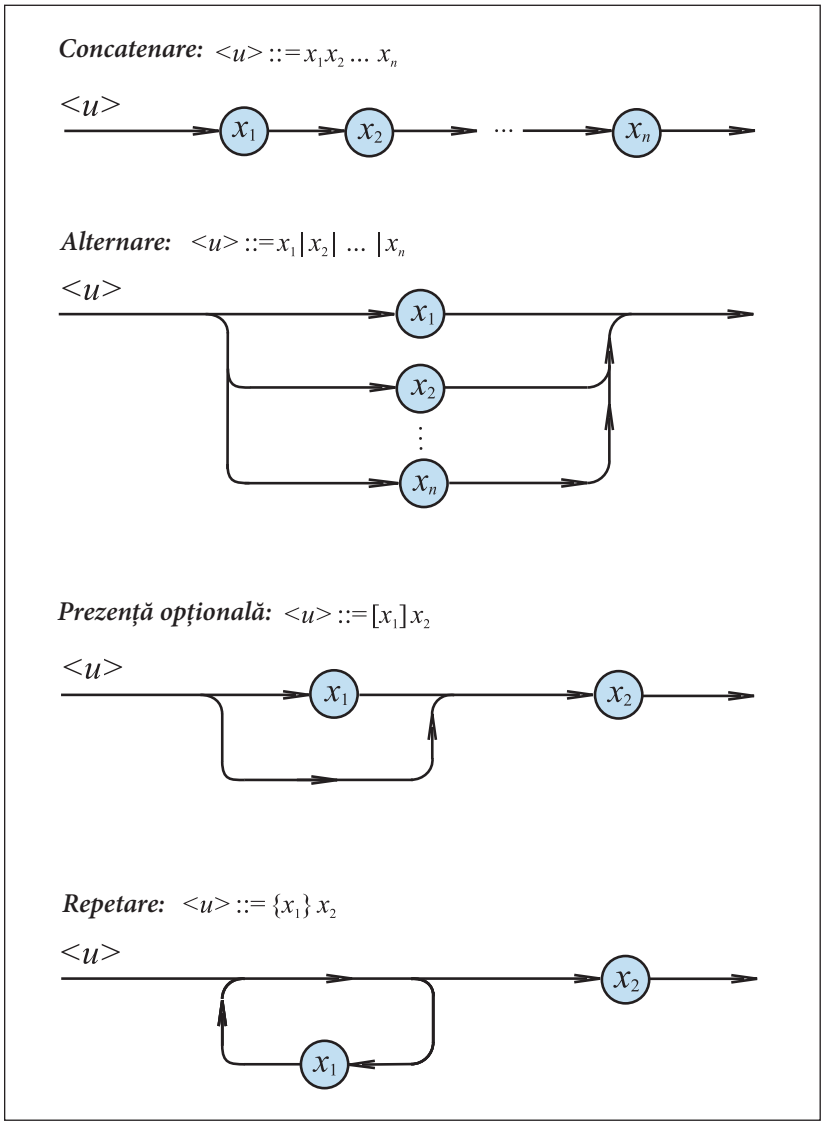


Fig. 1.1. Reprezentarea formulelor BNF prin diagrame sintactice

În *figura 1.2* sînt prezentate diagramele sintactice pentru unităţile gramaticale <*Întreg fără semn*>, <*Identificator*> şi <*Factor scală*>, definite în paragraful precedent. Observăm că fiecărui drum în diagrama sintactică îi corespunde o secvenţă de simboluri terminale corectă din punct de vedere sintactic.

Întrebări şi exerciţii

❶ Care este destinaţia diagramelor sintactice?
❷ Cum se reprezintă simbolurile terminale şi simbolurile neterminale pe diagramele sintactice?
❸ Cum se reprezintă formulele BNF pe diagramele sintactice?

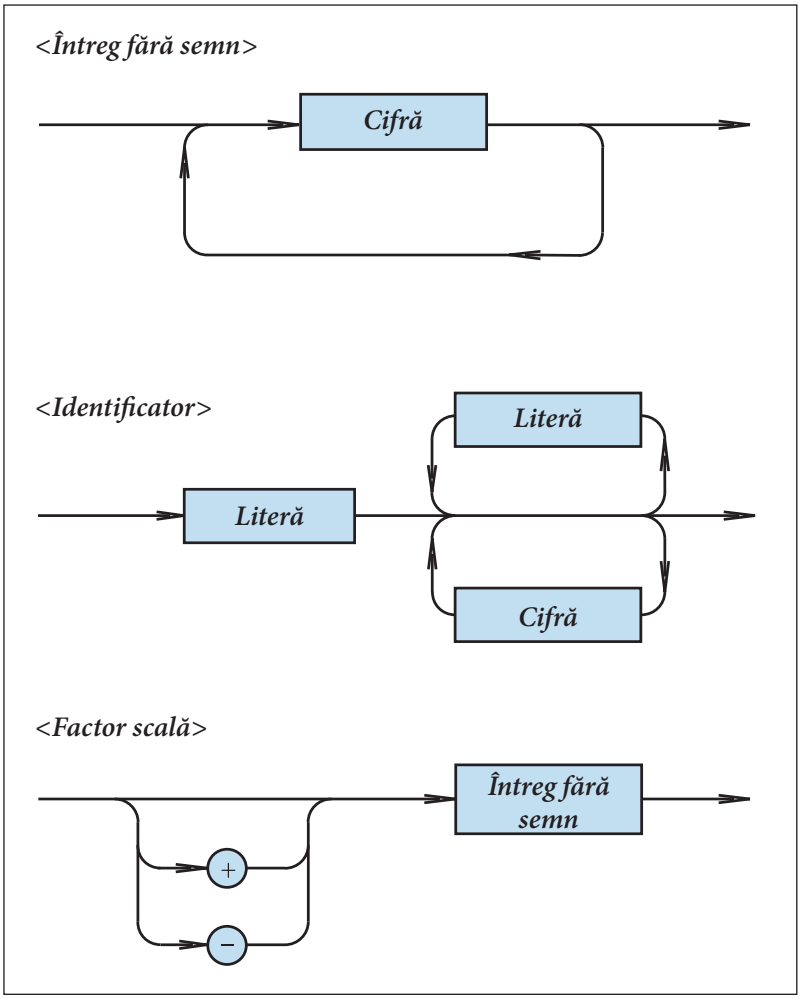


Fig. 1.2. Diagramele sintactice <*Întreg fără semn*>, <*Identificator*> şi <*Factor scală*>

❹ Reprezentați cu ajutorul diagramelor sintactice:

<*Cifră*> :: = `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`
<*Număr*> :: = <*Cifră*> {<*Cifră*>}
<*Semn*> ::= + | –
<*Expresie aritmetică*> ::= <*Număr*>{<*Semn*><*Număr*>}

❺ Reprezentați cu ajutorul diagramelor sintactice:

<*Disc*> ::= `A:|B:|C:|D:|E:`
<*Listă parametri* > ::= <*Disc*>{, <*Disc*>}
<*Nume comandă*> ::= `Citire | Copiere | Formatare`
<*Comandă*> ::= <*Nume comandă*> <*Listă parametri* >

❻ În *figura 1.3* sînt prezentate diagramele sintactice care definesc unitatea gramaticală <*Număr fracționar*>. Determinați care din secvențele ce urmează sînt conforme acestor diagrame:

a) `0.1`

b) `+0.1`

c) `-0.0`

d) `9.000`

e) `-538.`

f) `.538`

g) `721.386.`

h) `-421`

i) `247.532`

j) `+109.000`

❼ Scrieți formulele BNF care corespund diagramelor sintactice din *figura 1.3*.

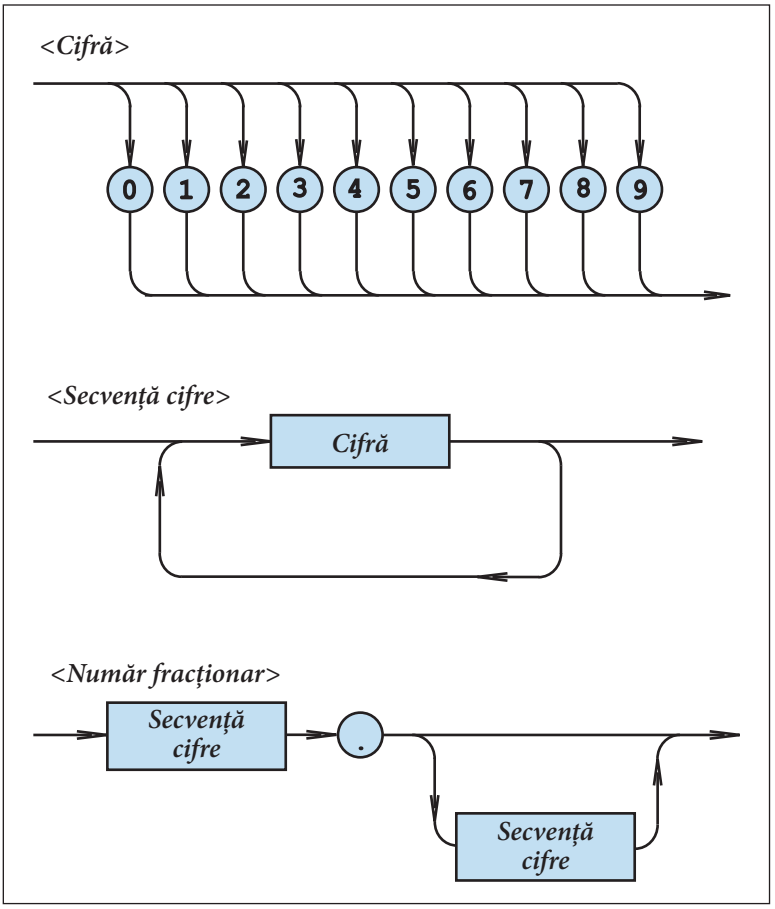


Fig. 1.3. Diagramele sintactice <*Cifră*>, <*Secvență cifre*>, <*Număr fracționar*>

1.4. Alfabetul limbajului

Alfabetul limbajului PASCAL este format din următoarele caractere ale codului *ASCII* (*American Standard Code for Information Interchange*):

- cifrele zecimale;

- literele mari și mici ale alfabetului englez;
- semnele de punctuație;
- operatorii aritmetici și logici;
- caracterele de control și editare (spațiu, sfîrșit de linie sau retur de car etc.).

În unele construcții ale limbajului pot fi folosite și literele alfabetelor naționale, de exemplu literele ă, â, î, ş, ţ ale alfabetului român.

1.5. Vocabularul limbajului

Cele mai simple elemente, alcătuite din caractere și înzestrate cu semnificație lingvistică, se numesc *lexeme* sau *unități lexicale*. Acestea formează **vocabularul** limbajului PASCAL.

Distingem următoarele **unități lexicale**:

- simboluri speciale și cuvinte-cheie;
- identificatori;
- numere;
- șiruri de caractere;
- etichete;
- directive.

1.5.1. Simboluri speciale şi cuvinte-cheie

Simbolurile speciale sînt formate din unul sau două caractere:

+	plus	<	mai mic
−	minus	>	mai mare
*	asterisc	[	paranteză pătrată din stînga
/	bară	]	paranteză pătrată din dreapta
=	egal	(	paranteză rotundă din stînga
,	virgulă	)	paranteză rotundă din dreapta
:	două puncte	;	punct și virgulă
.	punct	^	accent circumflex
@	la	$	dolar
{	acoladă din stînga	<=	mai mic sau egal
}	acoladă din dreapta	>=	mai mare sau egal

`#`	număr	`:=`	atribuire
`..`	puncte de suspensie	`<>`	neegal
`(*`	echivalentul acoladei {	`(.`	echivalentul parantezei [
`*)`	echivalentul acoladei }	`.)`	echivalentul parantezei]

Menționăm că dacă un simbol special este alcătuit din două caractere, de exemplu <= sau :=, între ele nu trebuie să apară niciun spațiu intermediar.

Cuvintele-cheie sînt formate din două sau mai multe litere:

`and`	și	**`nil`**	zero
`array`	tablou	**`not`**	nu
`begin`	început	**`of`**	din
`case`	caz	**`or`**	sau
`const`	constante	**`packed`**	împachetat
`div`	cîtul împărțirii	**`procedure`**	procedură
`do`	execută	**`program`**	program
`downto`	în descreștere la	**`record`**	articol (înregistrare)
`else`	altfel	**`repeat`**	repetare
`end`	sfîrșit	**`set`**	mulțime
`file`	fișier	**`then`**	atunci
`for`	pentru	**`to`**	la
`function`	funcție	**`type`**	tip
`goto`	treci la	**`until`**	pînă ce
`if`	dacă	**`var`**	variabile
`in`	în	**`while`**	cît
`label`	etichetă	**`with`**	cu
`mod`	restul împărțirii		

Cuvintele-cheie sînt rezervate și nu pot fi folosite în alt scop decît cel dat prin definiția limbajului.

Unitățile lexicale în studiu se definesc cu ajutorul următoarelor formule BNF:

<Simbol special> ::= + | − | * | / | = | < | > |] | [| , | (|)
| : | ; | ^ | . | @ | { | } | $ | # | <=

| >= | <> | := | .. | <*Cuvînt-cheie*> | <*Simbol echivalent*>

<*Simbol echivalent*> ::= (* | *) | (. | .)

<*Cuvînt-cheie*>::= **and** | **array** | **begin** | **case** | **const** | **div** | **do** | **downto** | **else** | **end** | **file** | **for** | **function** | **goto** | **if** | **in** | **label** | **mod** | **nil** | **not** | **of** | **or** | **packed** | **procedure** | **program** | **record** | **repeat** | **set** | **then** | **to** | **type** | **until** | **var** | **while** | **with**

De reținut că simbolurile {, }, [și] utilizate în notația BNF sînt în același timp și elemente ale vocabularului PASCAL. Pentru a evita confuziile, aceste simboluri, ca elemente ale vocabularului, pot fi redate prin simbolurile echivalente (*, *), (. și respectiv .).

Întrebări și exerciții

❶ Memorați cuvintele-cheie ale limbajului PASCAL.
❷ Care este diferența dintre caractere și simboluri?
❸ Desenați diagramele sintactice pentru unitățile lexicale <*Simbol special*>, <*Simbol echivalent*> și <*Cuvînt-cheie*>.

1.5.2. Identificatori

Identificatorii sînt unități lexicale care desemnează variabile, constante, funcții, programe ș.a. Un identificator începe cu o literă, care poate fi urmată de orice combinație de litere și cifre. Lungimea identificatorilor nu este limitată, dar sînt semnificative doar primele 63 de caractere.

Amintim formulele BNF care definesc unitatea lexicală <*Identificator*>:

<*Cifră*> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<*Literă*> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<*Identificator*> ::= <*Literă*> { <*Literă*> | <*Cifră*> }

Exemple de identificatori:

1) `x`
2) `y`
3) `z`
4) `x1`
5) `y10`
6) `z01b`
7) `lista`
8) `listaelevilor`
9) `listatelefoanelor`
10) `registru`

11) `adresa`

12) `adresadomiciliu`

13) `clasa10`

14) `anul2011`

În construcțiile gramaticale ale limbajului PASCAL, cu excepția șirurilor de caractere, literele mari și mici se consideră echivalente. Prin urmare sînt echivalenți și identificatorii:

1) `x` și `X`

2) `y` și `Y`

3) `z` și `Z`

4) `x1` și `X1`

5) `y10` și `Y10`

6) `z01b`, `Z01b`, `Z01B` și `z01B`

7) `lista`, `Lista`, `LIsta`, `ListA`, `LISTa`

Utilizarea literelor mari și mici ne permite să scriem identificatorii mai citeț, de exemplu:

1) `ListaElevilor`

2) `ListaTelefoanelor`

3) `AdresaDomiciliu`

4) `BugetulAnului2011`

Menționăm că în construcțiile de bază ale limbajului PASCAL nu se utilizează literele `ă`, `â`, `î`, `ş`, `ţ` ale alfabetului român. Aşadar în scrierea identificatorilor semnele diacritice respective vor fi omise.

Exemple:

1) `Suprafata`

2) `Numar`

3) `NumarElevi`

4) `Patrat`

5) `SirDeCaractere`

6) `NumarIncercari`

Întrebări și exerciții

❶ Desenați diagramele sintactice pentru unitățile gramaticale <*Cifră*>, <*Literă*> și <*Identificator*>.

❷ Care din secvențele propuse sînt conforme definiției unității lexicale <*Identificator*>?

a) `x1`

b) `X1`

c) `1x`

d) `1X`

e) `xy`

f) `Suprafata`

g) `SUPRAFATA`

h) `rădăcina`

i) `radacina`

j) `R1`

k) `A1x`

l) `ListaA`

m)	`Lista1`	*p)*	`Dreptunghi`	*s)*	`Luni`
n)	`B-1`	*q)*	`iI`	*t)*	`Luna`
o)	`abc`	*r)*	`I1j`	*u)*	`20.07.2011`

Pentru secvențele corecte indicați drumurile respective din diagrama sintactică <*Identificator*>.

❸ Găsiți perechile de identificatori echivalenți:

a)	`x101`	*k)*	`CERCURI`
b)	`ya15`	*l)*	`SirDeCaractere`
c)	`radacinaX1`	*m)*	`Triunghiuri`
d)	`radacinaX2`	*n)*	`RegistruClasa10`
e)	`triunghi`	*o)*	`zilE`
f)	`cerc`	*p)*	`X101`
g)	`sirdecaractere`	*q)*	`RegistruClasa10`
h)	`registruclasa10`	*r)*	`radaciniX1X2`
i)	`COTIDIAN`	*s)*	`RADACINAX1`
j)	`ZILE`	*t)*	`yA101`

❹ Care este destinația identificatorilor din programele PASCAL?

❺ Pentru a găsi soluțiile x_1, x_2 ale ecuației pătrate $ax^2 + bx + c = 0$, mai întîi se calculează discriminantul d. Propuneți cîteva variante de reprezentare a coeficienților a, b, c ai discriminantului d și a soluțiilor x_1, x_2 prin identificatori.

1.5.3. Numere

Numerele pot fi întregi sau reale. În mod obișnuit, se folosește sistemul zecimal de numerație. În *figura 1.4* sînt prezentate diagramele sintactice pentru unitățile lexicale <*Număr întreg*> și <*Număr real*>.

Exemple de numere întregi:

`23`	`-23`	`0023`	`+023`
`318`	`00318`	`+0318`	`-318`
`1996`	`+1996`	`-1996`	`0001996`
`-0023`	`-0318`	`+001996`	`-000199`

În cazul numerelor reale, partea fracționară se separă de partea întreagă prin punct. Punctul zecimal trebuie să fie precedat și urmat de cel puțin o cifră zecimală.

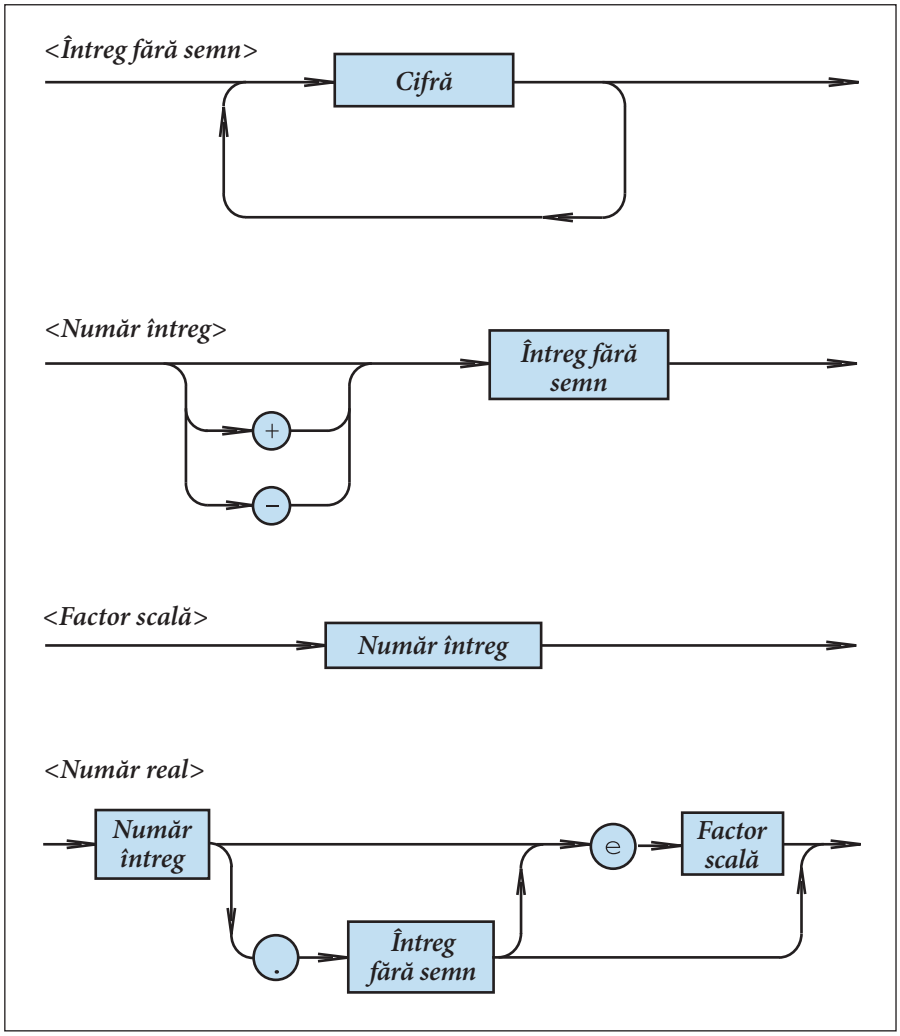


Fig. 1.4. Diagramele sintactice <*Număr întreg*> și <*Număr real*>

Exemple de numere reale:

`3.1415`	`+3.04`	`0.0001`	`283.19`
`-3.04`	`-0.0001`	`-256.19`	`28.17`
`+0.0001`	`6.28`	`+3.12421`	`63906.734`

În scrierea numerelor reale se poate utiliza și un **factor de scală**. Acesta este un număr întreg precedat de litera `e` (sau `E`), care indică că numărul urmat de factorul de scală se înmulțește cu 10 la puterea respectivă.

Exemple:

	Forma uzuală	*Notația în PASCAL*
1)	$8{,}12 \cdot 10^{-5}$	`8.12e-5`

2)	$749{,}512 \cdot 10^{8}$	`749.512e+8`
3)	$-0{,}0823 \cdot 10^{-12}$	`-0.0823e-12`
4)	$3250{,}4 \cdot 10^{6}$	`3250.4e06`
5)	$3{,}421 \cdot 10^{16}$	`3.421e16`

Evident, `8.12e-05`, `812e-7`, `0.812e-4`, `81.2e-6` reprezintă una și aceeași valoare $8{,}12 \cdot 10^{-5}$.

Întrebări și exerciții

❶ Care din secvențele de caractere ce urmează sînt conforme definiției unității lexicale <*Număr întreg*>?

a) `-418`	*f)* `0+2469`	*k)* `32,014`
b) `0-418`	*g)* `32,14`	*l)* `-719`
c) `621+`	*h)* `+00621`	*m)* `+62.1`
d) `2469`	*i)* `24693.`	*n)* `-00418`
e) `-6210`	*j)* `-621`	*o)* `-00621`

Găsiți numerele întregi care reprezintă una și aceeași valoare.

❷ Pornind de la diagramele sintactice din *figura 1.4*, scrieți formulele BNF pentru definirea unității lexicale <*Număr întreg*>.

❸ Care din secvențele de mai jos sînt conforme definiției unității lexicale <*Număr real*>?

a) `3.14`	*h)* `281.3`	*o)* `0,618284e00`
b) `2.514e+5`	*i)* `591328`	*p)* `1961.`
c) `591328E+3`	*j)* `2514e+2`	*q)* `28130E-2`
d) `.000382`	*k)* `-464.597e+3`	*r)* `591.328`
e) `0.1961E+4`	*l)* `+519.328e-4`	*s)* `-658.14e-6`
f) `+314629.`	*m)* `591328e-3`	*t)* `2514e+2`
g) `0.000314E4`	*n)* `28130e-2`	*u)* `618.248e-3`

Găsiți numerele reale care reprezintă una și aceeași valoare. Scrieți aceste numere în forma uzuală.

❹ Pornind de la diagramele sintactice din *figura 1.4*, scrieți formulele BNF pentru definirea unității lexicale <*Număr real*>.

❺ Indicați pe diagramele sintactice din *figura 1.4* drumurile care corespund numerelor:

a) `-418`	*b)* `1961.0`	*c)* `2514E+2`

d) `281.3` *g)* `32.014` *j)* `+0001`

e) `2.514e+5` *h)* `591.328` *k)* `-614.85e-3`

f) `-1951.12` *i)* `+19.511e+2` *l)* `2013e-4`

1.5.4. Şiruri de caractere

Şirurile de caractere sînt şiruri de caractere imprimabile, delimitate de apostrof. În şirul de caractere apostroful apare dublat. Accentuăm că în cazul şirurilor de caractere literele mari şi mici apar drept caractere distincte.

Exemple:

1) `'Variabila x'`

2) `'Calculul aproximativ'`

3) `'Apostroful '' este dublat'`

Spre deosebire de alte unităţi lexicale ale limbajului PASCAL, în şirurile de caractere pot fi utilizate şi literele ă, â, î, ş, ţ ale alfabetului român. În acest scop e necesar ca pe calculatorul la care lucraţi să fie instalate **programele-pilot** ce asigură introducerea, afişarea şi imprimarea literelor date.

Exemple:

1) `'Şir de caractere'`

2) `'Limba engleză'`

3) `'Suprafaţă'`

4) `'Număr încercări'`

Unitatea lexicală <*Şir de caractere*> se defineşte cu ajutorul următoarelor formule BNF:

<*Şir de caractere*> ::= ' <*Element şir*> {<*Element şir*>} '

<*Element şir*> ::= '' | <*Orice caracter imprimabil*>

Diagrama sintactică a unităţii lexicale în studiu este prezentată în *figura 1.5.*

Întrebări şi exerciţii

❶ Indicaţi pe diagramele sintactice din *figura 1.5* drumurile care corespund şirurilor de caractere:

a) `'variabila z'`

b) `''''`

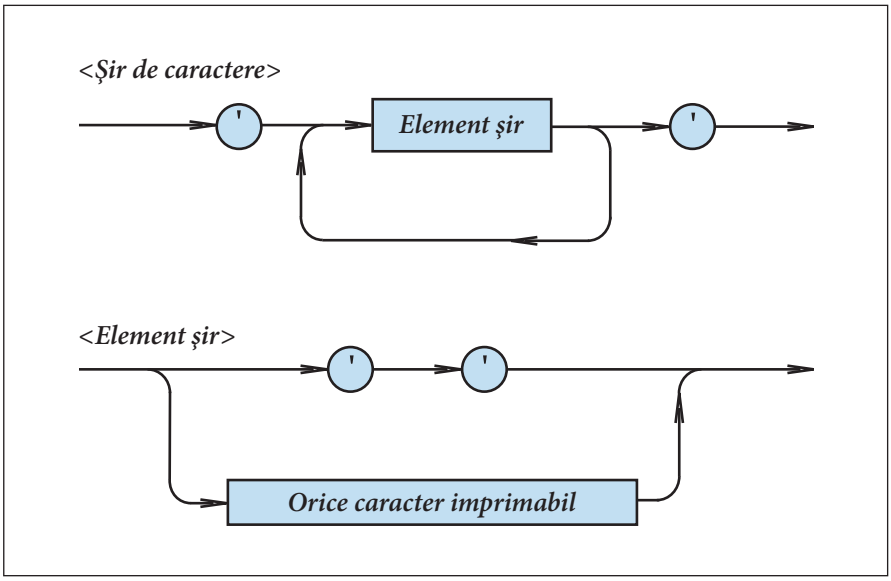


Fig. 1.5. Diagrama sintactică <*Şir de caractere*>

c) `'Caracterele ''x'', ''y'''`

d) `'UNITĂŢI LEXICALE'`

❷ Care dintre secvenţele ce urmează sînt conforme definiţiei unităţii lexicale <*Şir de caractere*>:

a) `'Număr întreg'`

b) `'Sfîrşitul programului'`

c) `'APOSTROF'`

d) `''x''`

e) `'funcţie'`

f) `'Anul 1997'`

g) `'Rădăcină pătrată'`

h) `'Anul '97'`

i) `'Lista telefoanelor'`

j) `'''`

1.5.5. Etichete

Etichetele sînt numere întregi fără semn din domeniul 0, 1, ..., 9999 şi se utilizează pentru a marca instrucţiunile limbajului PASCAL.

Exemple:

```
1    100    999    582    1004
```

Evident, formula BNF care defineşte unitatea lexicală în studiu are forma:

<*Etichetă*> ::= <*Întreg fără semn*>

1.5.6. Directive

Unitatea lexicală <*Directivă*> se definește exact ca identificatorul:

<*Directivă*> ::= <*Literă*> {<*Literă*> | <*Cifră*>}

Efectiv, directivele sînt cuvinte rezervate care au o semnificație specială. Limbajul-standard conține o singură directivă:

```
forward
```

Aceasta se folosește la descrierea unor proceduri (subalgoritmi) și funcții definite de utilizator.

1.6. Separatori

Orice program PASCAL constă din lexeme și separatori. Separatorii folosiți în limbaj sînt spațiul, sfîrșitul de linie (retur de car) și comentariul.

Exemple:

1)
```
x div y
```

2)
```
not x
```

3)
```
begin
  writeln(x);
  writeln(y);
end.
```

În lipsa separatorilor, la scrierea consecutivă a identificatorilor, a cuvintelor-cheie, a numerelor fără semn și a directivelor, începutul unei unități lexicale ar putea fi interpretat în unele cazuri drept o continuare a celei precedente.

În particular, construcția "`x div y`" din primul exemplu comunică calculatorului "împarte valoarea variabilei `x` la valoarea variabilei `y`". Însă, în lipsa spațiior de separare, construcția "`xdivy`" va fi interpretată ca un identificator.

Menționăm că simbolurile speciale compuse din două caractere <=, >=, <>, `:=`, `..` etc., identificatorii, numerele ș.a.m.d. sînt unități lexicale ale programului. Prin urmare nu se pot introduce spații sau retururi de car între caracterele componente.

Exemple:

	Corect	*Incorect*
1)	`CitireDisc`	`Citire Disc`
2)	`Program`	`Pro gram`
3)	`:=`	`: =`
4)	`..`	`. .`

5)	`345`	`3 45`
6)	**`downto`**	**`down to`**
7)	**`begin`**	**`be gin`**

Comentariile sînt secvențe de caractere precedate de `{` și urmate de `}`.

Exemple:

1)
```
{ Program elaborat de Radu Ion }
```

2)
```
{ Introducerea datelor initiale }
```

3)
```
{ Datele initiale se introduc de la tastatura.
  Rezultatele vor fi afisate pe ecran si tiparite
  la imprimanta peste 3-4 minute }
```

Comentariile nu influențează în niciun fel derularea programelor PASCAL și se utilizează pentru a include în ele unele precizări, explicații, informații suplimentare etc. Desigur, comentariile nu sînt destinate calculatorului, ci persoanelor care citesc programul respectiv.

Accentuăm că utilizarea rațională a comentariilor, spațiilor și retururilor de car asigură scrierea unor programe lizibile (ușor de citit).

Test de autoevaluare nr. 1

1. Sintaxa limbajului de programare a executantului ***Robot*** este descrisă cu ajutorul formulelor metalingvistice:

<*Cifră*> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

<*Număr*> ::= <*Cifră*> {<*Cifră*>}

<*Comandă*> ::= `sus | jos | dreapta | stînga`

<*Instrucțiune*> ::= <*Comandă*> `(`<*Număr*>`)`

<*Program*> ::= **`început`** {< *Instrucțiune* >`;`} **`sfîrșit`**

Indicați programele corecte sintactic:

a)
```
început sus(1); dreapta(4); jos(0); stînga(00); sfîrșit
```

b)
```
început sus(1); dreapta(73); jos(0); stînga(00+23); sfîrșit
```

c)
```
început jos(30); dreapta(45); sus(980); sfîrșit
```

d)
```
început stînga(21); jos(50); dreapta(45); sus(980); sfîrșit
```

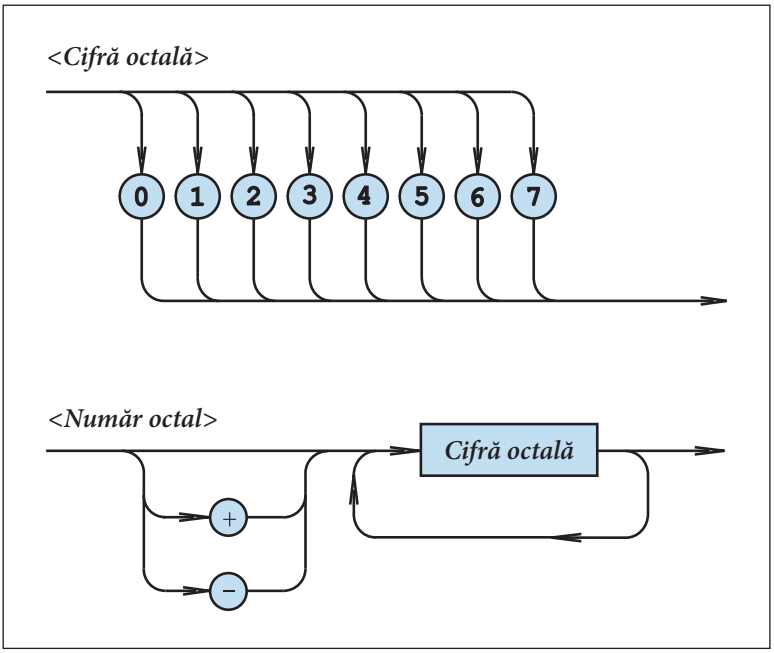
e)
```
început stînga(3); jos(13); stînga(21) sfîrșit; sfîrșit
```

f)
```
început jos(73); dreapta(5); sus(71) stînga(13); sfîrșit
```

g)
```
început sus(1); dreapta(-4); jos(0); stînga(10950); sfîrșit
```

2. Desenați diagramele sintactice ce corespund formulelor metalingvistice <*Comandă*>, <*Instrucțiune*> și <*Program*> din itemul 1.

3. În figura de mai jos sînt prezentate diagramele sintactice care definesc unitatea gramaticală <*Număr octal*>.



Determinați care din secvențele ce urmează sînt conforme diagramei <*Număr octal*>:

a)	`+0`	*f)*	`-34637`	*k)*	`+123146482351`
b)	`18`	*g)*	`2347-523`	*l)*	`614,45`
c)	`-17250`	*h)*	`-0000007`	*m)*	`-152`
d)	`+6362,1`	*i)*	`527345372`	*n)*	`+35,1`
e)	`717424410571`	*j)*	`614.45`	*o)*	`-412`

4. Scrieți formulele metalingvistice ce corespund diagramelor sintactice din itemul 3.

5. În limbajul PASCAL un identificator începe cu o literă, care poate fi urmată de orice combinație de litere și cifre. Scrieți formulele metalingvistice care definesc unitatea lexicală <*Identificator*>.

6. Compuneți cel puțin zece identificatori care ar reflecta specificul problemelor din fizică, matematică, chimie, prelucrarea textelor și imaginilor.

7. Transcrieți din forma uzuală în notația PASCAL numerele:

a)	3,14	*b)*	265	*c)*	23,4635

d)	+0,000001	*h)*	+28	*l)*	–38,00001
e)	$6{,}1532 \cdot 10^{-5}$	*i)*	+28000000	*m)*	$35728{,}345452 \cdot 10^{-8}$
f)	–984,52	*j)*	$614{,}45 \cdot 10^{-12}$	*n)*	24815
g)	–523	*k)*	$-3628{,}297 \cdot 10^{12}$	*o)*	–296,0020001

8. Transcrieți din notația PASCAL în forma uzuală numerele:

a)	`6124.485`	*f)*	`-0.03428e-08`	*k)*	`2005`
b)	`+18.315`	*g)*	`232847.5213`	*l)*	`+23.08e-5`
c)	`-218.034e-3`	*h)*	`-0000012e+2`	*m)*	`-17502`
d)	`193526`	*i)*	`18.45`	*n)*	`+1`
e)	`1000.01e+23`	*j)*	`623.495e-6`	*o)*	`-46341.2e-06`

9. Este cunoscut faptul că vocabularul limbajului PASCAL include unitățile lexicale: simboluri speciale, cuvinte-cheie, identificatori, numere, șiruri de caractere, etichete și directive. Indicați unitățile lexicale din programul ce urmează:

```
1  Program TA1;
2  var a, b, x : real;
3  begin
4    readln(a, b);
5    if a<>0 then
6     begin
7       x:=-b/a;
8       writeln('Ecuatia are o singura radacina');
9       writeln(x);
10    end;
11   if (a=0) and (b=0) then
12      writeln( 'Ecuatia are o multime
                  infinita de radacini');
13   if (a=0) and (b<>0) then
14      writeln('Ecuatia nu are sens');
15 end.
```

Amintim că numerele *1, 2, 3, ..., 15* din partea stîngă a paginii nu fac parte din programul PASCAL. Ele servesc doar pentru referirea liniilor.

Exemplu: **Program** – cuvînt-cheie; `TA1` – identificator; `;` – simbol special; **var** – cuvînt-cheie ș.a.m.d.

10. Indicați antetul, partea declarativă și partea executabilă ale programului `TA1` din itemul 9.

Capitolul 2

TIPURI DE DATE SIMPLE

2.1. Conceptul de dată

Informația care va fi supusă unei prelucrări este accesibilă calculatorului în formă de date. **Datele** sînt constituite din cifre, litere, semne, numere, șiruri de caractere ș.a.m.d.

Într-un limbaj cod-calculator datele sînt reprezentate prin secvențe de cifre binare. De exemplu, la nivelul procesorului numărul natural 1039 se reprezintă în sistemul de numerație binar ca:

```
10000001111
```

Pentru a scuti utilizatorul de toate detaliile legate de reprezentarea internă a datelor, în PASCAL se folosesc diverse tipuri de date.

Prin **tip de date** se înțelege o **mulțime de valori** și o **mulțime de operații** care pot fi efectuate cu valorile respective.

De exemplu, în versiunea Turbo PASCAL 7.0 tipul `integer` include mulțimea numerelor întregi

```
{-32768, -32767, ..., -2, -1, 0, 1, 2, ..., 32767}.
```

Cu aceste numere pot fi efectuate următoarele operații:

`+` adunarea;
`-` scăderea;
`*` înmulțirea;
`mod` restul împărțirii;
`div` cîtul împărțirii ș.a.

Tipul de date `real` (real) include o submulțime a numerelor reale, cu ajutorul cărora se realizează operațiile `+`, `-`, `*`, `/` (împărțirea) etc.

Operațiile **`mod`** și **`div`**, admise în cazul datelor de tip `integer`, sînt inadmisibile în cazul datelor de tip `real`.

Într-un program PASCAL datele sînt reprezentate prin **mărimi**, și anume: prin variabile și constante. Termenul "mărime" a fost împrumutat din matematică și fizică, unde mărimile sînt utilizate pentru descrierea anumitor fenomene. Pentru exemplificare, amintim unele mărimi studiate în cadrul lecțiilor respective: masa m, lungimea l, aria S, volumul V, accelerația căderii libere $g \approx 9{,}8$ m/s^2, numărul irațional $\pi \approx 3{,}14$ ș.a.

Variabila este o mărime valorile căreia pot fi modificate pe parcursul execuției programului. Fiecare variabilă are nume, valoare și tip. Numele variabilei (de exemplu, `m`, `l`, `S`, `V`, `delta`) servește pentru notarea ei în program. În timpul execuției programului, în

orice moment concret, fiecare variabilă are o valoare curentă (de exemplu, 105 sau –36) ori nu este definită.

Mulțimea de valori pe care le poate lua fiecare variabilă și operațiile admise se indică prin asocierea numelui de variabilă cu un anumit tip de date. În acest scop, numele de variabilă și tipul dorit de date se declară explicit cu ajutorul cuvîntului-cheie **`var`**.

Exemplu:

```
var x, y : integer;
    z : real;
```

În procesul derulării programului, variabilele `x` și `y` pot lua orice valori ale tipului de date `integer`, iar variabila `z` – orice valori ale tipului de date `real`.

Constanta este o mărime valoarea căreia nu poate fi modificată pe parcursul execuției programului. Tipul unei **constante** se declară implicit prin forma ei textuală. De exemplu, `10` este o constantă de tip `integer`, iar `10.0` este o constantă de tip `real`.

Pentru a face programele mai lizibile, constantele pot avea denumiri simbolice. Denumirile respective se definesc cu ajutorul cuvîntului-cheie **`const`**.

Exemplu:

```
const g = 9.8;
      pi = 3.14;
```

Evident, constantele `g` și `pi` sînt de tipul `real` și valorile lor nu pot fi schimbate pe parcursul derulării programului.

Conceptul de dată realizat în limbajul PASCAL presupune:

1) fiecare mărime (variabilă sau constantă) într-un program în mod obligatoriu se asociază cu un anumit tip de date;

2) tipul unei variabile definește mulțimea de valori pe care le poate lua variabila și operațiile care pot fi efectuate cu aceste valori;

3) există tipuri de date de interes general, definiția cărora se consideră cunoscută: `integer`, `real`, `char` (caracter), `boolean` (logic), `text` ș.a.;

4) pe baza tipurilor cunoscute programatorul poate crea tipuri noi, adecvate informațiilor de prelucrat.

Întrebări și exerciții

❶ Cum se reprezintă datele în limbajul cod-calculator? Care sînt avantajele și deficiențele acestei reprezentări?

❷ Cum se reprezintă datele într-un program PASCAL? Care este diferența dintre variabile și constante?

❸ Explicați semnificația termenului *tip de date*. Dați exemple.

❹ Cum se asociază o variabilă la un anumit tip de date?

❺ Determinați tipul variabilelor `r`, `s`, `t`, `x`, `y` și `z` din declarația ce urmează:

```
var r, y : integer;
    s, z : real;
    t, x : boolean;
```

❻ Scrieți o declarație care ar defini *a*, *b* și *c* ca variabile întregi, iar *p* și *q* ca variabile *text*.

❼ Precizați tipul următoarelor constante:

a) `-301` *d)* `-61.00e+2` *g)* `314.0`

b) `-301.0` *e)* `3.14` *h)* `0314`

c) `+6100` *f)* `-0.0001` *i)* `-0.000672`

2.2. Tipul de date `integer`

Mulțimea de valori ale tipului de date `integer` este formată din numerele întregi care pot fi reprezentate pe calculatorul-gazdă al limbajului. Valoarea maximă poate fi referită prin constanta `MaxInt`, cunoscută oricărui program PASCAL. De obicei, valoarea minimă, admisă de tipul de date în studiu, este `-MaxInt` sau `-(MaxInt+1)`.

Programul ce urmează afișează pe ecran valoarea constantei predefinite `MaxInt`:

```
Program P2;
 { Afisarea constantei predefinite MaxInt }
begin
  writeln('MaxInt=', MaxInt);
end.
```

Pe un calculator IBM PC, versiunea Turbo PASCAL 7.0, constanta `MaxInt` are valoarea `32767`, iar mulțimea de valori ale tipului `integer` este:

```
{-32768, -32767, ..., -2, -1, 0, 1, 2, ..., 32767}.
```

Operațiile care se pot face cu valorile întregi sînt: `+`, `-`, `*`, **`mod`**, **`div`** ș.a. Rezultatele acestor operații pot fi vizualizate cu ajutorul programului `P3`:

```
Program P3;
 { Operatii cu date de tipul integer }
var x, y, z : integer;
begin
  writeln('Introduceti numerele intregi x, y:');
  readln(x, y);
  writeln('x=', x);
  writeln('y=', y);
  z:=x+y; writeln('x+y=', z);
  z:=x-y; writeln('x-y=', z);
  z:=x*y; writeln('x*y=', z);
  z:=x mod y; writeln('x mod y=', z);
  z:=x div y; writeln('x div y=', z);
end.
```

Rezultatele operațiilor +, –, * cu valori întregi trebuie să aparțină mulțimii de valori ale tipului de date `integer`. Dacă programatorul nu acordă atenția cuvenită acestei reguli, apar erori de depășire. Aceste erori vor fi semnalate în procesul compilării sau execuției programului respectiv. Pentru exemplificare, prezentăm programele `P4` și `P5`:

```
Program P4;
 { Eroare de depasire semnalata in procesul compilarii }
var x : integer;
begin
  x:=MaxInt+1; { Eroare, x>MaxInt }
  writeln(x);
end.
```

```
Program P5;
 { Eroare de depasire semnalata in procesul executiei }
var x, y : integer;
begin
  x:=MaxInt;
  y:=x+1; { Eroare, y>MaxInt }
  writeln(y);
end.
```

Prioritățile operațiilor +, –, *, **mod**, **div** vor fi studiate mai tîrziu (*tabelul 3.2*).

Întrebări și exerciții

❶ Care este mulțimea de valori ale tipului de date `integer`? Ce operații se pot face cu aceste valori?
❷ Cînd apar erori de depășire? Cum se depistează aceste erori?
❸ Determinați valoarea constantei `MaxInt` a versiunii PASCAL cu care lucrați dvs.
❹ Se consideră programele:

```
Program P6;
 { Eroare de depasire }
var x : integer;
begin
  x:=-2*MaxInt;
  writeln(x);
end.
```

```
Program P7;
 { Eroare de depasire }
var x, y : integer;
begin
  x:=-MaxInt;
```

```
  y:=x-10;
  writeln(y);
end.
```

Cînd vor fi semnalate erori de depășire: la compilare sau la execuție?

❺ Dați exemple de valori ale variabilelor `x` și `y` din programul `P3` pentru care apar erori de depășire.

2.3. Tipul de date `real`

Mulțimea de valori ale tipului de date `real` este formată din numerele reale care pot fi reprezentate pe calculatorul-gazdă al limbajului.

De exemplu, în versiunea Turbo PASCAL 7.0 domeniul de valori ale tipului `real` este $-1{,}7\cdot10^{38}$, ..., $+1{,}7\cdot10^{38}$, numerele fiind reprezentate cu o precizie de 11–12 cifre zecimale.

În programul ce urmează variabilelor reale `x`, `y` și `z` li se atribuie valorile, respectiv, 1,1, $-6{,}4\cdot10^{8}$ și $90{,}3\cdot10^{-29}$, afișate ulterior pe ecran.

```
Program P8;
 { Date de tip real }
var x, y, z : real;
begin
  x:=1.1;
  y:=-6.14e8;
  z:=90.3e-29;
  writeln('x=', x);
  writeln('y=', y);
  writeln('z=', z);
end.
```

Amintim că la scrierea numerelor reale virgula zecimală este redată prin punct, iar puterea lui 10 – prin factorul de scală (vezi paragraful 1.5.3).

Operațiile care se pot face cu valorile reale sînt +, –, *, / (împărțirea) ș.a.

Operațiile asupra valorilor reale sînt în general aproximative din cauza erorilor de rotunjire. Rezultatele operațiilor în studiu trebuie să aparțină domeniului de valori ale tipului de date `real`. În caz contrar, apar erori de depășire.

Proprietățile operațiilor +, –, * și / pot fi studiate cu ajutorul programului ce urmează:

```
Program P9;
 { Operatii cu date de tipul real }
var x, y, z : real;
begin
  writeln('Introduceti numerele reale x, y:');
  readln(x,y);
```

```
  writeln('x=', x);
  writeln('y=', y);
  z:=x+y; writeln('x+y=', z);
  z:=x-y; writeln('x-y=', z);
  z:=x*y; writeln('x*y=', z);
  z:=x/y; writeln('x/y=', z);
end.
```

În *tabelul 2.1* sînt prezentate datele afișate de programul `P9` (versiunea Turbo PASCAL 7.0) pentru unele valori ale variabilelor `x` și `y`. Se observă că rezultatele operațiilor `x+y` și `x-y` din primele două linii ale *tabelului 2.1* sînt exacte. În cazul valorilor `x` = 1,0, `y` = $1{,}0 \cdot 10^{-11}$ (linia 3 a tabelului în studiu), rezultatul adunării este aproximativ, iar cel al scăderii – exact. Ambele rezultate din linia a patra sînt aproximative. În cazul valorilor `x` = `y` = $1{,}7 \cdot 10^{38}$ (linia 5) are loc o depășire la efectuarea adunării. Pentru valorile `x` = $3{,}1 \cdot 10^{-39}$, `y` = $3{,}0 \cdot 10^{-39}$ (linia 6) rezultatul adunării este exact, iar rezultatul scăderii este aproximativ.

Tabelul 2.1

Rezultatele programului P9

Nr. crt.	x	y	$x + y$	$x - y$
1	1,0	1,0	2.0000000000E+00	0.0000000000E+00
2	1,0	$1{,}0 \cdot 10^{-10}$	1.0000000000E+00	9.9999999999E-01
3	1,0	$1{,}0 \cdot 10^{-11}$	1.0000000000E+00	9.9999999999E-01
4	1,0	$1{,}0 \cdot 10^{-12}$	1.0000000000E+00	1.0000000000E+00
5	$1{,}7 \cdot 10^{38}$	$1{,}7 \cdot 10^{38}$	depăşire	0.0000000000E+00
6	$3{,}1 \cdot 10^{-39}$	$3{,}0 \cdot 10^{-39}$	6.1000000000E-39	0.0000000000E+00

Însumîndu-se, erorile de calcul, proprii tipului de date `real`, pot compromite rezultatele execuției unui program. Evaluarea și, dacă e necesar, suprimarea erorilor semnificative cade în sarcina programatorului.

Prioritățile operațiilor +, –, *, / vor fi studiate mai tîrziu (*tabelul 3.2*).

Întrebări și exerciții

❶ Cum se scriu numerele reale în limbajul PASCAL?

❷ Determinați domeniul de valori ale tipului de date `real` din versiunea PASCAL cu care lucrați. Care este precizia numerelor respective?

❸ Ce operații se pot face cu datele de tip `real`? Sînt oare exacte aceste operații?

❹ Lansați în execuție programul `P9` pentru următoarele valori ale variabilelor `x`, `y`:

a) $x = 2{,}0$; $y = -3{,}0$;

b) $x = 14{,}3 \cdot 10^{2}$; $y = 15{,}3 \cdot 10^{-3}$;

c) $x = 3{,}0$; $y = 2{,}0 \cdot 10^{12}$;

d) $x = 3{,}0$; $y = 2{,}0 \cdot 10^{-12}$;

e) $x = 2{,}9 \cdot 10^{-39}$; $y = 6{,}4 \cdot 10^{-3}$;

f) $x = 7{,}51 \cdot 10^{21}$; $y = -8{,}64 \cdot 10^{17}$;

g) $x = 1{,}0$; $y = 2{,}9 \cdot 10^{-39}$; *h)* $x = 1{,}7 \cdot 10^{38}$; $y = 2{,}9 \cdot 10^{-39}$.

Verificați rezultatele operațiilor respective. Explicați mesajele afișate pe ecran.

❺ Care sînt cauzele erorilor de calcul cu date de tip `real`?

2.4. Tipul de date `boolean`

Tipul de date `boolean` (logic) include valorile de adevăr `false` (fals) și `true` (adevărat). În programul de mai jos variabilei `x` i se atribuie consecutiv valorile `false` și `true`, afișate ulterior pe ecran.

```
Program P10;
 { Date de tip boolean }
var x : boolean;
begin
 x:=false;
 writeln(x);
 x:=true;
 writeln(x);
end.
```

Operațiile predefinite ale tipului de date `boolean` sînt:

`not`	negația (inversia logică, operația logică *NU*);
`and`	conjuncția (produsul logic, operația logică *ȘI*);
`or`	disjuncția (suma logică, operația logică *SAU*).

Tabelele de adevăr ale operațiilor în studiu sînt prezentate în *figura 2.1*.

Proprietățile operațiilor logice **`not`**, **`and`** și **`or`** pot fi cercetate cu ajutorul programului `P11`.

```
Program P11;
 { Operatii cu date de tip boolean }
var x, y, z : boolean;
begin
 x:=false; y:=false;
 writeln('x=', x, 'y=', y);
 z:=not x; writeln('not x = ', z);
 z:=x and y; writeln('x and y = ', z);
 z:=x or y; writeln('x or y = ', z);
 writeln;
 x:=false; y:=true;
 writeln('x=', x, 'y=', y);
 z:=not x; writeln('not x = ', z);
 z:=x and y; writeln('x and y = ', z);
 z:=x or y; writeln('x or y = ', z);
```

```
  writeln;
  x:=true; y:=false;
  writeln('x=', x, 'y=', y);
  z:=not x; writeln('not x = ', z);
  z:=x and y; writeln('x and y = ', z);
  z:=x or y; writeln('x or y = ', z);
  writeln;
  x:=true; y:=true;
  writeln('x=', x, 'y=', y);
  z:=not x; writeln('not x = ', z);
  z:=x and y; writeln('x and y = ', z);
  z:=x or y; writeln('x or y = ', z);
  writeln;
end.
```

x	not x
false	true
true	false

x	y	x and y
false	false	false
false	true	false
true	false	false
true	true	true

x	y	x or y
false	false	false
false	true	true
true	false	true
true	true	true

Fig. 2.1. Tabelele de adevăr ale operaţiilor logice **not**, **and** şi **or**

Spre deosebire de variabilele de tip întreg sau real, valorile curente ale variabilelor `boolean` nu pot fi citite de la tastatură cu ajutorul procedurii-standard `readln`. Din acest motiv, în programul `P11` valorile curente ale variabilelor `x` și `y` sînt date prin atribuire.

Prioritățile operațiilor **`not`, `and`, `or`** vor fi studiate mai tîrziu (*tabelul 3.2*).

Întrebări și exerciții

❶ Numiți mulțimea de valori și operațiile cu date de tip `boolean`.
❷ Memorizați tabelele de adevăr ale operațiilor logice.
❸ Creați un program care afișează pe ecran tabelul de adevăr al operației logice **`not`**.
❹ Elaborați un program care calculează valorile funcției logice $z = x \,\&\, y$ pentru toate valorile posibile ale argumentelor x, y.
❺ Elaborați un program care afișează valorile funcției logice $z = x \vee y$.

2.5. Tipul de date `char`

Mulțimea valorilor acestui tip de date este o mulțime finită și ordonată de caractere. Valorile în studiu se desemnează prin includerea fiecărui caracter între două semne `'` (apostrof), de exemplu, `'A'`, `'B'`,`'C'` etc. Însuși apostroful se dublează, reprezentîndu-se prin `''''`.

În programul ce urmează variabilei `x` de tip `char` i se atribuie consecutiv valorile `'A'`, `'+'` și `''''`, afișate ulterior pe ecran.

```
Program P12;
 { Date de tip char }
var x : char;
begin
 x:='A';
 writeln(x);
 x:='+';
 writeln(x);
 x:='''';
 writeln(x);
end.
```

Valorile curente ale unei variabile de tip `char` pot fi citite de la tastatură cu ajutorul procedurii-standard `readln`. Pentru exemplificare, prezentăm programul `P13`, care citește de la tastatură și afișează pe ecran valori de tipul `char`.

```
Program P13;
 { Citirea si afisarea caracterelor }
var x : char;
```

```
begin
 readln(x); writeln(x);
 readln(x); writeln(x);
 readln(x); writeln(x);
end.
```

Caracterele respective se introduc de la tastatură și se afișează pe ecran fără apostrofurile care le încadrează în textul unui program PASCAL.

De regulă, caracterele unei versiuni concrete a limbajului PASCAL sînt **ordonate** conform tabelului de cod *ASCII* (vezi paragraful 1.4).

Numărul de ordine al oricărui caracter din mulțimea de valori ale tipului `char` poate fi aflat cu ajutorul funcției predefinite `ord`. De exemplu:

1) `ord('A')= 65`

2) `ord('B')= 66`

3) `ord('C')= 67`

ș.a.m.d.

Programul `P14` afișează pe ecran numărul de ordine a patru caractere citite de la tastatură.

```
Program P14;
 { Studierea functiei ord }
var x : char; { caracter }
    i : integer; { numar de ordine }
begin
 readln(x); i:=ord(x); writeln(i);
 readln(x); i:=ord(x); writeln(i);
 readln(x); i:=ord(x); writeln(i);
 readln(x); i:=ord(x); writeln(i);
end.
```

Funcția predefinită `chr` returnează caracterul care corespunde numărului de ordine indicat. Așadar:

1) `chr(65)='A';`

2) `chr(66)='B';`

3) `chr(67)='C'`

ș.a.m.d.

Programul `P15` afișează pe ecran caracterele ce corespund numerelor de ordine citite de la tastatură.

```
Program P15;
 { Studierea functiei chr }
var i : integer; { numar de ordine }
```

```
    x : char; { caracter }
begin
 readln(i); x:=chr(i); writeln(x);
 readln(i); x:=chr(i); writeln(x);
 readln(i); x:=chr(i); writeln(x);
 readln(i); x:=chr(i); writeln(x);
end.
```

Amintim că un set extins *ASCII* include 256 de caractere, numerotate cu 0, 1, 2, ..., 255. Tipul de date `char` se utilizează pentru formarea unor structuri de date mai complexe, în particular, a șirurilor de caractere.

Întrebări și exerciții

❶ Care este mulțimea de valori ale tipului de date `char`?
❷ Cum este ordonată mulțimea de valori ale tipului `char`?
❸ Determinați numerele de ordine ale următoarelor caractere:
- cifrele zecimale;
- literele mari ale alfabetului englez;
- semnele de punctuație;
- operatorii aritmetici și logici;
- caracterele de control și editare;
- literele alfabetului român (dacă sînt implementate pe calculatorul dvs.).

❹ Determinați caracterele care corespund următoarelor numere de ordine:

`77` `109` `79` `111` `42` `56` `91` `123`

❺ Elaborați un program care afișează pe ecran setul de caractere al calculatorului cu care lucrați.

2.6. Tipuri de date *enumerare*

Tipurile `integer`, `real`, `boolean` și `char`, studiate pînă acum, sînt tipuri predefinite, cunoscute oricărui program PASCAL. În completare la tipurile predefinite, programatorul poate defini și utiliza tipuri proprii de date, în particular, tipuri *enumerare*.

Un tip *enumerare* include o mulțime ordonată de valori specificate prin identificatori. Denumirea unui tip de date *enumerare* și mulțimea lui de valori se indică în partea declarativă a programului după cuvîntul-cheie **`type`** (tip).

Exemplu:

```
type Culoare = (Galben, Verde, Albastru, Violet);
     Studii = (Elementare, Medii, Superioare);
     Raspuns = (Nu, Da);
```

Primul identificator din lista de enumerare desemnează cea mai mică valoare, cu numărul de ordine zero. Identificatorul al doilea va avea numărul de ordine unu, al trei-

lea – numărul doi etc. Numărul de ordine al unei valori poate fi aflat cu ajutorul funcției predefinite `ord`.

Exemple:

1) `ord(Galben)= 0`
2) `ord(Verde)= 1`
3) `ord(Albastru)= 2`
4) `ord(Violet)= 3`
5) `ord(Elementare)= 0`
6) `ord(Medii)= 1`

ș.a.m.d.

Programul ce urmează afișează pe ecran numerele de ordine ale valorilor tipului de date `Studii`.

```
Program P16;
 { Tipul de date Studii }
type Studii = (Elementare, Medii, Superioare);
var i : integer; { numar de ordine }
begin
 i:=ord(Elementare); writeln(i);
 i:=ord(Medii); writeln(i);
 i:=ord(Superioare);
 writeln(i);
end.
```

Variabilele de tip *enumerare* se declară cu ajutorul cuvîntului-cheie **`var`**. Ele pot lua numai valori din lista de enumerare a tipului de date cu care sînt asociate.

În programul `P17` variabila `x` ia valoarea `Albastru`; variabila `y` ia valoarea `Nu`. Numerele de ordine ale acestor valori se afișează pe ecran.

```
Program P17;
 { Variabile de tip enumerare }
type Culoare = (Galben, Verde, Albastru, Violet);
     Raspuns = (Nu, Da);
var  x : Culoare; { variabila de tip Culoare }
     y : Raspuns; { variabila de tip Raspuns }
     i : integer; { numar de ordine }
begin
 x:=Albastru;
 i:=ord(x); writeln(i);
 y:=Nu; i:=ord(y); writeln(i);
end.
```

În cazurile în care într-un program PASCAL se definesc mai multe tipuri de date, listele de enumerare nu trebuie să conțină identificatori comuni.

De exemplu, declarația

```
type Studii = (Elementare, Medii, Superioare);
```

```
Grade = (Inferioare, Superioare)
```

este incorectă, întrucît identificatorul `Superioare` apare în ambele liste.

Valorile curente ale variabilelor de tip *enumerare* nu pot fi citite de la tastatură sau afișate pe ecran cu ajutorul procedurilor-standard `readln` și `writeln`. Totuși utilizarea tipurilor de date în studiu permite elaborarea unor programe lizibile, simple și eficiente.

Întrebări și exerciții

❶ Cum se definește un tip de date *enumerare*? Care este mulțimea de valori ale unui tip *enumerare*?

❷ Contează oare ordinea în care apar identificatorii într-o listă de enumerare?

❸ Elaborați un program care afișează pe ecran numerele de ordine ale valorilor următoarelor tipuri de date:

a)
```
Continente = (Europa, Asia, Africa, AmericaDeNord,
              AmericaDeSud, Australia, Antarctida);
```

b)
```
Sex = (Barbat, Femeie);
```

c)
```
PuncteCardinale = (Nord, Sud, Est, Vest);
```

d)
```
Etaje = (Unu, Doi, Trei, Patru, Cinci);
```

❹ Numiți tipul fiecărei variabile din programul `P18`:

```
Program P18;
type Litere = (A, B, C, D, E, F, G);
var x : Litere; y : char; i : integer;
begin
 x:=A; i:=ord(x); writeln(i);
 y:='A'; i:=ord(y); writeln(i);
end.
```

Ce va afișa pe ecran acest program?

❺ Se consideră declarațiile:

```
type Culoare = (Galben, Verde, Albastru, Violet);
     Fundal = (Alb, Negru, Gri);
var  x, y : Culoare;
     z : Fundal;
```

Care din instrucțiunile ce urmează sînt corecte?

a) `x:=Verde`

b) `y:=Negru`

c) `z:=Alb`

d) `x:=Gri`

e) `y:=Gri`

f) `z:=Violet`

g) `x:=Albastru`

h) `y:=Azuriu`

❻ Inserați înainte de cuvîntul-cheie **end** al programului `P17` una din liniile ce urmează:

a) `readln(x);`

b) `writeln(x);`

Explicați mesajele afișate pe ecran în procesul compilării programului modificat.

2.7. Tipuri de date *subdomeniu*

Un tip de date *subdomeniu* include o submulțime de valori ale unui tip deja definit, denumit tip de bază. Tipul de bază trebuie să fie `integer`, `boolean`, `char` sau *enumerare*.

Denumirea unui tip de date *subdomeniu*, valoarea cea mai mică și valoarea cea mai mare (în sensul numărului de ordine) se indică în partea declarativă a programului după cuvîntul-cheie **`type`**.

Exemple:

1)
```
type Indice = 1..10;
     Litera = 'A'..'Z';
     Cifra = '0'..'9';
```

Tipul `Indice` este un subdomeniu al tipului predefinit `integer`. Tipurile `Litera` și `Cifra` sînt subdomenii ale tipului predefinit `char`.

2)
```
type Zi = (L, Ma, Mi, J, V, S, D);
     ZiDeLucru = L..V;
     ZiDeOdihna = S..D;
```

Tipurile `ZiDeLucru` și `ZiDeOdihna` sînt subdomenii ale tipului *enumerare* `Zi`, definit de utilizator.

3)
```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = A..F;
     T3 = C..H;
```

Tipurile `T2` și `T3` sînt subdomenii ale tipului *enumerare* `T1`. Tipurile de bază ale tipurilor de date *subdomeniu* din exemplele în studiu sînt:

	Tip subdomeniu	*Tipul de bază*
1)	`Indice`	`integer`
2)	`Litera`	`char`
3)	`Cifra`	`char`
4)	`ZiDeLucru`	`Zi`
5)	`ZiDeOdihna`	`Zi`
6)	`T2`	`T1`
7)	`T3`	`T1`

Variabilele unui tip de date *subdomeniu* se declară cu ajutorul cuvîntului-cheie **var**. O variabilă de tip *subdomeniu* moștenește toate proprietățile variabilelor tipului de bază, dar valorile ei trebuie să fie numai din intervalul specificat. În caz contrar, este semnalată o eroare și programul se oprește.

Exemplu:

```
Program P19;
 { Valorile variabilelor de tip subdomeniu }
type Indice = 1..10;
     Zi = (L, Ma, Mi, J, V, S, D);
     ZiDeLucru = L..V;
     ZiDeOdihna = S..D;
var  i : Indice;      { valori posibile: 1, 2, ..., 10 }
     z : Zi;          { valori posibile: L, Ma, ..., D }
     zl : ZiDeLucru; { valori posibile: L, Ma, ..., V }
     zo : ZiDeOdihna; { valori posibile: S, D }
begin
 i:=5; i:=11;          { Eroare, i>10 }
 z:=L; zl:=J; zl:=S;   { Eroare, zl>V }
 zo:=S; zo:=V;         { Eroare, zo<S }
 writeln('Sfirsit');
end.
```

Programul P20 demonstrează cum tipul Pozitiv moștenește proprietățile tipului de bază integer.

```
Program P20;
 { Tipul Pozitiv mosteneste proprietatile
   tipului integer }
type Pozitiv = 1..32767;
var x, y, z : Pozitiv;
begin
 writeln('Introduceti numerele pozitive x, y:');
 readln(x,y);
 writeln('x=', x);
 writeln('y=', y);
 z:=x+y; writeln('x+y=', z);
 z:=x-y; writeln('x-y=', z);
 z:=x*y; writeln('x*y=', z);
 z:=x mod y; writeln('x mod y=', z);
 z:=x div y; writeln('x div y=', z);
end.
```

Se observă că operațiile +, -, *, **mod** și **div** ale tipului de bază integer sînt moștenite de tipul *enumerare* Pozitiv. Dar, spre deosebire de variabilele de tip integer, variabilele de tip Pozitiv nu pot lua valori negative.

Utilizarea tipurilor de date *subdomeniu* face programele mai intuitive și simplifică verificarea lor. Subliniem faptul că în limbajul PASCAL nu este permisă definirea unui subdomeniu al tipului `real`, deoarece valorile acestuia nu au numere de ordine.

Întrebări și exerciții

❶ Cum se definește un tip *subdomeniu*? Care este mulțimea de valori ale unui tip *subdomeniu*?

❷ Numiți tipul de bază al fiecărui tip *subdomeniu*:

```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = -60..60;
     T3 = 5..9;
     T4 = '5'..'9';
     T5 = A..E;
     T6 = 'A'..'E';
```

❸ Ce valori poate lua fiecare variabilă din următoarele declarații:

```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = 1..9;
     T3 = 6..15;
     T4 = -100..100;
     T5 = 'A'..'Z';
     T6 = '0'..'9';
     T7 = C..F;
var  i : integer;
     j : T2;
     m : T4;
     p : T5;
     q : char;
     r : T6;
     s : T1;
     t : T7;
```

Numiți tipul de bază al fiecărui tip *subdomeniu*. Indicați setul de operații moștenit de la tipul de bază.

❹ Care din următoarele definiții sînt corecte? Argumentați răspunsul.

a)
```
type Lungime = 1.0e-2..1.0;
     Latime = 1.0e-2..0.5;
```

b)
```
type Indice = 1..10;
     Abatere = +5..-5;
     Deviere = -10..+10;
```

c)
```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = C..H;
     T3 = F..B;
```

d)
```
type Luni = (Ianuarie, Februarie, Martie, Aprilie, Mai,
             Iunie, Iulie, August, Septembrie,
             Octombrie, Noiembrie, Decembrie);
```

```
LuniDeIarna = (Decembrie..Februarie);
LuniDePrimavara = (Martie..Mai);
LuniDeVara=(Iunie..August);
LuniDeToamna=(Septembrie..Noiembrie);
```

❺ Se consideră următorul program:

```
Program P21;
 type Indice=1..10;
 var  i, j, k, m : Indice;
begin
 writeln('Introduceti indicii i, j:');
 readln(i, j);
 k:=i+j; writeln('k=', k);
 m:=i-j; writeln('m=', m);
end.
```

Pentru care valori ale variabilelor `i`, `j` se vor declanșa erori de execuție?

a) `i=3, j=2;`
b) `i=7, j=4;`
c) `i=4, j=7;`
d) `i=6, j=3;`
e) `i=2, j=2;`
f) `i=3, j=11;`
g) `i=8, j=4;`
h) `i=5, j=3.`

❻ Se consideră programul `P20`. După lansarea în execuție utilizatorul introduce `x=1`, `y=2`. Evident, `x-y=-1`. Întrucît valoarea –1 nu aparține tipului de date `Pozitiv`, la execuția instrucțiunii

```
z:=x-y
```

va surveni o eroare.
Indicați instrucțiunile la execuția cărora se vor declanșa erori, dacă:

a) `x=1000, y=1000;`
b) `x=1000, y=1001;`
c) `x=1001, y=1000;`
d) `x=30000, y=30000;`
e) `x=1, y=2;`
f) `x=1000, y=100;`
g) `x=0, y=1;`
h) `x=1, y=0.`

2.8. Generalități despre tipurile ordinale de date

Tipurile de date `integer`, `boolean`, `char`, *enumerare* și *subdomeniu* se numesc **tipuri ordinale**. Fiecare valoare a unui tip ordinal are un număr de ordine, definit după cum urmează:

1) numărul de ordine al unui număr de tip `integer` este însuși numărul considerat;

2) valorile de adevăr `false` și `true` ale tipului `boolean` au numerele de ordine, respectiv, 0 și 1;

3) numărul de ordine al unui caracter (tipul `char`) este dat de poziția lui în tabelul de codificare, obișnuit *ASCII*;

4) numărul de ordine al unei valori de tip *enumerare* este dat de poziția ei în lista de enumerare. De remarcat că valorile unei liste sînt numerotate prin 0, 1, 2, ... ș.a.m.d.;

5) valorile unui tip *subdomeniu* moștenesc numerele de ordine de la tipul de bază.

Numărul de ordine al unei valori de tip ordinal poate fi aflat cu ajutorul funcției predefinite `ord`.

Programul `P22` afișează pe ecran numerele de ordine ale valorilor `-32`, `true`, `'A'`, `A` și `B`.

```
Program P22;
 { Numerele de ordine ale valorilor de tip ordinal }
type T1 = (A, B, C, D, E, F, G, H);
     T2 = B..G;
begin
 writeln(ord(-32));    { -32 }
 writeln(ord(true));   { 1 }
 writeln(ord('A'));    { 65 }
 writeln(ord(A));      { 0 }
 writeln(ord(B));      { 1 }
end.
```

Asupra valorilor oricărui tip ordinal de date sînt permise operațiile relaționale cunoscute:

`<`	mai mic;
`<=`	mai mic sau egal;
`=`	egal;
`>=`	mai mare sau egal;
`>`	mai mare;
`<>`	diferit.

Rezultatul unei operații relaționale este de tip `boolean`, `false` sau `true`, în funcție de numerele de ordine ale valorilor operanzilor.

De exemplu, în prezența declarațiilor

```
type Culoare = (Galben, Verde, Albastru, Violet);
```

rezultatul operației

```
Verde < Violet
```

este `true`, deoarece `ord(Verde)=1`, `ord(Violet)=3` și 1 este mai mic decît 3.

Rezultatul operației

```
Galben > Violet
```

este `false`, fiindcă `ord(Galben)= 0`, `ord(Violet)=3`, iar 0 nu este mai mare decît 3.

Programul ce urmează afișează pe ecran rezultatele operațiilor relaționale pentru valorile `Verde` și `Violet` ale tipului de date `Culoare`.

```
Program P23;
 { Operatii relationale asupra valorilor de tip ordinal }
type Culoare = (Galben, Verde, Albastru, Violet);
begin
 writeln(Verde<Violet);    { true }
 writeln(Verde<=Violet);   { true }
 writeln(Verde=Violet);    { false }
 writeln(Verde>=Violet);   { false }
 writeln(Verde>Violet);    { false }
 writeln(Verde<>Violet);   { true }
end.
```

Pentru tipurile ordinale de date există funcțiile predefinite `pred` (*predecesor*) și `succ` (*succesor*).

Predecesorul valorii ordinale cu numărul de ordine *i* este valoarea cu numărul de ordine *i*–1. *Succesorul* valorii ordinale în studiu este valoarea cu numărul de ordine *i*+1.

De exemplu, pentru valorile tipului ordinal de date `Culoare` obținem:

```
pred(Verde) = Galben;

succ(Verde) = Albastru;

pred(Albastru) = Verde;

succ(Albastru) = Violet.
```

Valoarea cea mai mică nu are predecesor, iar valoarea cea mai mare nu are succesor.

Programul `P24` afișează pe ecran predecesorii și succesorii valorilor ordinale `'B'`, `0` și `'0'`.

```
Program P24;
 { Predecesorii si succesorii valorilor ordinale }
begin
 writeln(pred('B'));   { 'A' }
 writeln(succ('B'));   { 'C' }
 writeln(pred(0));     { -1 }
 writeln(succ(0));     { 1 }
 writeln(pred('0'));   { '/' }
 writeln(succ('0'));   { '1' }
end.
```

Se observă că predecesorii și succesorii valorilor ordinale `0` (tip `integer`) și `'0'` (tip `char`) nu coincid, deoarece tipurile valorilor respective diferă.

Subliniem faptul că tipul de date `real` nu este un tip ordinal. Prin urmare asupra valorilor reale nu pot fi aplicate funcțiile `ord` (numărul de ordine), `pred` (predecesor) și `succ` (succesor). Nerespectarea acestei reguli va produce erori.

Întrebări și exerciții

❶ Numiți tipurile ordinale de date. Care sînt proprietățile lor comune?

❷ Cum se definesc numerele de ordine ale valorilor unui tip ordinal de date?

❸ Ce va afișa pe ecran programul ce urmează?

```
Program P25;
type Zi = (L, Ma, Mi, J, V, S, D);
var z1, z2 : Zi;
begin
 z1:=Ma;
 writeln(ord(z1));
 z2:=pred(z1);
 writeln(ord(z2));
 z2:=succ(z1);
 writeln(ord(z2));
 z1:=Mi; z2:=V;
 writeln(z1<z2);
 writeln(z1>z2);
 writeln(z1<>z2);
end.
```

❹ Excludeți din programul de mai jos linia care conține o eroare:

```
Program P26;
 { Eroare }
var i : integer;
begin
 i:=MaxInt;
 writeln(pred(i));
 writeln(succ(i));
end.
```

Ce rezultate vor fi afișate pe ecran după execuția programului modificat?

❺ Comentați programul ce urmează:

```
Program P27;
 { Eroare }
var i : integer; x : real;
begin
 i:=1; x:=1.0;
 writeln(ord(i));
 writeln(ord(x));
 writeln(pred(i));
 writeln(pred(x));
```

```
  writeln(succ(i));
  writeln(succ(x));
end.
```

Excludeți liniile care conțin erori. Ce rezultate vor fi afișate pe ecran după execuția programului modificat?

2.9. Definirea tipurilor de date

Limbajul PASCAL oferă utilizatorului tipurile predefinite de date `integer`, `real`, `boolean`, `char` ș.a. Dacă este necesar, programatorul poate defini tipuri proprii de date, de exemplu, *enumerare* și *subdomeniu*.

Denumirea unui tip de date și mulțimea lui de valori se definesc cu ajutorul următoarelor unități gramaticale:

<*Tipuri* > ::= **type** <*Definiție tip*>**;** { <*Definiție tip*>**;**}
<*Definiție tip*> ::= <*Identificator*> = <*Tip*>

<*Tip*> ::= <*Identificator*> | <*Tip enumerare*> | <*Tip subdomeniu*> | <*Tip tablou*> | <*Tip articol*> | <*Tip mulțime*> | <*Tip fișier*> | <*Tip referință*>

<*Tip enumerare*> ::= (<*Identificator*> {**,** <*Identificator*>})

<*Tip subdomeniu*> ::= <*Constantă*>**..**<*Constantă*>

Diagramele sintactice corespunzătoare sînt prezentate în *figura 2.2.*

Exemple:

1)
```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = B..F;
     T3 = C..H;
```

2)
```
type Pozitiv = 1..MaxInt;
     Natural = 0..MaxInt;
     Negativ = -MaxInt..-1;
```

3)
```
type Abatere = -10...+10;
     Litera = 'A'..'Z';
     Cifra = '0'..'9';
```

Clasificarea tipurilor de date ale limbajului PASCAL este prezentată în *figura 2.3*. Tipurile studiate deja sînt evidențiate printr-un fundal mai închis.

În anumite construcții ale limbajului PASCAL variabilele și constantele trebuie să fie de tipuri identice sau compatibile.

Două tipuri sînt **identice**, dacă ele au fost definite cu același nume de tip.

De exemplu, fie

```
type T4 = integer;
     T5 = integer;
```

Aici tipurile `integer`, `T4` și `T5` sînt identice.

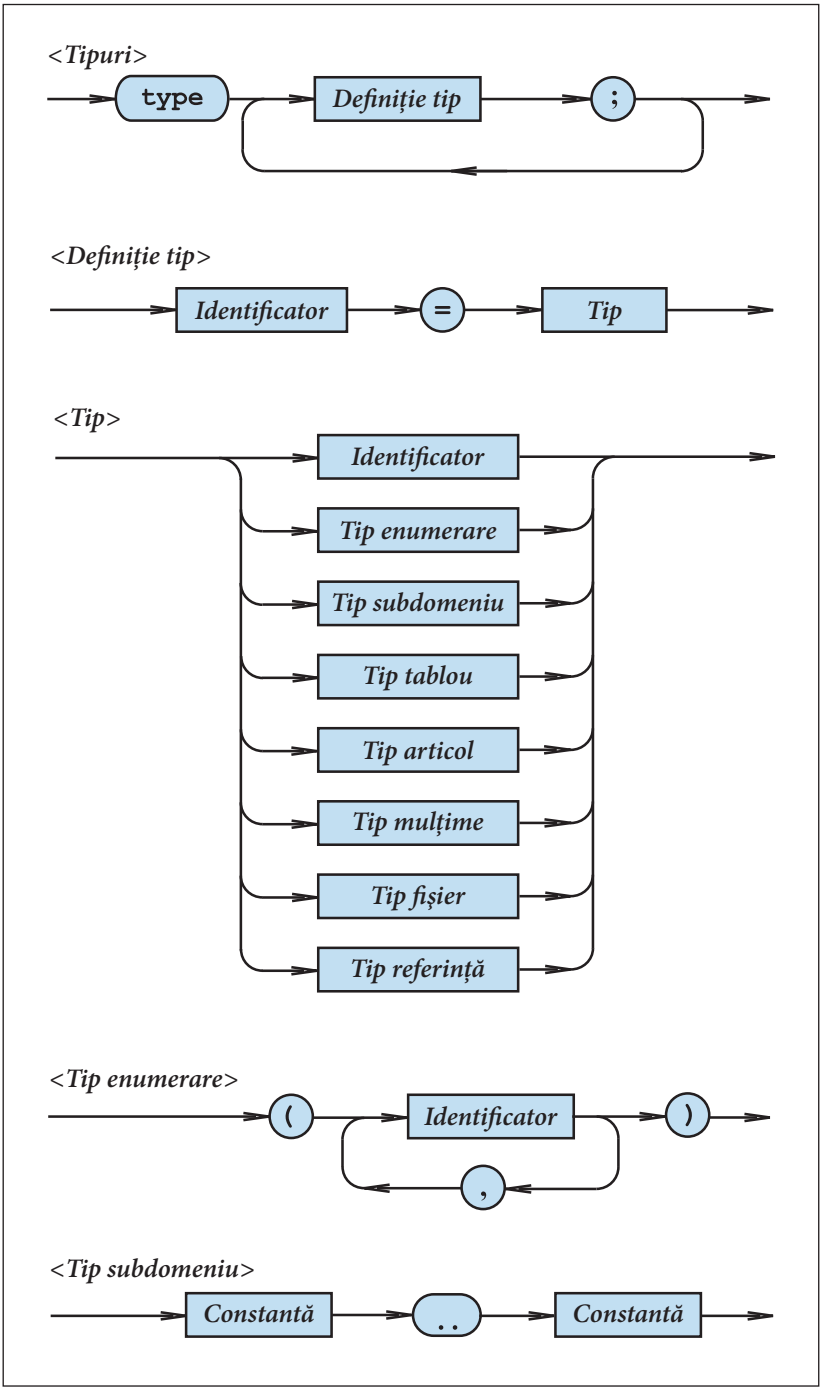


Fig. 2.2. Diagramele sintactice pentru definirea tipurilor de date

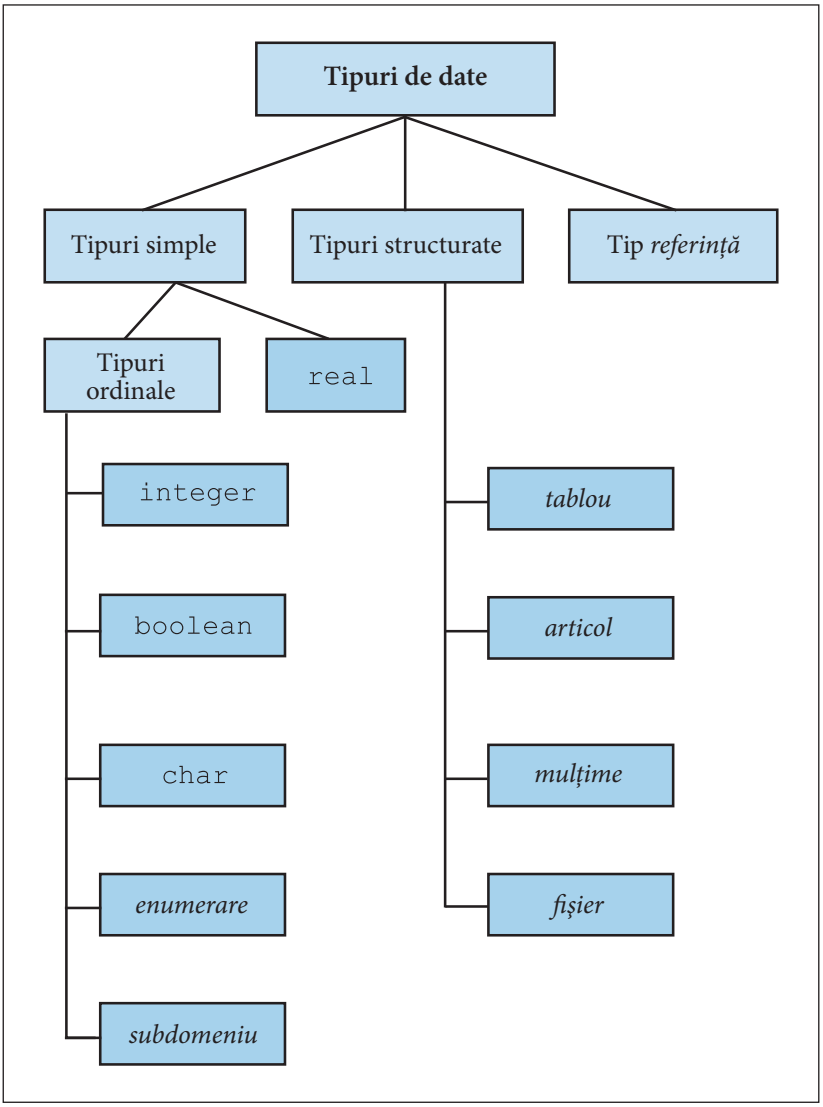


Fig. 2.3. Clasificarea tipurilor de date

Două tipuri sînt identice și atunci cînd sînt definite cu nume diferite, dar aceste nume sînt echivalente prin tranzitivitate.

De exemplu, fie

```
type T6 = real;
     T7 = T6;
     T8 = T7;
```

Aici `real`, `T6`, `T7` și `T8` sînt tipuri identice.

Două tipuri sînt **compatibile** atunci cînd este adevărată cel puțin una din următoarele afirmații:

1) cele două tipuri sînt identice;

2) un tip este un subdomeniu al celuilalt tip;
3) ambele tipuri sînt subdomenii ale aceluiași tip de bază.
De exemplu, în prezența declarațiilor

```
type Zi = (L, Ma, Mi, J, V, S, D);
     ZiDeLucru = (L, Ma, Mi, J, V);
     ZiDeOdihna = (S, D);
     Culoare = (Galben, Verde, Albastru, Violet);
```

tipurile `Zi`, `ZiDeLucru`, `ZiDeOdihna` sînt compatibile. Tipurile `Zi` și `Culoare` sînt tipuri incompatibile. Prin urmare sînt admise operațiile relaționale:

```
L < D        Mi <> D        Verde <> Violet
```

etc. și nu sînt admise operațiile de tipul:

```
L < Violet        Verde = V        S <> Albastru
```

ș.a.m.d.

În completare la tipurile de date definite de utilizator explicit cu ajutorul cuvîntului-cheie **`type`**, într-un program PASCAL pot fi definite și tipuri anonime (fără denumire).

Un **tip anonim** se definește implicit într-o declarație de variabile.

Exemplu:

```
var i : 1..20;
    s : (Alfa, Beta, Gama, Delta);
    t : Alfa..Gama;
```

Se observă că tipul *subdomeniu* `1..20`, tipul *enumerare* `(Alfa, Beta, Gama, Delta)` și tipul *subdomeniu* `Alfa..Gama` nu au denumiri proprii.

De regulă, tipurile anonime se utilizează în programele cu un număr mic de variabile.

Întrebări și exerciții

❶ Se consideră următorul program:

```
Program P28;
type T1 = -100..100;
     T2 = 'A'..'H';
     T3 = (A, B, C, D, E, F, G, H);
     T4 = A..E;
     T5 = integer;
     T6 = real;
     T7 = char;
     T8 = boolean;
var  i : T1;
     j : T5;
     k : T2;
     m : T3;
```

```
      n : T4;
      p : real;
      q : T6;
      r : char;
      s : T7;
      t : boolean;
      y : real;
      z : T8;
begin
  { calcule ce utilizeaza }
  { variabilele in studiu }
  writeln('Sfirsit');
end.
```

Precizați tipurile de date ale programului. Ce valori poate lua fiecare variabilă din acest program? Care tipuri sînt compatibile?

❷ Indicați pe diagramele sintactice din *figura 2.2* drumurile care corespund definițiilor tipurilor de date din programul `P28`.

❸ Precizați tipurile anonime de date din următorul program:

```
Program P29;
type  T1 = integer;
      T2 = -150..150;
      T3 = 1..5;
var   i : T1;
      j : T2;
      k : T3;
      m : 1..5;
      n : (Unu, Doi, Trei, Patru, Cinci);
begin
  { calcule ce utilizeaza }
  { variabilele in studiu }
  writeln('Sfirsit')
end.
```

Ce valori poate lua fiecare variabilă din acest program?

❹ Se consideră următoarele declarații:

```
type  T1 = boolean;
      T2 = T1;
      T3 = T2;
      T4 = T3;
var x : T4;
```

Precizați valorile pe care le poate lua variabila `x` și operațiile tipului corespunzător de date.

❺ Cînd două tipuri de date sînt identice? Dați exemple.

❻ Cînd două tipuri de date sînt compatibile? Dați exemple.

❼ Se consideră declarațiile:

```
type  T1 = integer;
      T2 = T1;
```

```
T3 = -5..+5;
T4 = T3;
T5 = -10..+10;
T6 = (A, B, C, D, E, F, G, H);
T7 = A..D;
T8 = E..H;
T9 = 'A'..'D';
T10 = 'E' ..'H';
```

Precizați tipurile identice și tipurile compatibile de date.

2.10. Declarații de variabile

Cunoaștem deja că fiecare variabilă care apare într-un program PASCAL în mod obligatoriu se asociază cu un anumit tip de date. Pentru aceasta se utilizează următoarele construcții gramaticale:

<*Variabile*> ::= **var** <*Declarație variabile*>**;** {<*Declarație variabile*>**;**}

<*Declarație variabile*> ::= <*Identificator*> {**,** <*Identificator*>} **:** <*Tip*>

Diagramele sintactice ale unităților gramaticale în studiu sînt prezentate în *figura 2.4.*

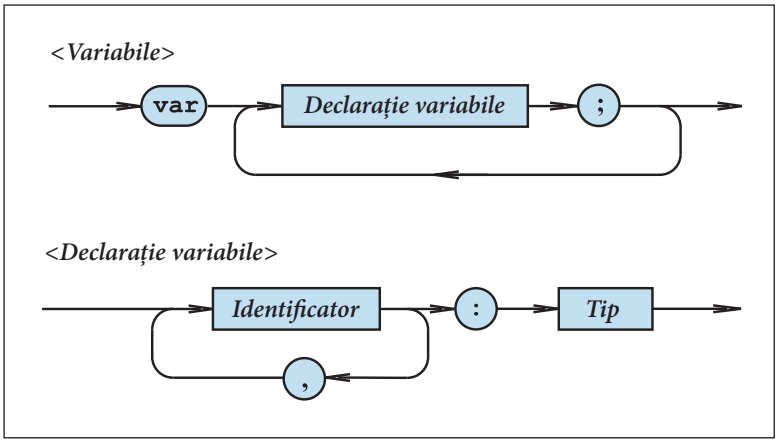


Fig. 2.4. Diagramele sintactice <*Variabile*>, <*Declarație variabile*>

Declarațiile de variabile pot utiliza tipuri predefinite de date (`integer`, `real`, `char`, `boolean` ș.a.) și tipuri definite de utilizator (*enumerare, subdomeniu* etc.).

Exemple:

1)
```
var  i, j : integer;
     x : real;
     p : boolean;
     r, s : char;
```

2)
```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = C..F;
     T3 = 1..10;
     T4 = 'A'..'Z';
var  x, y, z : real;
     r, s : char;
     i, j : integer;
     k : T3;
     p : T1;
     q : T2;
     v : T4;
```

3)
```
type Zi = (L, Ma, Mi, J, V, S, D);
var  x, y : real;
     z : Zi;
     zl : L..V;
     m, n : 1..10;
```

Menționăm că în ultimul exemplu tipul variabilelor `zl`, `m` și `n` este definit direct în declarațiile de variabile. Prin urmare variabilele în studiu aparțin unor tipuri anonime de date.

Întrebări și exerciții

❶ Determinați tipul variabilelor din următorul program:

```
Program P30;
type T1 = integer;
     Studii = (Elementare, Medii, Superioare);
     T2 = real;
     Culoare = (Galben, Verde, Albastru, Violet);
var  x : real;
     y : T1;
     i : integer;
     j : T2;
     p : boolean;
     c : Culoare;
     s : Studii;
     q : Galben..Albastru;
     r : 1..9;
begin
  { calcule în care se utilizeaza }
  { variabilele in studiu }
  writeln('Sfirsit');
end.
```

Precizați valorile pe care le poate lua fiecare variabilă și operațiile tipului corespunzător de date.

❷ Indicați pe diagramele sintactice din *figura 2.4* drumurile care corespund declarațiilor de variabile din programul `P30`.

❸ Ce mesaje vor fi afișate pe ecran pe parcursul compilării următorului program?

```
Program P31;
var i, j : integer;
begin
  i:=1; j:=2; k:=i+j;
  writeln('k=', k);
end.
```

❹ Cum se declară variabilele unui tip anonim de date?

2.11. Definiții de constante

Se știe că valorile unui tip de date pot fi referite prin variabile și constante. Pentru a face programele mai lizibile și ușor de modificat, limbajul PASCAL permite reprezentarea constantelor prin denumiri simbolice. Identificatorul care reprezintă o constantă se numește *nume de constantă* sau, pur și simplu, *constantă*. Peste tot în program, unde apare un astfel de nume, el va fi înlocuit cu valoarea corespunzătoare.

Definirea numelor de constante se face cu ajutorul următoarelor construcții gramaticale:

<*Constante*> ::= **const** <*Definiție constantă*>; { <*Definiție constantă*>;}

<*Definiție constantă*> ::= <*Identificator*> = <*Constantă*>

<*Constantă*> ::= [+ | −] <*Număr fără semn*> | [+ | −] <*Nume de constantă*> | <*Șir de caractere*>

Diagramele sintactice ale unităților gramaticale în studiu sînt prezentate în *figura 2.5.*

Exemple:

1)
```
const a = 10;
      b = 9.81;
      c = '*';
      t = 'TEXT';
```

2)
```
const NumarCaractere = 60;
      LungimePagina = 40;
```

3)
```
const n = 10;
      m = 20;
      Pmax = 2.15e+8;
      Pmin = -Pmax;
      S = 'STOP';
```

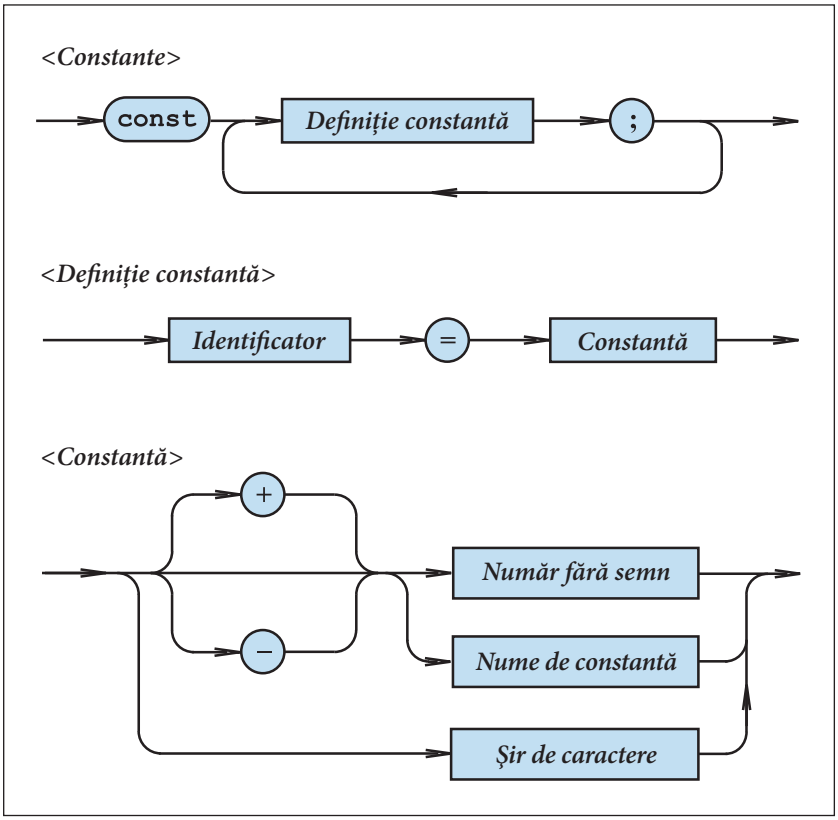


Fig. 2.5. Diagramele sintactice <*Constante*>, <*Definiție constantă*> şi <*Constantă*>

Spre deosebire de variabile, tipul cărora se indică explicit în declarațiile de variabile, tipul constantelor se indică implicit prin forma lor textuală. În exemplele de mai sus tipul constantelor este:

`a, NumarCaractere, LungimePagina, n, m` – *integer;*
`b, Pmax, Pmin` – *real;*
`c` – *char;*
`t, S` – *şir de caractere.*

În programul `P32` se definesc constantele `Nmax`, `Nmin`, `Pi`, `Separator`, `Indicator` şi `Mesaj`. Valorile acestor constante se afişează pe ecran.

```
Program P32;
 { Definitii de constante }
const Nmax = 40;          { Constanta de tip integer }
      Nmin = -Nmax;       { Constanta de tip integer }
      Pi = 3.14;          { Constanta de tip real }
      Separator = '\';    { Constanta de tip char }
      Indicator = TRUE;   { Constanta de tip boolean }
      Mesaj = 'Verificati imprimanta'; { Sir de caractere }
```

```
begin
 writeln(Nmax);
 writeln(Nmin);
 writeln(Pi);
 writeln(Separator);
 writeln(Indicator);
 writeln(Mesaj);
 { calcule in care se utilizeaza }
 { constantele in studiu }
end.
```

În mod obișnuit, datele constante ale unui program, de exemplu, numărul de linii ale unui tabel, numărul π, accelerația căderii libere *g* ș.a.m.d., se includ în definiții de constante. Acest fapt permite modificarea constantelor, fără a schimba programul în întregime.

În programul `P33` lungimea *L* și aria cercului *S* se calculează conform formulelor:

$$L=2\pi r;\ S=\pi r^2,$$

unde *r* este raza cercului. Numărul π este reprezentat prin constanta `Pi=3.14`.

```
Program P33;
 { Lungimea si aria cercului }
const Pi = 3.14;
var L, S , r : real;
begin
 writeln('Introduceti raza r:');
 readln(r);
 L:=2*Pi*r;
 writeln('Lungimea L=', L);
 S:=Pi*r*r;
 writeln('Aria S=', S);
end.
```

Dacă utilizatorul are nevoie de rezultate mai exacte, se modifică numai linia a treia a programului `P33`:

```
const Pi = 3.141592654;
```

Evident, restul programului rămîne neschimbat.

Spre deosebire de variabile, valorile constantelor nu pot fi modificate prin atribuire sau operații de citire. Nerespectarea acestei reguli va produce erori de compilare.

Întrebări și exerciții

❶ Determinați tipul următoarelor constante:

a)
```
const a = 29.1;
      b = TRUE;
      c = 18;
```

b)
```
const t = 'F';
      q = '1';
      c = '18';
```

c)
```
const  d = -16.82e-14;
       f = -d;
       i = 15;
       j = '15';
       n = '-d';
```

d)
```
const  x = 65;
       q = FALSE;
       y = -x;
       m = 'PAUZĂ';
       z = -y;
```

❷ Elaborați un program care afișează pe ecran valorile constantelor din exercițiul 1.

❸ Indicați pe diagramele sintactice din *figura 2.5* drumurile care corespund definițiilor de constante din programul `P32`.

❹ Inserați între cuvintele-cheie **`begin`** și **`end`** ale programului `P32` una din liniile ce urmează:

a)
```
Nmax:=10;
```

b)
```
readln(Nmax);
```

Explicați mesajele afișate pe ecran pe parcursul compilării programului modificat.

❺ Se consideră programul:

```
Program P34;
const t = '1';
      s = -t;
begin
      writeln(s);
end.
```

Ce mesaje vor fi afișate pe ecran în procesul compilării?

❻ Ce rezultate se vor afișa după execuția următorului program:

```
Program P35;
const i = 1;
      j = i;
      k = j;
var x, y, z : integer;
begin
 x:=i+1; writeln(x);
 y:=j+2; writeln(y);
 z:=k+3; writeln(z);
end.
```

❼ Se consideră programul:

```
Program P36;
const a = 1;
      b = 2;
      c = 3;
      d = 4;
var i, j, k : integer;
begin
 i:=a+1; writeln(i);
```

```
  j:=b+1; writeln(j);
  k:=c+1; writeln(k);
  d:=a+1; writeln(d);
end.
```

Explicați mesajele afișate pe ecran.

❽ Excludeți din programul ce urmează linia care conține o eroare.

```
Program P37;
const a = 1;
var i, j : integer;
begin
 writeln('Introduceti i=');
 readln(i); j:=i+a;
 writeln(j);
 writeln('Introduceti a=');
 readln(a);
 j:=i+a;
 writeln(j);
end.
```

Ce rezultate vor fi afișate pe ecran după execuția programului modificat?

Test de autoevaluare nr. 2

1. Explicați semnificația termenului *tip de date*. Numiți cel puțin două tipuri de date și dați câteva exemple de astfel de date.

2. Cum se reprezintă datele într-un program PASCAL? Explicați termenii *mărime*, *variabilă* și *constantă*. Care este diferența dintre variabile și constante?

3. Indicați tipul datelor din programul de mai jos:

```
Program TA2;
 { Tipuri de date simple }
var i, j : integer;
    a, b, c : real;
    s : char;
    p : boolean;
begin
  i:=5; j:=i+9;
  writeln(i); writeln(j);
  a:=1.0; b:=1.0e-01; c:=-2.001;
  writeln(a); writeln(b); writeln(c);
  s:='A'; writeln(s);
  p:=true; writeln(p);
end.
```

Exemplu: `i` – variabilă de tip `integer`; `a` – variabilă de tip `real`; `5` – constantă de tip `integer` ș.a.m.d.

4. Care este mulțimea de valori ale tipului de date `integer`? Ce operații se pot face cu aceste valori?

5. Se consideră următorul program PASCAL:

```
Program TA3;
 { Erori de depasire }
var x, y, z : integer;
begin
  writeln('Introduceti numerele intregi x, y:');
  readln(x, y);
  z:=x*y; writeln('x*y=', z);
end.
```

Dați exemple de valori ale variabilelor `x` și `y` din programul `TA3` pentru care apar erori de depășire.

6. Care este mulțimea de valori ale tipului de date `real`? Ce operații se pot face cu aceste valori? Sînt oare exacte aceste operații?

7. Se consideră următorul program PASCAL:

```
Program TA4;
 { Erori de depasire }
var x, y, z : real;
begin
  writeln('Introduceti numerele reale x, y:');
  readln(x, y);
  z:=x*y; writeln('x*y=', z);
end.
```

Dați exemple de valori ale variabilelor `x` și `y` din programul `TA4` pentru care apar erori de depășire.

8. Care este mulțimea de valori ale tipului de date `boolean`? Ce operații se pot face cu aceste valori?

9. Alcătuiți tabelele de adevăr ale operațiilor logice **`not`**, **`and`** și **`or`**.

10. Indicați eroarea din programul `TA5`:

```
Program TA5;
 { Eroare }
var p, q, r : boolean;
begin
  writeln('Introduceti valorile logice p, q:');
  readln(p, q);
  r:=p and q;
```

```
  writeln(q);
end.
```

11. Care este mulțimea de valori ale tipului de date `char`? Cum este ordonată această mulțime? Ce operații se pot face cu valorile de tip `char`?

12. Elaborați un program care afișează pe ecran numerele de ordine ale cifrelor zecimale.

13. Care este mulțimea de valori ale unui tip de date *enumerare*? Cum este ordonată această mulțime? Ce operații se pot face cu astfel de valori?

14. Creați un program care afișează pe ecran numerele de ordine ale valorilor următoarelor tipuri de date *enumerare*:

a)
```
FunctiaOcupata = (Muncitor, SefDeEchipa, Maistru,
                  SefDeSantier, Director);
```

b)
```
StareaCivila = (Casatorit, Necasatorit);
```

15. Cum se definește un tip de date *subdomeniu*? Care este mulțimea de valori ale unui tip de date *subdomeniu*? Ce operații se pot face cu astfel de valori?

16. Ce valori poate lua fiecare variabilă din următoarele declarații:

```
type  T1 = 'A'..'Z';
      T2 = 1..9;
      T3 = '0'..'9';
      T4 = (Alfa, Beta, Gama, Delta);
var   p : T1;
      q : T2;
      r : T3;
      s : char;
      t : integer;
      u : T4;
```

Numiți tipul de bază al fiecărui tip de date *subdomeniu*.

17. Numiți tipurile ordinale de date. Care sînt proprietățile lor comune?

18. Ce va afișa pe ecran programul ce urmează?

```
Program TA6;
type Culoare = (Galben, Verde, Albastru, Violet);
begin
  writeln(pred('Z'));
  writeln(succ('D'));
  writeln(pred(-5));
  writeln(succ(9));
  writeln(ord(Verde));
  writeln(ord(Violet));
end.
```

19. Ce va afișa pe ecran programul ce urmează?

```
Program TA7;
type Nivel = (A, B, C, D, E, F, G);
var n, m : Nivel;
begin
  n:=B; writeln(ord(n));
  m:=pred(n); writeln(ord(m));
  m:=succ(n); writeln(ord(m));
  n:=C; m:=E;
  writeln(n<m);
  writeln(n>m);
  writeln(n<>m);
end.
```

20. Explicați următorii termeni: tipuri identice, tipuri compatibile, tipuri anonime.

21. Se consideră declarațiile:

```
type  T1 = integer;
      T2 = T1;
      T3 = -5..+5;
      T4 = T3;
      T5 = -10..+10;
      T6 = (A, B, C, D, E, F, G, H);
      T7 = A..D;
      T8 = E..H;
      T9 = 'A'..'D';
      T10 = 'E' ..'H';
var   x : 1..100;
      y : (Alfa, Beta, Gama, Delta);
```

Precizați tipurile identice, tipurile compatibile și tipurile anonime de date.

22. Determinați tipul următoarelor constante:

```
const  Alfa = 5;
       Beta = 12.485;
       Indicator = true;
       Mesaj = 'Eroare de executie';
       Semn = '+';
       Inscris = '12.485';
```

23. Într-un algoritm se utilizează variabilele întregi *i*, *j*, variabilele reale *x*, *y*, constantele 3,14 și 9,8. Scrieți declarațiile PASCAL pentru variabilele și constantele în studiu.

Capitolul 3

INSTRUCȚIUNI

3.1. Conceptul de acțiune

Conform **conceptului de acțiune** realizat în limbajul PASCAL, calculatorul reprezintă un executant, mediul de lucru al căruia este format din mulțimea tuturor variabilelor și constantelor declarate în programul respectiv. În procesul derulării programului, executantul efectuează asupra mărimilor din mediul de lucru anumite acțiuni (operații), de exemplu: adunarea sau scăderea, citirea de la tastatură sau afișarea pe ecran etc. În urma acestor acțiuni valorile variabilelor pot fi schimbate, iar cele ale constantelor – nu.

Operațiile necesare pentru a prelucra datele unui program și ordinea executării lor se definesc cu ajutorul **instrucțiunilor**. Există două categorii de instrucțiuni:

1) instrucțiuni simple;
2) instrucțiuni structurate.

Instrucțiunile simple nu conțin alte instrucțiuni. Instrucțiunile simple sînt:
– instrucțiunea de atribuire;
– instrucțiunea de apel de procedură;
– instrucțiunea de salt necondiționat;
– instrucțiunea de efect nul.

Instrucțiunile structurate sînt construite din alte instrucțiuni. Instrucțiunile structurate sînt:
– instrucțiunea compusă;
– instrucțiunile condiționale **`if`** și **`case`**;
– instrucțiunile iterative **`for`**, **`while`** și **`repeat`**;
– instrucțiunea **`with`**.

Instrucțiunile **`if`** și **`case`** se utilizează pentru programarea algoritmilor cu ramificări, iar instrucțiunile **`for`**, **`while`** și **`repeat`** – pentru programarea algoritmilor repetitivi. Amintim că algoritmii repetitivi se folosesc pentru descrierea unor prelucrări care trebuie executate de mai multe ori, iar cei cu ramificări – pentru a selecta prelucrările dorite în funcție de condițiile din mediul de lucru al executantului.

În cadrul unui program instrucțiunile pot fi prefixate de etichete. Etichetele pot fi referite în instrucțiunile de salt necondiționat **`goto`**. Amintim că eticheta este un număr întreg fără semn (vezi paragraful 1.5.5).

Diagrama sintactică <*Instrucțiune*> este prezentată în *figura 3.1*. Menționăm că eticheta se separă de instrucțiunea propriu-zisă prin simbolul "`:`" (două puncte).

Într-un program PASCAL scrierea instrucțiunilor pe linii nu este limitată, o instrucțiune poate ocupa una sau mai multe linii, sau într-o linie pot fi mai multe instrucțiuni. Ca separator de instrucțiuni se folosește simbolul "`;`" (punct și virgulă).

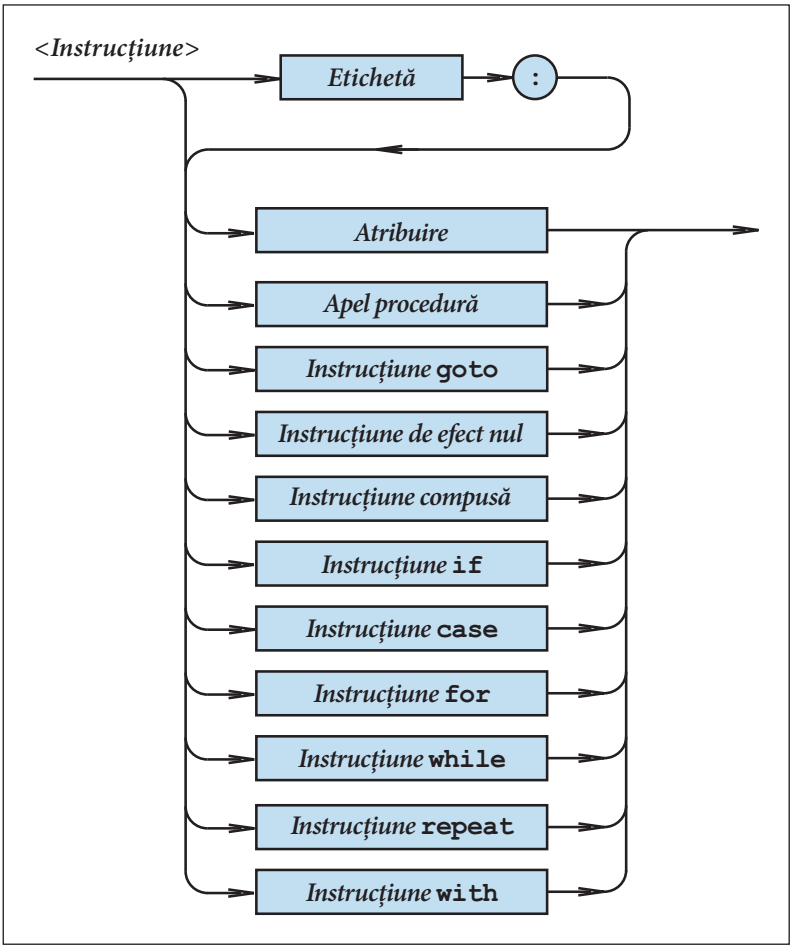


Fig. 3.1. Diagrama sintactică <*Instrucțiune*>

3.2. Expresii

Formulele pentru calculul unor valori se reprezintă în PASCAL prin **expresii**. Acestea sînt formate din operanzi (constante, variabile, referințe de funcții) și operatori (simbolurile operațiilor). Operatorii se clasifică după cum urmează:

<*Operator multiplicativ*> ::= * | / | **div** | **mod** | **and**

<*Operator aditiv*> ::= + | – | **or**

<*Operator relațional*> ::= < | <= | = | >= | > | <> | **in**

În componența expresiilor intră factori, termeni și expresii simple.

Factorul poate fi o variabilă, o constantă fără semn, apelul unei funcții ș.a. Mai exact:

<*Factor*> ::= <*Variabilă*> | <*Constantă fără semn*> | <*Apel funcție*> | **not** <*Factor*> | (<*Expresie*>) | <*Constructor mulțime*>

Exemple:

1) `15`

2) `x`

3) `p`

4) `sin(x)`

5) `not p`

6) `cos(x)`

Un **termen** are forma:

<*Termen*> ::= <*Factor*> {<*Operator multiplicativ*> <*Factor*>}

Exemple:

1) `15`

2) `x`

3) `15*x`

4) `x*y*z`

5) `p and q`

6) `sin(x)/3`

Prin **expresie simplă** se înțelege:

<*Expresie simplă*> ::= [+|-]<*Termen*>{<*Operator aditiv*> <*Termen*>}

Exemple:

1) `15`

2) `+15`

3) `15*x+sin(x)/3`

4) `p or q`

5) `-a+b-c`

6) `sin(x)/3+cos(x)`

La rîndul său, o **expresie** are forma:

<*Expresie*> ::= <*Expresie simplă*> {<*Operator relațional*> <*Expresie simplă*>}

Exemple:

1) `-15`

2) `x*y+cos(z)/4`

3) `x<15`

4) `15*x+sin(x)/3<>11`

5) `a>c`

6) `x+6>z-3.0`

Diagramele sintactice ale unităților gramaticale în studiu sînt prezentate în *figurile 3.2* și *3.3*.

O expresie luată între paranteze se transformă într-un factor. Cu astfel de factori se pot forma noi termeni, expresii simple, expresii ș.a.m.d.

Exemple:

	Notația matematică	*Notația în PASCAL*
1)	$\frac{a+b}{c+d}$	`(a+b)/(c+d)`

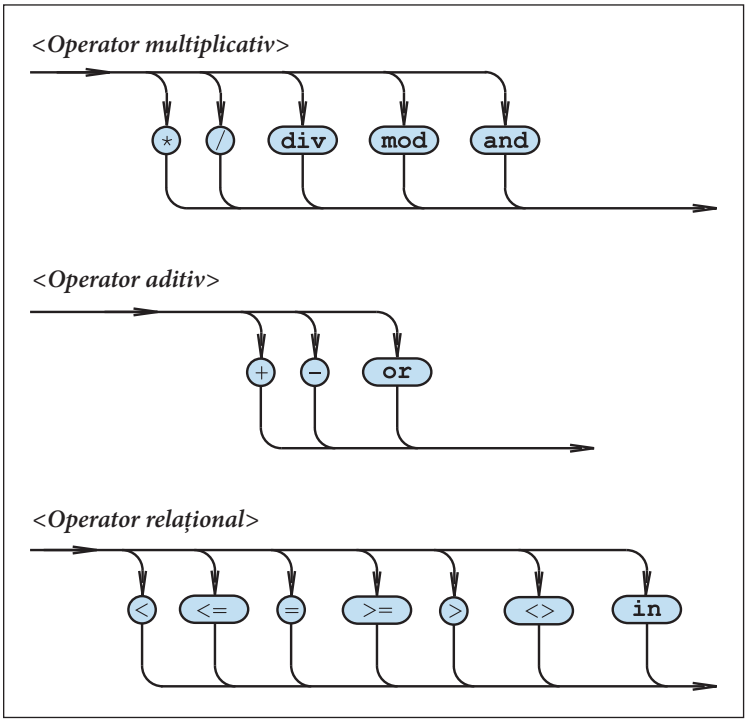


Fig. 3.2. Diagramele sintactice pentru operatori

2)	$\frac{-b+\sin(b-c)}{2a}$	`(-b+sin(b-c))/(2*a)`
3)	$-\frac{1}{xy}$	`-1/(x*y)`
4)	$p<q \,\&\, r>s$	`(p<q)and(r>s)`
5)	$\overline{x \vee y}$	`not (x or y)`
6)	$\frac{1}{a+b}>\frac{1}{c+d}$	`1/(a+b)>1/(c+d)`

În cadrul unei expresii, un apel de funcție poate să apară peste tot unde apare o variabilă sau o constantă (*fig. 3.3*). Limbajul PASCAL conține un set de **funcții predefinite**, cunoscute oricărui program. Aceste funcții sînt prezentate în *tabelul 3.1.*

Întrebări și exerciții

❶ Scrieți conform regulilor limbajului PASCAL expresiile:

a) a^2+b^2; *b)* $a^2+2ab+b^2$;

Tabelul 3.1

Funcțiile predefinite ale limbajului PASCAL

Denumirea funcției	Notația în PASCAL
valoarea absolută $\|x\|$	`abs(x)`
sinus sin x	`sin(x)`
cosinus cos x	`cos(x)`
arctangenta arctg x	`arctan(x)`
pătratul lui x x^2	`sqr(x)`
rădăcina pătrată $\sqrt{x}$	`sqrt(x)`
puterea numărului e e^x	`exp(x)`
logaritmul natural ln x	`ln(x)`
rotunjirea lui x	`round(x)`
trunchierea lui x	`trunc(x)`
paritatea numărului i (`false` pentru i par și `true` în caz contrar)	`odd(i)`
numărul valorii ordinale v	`ord(v)`
predecesorul lui v	`pred(v)`
succesorul lui v	`succ(v)`
caracterul cu numărul i	`chr(i)`
testarea sfîrșitului de fișier	`eof(f)`
testarea sfîrșitului de linie	`eoln(f)`

c) $(a + b)^2$;

d) $v_0 t + \frac{at^2}{2}$;

e) $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$;

f) $\cos \alpha + \cos \beta$;

g) $\cos (\alpha + \beta)$;

h) $2\pi r$;

i) πr^2;

j) $x_1 x_2 \vee x_3 x_4$;

k) $\overline{x_1 \vee x_2}$;

l) $|x| < 3$;

m) $|z| < 6 \;\&\; |q| > 3{,}14$;

n) $x > 0 \;\&\; y > 8 \;\&\; R < 15$.

❷ Indicați pe diagramele sintactice din *figura 3.3* drumurile care corespund următoarelor expresii PASCAL:

a) `x`

b) `3.14`

c) `sin(x)`

d) `not q`

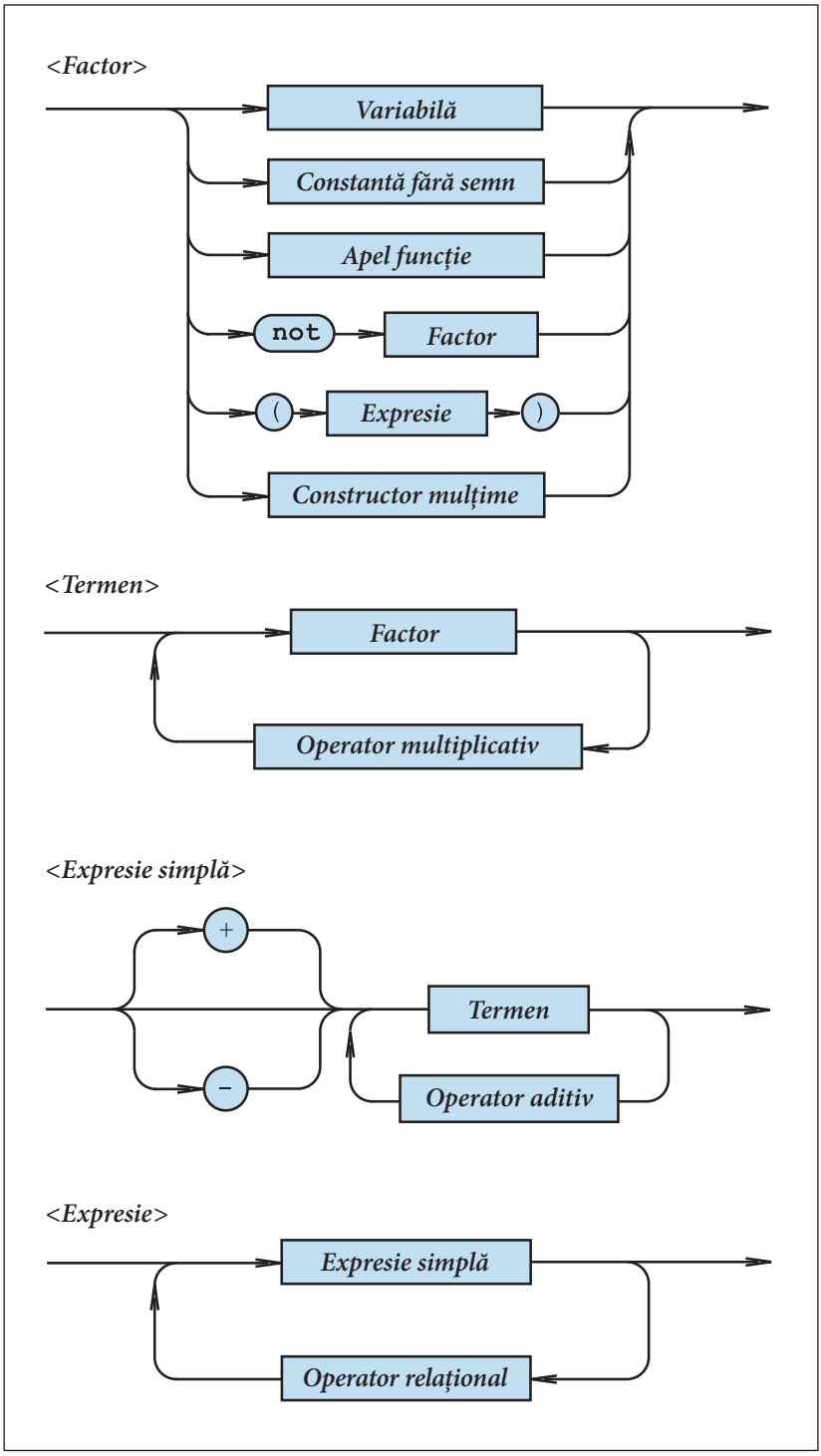


Fig. 3.3. Diagramele sintactice pentru definirea expresiilor

e) `(+3.14)`

f) `x>2.85`

g) `sqr(b)-4*a*c>0`

h) `(a>b)and(c>d)`

❸ Transpuneți expresiile PASCAL în notații obișnuite:

a) `sqr(a)+sqr(b)`

b) `2*a*(b+c)`

c) `sqrt((a+b)/(a-b))`

d) `exp(x+y)`

e) `cos(ALFA-BETA)`

f) `sqr(a+b)/(a-b)`

g) `x>0 or q<p`

h) `not(x and y)`

❹ Care din expresiile PASCAL ce urmează sînt greșite? Pentru a argumenta răspunsul, utilizați diagramele sintactice din *figura 3.3*.

a) `((((+x))))`

b) `((((x))))`

c) `sinx+cosx`

d) `sqr(x)+sqr(y)`

e) `a<<b or c>d`

f) `not not not p`

g) `q and not p`

h) `not q and p`

i) `a+-b`

j) `sin(-x)`

k) `sin-x`

l) `cos(x+y)`

m) `sin(abs(x)+abs(y))`

n) `sqrt(-y)`

3.3. Evaluarea expresiilor

Prin evaluarea unei expresii se înțelege calculul valorii ei. Rezultatul furnizat depinde de valorile operanzilor și de operatorii care acționează asupra acestora. Regulile de evaluare a unei expresii sînt cele obișnuite în matematică:

– operațiile se efectuează conform priorității operatorilor;

– în cazul priorităților egale, operațiile se efectuează de la stînga spre dreapta;

– mai întîi se calculează expresiile dintre paranteze.

Prioritățile operatorilor sînt indicate în *tabelul 3.2*.

Tabelul 3.2

Prioritățile operatorilor limbajului PASCAL

Categorie	Operatori	Prioritate
operatori unari	**not**, @	prima (cea mai mare)
operatori multiplicativi	*, /, **div**, **mod**, **and**	a doua
operatori aditivi	+, -, **or**	a treia
operatori relaționali	<, <=, =, >=, >, <>, **in**	a patra (cea mai mică)

Exemplu:

Fie $x = 2$ și $y = 6$. Atunci:

1) `2*x+y = 2*2+6 = 4+6 = 10;`

2) `2*(x+y) = 2(2+6) = 2·8 = 16;`

3) `x+y/x–y = 2+6/2–6 = 2+3–6 = 5–6 = –1;`

4) `(x+y)/x–y = (2+6)/2–6 = 8/2–6 = 4–6 = –2;`

5) `x+y/(x–y) = 2+6/(2–6) = 2+6/(–4) = 2+(–1,5) = 0,5;`

6) `x+y<15 = 2+6<15 = 8<15 = true;`

7)
```
(x+y<15)and(x>3) = (2+6<15)and(2>3) = (8<15)and(2>3) =
                 = true and false = false.
```

Se observă că părțile componente ale unei expresii (*fig. 3.3*) se calculează în următoarea ordine:

1) factorii;
2) termenii;
3) expresiile simple;
4) expresia propriu-zisă.

Valoarea curentă a unei expresii poate fi afișată pe ecran cu ajutorul procedurii `writeln`:

`writeln` (<*Expresie*>)

Programul `P38` afișează pe ecran rezultatele evaluării expresiilor `x*y+z` și `x+y<z–1.0`. Valorile curente ale variabilelor `x`, `y` și `z` sînt citite de la tastatură.

```
Program P38;
 { Evaluarea expresiilor }
var x, y, z : real;
begin
 writeln('Introduceti numerele reale x, y, z:');
 readln(x, y, z);
 writeln(x*y+z);
 writeln(x+y<z–1.0);
end.
```

Întrebări și exerciții

❶ Fie $x = 1$, $y = 2$ și $z = 3$. Evaluați următoarele expresii:

a) `x+y+2*z`

b) `(x+y+2)*z`

c) `x*y+y*z`

d) `x*(y+y)*z`

e) `(x*y+y)*z`

f) `x*(y+y*z)`

g) `x*y<y*z`

h) `(x>y)or(6*x>y+z)`

i) `not(x+y+z>0)`

j) `not(x+y>0) and not(z<0)`

❷ Care sînt regulile de evaluare a unei expresii PASCAL?

❸ Indicați prioritatea fiecărui operator al limbajului PASCAL.

❹ Precizați ordinea în care se calculează componentele unei expresii PASCAL.

❺ Elaborați un program care evaluează expresiile *c* și *g* din exercițiul 1. Valorile curente ale variabilelor reale `x`, `y` și `z` se citesc de la tastatură.

3.4. Tipul expresiilor PASCAL

În funcție de mulțimea valorilor pe care le poate lua, fiecare expresie se asociază cu un anumit tip de date. Conform conceptului de dată realizat în limbajul PASCAL, **tipul expresiei** derivă (rezultă) din tipul operanzilor și operatorilor care acționează asupra acestora. Prin urmare tipul unei expresii poate fi dedus fără a calcula valoarea ei.

Tipul rezultatelor furnizate de operatori este indicat în *tabelul 3.3*. *Tabelul 3.4* conține tipul rezultatelor furnizate de funcțiile predefinite ale limbajului PASCAL.

Indiferent de tipul operanzilor, operatorul `/` (împărțirea) furnizează numai rezultate de tip `real`, iar operatorii relaționali – numai rezultate de tip `boolean`.

Pentru a afla tipul unei expresii, factorii, termenii și expresiile simple se examinează în ordinea evaluării lor. Tipul fiecărei părți componente se deduce cu ajutorul *tabelelor 3.3* și *3.4*.

De exemplu, fie expresia:

```
(x>i) or (6*i<sin(x/y))
```

unde `i` este de tipul `integer`, iar `x` și `y` de tipul `real`.

Tabelul 3.3

Tipul rezultatelor furnizate de operatori

Operator	Tipul operanzilor	Tipul rezultatului
+, -, *	`integer`	`integer`
	unul `integer`, altul `real`	`real`
/	`integer` sau `real`	`real`
div	`integer`	`integer`
mod	`integer`	`integer`
not, **and**, **or**	`boolean`	`boolean`
<, <=, =, >=, >, <>	tipuri identice	`boolean`
	tipuri compatibile	`boolean`
	unul `integer`, altul `real`	`boolean`

Tabelul 3.4

Tipul rezultatelor furnizate de funcțiile predefinite

Funcția	Tipul argumentului	Tipul rezultatului
`abs(x)`	`integer` sau `real`	coincide cu tipul lui `x`
`sin(x)`	`integer` sau `real`	`real`
`cos(x)`	`integer` sau `real`	`real`
`arctan(x)`	`integer` sau `real`	`real`
`sqr(x)`	`integer` sau `real`	coincide cu tipul lui `x`
`sqrt(x)`	`integer` sau `real`	`real`
`exp(x)`	`integer` sau `real`	`real`
`ln(x)`	`integer` sau `real`	`real`
`round(x)`	`real`	`integer`
`trunc(x)`	`real`	`integer`
`odd(i)`	`integer`	`boolean`
`ord(v)`	ordinal	`integer`
`pred(v)`	ordinal	coincide cu tipul lui `v`
`succ(v)`	ordinal	coincide cu tipul lui `v`
`chr(i)`	`integer`	`char`
`eof(f)`	fișier	`boolean`
`eoln(f)`	fișier	`boolean`

Aflăm tipul fiecărei părți componente și al expresiei în ansamblu în ordinea de evaluare:

1) `x>I` — `boolean`;

2) `6*i` — `integer`;

3) `x/y` — `real`;

4) `sin(x/y)` — `real`;

5) `6*i < sin(x/y)` — `boolean`;

6) `(x>i)or(6*i<sin(x/y))` — `boolean`.

Prin urmare expresia în studiu este de tip `boolean`.

Întrucît în procesul deducției valorile concrete ale expresiilor în studiu nu se calculează, tipurile subdomeniu se extind asupra tipurilor de bază.

De exemplu, în prezența declarațiilor:

```
type T1=1..10;  { subdomeniu de integer }
     T2=11..20; { subdomeniu de integer }
```

```
var i:T1;
    j:T2;
```

avem:

	Expresie	*Tipul expresiei*
1)	`i+j`	`integer`;
2)	`i mod j`	`integer`;
3)	`i/j`	`real`;
4)	`sin(i+j)`	`real`;
5)	`i>j`	`boolean`.

ş.a.m.d.

În funcție de tipul expresiei, distingem:
– expresii aritmetice (`integer` sau `real`);
– expresii ordinale (`integer`, `boolean`, `char`, *enumerare*);
– expresii booleene (`boolean`).

De obicei, expresiile aritmetice se utilizează în calcule (instrucțiunea de atribuire), expresiile ordinale – în instrucțiunile **`case`** și **`for`**, iar expresiile booleene – în instrucțiunile **`if`**, **`repeat`** și **`while`**.

Întrebări și exerciții

❶ Prin ce metodă se află tipul unei expresii PASCAL?

❷ În prezența declarațiilor:

```
var x, y : real;
    i, j : integer;
    p, q : boolean;
    r : char;
    s : (A, B, C, D, E, F, G, H);
```

aflați tipul următoarelor expresii:

a)	`i mod 3`	*i)*	`sqr(i)-sqr(j)`
b)	`i/3`	*j)*	`sqr(x)-sqr(y)`
c)	`i mod 3 > j div 4`	*k)*	`trunc(x)+trunc(y)`
d)	`x+y/(x-y)`	*l)*	`chr(i)`
e)	`not(x<i)`	*m)*	`ord(r)`
f)	`sin(abs(i)+abs(j))`	*n)*	`ord(s)>ord(r)`
g)	`sin(abs(x)+abs(y))`	*o)*	`pred(E)`
h)	`p and (cos(x)<=sin(y))`	*p)*	`(-x+sin(x-y))/(2*i)`

❸ Tipul unei expresii poate fi aflat din forma textuală a rezultatelor afișate pe ecran de instrucțiunea `writeln` (*<Expresie>*).
Exemple:

	Rezultatul afișat pe ecran	*Tipul expresiei*
1)	`100`	`integer`;
2)	`1.0000000000E+02`	`real`;
3)	`true`	`boolean`.

Elaborați programele respective și precizați tipul expresiilor ce urmează, pornind de la forma textuală a rezultatelor afișate:

a) `1+1.0`
b) `1/1+1`
c) `9*3 mod 4`
d) `4*x>9*y`
e) `chr(65)`
f) `not(x>y)`
g) `pred(9)>succ(7)`
h) `15 div ord(3)`
i) `trunc(x)+round(6*y)`
j) `sqr(3)-sqrt(16)`

Se consideră că variabilele `x` și `y` sînt de tip `real`.

❹ Se consideră declarațiile:

```
type T1=1..10;
     T2=11..20;
     T3='A'..'Z';
     T4=(A, B, C, D, E, F, G, H);
var  i : T1;
     j : T2;
     k : T3;
     m : 'C'..'G';
     n : T4;
     p : C..G;
     q : boolean;
```

Aflați tipul următoarelor expresii:

a) `i-j`
b) `i div j`
c) `6.3*i`;
d) `cos(3*i-6*j)`
e) `4*i>5*j`
f) `k<m`
g) `k<>m`
h) `chr(i)`
i) `ord(k)`
j) `ord(m)`
k) `n>p`
l) `ord(n)`
m) `succ(n)`
n) `pred(p)`
o) `ord(p)`
p) `ord(k)>ord(m)`
q) `(i>j) and q`
r) `not(i+j>0) or q`

3.5. Instrucțiunea de atribuire

Instrucțiunea în studiu are forma:

<*Variabilă*> `:=` <*Expresie*>

Execuția unei instrucțiuni de atribuire presupune:
a) evaluarea expresiei din partea dreaptă;
b) atribuirea valorii obținute variabilei din partea stîngă.
Exemple:

1) `x:=1`

2) `y:=x+3`

3) `z:=sin(x)+cos(y)`

4) `p:=not q`

5) `q:=(a<b) or (x<y)`

6) `c:='A'`

De reținut că simbolul "`:=`" (se citește "atribuire") desemnează o atribuire și nu trebuie confundat cu operatorul de relație "=" (egal).

O atribuire are loc dacă variabila și rezultatul evaluării expresiei sînt compatibile din punctul de vedere al atribuirii. În caz contrar, se va produce o eroare.

Variabila și rezultatul evaluării expresiei sînt **compatibile din punctul de vedere al atribuirii** dacă este adevărată una din următoarele afirmații:

1) variabila și rezultatul evaluării sînt de tipuri identice;

2) tipul rezultatului este un subdomeniu al tipului variabilei;

3) ambele tipuri sînt subdomenii ale aceluiași tip, iar rezultatul este în subdomeniul variabilei;

4) variabila este de tip `real`, iar rezultatul – de tip `integer` sau un subdomeniu al acestuia.

Pentru exemplificare, să examinăm următorul program:

```
Program P39;
 { Compatibilitate din punctul de vedere al atribuirii }
type T1=1..10; { subdomeniu de integer }
     T2=5..15; { subdomeniu de integer }
var i : T1;
    j : T2;
    k, m, n : integer;
    x : real;
begin
 write('k=');
 readln(k);
 i:=k; { corect pentru 1<=k<=10 }
 write('m=');
 readln(m);
 j:=m; { corect pentru 5<=m<=15 }
 write('n=');
 readln(n);
```

```
 i:=n+5; { corect pentru -4<=n<=5 }
 j:=n+2; { corect pentru 3<=n<=13 }
 x:=i+j;
 writeln('x=', x);
end.
```

Programul va derula fără erori numai pentru următoarele valori de intrare:

$$1 \le k \le 10;\ 5 \le m \le 15;\ 3 \le n \le 5.$$

Evident, în programul P39 instrucțiunile de tipul

```
k:=x        i:=x+1        j:=sin(i)
```

ș.a.m.d. ar fi incorecte, întrucît `x`, `x+1`, `sin(i)` sînt expresii de tip `real`, iar variabilele `k`, `i`, `j` sînt de tip `integer` sau subdomenii ale acestuia.

Întrebări și exerciții

❶ Cum se execută o instrucțiune de atribuire?

❷ Explicați termenul "compatibilitate din punctul de vedere al atribuirii".

❸ Sînt date următoarele declarații:

```
type Zi = (L, Ma, Mi, J, V, S, D);
     Culoare = (Galben, Verde, Albastru, Violet);
var i, j, k : integer;
    z : Zi;
    c : Culoare;
    x : real;
```

Care din instrucțiunile ce urmează sînt corecte?

a) `i:=12`

b) `j:=ord(i)`

c) `x:=ord(z)+1`

d) `k:=ord(x)+2`

e) `c:=i+4`

f) `c:=Verde`

g) `z:=D`

h) `c:=Pred(Galben)`

i) `x:=Succ(z)`

j) `i:=Succ(c)`

❹ Precizați pentru care valori ale variabilei `j` programul P40 va derula fără erori?

```
Program P40;
var i : -10..+10;
    j : integer;
begin
 write('j=');
 readln(j);
 i:=j+15;
 writeln('i=', i);
end.
```

3.6. Instrucțiunea *apel de procedură*

Procedura este un subalgoritm scris în limbaj de programare ce poate fi apelat din mai multe puncte ale unui program. Fiecare procedură are un nume, de exemplu, `readln`, `writeln`, `CitireDate`, `A15` ș.a.m.d. Limbajul PASCAL include un set de proceduri predefinite, cunoscute oricărui program: `read`, `readln`, `write`, `writeln`, `get`, `put`, `new` etc. În completare, programatorul poate defini proceduri proprii.

Instrucțiunea *apel de procedură* lansează în execuție procedura cu numele specificat. Sintaxa instrucțiunii date este:

<*Apel procedură*> ::= <*Nume procedură*> [<*Listă parametri actuali*>]

<*Nume procedură*> ::= <*Identificator*>

<*Listă parametri actuali*> ::= (<*Parametru actual*> {, <*Parametru actual*>})

Diagramele sintactice ale formulelor în studiu sînt prezentate în *figura 3.4*.

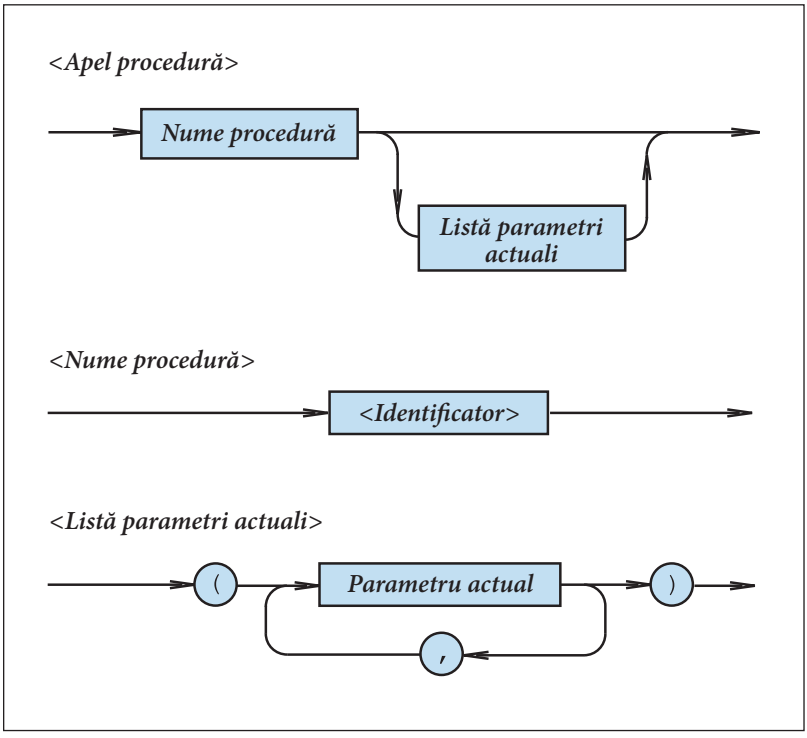


Fig. 3.4. Diagramele sintactice ale instrucțiunii *apel de procedură*

În mod obișnuit, <*Parametrul actual*> este o expresie.

Exemple:

1) `readln(x)`

2) `readln(x, y, z)`

3) `CitireDate(f, t)`

4) `Exit`

5) `writeln(x+y, sin(x))` *6)* `writeln(2*x)`

Tipul parametrilor actuali și ordinea în care aceștia apar în listă sînt impuse de declarațiile procedurii respective. În consecință, regulile de formare a listelor de parametri actuali vor fi studiate mai amănunțit în capitolele următoare.

Întrebări și exerciții

❶ Care este destinația instrucțiunii *apel de procedură*?

❷ Se consideră următoarele instrucțiuni:

a)
```
readln(x, y, z, q)
```

b)
```
CitireDate(ff, tt)
```

c)
```
Halt
```

d)
```
writeln('x=', x, 'y=', y)
```

e)
```
writeln('x+y=', x+y, 'sin(x)=', sin(x))
```

Precizați numele procedurilor apelate, numărul de parametri actuali din fiecare apel și parametrii propriu-ziși.

❸ Indicați pe diagramele sintactice din *figura 3.4* drumurile ce corespund instrucțiunilor din exercițiul 2.

3.7. Afișarea informației alfanumerice

În versiunile uzuale ale limbajului PASCAL, ecranul vizualizatorului este desemnat ca dispozitiv-standard de ieșire. De regulă, ecranul este împărțit în zone convenționale, numite zone-caracter. De obicei, aceste zone formează 25 de linii – cîte 80 de caractere pe linie. Zona în care va fi afișat caracterul curent este indicată de cursor.

Datele de ieșire ale unui program PASCAL pot fi afișate pe ecran printr-un apel `write(x)` sau `writeln(x)`.

Apelul

```
write(x1, x2, ..., xn)
```

este echivalent cu

```
write(x1); write(x2); ...; write(xn).
```

Parametrii actuali dintr-un apel `write` sau `writeln` se numesc ***parametri de ieșire***. Aceștia pot avea una din formele:

e $\qquad$ $e : w$ $\qquad$ $e : w : f$

unde e este o expresie de tip `integer`, `real`, `boolean`, `char` sau *șir de caractere*, a cărei valoare trebuie afișată; w și f sînt expresii de tip `integer`, numite ***specificatori de***

format. Expresia *w* specifică prin valoarea sa numărul minim de caractere ce vor fi folosite la afișarea valorii lui *e*; dacă sînt necesare mai puțin de *w* caractere, atunci forma externă a valorii lui *e* va fi completată cu spații la stînga pînă la *w* caractere (*fig. 3.5*).

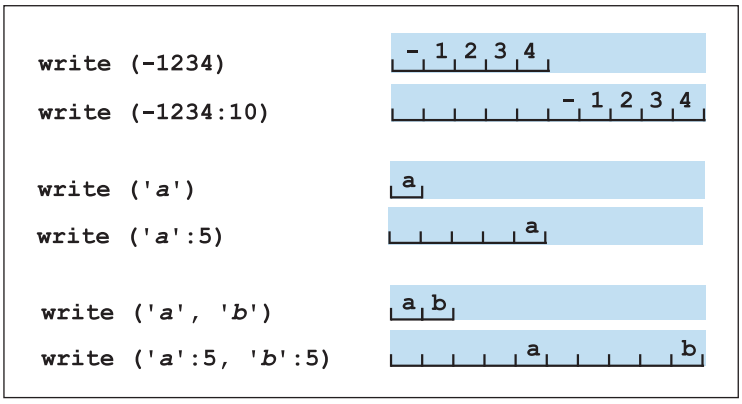


Fig. 3.5. Semnificația specificatorului de format *w*

Specificatorul de format *f* are sens în cazul în care *e* este de tip `real` și indică numărul de cifre care urmează punctul zecimal în scrierea valorii lui *e* în virgulă fixă, fără factor de scală. În lipsa lui *f*, valoarea lui *e* se scrie în virgulă mobilă, cu factor de scală (*fig. 3.6*).

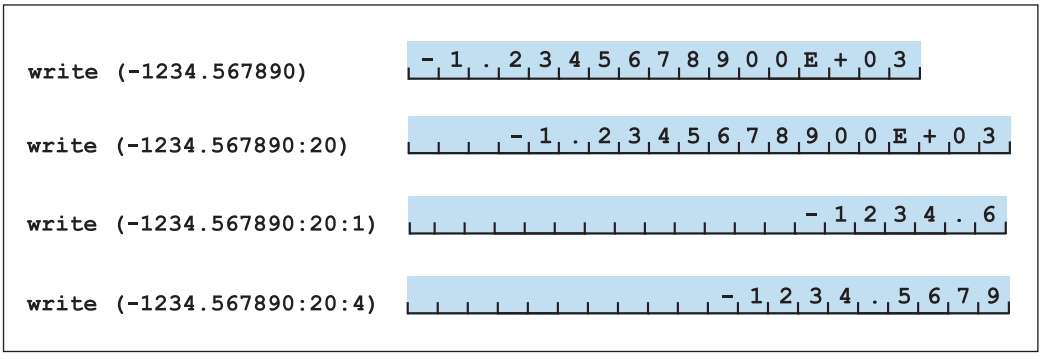


Fig. 3.6. Semnificația specificatorului de format *f*

Diferența dintre procedurile `write` și `writeln` constă în faptul că, după afișarea datelor, `write` lasă cursorul în linia curentă, în timp ce `writeln` îl trece la începutul unei linii noi. Utilizarea rațională a apelurilor `write`, `writeln` și a specificatorilor de format asigură o afișare lizibilă a datelor de ieșire. Dacă pe ecran se afișează mai multe valori, se recomandă ca acestea să fie însoțite de identificatorii respectivi sau de mesaje sugestive.

Exemple:

1) `write('Suma numerelor introduse este')`

2) `writeln(s:20)`

3) `writeln('Suma=', s)`

4) `writeln('s=', s)`

5) `writeln('x=', x, 'y=':5, y, 'z=':5, z)`

Întrebări și exerciții

❶ Care este destinația specificatorului de format?
❷ Cum se numesc parametrii actuali ai unui apel de procedură `write` sau `writeln`?
❸ Precizați formatul datelor afișate pe ecran de programele ce urmează:

```
Program P41;
 { Afisarea datelor de tip integer }
var i : integer;
begin
 i:=-1234;
 writeln(i);
 writeln(i:1);
 writeln(i:8);
 writeln(i, i);
 writeln(i:8, i:8);
 writeln(i, i, i);
 writeln(i:8, i:8, i:8);
end.
```

```
Program P42;
 { Afisarea datelor de tip real }
var x : real;
begin
 x:=-1234.567890;
 writeln(x);
 writeln(x:20);
 writeln(x:20:1);
 writeln(x:20:2);
 writeln(x:20:4);
 writeln(x, x, x);
 writeln(x:20, x:20, x:20);
 writeln(x:20:4, x:20:4, x:20:4);
end.
```

```
Program P43;
 { Afisarea datelor de tip boolean }
var p : boolean;
begin
 p:=false;
 writeln(p);
```

```
 writeln(p:10);
 writeln(p, p);
 writeln(p:10, p:10);
end.
```

```
Program P44;
 { Afisarea sirurilor de caractere }
begin
 writeln('abc');
 writeln('abc':10);
 writeln('abc', 'abc');
 writeln('abc':10, 'abc':10);
end.
```

❹ Elaborați un program care afișează pe ecran valorile `1234567890`, `123`, `123.0` și `true` după cum urmează:

```
1234567890
123
123.0
true
1234567890
                    123
                    123.000
                    true
```

3.8. Citirea datelor de la tastatură

În mod obișnuit, tastatura vizualizatorului este desemnată ca **dispozitiv-standard de intrare**. Citirea datelor de la tastatură se realizează prin apelul procedurilor predefinite `read` sau `readln`. Lista parametrilor actuali ai unui apel `read` sau `readln` poate să includă variabile de tip `integer`, `real`, `char`, inclusiv *șir de caractere*.

Apelul

```
read(x)
```

are următorul efect. Dacă variabila `x` este de tip `integer` sau `real`, atunci este citit întregul șir de caractere care reprezintă valoarea întreagă sau reală. Dacă `x` este de tip `char`, procedura citește un singur caracter.

Apelul

```
read(x1, x2, ..., xn)
```

este echivalent cu

```
read(x1); read(x2); ...; read(xn).
```

Datele numerice introduse de la tastatură trebuie separate prin spații sau caractere de sfîrșit de linie. Spațiile dinaintea unei valori numerice sînt ignorate. Șirul de caractere care

reprezintă o valoare numerică se conformează sintaxei constantelor numerice de tipul respectiv. În caz contrar, este semnalată o eroare de intrare-ieșire.

De exemplu, fie programul:

```
Program P45;
 { Citirea datelor numerice de la tastatura }
var i, j : integer;
    x, y : real;
begin
  read(i, j, x, y);
  writeln('Ati introdus:');
  writeln('i=', i);
  writeln('j=', j);
  writeln('x=', x);
  writeln('y=', y);
end.
```

în care sînt citite de la tastatură valorile variabilelor `i`, `j`, `x`, `y`. După lansarea programului în execuție, utilizatorul tastează:

```
1<ENTER>
2<ENTER>
3.0<ENTER>
4.0<ENTER>
```

Pe ecran se va afișa:

```
Ati introdus:
i=1
j=2
x=3.0000000000E+00
y=4.0000000000E+00
```

Același efect se va obține și la tastarea numerelor într-o singură linie:

```
1 2 3.0 4.0<ENTER>
```

Dacă e necesar, numerele întregi, introduse de utilizator, sînt convertite în valori reale. De exemplu, în cazul programului `P45` utilizatorul poate tasta

```
1 2 3 4<ENTER>
```

Procedura `readln` citește datele în același mod ca și procedura `read`. Însă după citirea ultimei valori, restul caracterelor din linia curentă se ignoră. Pentru exemplificare, prezentăm programul `P46`:

```
Program P46;
 { Apelul procedurii readln }
var i, j : integer;
    x, y : real;
```

```
begin
  writeln('Apelul procedurii read');
  read(i, j);
  read(x, y);
  writeln('Ati introdus:');
  writeln('i=', i, ' j=', j, ' x=', x, ' y=', y);
  writeln('Apelul procedurii readln');
  readln(i, j);
  readln(x, y);
  writeln('Ati introdus:');
  writeln('i=', i, ' j=', j, ' x=', x, ' y=', y);
end.
```

La execuția instrucțiunilor

```
read(i, j);
read(x, y);
```

valorile numerice din linia introdusă de utilizator

```
1 2 3 4<ENTER>
```

vor fi atribuite variabilelor respectiv `i`, `j`, `x`, `y`. La execuția instrucțiunii

```
readln(i, j)
```

valorile numerice `1` și `2` din linia

```
1 2 3 4<ENTER>
```

vor fi atribuite variabilelor `i` și `j`. Numerele `3` și `4` se ignoră. În continuare calculatorul execută instrucțiunea

```
readln(x, y)
```

adică va aștepta introducerea unor valori pentru `x` și `y`.

Subliniem faptul că apelul procedurii `readln` fără parametri va forța calculatorul să aștepte acționarea tastei `<ENTER>`. Acest apel se folosește pentru a suspenda derularea programului, oferindu-i utilizatorului posibilitatea să analizeze rezultatele afișate anterior pe ecran.

Pentru a înlesni introducerea datelor, se recomandă ca apelurile `read(...)` și `readln(...)` să fie precedate de afișarea unor mesaje sugestive.

Exemple:

1) `write('Dati doua numere:'); readln(x, y);`

2) `write('Dati un numar intreg:'); readln(i);`

3) `write('x='); readln(x);`

4) `write('Raspundeti D sau N:'); readln(c);`

Întrebări și exerciții

❶ Cum se separă datele numerice care se introduc de la tastatură?
❷ Care este diferența dintre procedurile `read` și `readln`?
❸ Se consideră următorul program:

```
Program P47;
var i : integer;
    c : char;
    x : real;
begin
 readln(i);
 readln(c);
 readln(x);
 writeln('i=', i);
 writeln('c=', c);
 writeln('x=', x);
 readln;
end.
```

Precizați rezultatele afișate de acest program după tastarea datelor de intrare:

a)
```
1
2
3
```

b)
```
1 2 3
5 6 7
8 9 0
```

c)
```
123
456
789
```

d)
```
123  456  789
abc def ghi
890 abc def
```

3.9. Instrucțiunea de efect nul

Executarea acestei instrucțiuni nu are niciun efect asupra variabilelor programului. Sintaxa instrucțiunii în studiu este:

<*Instrucțiunea de efect nul*> ::=

Prin urmare în textul unui program instrucțiunea de efect nul nu este reprezentată prin nimic. Întrucît instrucțiunile unui program sînt despărțite între ele prin delimitatorul "`;`", prezența instrucțiunii de efect nul este marcată de apariția acestui delimitator.

De exemplu, în textul

```
x:=4;;;; y:=x+1
```

există 5 instrucțiuni dintre care 3 de efect nul.

În mod obișnuit, instrucțiunea de efect nul se utilizează la etapa elaborării și depanării unor programe complexe. Deși efectul său la execuție este nul, inserarea sau eliminarea unei astfel de instrucțiuni (mai exact, a simbolului "`;`") poate să altereze semnificația programului.

3.10. Instrucțiunea `if`

Instrucțiunea de ramificare simplă **`if`**, în funcție de valoarea unei expresii de tip boolean, decide fluxul execuției. Sintaxa instrucțiunii este:

<*Instrucțiune* **`if`**> ::= **`if`** <*Expresie booleană*> **`then`** <*Instrucțiune*>
[**`else`** <*Instrucțiune*>]

Diagrama sintactică a instrucțiunii în studiu este prezentată în *figura 3.7*. Expresia booleană din componența instrucțiunii **`if`** se numește **condiție**.

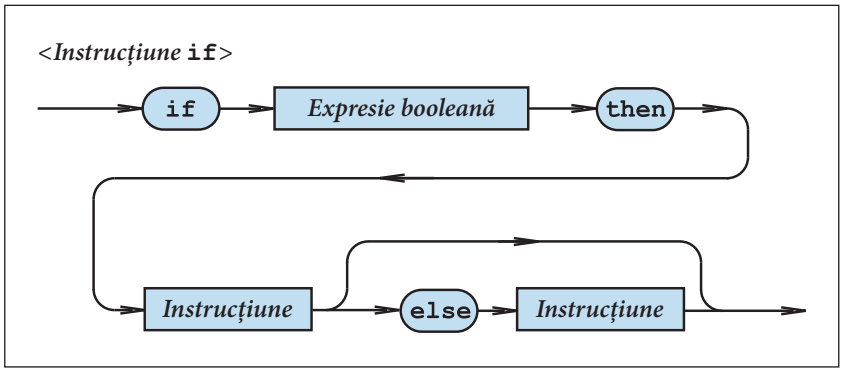


Fig. 3.7. Diagrama sintactică <*Instrucțiune* **`if`**>

Execuția instrucțiunii **`if`** începe prin evaluarea condiției. Dacă rezultatul evaluării este `true`, atunci se execută instrucțiunea situată după cuvîntul-cheie **`then`**. Dacă condiția are valoarea `false`, atunci: fie că se execută instrucțiunea situată după cuvîntul-cheie **`else`** (dacă există), fie că se trece la instrucțiunea situată după instrucțiunea **`if`**.

În programul ce urmează, instrucțiunea **`if`** este utilizată pentru determinarea maximului a două numere `x` și `y`, citite de la tastatură.

```
Program P48;
{ Determinarea maximului a doua numere }
var x, y, max : real;
begin
 writeln('Dati doua numere:');
 write('x='); readln(x);
 write('y='); readln(y);
 if x>=y then max:=x else max:=y;
 writeln('max=', max);
 readln;
end.
```

Următorul program transformă cifrele romane *I* (unu), *V* (cinci), *X* (zece), *L* (cincizeci), *C* (o sută), *D* (cinci sute) sau *M* (o mie), citite de la tastatură, în numerele corespunzătoare din sistemul zecimal.

```
Program P49;
 { Conversia cifrelor romane }
var i : integer; c : char;
begin
 i:=0;
 writeln('Introduceti una din cifrele');
 writeln('romane I, V, X, L, C, D, M');
 readln(c);
 if c='I' then i:=1;
 if c='V' then i:=5;
 if c='X' then i:=10;
 if c='L' then i:=50;
 if c='C' then i:=100;
 if c='D' then i:=500;
 if c='M' then i:=1000;
 if i=0 then writeln(c, ' – nu este o cifra romana')
        else writeln(i);
 readln;
end.
```

De reținut că limbajul PASCAL nu consideră simbolul "`;`" ca făcînd parte din instrucțiune, ci îl folosește ca delimitator. Prin urmare dacă într-o instrucțiune

```
if B then S
```

introducem înainte de S instrucțiunea cu efect nul

```
if B then; S
```

atunci `S` nu mai intră în componența instrucțiunii condiționale, deci este executată indiferent de valoarea lui `B`.

Dacă într-o instrucțiune

```
if B then I else J
```

introducem după `I` simbolul "`;`", obținem un program incorect sub aspect sintactic:

```
if B then I; else J
```

În acest caz, secvența **`else`** `J` este interpretată ca fiind instrucțiunea ce urmează celei condiționale.

Întrebări și exerciții

❶ Care este destinația instrucțiunii **`if`**?

❷ Ce valori va lua variabila `x` după executarea fiecăreia dintre instrucțiunile ce urmează? Se consideră că `a=18`, `b=-15` și `p=true`.

a)
```
if a>b then x:=1 else x:=4;
```

b)
```
if a<b then x:=15 else x:=-21;
```

c) `if p then x:=32 else x:=638;`

d) `if not p then x:=0 else x:=1;`

e) `if (a<b) and p then x:=-1 else x:=1;`

f) `if (a>b) or p then x:=-6 else x:=-5;`

g) `if not (a>b) then x:=19 else x:=-2;`

h) `if (a=b) or p then x:=89 else x:=-15.`

❸ Elaborați un program care calculează valorile uneia dintre funcțiile:

a) $y=\begin{cases}2x, x\geq 0;\\ \frac{x}{2}, x<0;\end{cases}$

b) $y=\begin{cases}x+3, x>5;\\ x-3, x\leq 5;\end{cases}$

c) $y=\begin{cases}x, x\geq 3;\\ x+4, x<3;\end{cases}$

d) $y=\begin{cases}x, |x|>5;\\ 2x, |x|\leq 5.\end{cases}$

Exemplu: $y=\begin{cases}x+6, x>4;\\ x-3, x\leq 4.\end{cases}$

```
Program P50;
var x, y : real;
begin
 write('x='); readln(x);
 if x>4 then y:=x+6 else y:=x-3;
 writeln('y=', y);
 readln;
end.
```

❹ Ce rezultate va afișa următorul program?

```
Program P51;
var x, y : real;
begin
 write('x='); readln(x);
 y:=x;
 if x>0 then; y:=2*x;
 writeln('y=', y);
 readln;
end.
```

❺ Comentați mesajele afișate pe ecran pe parcursul compilării programului P52:

```
Program P52;
 { Eroare }
var x, y : real;
```

```
begin
 write('x=');
 readln(x);
 if x>4 then y:=x+6;
        else y:=x-3;
 writeln('y=', y); readln;
end.
```

❻ Scrieți un program care transformă numerele zecimale 1, 5, 10, 50, 100, 500 și 1000, citite de la tastatură, în cifre romane.

3.11. Instrucțiunea case

Instrucțiunea de ramificare multiplă **case** conține o expresie numită **selector** și o listă de instrucțiuni. Fiecare instrucțiune este prefixată de una sau mai multe constante de caz. Sintaxa instrucțiunii în studiu este:

<*Instrucțiune* **case**> ::= **case** <*Expresie*> **of** [<*Caz*>{**;** <*Caz*>}] [**;**] **end**

<*Caz*> ::= <*Constantă*> {**,** <*Constantă*>} **:** <*Instrucțiune*>

Diagramele sintactice corespunzătoare sînt prezentate în *figura 3.8*.

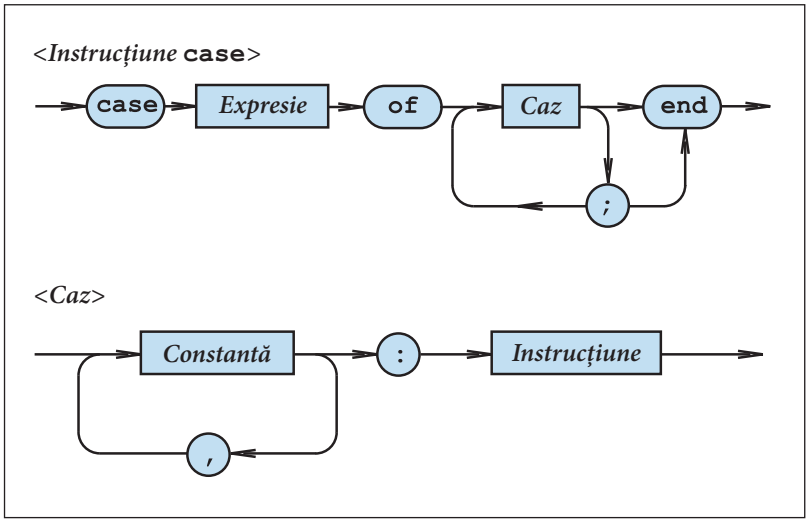


Fig. 3.8. Diagramele sintactice ale instrucțiunii **case**

Selectorul trebuie să fie de tip ordinal. Toate constantele de caz trebuie să fie unice și compatibile, din punctul de vedere al atribuirii, cu tipul selectorului.

Exemple:

```
var i : integer; c : char; a, b, y : real;
```

1)
```
case i of
  0, 2, 4, 6, 8 : writeln('Cifra para');
  1, 3, 5, 7, 9 : writeln('Cifra impara');
end;
```

2)
```
case c of
  '+' : y:=a+b;
  '-' : y:=a-b;
  '*' : y:=a*b;
  '/' : y:=a/b;
end.
```

Execuția instrucțiunii **case** începe prin evaluarea selectorului. În funcție de valoarea obținută, se execută instrucțiunea prefixată de constanta respectivă.

În programul ce urmează, instrucțiunea **case** este utilizată pentru conversia cifrelor romane în numere zecimale.

```
Program P53;
 { Conversia cifrelor romane }
var i : integer; c : char;
begin
 i:=0;
 writeln('Introduceti una din cifrele');
 writeln('romane I, V, X, L, C, D, M');
 readln(c);
 case c of
   'I' : i:=1;
   'V' : i:=5;
   'X' : i:=10;
   'L' : i:=50;
   'C' : i:=100;
   'D' : i:=500;
   'M' : i:=1000;
 end;
 if i=0 then writeln(c, ' - nu este o cifra romana')
        else writeln(i);
 readln;
end.
```

Subliniem faptul că în unele implementări ale limbajului sintaxa și semantica instrucțiunii **case** au fost modificate. Lista cazurilor poate să includă o instrucțiune precedată de cuvîntul-cheie **else** (în unele versiuni **otherwise**). Constantele de caz pot fi înlocuite cu intervale de forma:

```
<Constantă>..<Constantă>
```

Exemplu (Turbo PASCAL 7.0):

```
Program P54;
 { Simularea unui calculator de buzunar }
var a, b : real;
    c : char;
begin
 write('a='); readln(a);
 write('b='); readln(b);
 write('Cod operatie '); readln(c);
 case c of
   '+' : writeln('a+b=', a+b);
   '-' : writeln('a-b=', a-b);
   '*' : writeln('a*b=', a*b);
   '/' : writeln('a/b=', a/b);
 else writeln('Cod operatie necunoscut');
 end;
 readln;
end.
```

Întrebări și exerciții

❶ Indicați pe diagramele sintactice din *figura 3.8* drumurile care corespund instrucțiunilor **case** din programele P53 și P54.

❷ Cum se execută o instrucțiune **case**? De ce tip trebuie să fie selectorul?

❸ Ce fel de constante pot fi utilizate în calitate de constante de caz?

❹ Înlocuiți instrucțiunea **case** a programului P54 cu o secvență echivalentă de instrucțiuni **if**.

❺ Utilizînd instrucțiunea **case**, scrieți un program care transformă numerele zecimale 1, 5, 10, 50, 100, 500, 1000, citite de la tastatură, în cifre romane.

❻ Ce va apare pe ecran în urma execuției programului P55?

```
Program P55;
type Semnal=(Rosu, Galben, Verde);
var s : Semnal;
begin
 s:=Verde;
 s:=pred(s);
 case s of
   Rosu : writeln('STOP');
   Galben : writeln('ATENTIE');
   Verde : writeln('START');
 end;
 readln;
end.
```

❼ Comentați programele:

```
Program P56;
 { Eroare }
var x : real;
begin
 writeln('x='); readln(x);
 case x of
   0,2,4,6,8 : writeln('Cifra para');
   1,3,5,7,9 : writeln('Cifra impara');
 end;
 readln;
end.
```

```
Program P57;
 { Eroare }
var i : 1..4;
begin
 write('i='); readln(i);
 case i of
   1 : writeln('unu');
   2 : writeln('doi');
   3 : writeln('trei');
   4 : writeln('patru');
   5 : writeln('cinci');
 end;
 readln;
end.
```

```
Program P58;
 { Eroare }
type Semnal = (Rosu, Galben, Verde);
     Culoare = (Albastru, Portocaliu);
var s : Semnal;
    c : Culoare;
begin
 { ... }
 case s of
   Rosu : writeln('STOP');
   Galben : writeln('ATENTIE');
   Verde : writeln('START');
   Albastru : writeln('PAUZA');
 end;
 { ... }
end.
```

3.12. Instrucțiunea `for`

Instrucțiunea **`for`** indică execuția repetată a unei instrucțiuni în funcție de valoarea unei variabile de control. Sintaxa instrucțiunii în studiu este:

<*Instrucțiune* **`for`**> ::= **`for`** <*Variabilă*> **`:=`** <*Expresie*> <*Pas*> <*Expresie*> **`do`** <*Instrucțiune*>

<*Pas*> ::= **`to`** | **`downto`**

Diagramele sintactice sînt prezentate în *figura 3.9*.

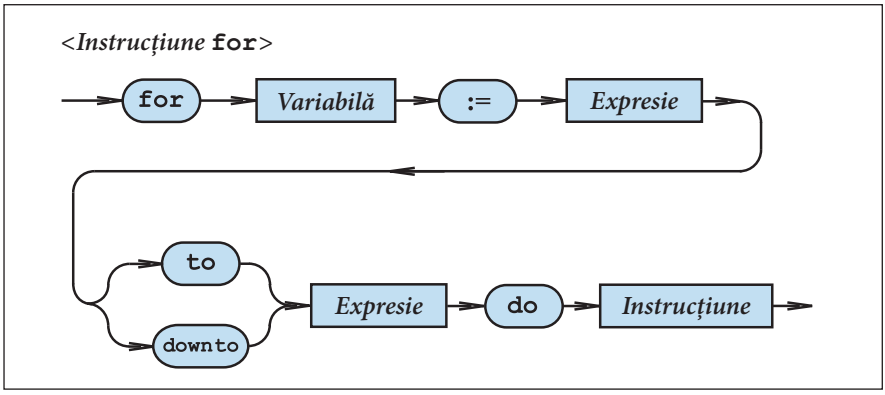


Fig. 3.9. Diagrama sintactică a instrucțiunii **`for`**

Variabila situată după cuvîntul-cheie **`for`** se numește **variabilă de control** sau **contor**. Această variabilă trebuie să fie de tip ordinal.

Valorile expresiilor din componența instrucțiunii **`for`** trebuie să fie compatibile, în aspectul atribuirii, cu tipul variabilei de control. Aceste expresii sînt evaluate o singură dată, la începutul ciclului. Prima expresie indică valoarea inițială, iar expresia a doua – valoarea finală a variabilei de control.

Instrucțiunea situată după cuvîntul-cheie **`do`** se execută pentru fiecare valoare din domeniul determinat de valoarea inițială și de valoarea finală.

Dacă instrucțiunea **`for`** utilizează pasul **`to`**, valorile variabilei de control sînt incrementate la fiecare repetiție, adică se trece la succesorul valorii curente. Dacă valoarea inițială este mai mare decît valoarea finală, instrucțiunea situată după cuvîntul-cheie **`do`** nu se execută niciodată.

Dacă instrucțiunea **`for`** utilizează pasul **`downto`**, valorile variabilei de control sînt decrementate la fiecare repetiție, adică se trece la predecesorul valorii curente. Dacă valoarea inițială este mai mică decît valoarea finală, instrucțiunea situată după cuvîntul-cheie **`do`** nu se execută niciodată.

Exemplu:

```
Program P59;
 { Instructiunea for }
var i : integer;
    c : char;
```

```
begin
  for i:=0 to 9 do write(i:2);
  writeln;
  for i:=9 downto 0 do write(i:2);
  writeln;
  for c:='A' to 'Z' do write(c:2);
  writeln;
  for c:='Z' downto 'A' do write(c:2);
  writeln;
  readln;
end.
```

Rezultatele afișate pe ecran:

```
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

Valorile variabilei de control nu pot fi modificate în interiorul ciclului, adică:

1) nu se fac atribuiri variabilei de control;

2) variabila actuală de control nu poate fi variabilă de control a altei instrucțiuni **for** incluse;

3) nu se admit apeluri de tipul `read`, `readln` în care apare variabila de control.

La ieșirea din instrucțiunea **for** valoarea variabilei de control nu este definită, în afara cazului cînd ieșirea din ciclu se face forțat, printr-o instrucțiune de salt necondiționat **goto**.

Instrucțiunea **for** este utilă pentru programarea algoritmilor iterativi, în care numărul de repetări este cunoscut. Pentru exemplificare prezentăm programele `P60`, `P61` și `P62` care calculează respectiv $n!$, x^n și suma $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+...+\frac{1}{n}$.

```
Program P60;
 { Calcularea factorialului }
var n, i, f : 0..MaxInt;
begin
  write('n='); readln(n);
  f:=1;
  for i:=1 to n do f:=f*i;
  writeln('n!=', f);
  readln;
end.
```

```
Program P61;
 { Calcularea lui x la puterea n }
var x, y : real;
    n, i : 0..MaxInt;
```

```
begin
  write('x='); readln(x);
  write('n='); readln(n);
  y:=1;
  for i:=1 to n do y:=y*x;
  writeln('y=', y);
  readln;
end.
```

```
Program P62;
 { Calcularea sumei 1 + 1/2 + 1/3 + ... + 1/n }
var n, i : 1..MaxInt;
    s : real;
begin
  write('n=');
  readln(n);
  s:=0;
  for i:=1 to n do s:=s+1/i;
  writeln('s=', s);
  readln;
end.
```

Întrebări și exerciții

❶ Indicați pe diagrama sintactică din *figura 3.9* drumurile care corespund instrucțiunilor **for** din programul P59.

❷ Cum se execută o instrucțiune **for**?

❸ Ce va afișa pe ecran programul P63?

```
Program P63;
type Zi = (L, Ma, Mi, J, V, S, D);
var z : Zi;
begin
  for z:=L to S do writeln(ord(z));
  readln;
  for z:=D downto Ma do writeln(ord(z));
  readln;
end.
```

❹ Se consideră declarațiile:

```
var i, j, n : integer;
    x, y : real;
    c : char;
```

Care din instrucțiunile ce urmează sînt corecte?

a) `for i:=-5 to 5 do j:=i+3;`

b)
```
for i:=-5 to 5 do i:=j+3;
```
c)
```
for j:=-5 to 5 do i:=j+3;
```
d)
```
for i:=1 to n do y:=y/i;
```
e)
```
for x:=1 to n do y:=y/x;
```
f)
```
for c:='A' to 'Z' do writeln(ord(c));
```
g)
```
for c:='Z' downto 'A' do writeln(ord(c));
```
h)
```
for i:=-5 downto -10 do readln(i);
```
i)
```
for i:=ord('A') to ord('A')+ 9 do writeln(i);
```
j)
```
for c:='0' to '9' do writeln(c, ord(c):3);
```
k)
```
for j:=i/2 to i/2+10 do writeln(j).
```

❺ Se consideră declarațiile:

```
var i, m, n : integer;
```

De cîte ori se vor executa apelurile `writeln(i)` și `writeln(2*i)` din componența instrucțiunilor

```
for i:=m to n do writeln(i);
for i:=m to n do writeln(2*i);
```

dacă:

a) `m=1, n=5;`

b) `m=3, n=5;`

c) `m=3, n=3;`

d) `m=5, n=3?`

❻ Creați un program care afișează pe ecran codurile caracterelor *'A'*, *'B'*, *'C'*, ..., *'Z'*.

❼ Calculați pentru primii *n* termeni:

a) $1 + 3 + 5 + 7 + \ldots$ și $1 \cdot 3 \cdot 5 \cdot 7 \cdot \ldots$;

b) $2 + 4 + 6 + 8 + \ldots$ și $2 \cdot 4 \cdot 6 \cdot 8 \cdot \ldots$;

c) $3 + 6 + 9 + 12 + \ldots$ și $3 \cdot 6 \cdot 9 \cdot 12 \cdot \ldots$;

d) $4 + 8 + 12 + 16 + \ldots$ și $4 \cdot 8 \cdot 12 \cdot 16 \cdot \ldots$.

Exemplu: Pentru $n=3$ avem $1 + 3 + 5 = 9$; $1 \cdot 3 \cdot 5 = 15$.

❽ Calculați suma primilor *n* termeni:

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\frac{1}{6}+\ldots.$$

Indicație: Utilizați în ciclu instrucțiunea **if** `odd(...)` **then** ... **else** ... .

3.13. Instrucțiunea compusă

Instrucțiunea în studiu are următoarea sintaxă:

<*Instrucțiune compusă*> ::= **begin** <*Instrucțiune*> {**;** <*Instrucțiune*>} **end**

Diagrama sintactică este prezentată în *figura 3.10*.

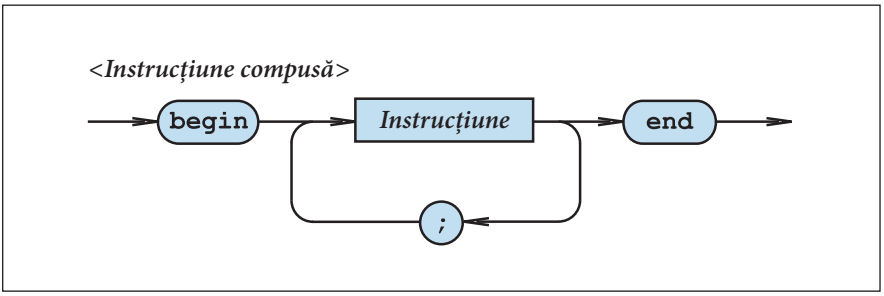


Fig. 3.10. Diagrama sintactică <*Instrucțiune compusă*>

Exemple:

1)
```
begin
  a:=x+12;
  p:=q and r;
  writeln(p)
end;
```

2)
```
begin
  write('x=');
  readln(x)
end;
```

Cuvintele-cheie **begin** și **end** joacă rolul unor "paranteze". Mulțimea instrucțiunilor cuprinse între aceste paranteze, din punctul de vedere al limbajului, formează o singură instrucțiune. Prin urmare instrucțiunea compusă este utilă pentru a plasa mai multe instrucțiuni în locurile din programe în care este permisă numai o singură instrucțiune (vezi instrucțiunile **if**, **case**, **for** ș.a.).

Exemple:

1)
```
if a>0 then begin x:=a+b; y:=a*b end
       else begin x:=a-b; y:=a/b end;
```

2)
```
case c of
  '+' : begin y:=a+b; writeln('Adunarea') end;
  '-' : begin y:=a-b; writeln('Scaderea') end;
  '*' : begin y:=a*b; writeln('Inmultirea') end;
  '/' : begin y:=a/b; writeln('Impartirea') end;
end;
```

3)
```
for i:=1 to n do
  begin
    write('x=');
    readln(x);
    s:=s+x
  end;
```

De menționat că partea executabilă a oricărui program este o instrucțiune compusă, adică o secvență de instrucțiuni încadrată între "parantezele" **begin** și **end**.

Întrucît simbolul "**;**" nu termină, ci separă instrucțiunile, el nu este necesar înaintea cuvîntului-cheie **end**. Cu toate acestea, mulți programatori inserează acest simbol cu scopul de a continua, în caz de necesitate, lista respectivă de instrucțiuni. Amintim că apariția adițională a unui simbol "**;**" semnifică inserarea unei instrucțiuni de efect nul.

Pentru a face programele mai lizibile, cuvintele **begin** și **end** se scriu strict unul sub altul, iar instrucțiunile dintre "paranteze" – cu cîteva poziții mai la dreapta. Dacă instrucțiunea compusă **begin...end** este inclusă în componența altor instrucțiuni (**if**, **case**, **for** ș.a.), cuvintele-cheie **begin** și **end** se scriu deplasate la dreapta.

Pentru exemplificare, prezentăm programul P64, în care se calculează media aritmetică a *n* numere, citite de la tastatură.

```
Program P64;
 { Media aritmetica a n numere }
var x, suma, media : real;
    i, n : integer;
begin
 write('n='); readln(n);
 suma:=0;
 writeln('Dati ', n, ' numere:');
 for i:=1 to n do
   begin
    write('x='); readln(x);
    suma:=suma+x;
   end;
 if n>0 then
          begin
            media:=suma/n;
            writeln('media=', media);
          end
        else writeln('media=*****');
 readln;
end.
```

Întrebări și exerciții

❶ Care este destinația instrucțiunii compuse?

❷ Indicați pe diagrama sintactică din *figura 3.10* drumurile care corespund instrucțiunii compuse din programul P64.

❸ Elaborați un program care citește de la tastatură *n* numere și afișează pe ecran:
a) suma și media aritmetică a numerelor citite;
b) suma și media aritmetică a numerelor pozitive;
c) suma și media aritmetică a numerelor negative.

❹ Elaborați un program care citește de la tastatură *n* caractere și afișează pe ecran:
a) numărul cifrelor zecimale citite;
b) numărul cifrelor pare;
c) numărul cifrelor impare;
d) numărul literelor citite;
e) numărul vocalelor;
f) numărul consoanelor.
Caracterele introduse se separă prin acționarea tastei <ENTER>. Sînt admise cifrele zecimale 0, 1, 2, ..., 9 și literele mari *A, B, C, ..., Z* ale alfabetului latin.

3.14. Instrucțiunea `while`

Instrucțiunea **`while`** conține o expresie booleană care controlează execuția repetată a altei instrucțiuni. Sintaxa instrucțiunii în studiu este:

<*Instrucțiune* **`while`**> ::= **`while`** <*Expresie booleană*> **`do`** <*Instrucțiune*>

Diagrama sintactică este prezentată în *figura 3.11*.

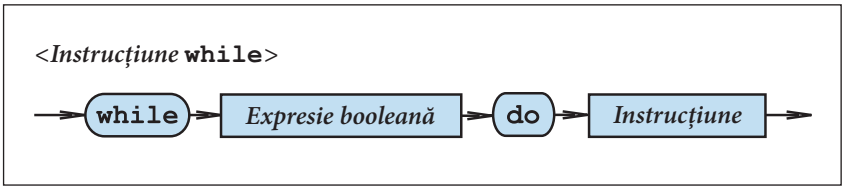


Fig. 3.11. Diagrama sintactică a instrucțiunii **`while`**

Exemple:

1)
```
while x>0 do x:=x-1;
```

2)
```
while x<3.14 do
  begin
    x:=x+0.001;
    writeln(sin(x));
  end;
```

3)
```
while p do
  begin
    x:=x+0.001;
    y:=10*x;
    p:=y<1000;
  end;
```

Instrucțiunea situată după cuvîntul-cheie **do** se execută repetat atîta timp, cît valoarea expresiei booleene este `true`. Dacă expresia booleană ia valoarea `false`, instrucțiunea de după **do** nu se mai execută. Se recomandă ca expresia booleană să fie cît mai simplă, deoarece ea este evaluată la fiecare iterație.

În mod obișnuit, instrucțiunea **while** se folosește pentru organizarea calculelor repetitive cu variabile de control de tip `real`.

În programul ce urmează, instrucțiunea **while** este utilizată pentru afișarea valorilor funcției $y = 2x$. Argumentul x ia valori de la x_1 la x_2 cu pasul Δx.

```
Program P65;
 { Tabelul functiei y=2*x }
var x, y, x1, x2, deltaX : real;
begin
  write('x1='); readln(x1);
  write('x2='); readln(x2);
  write('deltaX='); readln(deltaX);
  writeln('x':10, 'y':20);
  writeln;
  x:=x1;
  while x<=x2 do
    begin
      y:=2*x;
      writeln(x:20, y:20);
      x:=x+deltaX;
    end;
 readln;
end.
```

Instrucțiunea **while** se consideră deosebit de utilă în situația în care numărul de execuții repetate ale unei secvențe de instrucțiuni este dificil de evaluat.

Pentru exemplificare, prezentăm programul P66, care afișează pe ecran media aritmetică a numerelor pozitive citite de la tastatură.

```
Program P66;
 { Media aritmetica a numerelor pozitive
   citite de la tastatura }
var x, suma : real;
    n : integer;
begin
  n:=0;
```

```
  suma:=0;
  writeln('Dati numere pozitive:');
  readln(x);
  while x>0 do
    begin
      n:=n+1;
      suma:=suma+x;
      readln(x);
    end;
  writeln('Ati introdus ', n, ' numere pozitive.');
  if n>0 then writeln('media=', suma/n)
         else writeln('media=*****');
  readln;
end.
```

Se observă că numărul de execuții repetate ale instrucțiunii compuse **`begin ... end`** din componența instrucțiunii **`while`** nu poate fi calculat din timp. Execuția instrucțiunii **`while`** se termină cînd utilizatorul introduce un număr $x \le 0$.

Întrebări și exerciții

❶ Cum se execută o instrucțiune **`while`**?

❷ Indicați pe diagramele sintactice din *figura 3.11* drumurile care corespund instrucțiunilor **`while`** din programele P65 și P66.

❸ Utilizînd instrucțiunea **`while`**, scrieți un program care afișează pe ecran valorile funcției y pentru valori ale argumentului de la x_1 la x_2 cu pasul Δx:

a) $y = \frac{x}{3} + 2;$

b) $y = \frac{x}{2};$

c) $y = 3x - 4;$

d) $y = 4x - 13.$

❹ Utilizatorul introduce de la tastatură numere întregi pozitive, separate prin acționarea tastei <ENTER>. Sfîrșitul secvenței de numere e indicat prin introducerea numărului 0. Scrieți un program care afișează pe ecran:
a) suma și media aritmetică a numerelor pare;
b) suma și media aritmetică a numerelor impare.

❺ Scrieți un program care afișează pe ecran valorile funcției $y = f(x)$. Argumentul x ia valori de la x_1 la x_2 cu pasul Δx:

a) $y = \begin{cases} x, & x > 3; \\ 2x, & x \le 3; \end{cases}$

b) $y = \begin{cases} 6x, & x \ge 0; \\ 4x, & x < 0; \end{cases}$

c) $y = \begin{cases} x + 6, & x > 5; \\ x - 6, & x \le 5; \end{cases}$

d) $y = \begin{cases} 3 - x, & x > 4; \\ 3 + x, & x \le 4. \end{cases}$

Exemplu: $y = \begin{cases} x+1, & x > 8; \\ x-2, & x \le 8. \end{cases}$

```
Program P67;
 { Tabelul functiei }
var x, y, x1, x2, deltaX : real;
begin
  write('x1='); readln(x1);
  write('x2='); readln(x2);
  write('deltaX='); readln(deltaX);
  writeln('x':10, 'y':20);
  writeln;
  x:=x1;
  while x<=x2 do
    begin
      if x>8 then y:=x+1 else y:=x-2;
      writeln(x:20, y:20);
      x:=x+deltaX;
    end;
  readln;
end.
```

❻ Instrucțiunea repetitivă

```
for i:=i1 to i2 do writeln(ord(i))
```

este echivalentă cu secvența de instrucțiuni

```
i:=i1;
while i<=i2 do
  begin
    writeln(ord(i));
    i:=succ(i);
  end.
```

Scrieți o secvență echivalentă pentru instrucțiunea repetitivă

```
for i:=i1 downto i2 do writeln(ord(i))
```

❼ Se consideră declarațiile:

```
var x1, x2, deltaX : real;
    i, n : integer;
```

Care din secvențele de instrucțiuni ce urmează sînt echivalente?

a)
```
x:=x1;
while x<=x2 do
begin
  writeln(x);
  x:=x+deltaX;
end;
```

b)
```
n:=trunc((x2-x1)/deltaX)+1;
x:=x1;
for i:=1 to n do
```

```
  begin
    writeln(x);
    x:=x+deltaX;
  end;
```

c)
```
n:=round((x2-x1)/deltaX)+1;
x:=x1;
  for i:=1 to n do
    begin
      writeln(x);
      x:=x+deltaX;
    end;
```

Argumentați răspunsul.

3.15. Instrucțiunea `repeat`

Instrucțiunea **`repeat`** indică repetarea unei secvențe de instrucțiuni în funcție de valoarea unei expresii booleene. Sintaxa instrucțiunii este:

<*Instrucțiune* **`repeat`**> ::= **`repeat`** <*Instrucțiune*> {**`;`** <*Instrucțiune*>} **`until`** <*Expresie booleană*>

Diagrama sintactică este prezentată în *figura 3.12.*

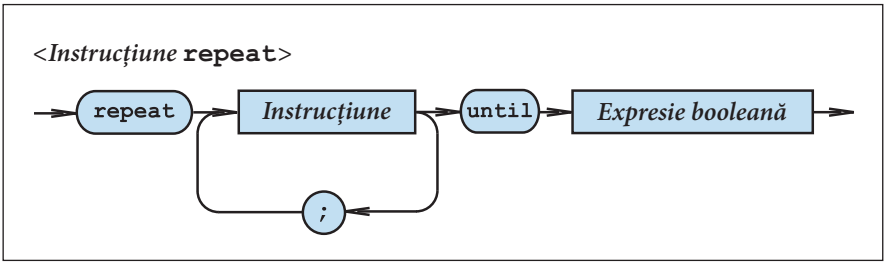


Fig. 3.12. Diagrama sintactică a instrucțiunii **`repeat`**

Exemple:

1)
```
repeat x:=x-1 until x<0;
```

2)
```
repeat
  y:=y+delta;
  writeln(y)
until y>20.5;
```

3)
```
repeat
  readln(i);
  writeln(odd(i))
until i=0;
```

Instrucțiunile situate între cuvintele-cheie **repeat** și **until** se execută repetat atît timp cît expresia booleană este falsă. Cînd această expresie devine adevărată, se trece la instrucțiunea următoare. Instrucțiunile aflate între **repeat** și **until** vor fi executate cel puțin o dată, deoarece evaluarea expresiei logice are loc după ce s-a executat această secvență.

În mod obișnuit, instrucțiunea **repeat** se utilizează în locul instrucțiunii **while** atunci cînd evaluarea expresiei care controlează repetiția se face după executarea secvenței de repetat.

Programul ce urmează afișează pe ecran paritatea numerelor întregi citite de la tastatură.

```
Program P68;
 { Paritatea numerelor citite de la tastatura }
var i : integer;
begin
 writeln('Dati numere intregi:');
 repeat
   readln(i);
   if odd(i) then writeln(i:6, ' - numar impar')
             else writeln(i:6, ' - numar par');
 until i=0;
 readln;
end.
```

Execuția instrucțiunii **repeat** se termină cînd utilizatorul introduce `i=0`.

Instrucțiunea **repeat** este utilizată foarte des pentru validarea (verificarea corectitudinii) datelor introduse de la tastatură.

De exemplu, presupunem că se cere scrierea unui program care citește de la tastatură numărul real x și afișează pe ecran rădăcina pătrată $y = \sqrt{x}$. Evident, valorile negative ale variabilei x sînt inadmisibile.

```
Program P69;
 { Calcularea radacinii patrate }
var x, y : real;
begin
  repeat
    write('Introduceti numarul nenegativ  x=');
    readln(x);
  until x>=0;
  y:=sqrt(x);
  writeln('Radacina patrata  y=', y);
  readln;
end.
```

După lansarea programului `P69` în execuție, utilizatorul este invitat să introducă un număr mai mare sau egal cu zero. Dacă, din greșeală, utilizatorul va introduce un număr negativ, instrucțiunile `write` și `readln` din componența instrucțiunii **repeat** vor fi

executate din nou. Procesul repetitiv va continua pînă cînd utilizatorul nu va introduce un număr corect.

Din exemplele studiate se observă că instrucțiunea **repeat** este utilă în situația în care numărul de executări repetate ale unei secvențe de instrucțiuni este dificil de evaluat.

Întrebări și exerciții

❶ Cum se execută o instrucțiune **repeat**?

❷ Indicați pe diagramele sintactice din *figura 3.12* drumurile care corespund instrucțiunilor **repeat** din programele P68 și P69.

❸ Se consideră instrucțiunile:

a)
```
repeat
   <Instructiune 1>;
   <Instructiune 2>;
   ...
   <Instructiune n>;
until p
```

b)
```
while not p do
begin
   <Instructiune 1>;
   <Instructiune 2>;
   ...
   <Instructiune n>;
end.
```

Sînt oare echivalente aceste instrucțiuni? Argumentați răspunsul.

❹ Elaborați un program care citește de la tastatură o secvență de caractere și afișează pe ecran:
a) numărul cifrelor zecimale citite;
b) numărul cifrelor pare;
c) numărul cifrelor impare.
Caracterele introduse se separă prin acționarea tastei <ENTER>. Sînt admise cifrele zecimale 0, 1, 2, ..., 9 și caracterul * care indică sfîrșitul secvenței.

❺ Elaborați un program care citește de la tastatură numărul real x și afișează pe ecran valoarea expresiei $\frac{1}{x}$. Evident, valoarea $x = 0$ este inadmisibilă. Validați datele introduse de la tastatură cu ajutorul instrucțiunii **repeat**.

❻ Este oare echivalentă instrucțiunea

```
for i:=i1 to i2 do writeln(ord(i))
```

cu secvența de instrucțiuni

```
i:=i1;
repeat
  writeln(ord(i));
  i:=succ(i);
until i>i2;
```

Argumentați răspunsul.

❼ Scrieți un program care afișează pe ecran valorile funcției $y = f(x)$. Argumentul x ia valori de la x_1 la x_2 cu pasul Δx. Ciclul se va organiza cu ajutorul instrucțiunii **repeat**.

a) $y = 2x;$ *c)* $y = x - 4;$

b) $y = \frac{x}{3} + 9;$ *d)* $y = \frac{x}{8} - 6.$

❽ Creați un program care citește de pe tastatură o secvență de caractere și afișează pe ecran:

a) numărul literelor citite;

b) numărul literelor mari;

c) numărul literelor mici.

Caracterele introduse se separă prin acționarea tastei <ENTER>. Sînt admise literele mari și mici ale alfabetului latin și caracterul * care indică sfîrșitul secvenței.

3.16. Instrucțiunea goto

Instrucțiunile unui program sînt executate secvențial, așa cum apar scrise în textul programului. Instrucțiunea de salt necondiționat **goto** oferă posibilitatea de a întrerupe această secvență și de a relua execuția dintr-un alt loc al programului. Instrucțiunea în studiu are sintaxa:

<*Instrucțiune* **goto**> ::= **goto** <*Etichetă*>

Amintim că eticheta este un număr întreg fără semn care prefixează o instrucțiune a programului (*fig. 3.1*). Etichetele unui program sînt listate în partea declarativă a programului după cuvîntul-cheie **label**. Sintaxa acestei declarații este:

<*Etichete*> ::= **label** <*Etichetă*> {, <*Etichetă*>}**;**

Diagramele sintactice ale unităților gramaticale propuse sînt prezentate în *figura 3.13*. De reținut că declararea prin **label** a etichetelor este obligatorie.

Execuția instrucțiunii **goto** are ca efect transferul controlului la instrucțiunea prefixată de eticheta respectivă.

Pentru exemplificare prezentăm programul P70 care calculează valoarea funcției:

$$y = \begin{cases} x, & x \geq 0; \\ 2x, & x < 0. \end{cases}$$

Valorile argumentului x se citesc de la tastatură.

```
Program P70;
 { Executia instructiunii goto }
label 1, 2;
var x, y : real;
```

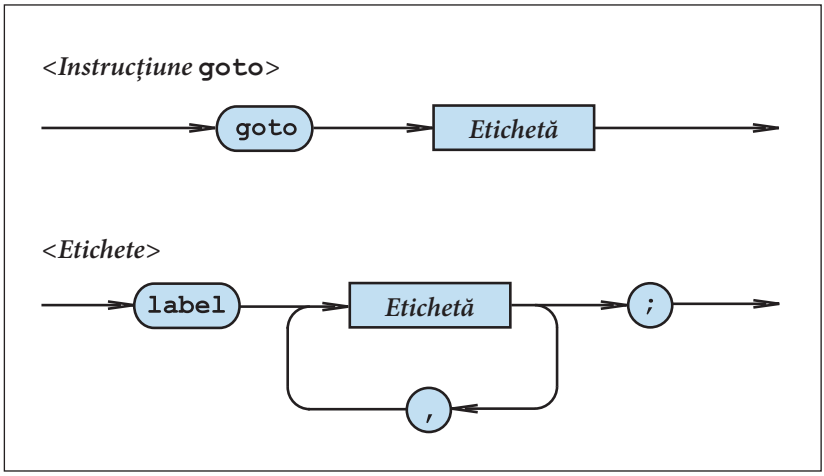


Fig. 3.13. Diagramele sintactice <*Instrucțiune* **goto**> și <*Etichete*>

```
begin
  write('x='); readln(x);
  if x>=0 then goto 1;
  y:=2*x;
  writeln('x<0, y=', y);
  goto 2;
1: y:=x;
  writeln('x>=0, y=', y);
2: readln;
end.
```

Dacă utilizatorul tastează o valoare $x \geq 0$, se va executa instrucțiunea **goto** 1 și controlul va trece la instrucțiunea de atribuire

```
1: y:=x
```

După aceasta, se vor executa instrucțiunile

```
   writeln('x>=0 , y=', y);
2: readln
```

Dacă utilizatorul tastează o valoare $x < 0$, se execută instrucțiunile

```
y:= 2*x;
writeln('x<0 , y=', y);
goto 2
```

Ultima instrucțiune va transfera controlul instrucțiunii

```
2: readln
```

Etichetele și instrucțiunile unui program trebuie să respecte următoarele reguli:
1) orice etichetă trebuie să fie declarată cu ajutorul cuvîntului-cheie **label**;
2) orice etichetă trebuie să prefixeze o singură instrucțiune;

3) este interzis saltul din afara unei instrucțiuni structurate (**if**, **for**, **while** etc.) în interiorul ei.

Exemple:

1)
```
if i>5 then 1: writeln('i>5') else writeln('i<=5');
...
goto 1; {Eroare}
```

2)
```
for i:=1 to 10 do
begin
10: writeln('i=', i);
end;
...
goto 10; {Eroare}
```

În lipsa lui **goto** instrucțiunile unui program sînt executate în ordinea în care sînt scrise. Prin urmare instrucțiunile **goto** încalcă concordanța dintre textul programului și ordinea de execuție a instrucțiunilor. Acest fapt complică elaborarea, verificarea și depanarea programelor. În consecință folosirea instrucțiunii **goto** nu este recomandată.

De exemplu, programul P70 poate fi refăcut după cum urmează:

```
Program P71;
 { Excluderea instructiunii goto din programul P70 }
var x, y : real;
begin
  write('x=');
  readln(x);
  if x>=0 then
  begin
    y:=x;
    writeln('x>=0, y=', y);
  end
    else
  begin
    y:=2*x;
    writeln('x<0, y=', y);
  end;
 readln;
end.
```

De regulă, instrucțiunea **goto** se utilizează în cazuri extraordinare, de exemplu, pentru a mări viteza de derulare sau pentru a micșora lungimea unui program.

Întrebări și exerciții

❶ Care este destinația instrucțiunii **goto**?

❷ Indicați pe diagramele sintactice din *figura 3.13* drumurile care corespund etichetelor și instrucțiunilor **goto** din programul P70.

❸ Transcrieți fără a aplica instrucțiunea **goto**:

```
Program P72;
 { Afisarea formulelor de salut }
label 1, 2, 3;
var i : 6..23;
begin
  write('Cit e ora?'); readln(i);
  if i>12 then goto 1;
  writeln('Buna dimineata!');
  goto 3;
1: if i>17 then goto 2;
  writeln('Buna ziua!');
  goto 3;
2: writeln('Buna seara!');
3: readln;
end.
```

❹ Comentați următorul program:

```
Program P73;
 { Eroare }
label 1;
var i : 1..5;
begin
  i:=1;
1: writeln(i);
  i:=i+1;
  goto 1;
end.
```

❺ Ce va afișa pe ecran programul ce urmează?

```
Program P74;
 { Eroare }
label 1;
var x : real;
begin
  x:=0;
1: writeln(x);
  x:=x+1e-30;
  goto 1;
end.
```

Amintim că derularea unui program poate fi întreruptă prin acționarea tastelor <CTRL+C> sau <CTRL+BREAK>.

❻ Comentați programul de mai jos:

```
Program P75;
 { Eroare }
label 1;
var i : integer;
begin
  i:=1;
while i<=20 do
    begin
      writeln(i);
      1: i:=i+1;
    end;
  goto 1;
end.
```

3.17. Generalități despre structura unui program PASCAL

Un program PASCAL are următoarea structură:

<*Program*> ::= <*Antet program*>
<*Corp*>**.**

Antetul specifică numele programului și include, opțional, o listă de parametri formali:

<*Antet program*> ::= **Program** <*Identificator*> [(<*Identificator*> {**,** <*Identificator*>})]**;**

Exemple:

1) **Program** A1;

2) **Program** B6(Intrare);

3) **Program** C15(Intrare, Iesire);

În mod obișnuit, parametrii formali se folosesc pentru comunicarea programului cu mediul său. Aceștia vor fi studiați mai amănunțit în capitolele următoare.

Corpul unui program este compus din partea declarativă și partea executabilă:

<*Corp*> ::= <*Declarații*>
<*Instrucțiune compusă*>

Partea declarativă a programului are următoarea sintaxă:

<*Declarații*> ::= [<*Etichete*>]
[<*Constante*>]
[<*Tipuri*>]

[<*Variabile*>]
[<*Subprograme*>]

Partea executabilă a unui program este o instrucțiune compusă **`begin...end`**.

Diagramele sintactice ale unităților gramaticale în studiu sînt prezentate în *figura 3.14*.

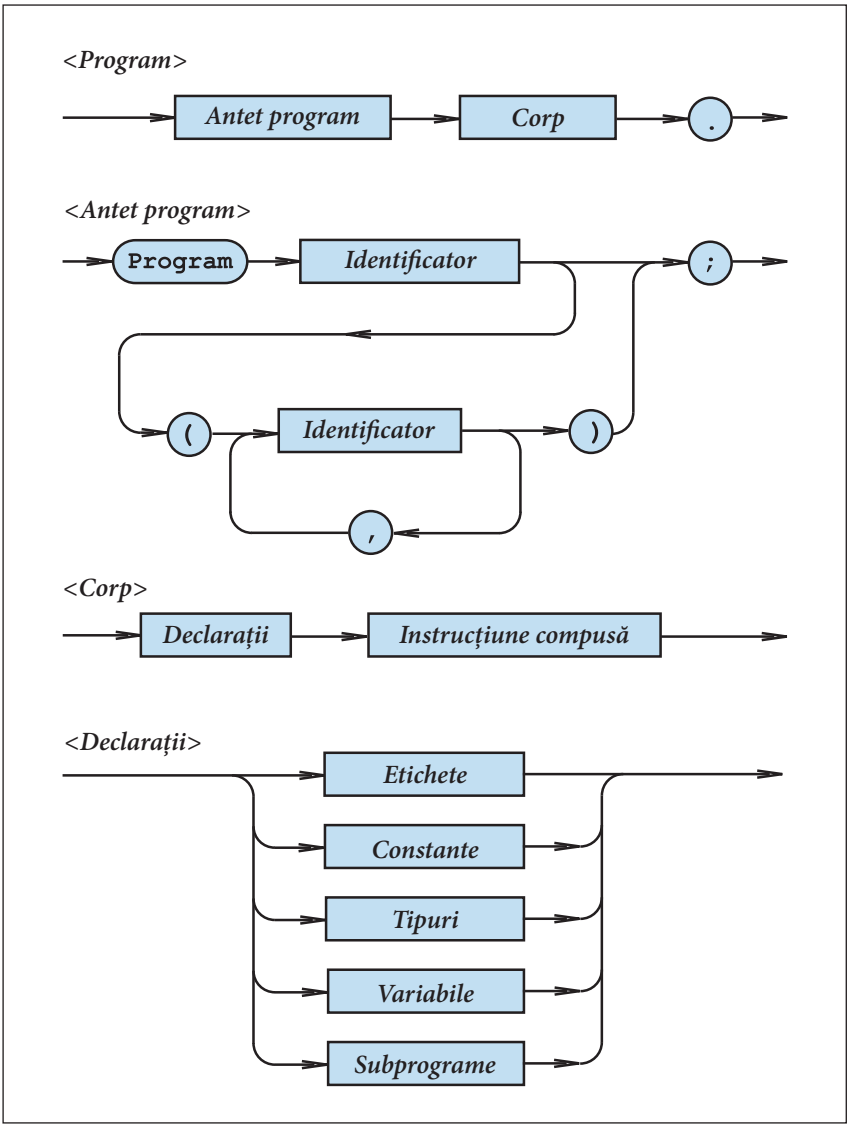


Fig. 3.14. Structura unui program PASCAL

Subliniem faptul că sfîrșitul unui program PASCAL este indicat de simbolul "`.`" (punct).

Pentru exemplificare, prezentăm în continuare programul `P76`, care calculează lungimea arcului de cerc de α grade și aria sectorului respectiv.

```
Program P76;
  { Lungimea arcului de cerc si aria sectorului
    respectiv }
label 1, 2;
const Pi=3.141592654;
type  grade=0..360;
var   alfa : grade;
      raza, lungimea, aria : real;
begin
  write('raza='); readln(raza);
  if raza<0 then goto 1;
  write('alfa='); readln(alfa);
  lungimea:=Pi*raza*alfa/180;
  writeln('lungimea=', lungimea);
  aria:=Pi*sqr(raza)*alfa/360;
  writeln('aria=', aria);
  goto 2;
1: writeln('Eroare: raza<0');
2: readln;
end.
```

Întrebări și exerciții

❶ Care este destinația antetului de program? Cum se indică sfîrșitul unui program PASCAL?
❷ Indicați pe diagramele sintactice din *figura 3.14* drumurile care corespund unităților gramaticale ale programului `P76`.
❸ Transcrieți programul `P76` fără a utiliza instrucțiunea **`goto`**. Indicați partea declarativă și partea executabilă a programului elaborat.
❹ Care este destinația parametrilor dintr-un antet de program?

Test de autoevaluare nr. 3

1. Scrieți conform regulilor limbajului PASCAL expresiile:

a) $(a + b) - 2ab$;
b) $6a^2 + 15ab - 13b^2$;
c) $(a + b)\,(a - b)$;
d) $2\alpha\beta - 5\pi r$;
e) $\pi r^2 + \alpha\beta^2$;
f) $xy \vee xz$.

2. Transpuneți expresiile PASCAL în notații obișnuite:

a) `sqr(a)+2/sqr(b)`
b) `2*a/(b+c)`
c) `15*sqrt(a/(a-b))`
d) `not(x and y) or z`
e) `sqr((a+b)/2)`
f) `(x<>0) and (q<p)`

3. Care din expresiile PASCAL ce urmează sînt greşite?

a) `2*a+2*b`

b) `4*sinx+4*cosy`

c) `3*sqr(x)+3/sin(y)`

d) `a+2*-b`

e) `not (q and p)`

f) `2*(+x)+((-y))`

4. Fie x=1, y=2 şi z=3. Evaluaţi expresiile:

a) `x+2*y+3*z`

b) `(1+x+y-2)*z`

c) `x*y+y*(-z)`

d) `not(x+y+z>0)`

e) `x*y<y+z`

f) `(x>y)or(2*x<y+z)`

5. În prezenţa declaraţiilor:

```
var  x   : real;
     i   : integer;
     p   : boolean;
     s   : char;
     Zi  : (Luni, Marti, Miercuri, Joi, Vineri, Simbata,
            Duminica);
```

aflaţi tipul următoarelor expresii:

a) `i mod 9`

b) `i/9`

c) `i+x`

d) `ord(pred(Zi))`

e) `ord(Zi)+trunc(x)`

f) `sqr(ord(s))`

g) `p or (x>i)`

h) `chr(i+ord(p))`

6. Scrieţi un program PASCAL care afişează pe ecran valorile expresiei $15i\,(x + y)$. Valoarea variabilei întregi *i* şi valorile variabilelor reale *x*, *y* se citesc de la tastatură.

7. Se consideră următoarele declaraţii:

```
type  FunctiaOcupata = (Muncitor, SefDeEchipa, Maistru,
                        SefDeSantier, Director);
      StareaCivila = (Casatorit, Necasatorit);
var  i : integer;
     x : real;
     f : FunctiaOcupata;
     s : StareaCivila;
```

Care din instrucţiunile ce urmează sînt corecte?

a) `i:=ord(f)+15`

b) `f:=Casatorit`

c) `x:=ord(f)+1`

d) `i:=2*x-15`

e) `s:=pred(s)`

f) `f:=succ(SefdeEchipa)`

8. Elaborați un program care calculează valorile funcției:

$$y=\begin{cases}9x+3x^2, x>15;\\ 3x-5\sqrt{x+28}, x\le 15.\end{cases}$$

Valorile variabilei reale *x* se citesc de la tastatură.

9. Monedele uzuale ale Republicii Moldova au valoarea de 1, 5, 10, 25 sau 50 de bani. Elaborați un program PASCAL care citește de la tastatură valoarea numerică a monedei și afișează pe ecran valoarea respectivă, exprimată prin cuvinte. De exemplu, dacă utilizatorul tastează "25", pe ecran se va afișa "douăzeci și cinci de bani". Dacă utilizatorul tastează un număr ce diferă de 1, 5, 10, 25 sau 50, pe ecran se va afișa mesajul "valoare inadmisibilă".

10. Utilizînd instrucțiunea **`for`**, scrieți un program care calculează pentru primii *n* termeni suma

$$s=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{n}$$

și produsul

$$p=\frac{1}{1}\cdot\frac{1}{2}\cdot\frac{1}{3}\cdot...\cdot\frac{1}{n}.$$

11. Utilizînd instrucțiunea **`while`**, scrieți un program care afișează pe ecran valorile funcției:

$$y=\begin{cases}2\sqrt{x+6}, x\ge 4;\\ 3-abs(x), x<4\end{cases}$$

pentru valori ale argumentului de la x_1 la x_2 cu pasul Δx.

12. Mesajele speciale ale telefoniei mobile se definesc cu ajutorul formulelor metalingvistice:

<*Cifră*> ::= `0|1|2|3|4|5|6|7|8|9`
<*Mesaj special*> ::= *{<*Cifră*>}#

Utilizînd instrucțiunea **`repeat`**, scrieți un program care afișează pe ecran numărul de cifre din mesajele speciale. Caracterele introduse se separă prin acționarea tastei <ENTER>. De exemplu, dacă utilizatorul tastează:

```
*<ENTER>
1<ENTER>
0<ENTER>
4<ENTER>
#<ENTER>
```

pe ecran se va afișa numărul 3.

Capitolul 4

TIPURI DE DATE STRUCTURATE UNIDIMENSIONALE

4.1. Tipuri de date *tablou* (`array`) *unidimensional*

Mulțimea de valori ale unui tip de date **`array`** este constituită din tablouri (tabele). Tablourile sînt formate dintr-un număr fixat de componente de același tip, denumit **tip de bază**. Referirea componentelor se face cu ajutorul unui **indice**.

Un tip de date *tablou unidimensional* se definește printr-o construcție de forma

```
type <Nume tip> = array [T1] of T2;
```

unde T_1 este tipul indicelui care trebuie să fie ordinal, iar T_2 este tipul componentelor (tipul de bază) care poate fi un tip oarecare.

Exemple:

1)
```
type  Vector = array [1..5] of real;
var   x : Vector;
```

2)
```
type  Zi = (L, Ma, Mi, J, Vi, S, D);
      Venit = array [Zi] of real;
var   v : Venit;
      z : Zi;
```

3)
```
type  Ora = 0..23;
      Grade = -40..40;
      Temperatura = array [Ora] of Grade;
var   t : Temperatura;
      h : Ora;
```

Structura datelor din exemplele în studiu este prezentată în *figura 4.1*.

Fiecare componentă a unei variabile de tip *tablou unidimensional* poate fi specificată explicit, prin numele variabilei urmat de indicele respectiv încadrat de paranteze pătrate.

Exemple:

1) `x[1], x[4];`

2) `v[L], v[Ma], v[J];`

3) `t[0], t[15], t[23];`

4) `v[z], t[h].`

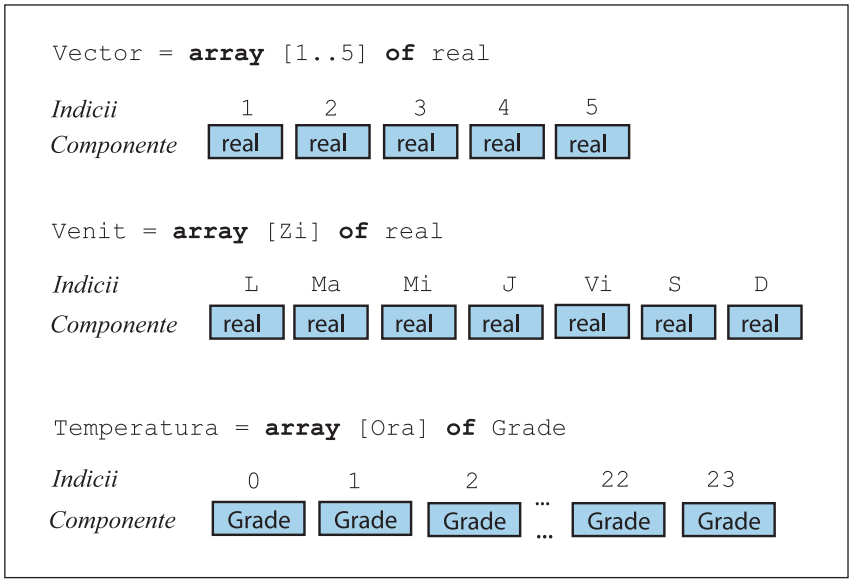


Fig. 4.1. Structura datelor de tip `Vector`, `Venit` şi `Temperatura`

Asupra componentelor datelor de tip *tablou unidimensional* se pot efectua toate operațiile admise de tipul de bază respectiv. Programul ce urmează afișează pe ecran suma componentelor variabilei `x` de tip `Vector`. Valorile componentelor `x[1]`, `x[2]`, ..., `x[5]` se citesc de la tastatură.

```
Program P77;
{ Suma componentelor variabilei x de tip Vector }
type Vector = array [1..5] of real;
var x : Vector;
    i : integer;
    s : real;
begin
  writeln('Dati 5 numere:');
  for i:=1 to 5 do readln(x[i]);
  writeln('Ati introdus:');
  for i:=1 to 5 do writeln(x[i]);
  s:=0;
  for i:=1 to 5 do s:=s+x[i];
  writeln('Suma=', s);
  readln;
end.
```

Pentru a extinde aria de aplicare a unui program, se recomandă ca numărul de componente ale datelor de tip **`array`** să fie specificat prin constante.

De exemplu, programul `P77` poate fi modificat pentru a însuma n numere reale, $n \leq 100$:

```
Program P78;
{ Extinderea domeniului de aplicare a programului P77 }
const nmax = 100;
type Vector = array [1..nmax] of real;
var x : Vector;
    n : 1..nmax;
    i : integer;
    s : real;
begin
  write('n='); readln(n);
  writeln('Dati ', n, ' numere:');
  for i:=1 to n do readln(x[i]);
  writeln('Ati introdus:');
  for i:=1 to n do writeln(x[i]);
  s:=0;
  for i:=1 to n do
    s:=s+x[i];
  writeln('Suma=', s);
  readln;
end.
```

În general, un tip *tablou unidimensional* se definește cu ajutorul diagramelor sintactice din *figura 4.2*. Atributul **packed** (împachetat) indică cerința de optimizare a spațiului de memorie pentru elementele tipului **array**. Menționăm că în majoritatea compilatoarelor actuale utilizarea acestui atribut nu are niciun efect, întrucît optimizarea se efectuează în mod automat.

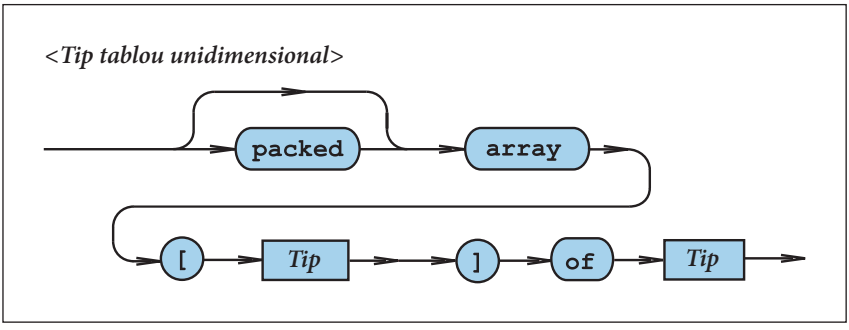


Fig. 4.2. Diagrama sintactică <*Tip tablou unidimensional*>

Fiind date două variabile de tip *tablou* de același tip, numele variabilelor pot apărea într-o instrucțiune de atribuire. Această atribuire înseamnă copierea tuturor componentelor din membrul drept în cel stîng.

De exemplu, în prezența declarațiilor

```
var a, b : Vector;
```

instrucțiunea

```
a:=b
```

este corectă.

În exemplele de mai sus tipul de bază (tipul componentelor) a fost de fiecare dată un tip simplu. Deoarece tipul de bază poate fi, în general, un tip arbitrar, devine posibilă definirea tablourilor cu componente de tip structurat. Considerăm acum un exemplu în care tipul de bază este el însuși un tip **`array`**:

```
Type Linie = array [1..4] of real;
     Tabel = array [1..3] of Linie;
var L : Linie;
    T : Tabel;
    x : real;
```

Variabila `T` este formată din 3 componente: `T[1]`, `T[2]` și `T[3]` de tip `Linie`. Variabila `L` este formată din 4 componente: `L[1]`, `L[2]`, `L[3]` și `L[4]` de tip `real`.

Prin urmare atribuirile

```
L[1]:=x; x:=L[3]; T[2]:=L; L:=T[1]
```

sînt corecte.

Elementele variabilei `T` pot fi specificate prin `T[i][j]` sau prescurtat `T[i,j]`. Aici `i` indică numărul componentei de tip `Linie` în cadrul variabilei `T`, iar `j` – numărul componentei de tip `real` în cadrul componentei `T[i]` de tip `Linie`.

În programele PASCAL tablourile se utilizează pentru a grupa sub un singur nume mai multe variabile cu caracteristici identice.

Întrebări și exerciții

❶ Precizați tipul indicilor și tipul componentelor din următoarele declarații:

```
type P = array [1..5] of integer;
     Culoare = (Galben, Verde, Albastru, Violet);
     R = array [Culoare] of real;
     T = array [boolean] of Culoare;
```

Reprezentați structura datelor de tipul `P`, `R` și `T` pe un desen (*fig. 4.1*).

❷ Indicați pe diagrama sintactică din *figura 4.2* drumurile care corespund declarațiilor din exercițiul 1.

❸ Scrieți formulele metalingvistice care corespund diagramei sintactice <*Tip tablou*> din *figura 4.2*.

❹ Se consideră declarațiile:

```
type Vector = array [1..5] of real;
var x, y : Vector;
```

Scrieți expresia aritmetică a cărei valoare este:

a) suma primelor trei componente ale variabilei `x`;

b) suma tuturor componentelor variabilei `y`;

c) produsul tuturor componentelor variabilei `x`;
d) valoarea absolută a componentei a treia a variabilei `y`;
e) suma primelor componente ale variabilelor `x` și `y`.

❺ Se consideră declarațiile:

```
type Zi = (L, Ma, Mi, J, Vi, S, D);
     Venit = array [Zi] of real;
var v : Venit;
```

Componentele variabilei `v` reprezintă venitul zilnic al unei întreprinderi. Elaborați un program care:
a) calculează venitul săptămînal al întreprinderii;
b) calculează media venitului zilnic;
c) indică ziua în care s-a obținut cel mai mare venit;
d) indică ziua cu venitul cel mai mic.

❻ Se consideră declarațiile:

```
type Ora = 0..23;
     Grade = -40..40;
     Temperatura = array [Ora] of Grade;
var t : Temperatura;
```

Componentele variabilei `t` reprezintă temperaturile măsurate din oră în oră pe parcursul a 24 de ore. Elaborați un program care:
a) calculează temperatura medie;
b) indică maximul și minimul temperaturii;
c) indică ora (orele) la care s-a înregistrat temperatura maximă;
d) indică ora (orele) la care s-a înregistrat temperatura minimă.

❼ Se consideră declarațiile:

```
type Oras = (Chisinau, Orhei, Balti, Tighina, Tiraspol);
     Consum = array [Oras] of real;
var C : Consum;
    r : Oras;
```

Componenta `C[r]` a variabilei `C` reprezintă consumul de energie electrică a orașului `r` pe parcursul unei săptămîni. Elaborați un program care:
a) calculează energia electrică consumată de toate orașele pe parcursul unei săptămîni;
b) indică orașul cu un consum săptămînal maxim;
c) indică orașul cu un consum săptămînal minim.

❽ Se consideră declarațiile:

```
type Vector = array [1..5] of real;
     Matrice = array [1..3] of boolean;
     Linie = array [1..4] of real;
     Tabel = array [1..3] of Linie;
var V : Vector;
    M : Matrice;
    L : Linie;
    T : Tabel;
```

```
    x : real;
    i : integer;
```

Care din atribuirile ce urmează sînt corecte?

a) `T[3]:=T[1]`

b) `M:=T`

c) `L:=V`

d) `L[3]:=x`

e) `x:=i`

f) `i:=x`

g) `L[3]:=i`

h) `i:=M[1,2]`

i) `x:=V[4]`

j) `L[3]:=V[4]`

k) `T[1]:=4`

l) `T[2]:=V`

m) `L:=T[3]`

n) `T[1,2]:=M[1,2]`

o) `T[1,2]:=M[1,2]`

p) `M[1]:=4`

q) `M[1,3]:=L[2]`

r) `x:=T[1][2]`

s) `x:=M[1]`

t) `L:=M[1]`

u) `V[5]:=M[3,4]`

v) `L:=M[3,4]`

❾ Utilizînd un tip de date *tablou*, elaborați un program care realizează algoritmul lui Eratostene pentru calcularea numerelor prime mai mici decît un număr natural dat n ($n \leq 200$).

4.2. Tipuri de date *şir de caractere*

În limbajul-standard tipul de date *șir de caractere* reprezintă un caz special al tipului **`array`** și se definește printr-o construcție de forma

```
<Nume tip> ::= packed array [1..n] of char;
```

Mulțimea de valori ale tipului de date în studiu este formată din toate șirurile ce conțin exact n caractere.

Exemplu:

```
Program P80;
{ Siruri de caractere de lungime constanta }
type Nume = packed array [1..8] of char;
     Prenume = packed array [1..5] of char;
var N : Nume;
    P : Prenume;
begin
  N:='Munteanu';
```

```
  P:='Mihai';
  writeln(N);
  writeln(P);
  readln;
end.
```

Rezultatul afișat pe ecran:

```
Munteanu
Mihai
```

Întrucît șirurile de lungime diferită aparțin unor tipuri distincte de date, în cadrul programului P80 nu sînt admise atribuiri de genul:

```
N:= 'Olaru';
P:= 'Ion';
```

În astfel de cazuri, programatorul va completa datele respective cu spațiul pînă la numărul stabilit de caractere n, de exemplu:

```
N:= 'Olaru ';
P:= 'Ion ';
```

Valorile unei variabile v de tip **packed array** [1..n] **of** char pot fi introduse de la tastatură numai prin citirea separată a componentelor respective:

```
read(v[1]); read(v[2]); ...; read(v[n]).
```

În schimb, o astfel de valoare poate fi afișată în totalitatea ei printr-un singur apel write(v) sau writeln(v).

Subliniem faptul că șirurile de caractere de tip **packed array** [1..n] **of** char conțin exact n caractere, adică sînt **șiruri de lungime constantă**. Evident, lungimea lor nu poate fi modificată pe parcursul derulării programului respectiv. Acest fapt complică elaborarea programelor destinate prelucrării unor șiruri arbitrare de caractere.

Pentru a elimina acest neajuns, versiunile actuale ale limbajului permit utilizarea șirurilor de caractere de lungime variabilă. În Turbo PASCAL un tip de date **șir de caractere de lungime variabilă** se declară printr-o construcție de forma:

type <*Nume tip*> = **string**;

sau

type <*Nume tip*> = **string** [*nmax*];

unde *nmax* este lungimea maximă pe care o pot avea șirurile respective. În lipsa parametrului *nmax* lungimea maximă se stabilește implicit, în mod obișnuit – 255 de caractere.

Asupra șirurilor de tip **string** se poate efectua operația de concatenare (juxtapunere), notată prin semnul "+". Lungimea curentă a unei valori v de tip **string** poate fi aflată cu ajutorul funcției predefinite length(v) care returnează o valoare de tip integer. Indiferent de lungime, toate șirurile de caractere de tip **string** sînt compatibile.

Exemplu:

```
Program P81;
{ Siruri de caractere de lungime variabila }
type Nume = string [8];
     Prenume = string [5];
     NumePrenume = string;
var N : Nume;
    P : Prenume;
    NP : NumePrenume;
    L : integer;
begin
  N:='Munteanu';  L:=length(N);   writeln(N, L:4);
  P:='Mihai';     L:=length(P);   writeln(P, L:4);
  NP:=N+' '+P;    L:=length(NP);  writeln(NP, L:4);
  N:='Olaru';     L:=length(N);   writeln(N, L:4);
  P:='Ion';       L:=length(P);   writeln(P, L:4);
  NP:=N+' '+P;    L:=length(NP);  writeln(NP, L:4);
  readln;
end.
```

Rezultatele afișate pe ecran:

```
Munteanu   8
Mihai   5
Munteanu Mihai  14
Olaru   5
Ion   3
Olaru Ion 9
```

Se observă că pe parcursul derulării programului în studiu lungimea șirurilor de caractere `N`, `P` și `NP` se schimbă.

Asupra șirurilor de caractere sînt admise operațiile relaționale <, <=, =, >=, >, <>. Șirurile se compară componentă cu componentă de la stînga la dreapta în conformitate cu ordonarea caracterelor în tipul de date `char`. Ambii operanzi trebuie să fie de tip **`packed array [1..n] of char`** cu același număr de componente sau de tip **`string`**. Evident, operanzii de tip **`string`** pot avea lungimi arbitrare.

De exemplu, rezultatul operației

```
'AC' < 'BA'
```

este `true`, iar rezultatul operației

```
'AAAAC' < 'AAAAB'
```

este `false`.

O variabilă de tip *șir de caractere* poate fi folosită fie în totalitatea ei, fie parțial, prin referirea unui caracter din șir.

De exemplu, pentru `P='Mihai'` avem `P[1]='M'`, `P[2]='i'`, `P[3]='h'` ș.a.m.d. După executarea secvenței de instrucțiuni

```
P[1]:='P';
P[2]:='e';
P[3]:='t';
P[4]:='r';
P[5]:='u';
```

variabila `P` va avea valoarea `'Petru'`.

Programul ce urmează citește de la tastatură șiruri arbitrare de caractere și afișează pe ecran numărul de spații în șirul respectiv. Derularea programului se termină după introducerea șirului `'Sfirsit'`.

```
Program P82;
{ Numarul de spatii intr-un sir de caractere }
var S : string;
    i, j : integer;
begin
  writeln('Dati siruri de caractere:');
  repeat
    readln(S);
    i:=0;
    for j:=1 to length(S) do
      if S[j]=' ' then i:=i+1;
    writeln('Numarul de spatii=', i);
  until S='Sfirsit';
end.
```

Întrebări și exerciții

❶ Cum se definește un tip de date *șir de caractere*?

❷ Ce operații pot fi efectuate asupra șirurilor de caractere?

❸ Comentați următorul program:

```
Program P83;
{ Eroare }
var S : packed array [1..5] of char;
begin
  S:='12345';
  writeln(S);
  S:='Sfat';
  writeln(S);
end.
```

❹ Elaborați un program care:
a) determină numărul de apariții ale caracterului `'A'` într-un șir;

b) substituie caracterul `'A'` prin caracterul `'*'`;
c) radiază din șir caracterul `'B'`;
d) determină numărul de apariții ale silabei `'MA'` într-un șir;
e) substituie silabele `'MA'` prin silaba `'TA'`;
f) radiază din șir silaba `'TO'`.

❺ Precizați rezultatul operațiilor relaționale:

a) `'B' < 'A'`
b) `'BB' > 'AA'`
c) `'BAAAA' < 'AAAAA'`
d) `'CCCCD' > 'CCCCA'`
e) `'A A' = 'AA'`
f) `'BB ' < 'B B'`
g) `'A' = 'a'`
h) `'Aa' > 'aA'`
i) `'123' = '321'`
j) `'12345' > '12345'`

❻ Se consideră șiruri de caractere formate din literele mari ale alfabetului latin și spații. Creați un program care afișează șirurile în studiu după regulile:
– fiecare literă de la `'A'` pînă la `'Y'` se înlocuiește prin următoarea literă din alfabet;
– fiecare literă `'Z'` se înlocuiește prin litera `'A'`;
– fiecare spațiu se înlocuiește prin `'-'`.

❼ Elaborați un program care descifrează șirurile cifrate conform regulilor din exercițiul 6.

❽ Se consideră m, $m \leq 100$, șiruri de caractere formate din literele mici ale alfabetului latin. Elaborați un program care afișează pe ecran șirurile în studiu în ordine alfabetică.

❾ Șirul S este compus din cîteva propoziții, fiecare terminîndu-se cu punct, semn de exclamare sau semnul întrebării. Creați un program care afișează pe ecran numărul de propoziții din șirul în studiu.

Test de autoevaluare nr. 4

1. Se consideră declarațiile

```
type Obiect = (Istoria, Geografia, Matematica,
     Informatica, Fizica);
     Nota = 1..10;
     SituatiaScolara = array [Obiect] of Nota;
```

Reprezentați pe un desen structura datelor de tipul `SituatiaScolara`.

2. Precizați tipul indicilor și tipul componentelor tipului de date `OrarulDeAstazi` din declarațiile ce urmează:

```
type Lectie = 1..6;
     Obiect = (LimbaRomana, LimbaModerna, Istoria,
     Geografia, Matematica, Informatica, Fizica, Chimia);
     OrarulDeAstazi = array [Lectie] of Obiect;
```

3. Care din tipurile de date ce urmează pot fi utilizate în calitate de tip de indice în cazul definirii tablourilor unidimensionale?

a) `real`

b) `integer`

c) `[-5..-1]`

d) `(AneniiNoi, Orhei, Chisinau)`

e) `boolean`

f) `char`

g) `[1..5]`

h) `array[1..5] of integer`

4. Se consideră declarațiile:

```
type RemunerareaLunara = array [1..12] of real;
var RMunteanu, RPetrescu : RemunerareaLunara;
    r : real;
    s : boolean;
    i : 1..12;
    t : real;
```

Care din atribuirile de mai jos sînt corecte?

a) `i:=t`

b) `RMunteanu[5]:= 1461.12`

c) `RMunteanu[3]:=true`

d) `r:=RPetrescu[9]`

e) `RPetrescu[11]:=t`

f) `t:=i`

g) `RMunteanu[i]:=t`

h) `s:=RPetrescu[i]`

i) `RPetrescu:=2489,81`

j) `Petrescu[16]:=3905.00`

k) `RMunteanu:=RPetrescu`

l) `Petrescu:=2*RMunteanu`

m) `RMunteanu[s]:=r`

n) `RMunteanu[5]:=RPetrescu[12]`

o) `RMunteanu[i]:=RPetrescu[5]`

p) `s:=RMunteanu[5]=RPetrescu[8]`

5. Se consideră declarațiile:

```
type Tablou = array [1..10] of integer;
var x, y : Tablou;
```

Scrieți expresia aritmetică a cărei valoare este:
a) suma primelor patru componente ale variabilei `x`;
b) suma ultimelor patru componente ale variabilei `y`;
c) valoarea absolută a componentei a treia a variabilei `x`;
d) valoarea absolută a componentei a șasea a variabilei `y`;
e) suma primei componente a variabilei `x` și ultimei componente a variabilei `y`.

6. Se consideră n ($n \leq 50$) numere întregi $a_1, a_2, a_3, ..., a_n$. Elaborați un program PASCAL care citește numerele respective de la tastatură și le afișează pe ecran în ordinea inversă citirii: $a_n, ..., a_3, a_2, a_1$.

7. Se consideră tablourile unidimensionale `A[1..100]` și `B[1..100]`, indicii și componentele cărora sunt numere naturale. Scrieți un program PASCAL care calculează numărul de cazuri în care componentele cu același indice ale ambelor tablouri sunt egale, adică `A[i]=B[i]`.

8. Ce operații pot fi efectuate asupra șirurilor de caractere? Precizați tipul rezultatelor acestor operații.

9. Elaborați un program PASCAL care afișează pe ecran în ordine inversă șirul de caractere citit de la tastatură. De exemplu, șirul `'soare'` va fi afișat pe ecran ca `'eraos'`.

10. Se consideră șiruri de caractere formate din literele mari ale alfabetului latin. Elaborați un program care afișează pe ecran numărul de vocale din șirul de caractere *S* citit de la tastatură.

11. Elaborați un program PASCAL care citește de la tastatură un număr natural din intervalul [1, 7] și afișează pe ecran denumirea corespunzătoare a zilei din săptămînă.

Exemple:

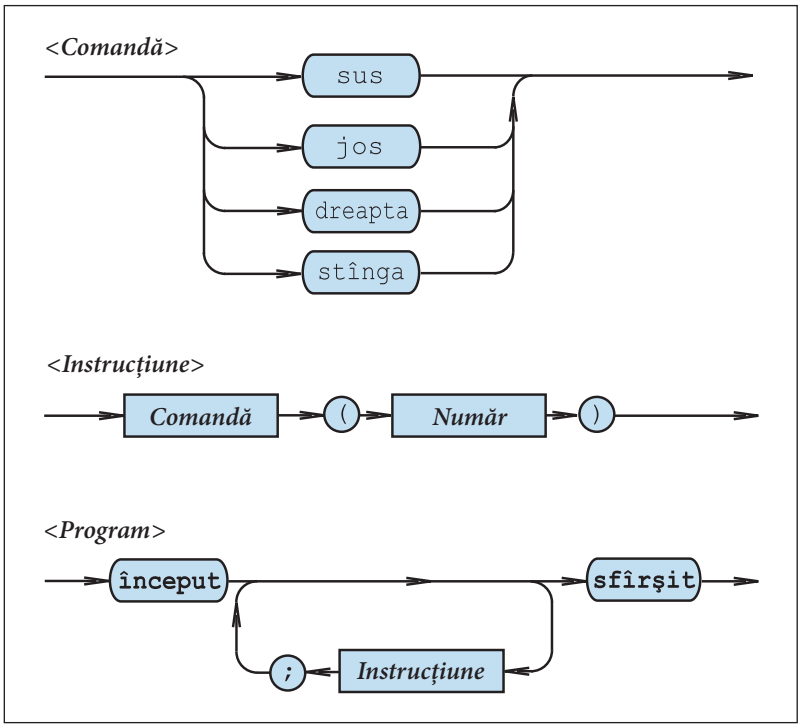
	Intrare	*Ieșire*
a)	`3`	`Miercuri`
b)	`1`	`Luni`
c)	`6`	`Simbata`

Răspunsuri la testele de autoevaluare

Testul nr. 1

1. *a, c, d* – corect; *b, e, f, g* – greșit.

2.



3. *a, c, e, f , h, i, m, o* – corect; *b, d, g, j, k, l, n* – greșit.

4. <*Cifră octală*> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7`

<*Număr octal*> ::= [+|-]<*Cifră octală*>{<*Cifră octală*>}

5. <*Cifră*> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

<*Literă*> ::= `a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z`

<*Identificator*> ::= <*Literă*> { <*Literă*> | <*Cifră*> }

6.

1) `x1` *2)* `x2`

3) `Delta`

4) `UnghiulAlfa`

5) `UnghiulBeta`

6) `DistantaParcursa`

7) `ListaElevilor`

8) `Vocala`

9) `Pixel`

10) `Culoare`

7.

a) `3.14`

b) `265`

c) `23.4635`

d) `+0.000001`

e) `6.1532e-5`

f) `-984.52`

g) `-523`

h) `+28`

i) `+28000000`

j) `614.45e-12`

k) `-3628.297e12`

l) `-38.00001`

m) `35728.345452e-8`

n) `24815`

o) `-296.0020001`

8.

a) 6124,485;

b) +18,315;

c) $-218{,}034 \cdot 10^{-3}$;

d) 193526;

e) $1000{,}01 \cdot 10^{23}$;

f) $-0{,}03428 \cdot 10^{-8}$;

g) 232847,5213;

h) $-0000012 \cdot 10^{+2}$;

i) 18,45;

j) $623{,}495 \cdot 10^{-6}$;

k) 2005;

l) $+23{,}08 \cdot 10^{-5}$;

m) −17502;

n) +1;

o) $-46341{,}2 \cdot 10^{-6}$.

9. Cuvinte-cheie: **`Program`**, **`var`**, **`begin`**, **`if`**, **`then`**, **`end`**, **`and`**.
Simboluri speciale: `;` , `,` , `:` , `(` , `)` , `<>` , `:=` , `/` , `'` , `=` , `..`
Identificatori: `TA1`, `a`, `b`, `x`, `real`, `readln`, `writeln`.
Numere: `0`.
Şiruri de caractere:
```
'Ecuatia are o singura radacina',
'Ecuatia are o multime infinita de radacini',
'Ecuatia nu are sens'.
```

10. *Linia 1* – antetul; *Linia 2* – partea declarativă; *Liniile 3–15* – partea executabilă.

Testul nr. 2

1. Prin *tip de date* se înțelege o mulțime de valori și o mulțime de operații care pot fi efectuate cu valorile respective. Exemple de tipuri de date: `integer`, `real`, `char`. Valorile `1`, `2` și `3` sînt de tip `integer`; valorile `1.0`, `2.0` și `0.5e+07` sînt de tip `real`, iar valorile `'A'`, `'B'` și `'+'` – de tip `char`.

2. Într-un program PASCAL mărimile sînt obiecte destinate pentru a reprezenta datele. Există două genuri de mărimi: *variabile* și *constante*. Pe parcursul execuției programului, valoarea oricărei variabile poate fi schimbată, în timp ce valorile constantelor nu pot fi schimbate.

3. `i`, `j` – variabile de tip `integer`; a, b, c – variabile de tip `real`; s – variabilă de tip `char`; p – variabilă de tip `boolean`; 5, 9 – constante de tip `integer`; `1.0`, `1.0e-01`, `-2.001` – constante de tip `real`; `'A'` – constantă de tip `char`; `true` – constantă de tip `boolean`.

4. Mulțimea de valori ale tipului de date `integer` este formată din numerele întregi care pot fi reprezentate pe calculatorul-gazdă al limbajului. Valoarea maximă poate fi referită prin constanta `MaxInt`, cunoscută oricărui program PASCAL. De obicei, valoarea minimă, admisă de tipul de date în studiu, este `-MaxInt` sau `-(MaxInt+1)`. Operațiile care se pot face cu valorile întregi sînt: +, -, *, **`mod`**, **`div`** ș.a.

5. Erorile de depășire vor apărea atunci cînd `x*y>MaxInt`. De exemplu, pentru `MaxInt=32767` (versiunea Turbo PASCAL 7.0) vor apărea erori de depășire dacă utilizatorul va tasta pentru `x` și `y` valori mai mari ca `200`.

6. Mulțimea de valori ale tipului de date în studiu este formată din numerele reale care pot fi reprezentate pe calculatorul-gazdă al limbajului. Operațiile care se pot face cu valorile reale sînt +, -, *, / (împărțirea) ș.a. Aceste operații sînt în general aproximative datorită erorilor de rotunjire.

7. Erorile de depășire vor apărea atunci cînd rezultatul operației `x*y` nu se va încadra în domeniul de valori ale tipului de date `real`. În versiunea Turbo PASCAL 7.0, domeniul de valori ale tipului `real` este $-1{,}7\cdot10^{38}, ..., +1{,}7\cdot10^{38}$. Prin urmare erori de depășire vor apărea atunci cînd utilizatorul va tasta pentru numerele `x` și `y` valori mai mari, de exemplu, 10^{20}.

8. Tipul de date `boolean` include valorile de adevăr `false` și `true`. Operațiile predefinite ale tipului de date `boolean` sînt: **`not`**, **`and`** și **`or`**.

9. Vezi *figura 2.1*.

10. Instrucțiunea `readln(p, q)` este greșită, întrucît valorile variabilelor logice nu pot fi citite de la tastatură.

11. Mulțimea valorilor tipului de date `char` include toate caracterele imprimabile, ordonate conform tabelului de cod *ASCII*. Operațiile ce pot fi efectuate cu valorile tipului de date `char` sînt: `chr`, `ord`, `pred`, `succ`, operații relaționale.

12.

```
Program RTA1;
 { Numerele de ordine ale cifrelor zecimale }
begin
  writeln(ord('0'));
  writeln(ord('1'));
  writeln(ord('2'));
  writeln(ord('3'));
  writeln(ord('4'));
  writeln(ord('5'));
  writeln(ord('6'));
  writeln(ord('7'));
  writeln(ord('8'));
  writeln(ord('9'));
end.
```

13. Mulțimea de valori ale oricărui tip date *enumerare* este specificată printr-o listă de identificatori. Primul identificator din listă desemnează cea mai mică valoare, cu numărul de ordine zero. Identificatorul al doilea va avea numărul de ordine unu, al treilea – numărul de ordine doi etc. Operațiile ce pot fi efectuate cu valorile oricărui tip de date *enumerare* sînt: `ord`, `pred`, `succ`, operațiile relaționale.

14.

```
Program RTA2;
 { Numerele de ordine ale valorilor de tip enumerare }
type FunctiaOcupata = ( Muncitor, SefDeEchipa, Maistru,
                        SefDeSantier, Director);
     StareaCivila = (Casatorit, Necasatorit);
begin
  writeln(ord(Muncitor));
  writeln(ord(SefDeEchipa));
  writeln(ord(Maistru));
  writeln(ord(SefDeSantier));
  writeln(ord(Director));
  writeln(ord(Casatorit));
  writeln(ord(Necasatorit));
end.
```

15. Un tip de date *subdomeniu* include o submulțime de valori ale unui tip deja definit – `integer`, `boolean`, `char` sau *enumerare* –, denumit tip de bază. Cu valorile unui tip de date *subdomeniu* pot fi efectuate toate operațiile tipului de bază.

16. p – tip *subdomeniu*, tipul de bază `char`. Poate lua valorile *'A'*, *'B'*, *'C'*, ..., *'Z'*;
q – tip *subdomeniu*, tipul de bază `integer`. Poate lua valorile 1, 2, 3, ..., 9;
r – tip *subdomeniu*, tipul de bază `char`. Poate lua valorile *'0'*, *'1'*, *'2'*, ..., *'9'*;
s – tip `char`. Ca valoare poate lua orice caracter imprimabil *ASCII*;
t – tip `integer`. Ca valoare poate lua orice număr întreg care poate fi reprezentat pe calculatorul-gazdă al limbajului;
u – tip *enumerare*. Poate lua valorile `Alfa, Beta, Gama, Delta`.

17. Tipurile ordinale de date: `integer`, `boolean`, `char`, *enumerare*, *subdomeniu*. Proprietățile comune:

a) fiecare valoare a unui tip ordinal are un număr de ordine, care poate fi aflat cu ajutorul funcției predefinite `ord`;

b) asupra valorilor oricărui tip ordinal de date sînt permise operațiile relaționale;

c) pentru tipurile ordinale de date există funcțiile predefinite pred (predecesor) și succ (succesor).

18.

```
Y    E    -6    10    1    3
```

19.

```
1    0    2    true    false    true
```

20. Două tipuri sînt identice dacă ele au fost definite cu același nume de tip.
Două tipuri sînt compatibile atunci cînd este adevărată cel puțin una din următoarele afirmații:
a) cele două tipuri sînt identice;
b) un tip este un subdomeniu al celuilalt tip;
c) ambele tipuri sînt subdomenii ale aceluiași tip de bază.
Un tip de date este anonim dacă el este definit implicit într-o declarație de variabile.

21. Tipuri identice: *a)* `T1`, `integer` și `T2`; *b)* `T3` și `T4`;
Tipuri compatibile: *a)* `T1`, `integer`, `T2`, `T3`, `T4`, `T5` și `1..100`; *b)* `T6`, `T7` și `T8`; *c)* `T9` și `T10`.
Tipuri anonime: *a)* `1..100`; *b)* `(Alfa, Beta, Gama, Delta)`.

22. `Alfa` – `integer`; `Beta` – `real`; `Indicator` – `boolean`; `Mesaj` – șir de caractere; `Semn` – șir de caractere; `Inscris` – șir de caractere.

23.

```
const  a = 3.14;
       b = 9.8;
var  i, j : integer;
     x, y : real;
```

Testul nr. 3

1.

a) `(a+b)-2*a*b`

b) `6*sqr(a)+15*a*b-13*sqr(b)`

c) `(a+b)*(a-b)`

d) `2*Alfa*Beta-5*Pi*r`

e) `Pi*sqr(r)+Alfa*sqr(Beta)`

f) `x and y or x and z`

2.

a) $a^2+\frac{2}{b^2}$;

b) $\frac{2a}{b+c}$;

c) $15\sqrt{\frac{a}{a-b}}$;

d) $\overline{xy}\vee z$;

e) $\left(\frac{a+b}{2}\right)^2$;

f) $(x\neq 0)\ \&\ (q<p)$.

3. *a*, *c*, *e*, *f* – corect; *b*, *d* – greșit.

4. *a*) 14; *b*) 6; *c*) –4; *d*) `false`; *e*) `true`; *f*) `true`.

5.

a) `integer`

b) `real`

c) `real`

d) `integer`

e) `integer`

f) `integer`

g) `boolean`

h) `char`

6.

```
Program RTA3;
var  i : integer;
     x, y : real;
begin
  writeln('Dati i='); readln(i);
  writeln('Dati x='); readln(x);
  writeln('Dati y='); readln(y);
  writeln(15*i*(x+y));
  readln;
end.
```

7. *a*, *c*, *e*, *f* – corect; *b*, *d* – greșit.

8.

```
Program RTA4;
var x, y : real;
begin
  write('x='); readln(x);
  if x>15  then y:=9*x+3*sqr(x)
           else y:=3*x-5*sqrt(x+28);
  writeln('y=', y);
  readln;
end.
```

9.

```
Program RTA5;
var i : integer;
begin
  write('Dati valoarea monedei: ');
  readln(i);
  case i of
    1 : writeln('un ban');
    5 : writeln('cinci bani');
    10 : writeln('zece bani');
    25 : writeln('douazeci si cinci de bani');
    50 : writeln('cincizeci de bani');
```

```
    else writeln('valoare inadmisibila');
  end;
  readln;
end.
```

10.

```
Program RTA6;
var  n, i : integer;
     s, p : real;
begin
  write('n='); readln(n);
  s:=0; p:=1;
  for i:= 1 to n do
    begin
      s:=s+(1/i); p:=p*(1/i);
    end;
  writeln('s=', s);
  writeln('p=', p);
  readln;
end.
```

11.

```
Program RTA7;
var y, x, x1, x2, deltaX : real;
begin
  write('x1='); readln(x1);
  write('x2='); readln(x2);
  write('deltaX='); readln(deltaX);
  writeln('x':10, 'y':20);
  writeln;
  x:=x1;
  while x<=x2 do
    begin
      if x>=4 then y:=2*sqrt(x+6) else y:=3-abs(x);
      writeln(x:20, y:20);
      x:=x+deltaX;
    end;
  readln;
end.
```

12.

```
Program RTA8;
var  c : char; { caracterul citit de la tastatura }
     n : integer; { numarul de cifre in mesaj }
```

```
begin
  n:=0;
  writeln('Dati mesajul:');
  repeat
    readln(c);
    if (c<>'*') and (c<>'#') then n:=n+1;
  until c='#';
  writeln('Numarul de cifre n=', n);
readln;
end.
```

Testul nr. 4

1.

```
type SituatiaScolara = array [Obiect] of Nota
```

2. Tipul indicelui – `Lectie`; tipul componentelor – `Obiect`.

3. În calitate de tip de indice pot fi utilizate (*b*), (*c*), (*d*), (*e*), (*f*), (g).

4. Corecte sînt atribuirile (*b*), (*d*), (*e*), (*f*), (*g*), (*k*), (*n*), (*o*), (*p*).

5.

a) `x[1]+ x[2]+ x[3]+ x[4]`

b) `6y[7]+ y[8]+ y[9]+ y[10]`

c) `abs(x[3])`

d) `abs(x[6])`

e) `x[1]+ y[10]`

6.

```
Program RTA9;
{ Raspuns la Testul 4, Itemul 6 }
type Numere = array [1..50] of integer;
var A : Numere;
    n, i : integer;
begin
  write('Dati n='); readln(n);
  { Citim numerele de la tastatura }
  for i:=1 to n do
    begin
      write('A[', i, ']='); readln(A[i]);
    end;
```

```
  { Afisam numerele la ecran }
  writeln('Numerele in ordine inversa:');
  for i:=n downto 1 do
    writeln('A[', i, ']=', A[i]);
  readln;
end.
```

7.

```
Program RTA10;
{ Raspuns la Testul 4, Itemul 7 }
type Natural = 0..MaxInt;
     Tablou = array [1..100] of Natural;
var A, B : Tablou;
    n : 1..100; { numarul curent de componente }
    c : 0..100; { numarul de cazuri A[i]=B[i] }
    i : 1..100; { indicele }
begin
   write('Dati numarul de componente n=');
   readln(n);
   writeln('Dati componentele tabloului A');
   for i:=1 to n do
     begin write('A[', i, ']='); readln(A[i]); end;
   writeln('Dati componentele tabloului B');
   for i:=1 to n do
      begin write('B[', i, ']='); readln(B[i]); end;
   c:=0;
   for i:=1 to n do
     if A[i]=B[i] then c:=c+1;
   writeln('Numarul de cazuri c=', c);
   readln;
end.
```

8. Asupra şirurilor de tip **`string`** se poate efectua operaţia de concatenare (juxtapunere), notată prin "+". Lungimea curentă a unei valori v de tip **`string`** poate fi aflată cu ajutorul funcţiei predefinite `length`(v) care returnează o valoare de tip `integer`. De asemenea, asupra şirurilor de caractere sînt admise operaţiile relaţionale <, <=, =, >=, >, <>. Rezultatele acestor operaţii sînt de tip `boolean`.

9.

```
Program RTA11;
{ Raspuns la Testul 4, Itemul 9 }
var S : string;
    i : integer;
begin
  writeln('Dati sirul de caractere:');
  readln(S);
```

```
  writeln('Sirul in ordine inversa:');
  for i:=length(S) downto 1 do
    write(S[i]);
    writeln;
  readln;
end.
```

10.

```
Program RTA12;
{ Raspuns la Testul 4, Itemul 10 }
var S : string;
    i, n : integer;
begin
  write('Dati un sir de caractere format din ');
  writeln('literele mari ale alfabetului latin:');
  readln(S);
  { Numaram vocalele in sir }
  n:=0;
  for i:=1 to length(S) do
    if (S[i]='A') or (S[i]='E') or (S[i]='I') or
   (S[i]='O') or (S[i]='U') then n:=n+1;
  writeln('Numarul de vocale = ', n);
readln;
end.
```

11.

```
Program RTA13;
{ Raspuns la Testul 4, Itemul 11 }
type ZileleSaptaminii = array [1..7] of string;
var Z : ZileleSaptaminii;
    i : 1..7;
begin
   Z[1]:='Luni';
   Z[2]:='Marti';
   Z[3]:='Miercuri';
   Z[4]:='Joi';
   Z[5]:='Vineri';
   Z[6]:='Simbata';
   Z[7]:='Duminica';
   write('Dati numarul zilei saptaminii i=');
   readln(i);
   writeln(Z[i]);
   readln;
end.
```

Anexa 1. Vocabularul limbajului PASCAL

1. <*Literă*> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

2. <*Cifră*> ::= 0|1|2|3|4|5|6|7|8|9

3. <*Simbol special*> ::= +|-|*|/|=|<|>|]|[|,|(|)|:|;|^|.|@|{|}|$|#|<=|>=
| <> | := |..| <*Cuvînt-cheie*> | <*Simbol echivalent*>

4. <*Simbol echivalent*> ::= (*|*)|(.|.)

5. <*Cuvînt-cheie*> ::= **and | array | begin | case| const | div | do | downto | else | end | file | for | function | goto | if | in | label | mod | nil | not | of | or | packed | procedure | program | record | repeat | set | then | to | type | until | var | while |with**

6. <*Identificator*> ::= <*Literă*> { <*Literă*> | <*Cifră*> }

7. <*Directivă*> ::= <*Literă*> {<*Literă*> | <*Cifră*>}

8. <*Întreg fără semn*> ::= <*Cifră*> {<*Cifră*>}

9. <*Semn*> ::= + | –

10. <*Număr întreg*> ::= [<*Semn*>] <*Întreg fără semn*>

11. <*Factor scală*> ::= <*Număr întreg*>

12. <*Număr real*> ::= <*Număr întreg*> e <*Factor scală*> |
<*Număr întreg*> . <*Întreg fără semn*> [e <*Factor scală*>]

13. <*Număr*> ::= <*Număr întreg*> | <*Număr real*>

14. <*Şir de caractere*> ::= ' <*Element şir*> { <*Element şir*> } '

15. <*Element şir*> ::= '' | <*Orice caracter imprimabil*>

16. <*Etichetă*> ::= <*Întreg fără semn*>

17. <*Comentariu*> ::= (* <*Orice secvenţă de caractere şi sfîrşit de linie neconţinînd acolade*> *)

Notă. Simbolurile terminale { şi } din formula (17) sînt redate prin simbolurile echivalente respectiv (* şi *).

Anexa 2. Sintaxa limbajului PASCAL

1. <*Program*> ::= <*Antet program*>
<*Corp*>**.**

2. <*Antet program*> ::= **Program** <*Identificator*> [**(**<*Identificator*> {**,** <*Identificator*>}**)**]**;**

3. <*Corp*> ::= <*Declarații*>
<*Instrucțiune compusă*>

4. <*Declarații*> ::= [<*Etichete*>]
[<*Constante*>]
[<*Tipuri*>]
[<*Variabile*>]
[<*Subprograme*>]

5. <*Etichete*> ::= **label** <*Etichetă*> {**,** <*Etichetă*>}**;**

6. <*Constante*> ::= **const** <*Definiție constantă*>**;** { <*Definiție constantă*>**;**}

7. <*Definiție constantă*> ::= <*Identificator*> **=** <*Constantă*>

8. <*Constantă*> ::= [**+** | **-**] <*Număr fără semn*> | [**+** | **-**] <*Nume de constantă*> | <*Șir de caractere*>

9. <*Tipuri* > ::= **type** <*Definiție tip*>**;** { <*Definiție tip*>**;**}

10. <*Definiție tip*> ::= <*Identificator*> **=** <*Tip*>

11. <*Variabile*> ::= **var** <*Declarație variabile*>**;** {<*Declarație variabile*>**;**}

12. <*Declarație variabile*> ::= <*Identificator*> {**,** <*Identificator*>} **:** <*Tip*>

13. <*Subprograme*> ::= { <*Funcție*>**;** | <*Procedură*>**;** }

14. <*Tip*> ::= <*Identificator*> |
<*Tip enumerare*> |
<*Tip subdomeniu*> |
<*Tip tablou*> |
<*Tip articol*> |
<*Tip mulțime*> |
<*Tip fișier*> |
<*Tip referință*>

15. <*Tip enumerare*> ::= **(**<*Identificator*> {**,** <*Identificator*>}**)**

16. <*Tip subdomeniu*> ::= <*Constantă*>**..**<*Constantă*>

17. <*Tip tablou*> ::= [**packed**] **array** **(** **.** <*Tip*> {**,** <*Tip*>} **.** **)** **of** <*Tip*>

18. <*Tip mulțime*> ::= [**packed**] **set of** <*Tip*>

19. <*Tip fișier*> ::= [**packed**] **file of** <*Tip*>

20. <*Tip referință*> ::= **^**<*Tip*>

21. <*Tip articol*> ::= [**packed**] **record** <*Listă cîmpuri*> [**;**] **end**

22. <*Listă cîmpuri*> ::= <*Parte fixă*> [**;** <*Parte variante*>] | <*Parte variante*>

23. <*Parte fixă*> ::= <*Secțiune articol*> { **;** <*Secțiune articol*> }

24. <*Secțiune articol*> ::= <*Nume cîmp*> {**,** <*Nume cîmp*> } **:** <*Tip*>

25. <*Parte variante*> ::= **case** [<*Identificator*> **:**] <*Tip*> **of** <*Variantă*> {**;** <*Variantă*> }

26. <*Variantă*> ::= <*Constantă*> {**,** <*Constantă*> } **:** ([<*Listă cîmpuri*>] [**;**])

27. <*Funcție*> ::= <*Antet funcție*>**;** <*Corp*> | <*Antet funcție*>**;** <*Directivă*> |
function <*Identificator*> **;** <*Corp*>

28. <*Antet funcție*> ::= **function** <*Identificator*> [<*Listă parametri formali*>] **:** <*Identificator*>

29. <*Procedură*>::=<*Antet procedură*>**;**<*Corp*> | <*Antet procedură*>**;** <*Directivă*> |
procedure <*Identificator*> **;** <*Corp*>

30. <*Antet procedură*> := **procedure** <*Identificator*> [<*Listă parametri formali*>]

31. <*Listă parametri formali*> ::= **(**<*Parametru formal*> {**;** <*Parametru formal*>}**)**

32. <*Parametru formal*> ::= [**var**] <*Identificator*> {**,** <*Identificator*>} **:** <*Identificator*> |
<*Antet funcție*> | <*Antet procedură*>

33. <*Instrucțiune*> ::= [<*Etichetă*> **:**] <*Instrucțiune neetichetată*>

34. <*Instrucțiune neetichetată*> ::= <*Atribuire*> | <*Apel procedură*> |
<*Instrucțiune compusă*> |
<*Instrucțiune* **if**> | <*Instrucțiune* **case**> |
<*Instrucțiune* **while**> | <*Instrucțiune* **repeat**> |
<*Instrucțiune* **for**> | <*Instrucțiune* **with**> |
<*Instrucțiune* **goto**> | <*Instrucțiune de efect nul*>

35. <*Atribuire*> ::= <*Variabilă*> **:=** <*Expresie*> | <*Nume funcție*> **:=** <*Expresie*>

36. <*Apel procedură*> ::= <*Nume procedură*> [<*Listă parametri actuali*> |
<*Listă parametri de ieșire*>]

37. <*Listă parametri actuali*> ::= **(**<*Parametru actual* > {**,**<*Parametru actual*>}**)**

38. <*Parametru actual* > ::= <*Expresie*> | <*Variabilă*> | <*Nume funcție*> | <*Nume procedură* >

39. <*Nume funcție*> ::= <*Identificator*>

40. <*Nume procedură*> ::= <*Identificator*>

41. <*Listă parametri de ieșire*> ::= **(**<*Parametru de ieșire*> { **,** <*Parametru de ieșire*> }**)**

42. <*Parametru de ieșire*> ::= <*Expresie*> [**:**<*Expresie*> [**:** <*Expresie*>]]

43. <*Instrucțiune compusă*> ::= **begin** <*Instrucțiune*> { **;** <*Instrucțiune*> } **end**

44. <*Instrucțiune* **if** > ::= **if** <*Expresie booleană*> **then** <*Instrucțiune*>
[**else** <*Instrucțiune*>]

45. <*Instrucțiune* **case**> ::= **case** <*Expresie*> **of** [<*Caz*>{**;** <*Caz*>}] [**;**] **end**

46. <*Caz*> ::= <*Constantă*> {**,** <*Constantă*>} **:** <*Instrucțiune*>

47. <*Instrucțiune* **while**> ::= **while** <*Expresie booleană*> **do** <*Instrucțiune*>

48. <*Instrucțiune* **repeat**> ::= **repeat** <*Instrucțiune*> {**;**<*Instrucțiune*>}
until <*Expresie booleană*>

49. <*Expresie booleană*> ::= <*Expresie*>

50. <*Instrucțiune* **for**> ::= **for** <*Variabilă*> **:=** <*Expresie*> <*Pas*> <*Expresie*>
do <*Instrucțiune*>

51. <*Pas*> ::= **to** | **downto**

52. <*Instrucțiune* **with**> ::= **with** <*Variabilă*> {**,** <*Variabilă*>} **do** <*Instrucțiune*>

53. <*Instrucțiune* **goto**> ::= **goto** <*Etichetă*>

54. <*Instrucțiune de efect nul*> ::=

55. <*Variabilă*> ::= <*Identificator*> | <*Variabilă*> **(.** <*Expresie*> { **,** <*Expresie*> } **.)** |
<*Variabilă*>**.**<*Nume cîmp*> | <*Variabilă*>

56. <*Nume cîmp*> ::= <*Identificator*>

57. <*Expresie*> ::= <*Expresie simplă*>{<*Operator relațional*> <*Expresie simplă*>}

58. <*Operator relațional*> ::= **<** | **<=** | **=** | **>=** | **>** | **<>** | **in**

59. <*Expresie simplă*> ::= [**+** | **–**] <*Termen*>{<*Operator aditiv*> <*Termen*>}

60. <*Operator aditiv*> ::= **+** | **–** | **or**

61. <*Termen*> ::= <*Factor*> {<*Operator multiplicativ*> <*Factor*>}

62. <*Operator multiplicativ*> ::= ***** | **/** | **div** | **mod** | **and**

63. <*Factor*> ::= <*Variabilă*> | <*Constantă fără semn*> | <*Apel funcție*> | **not** <*Factor*> |
(<*Expresie*>**)**|<*Constructor mulțime*>

64. <*Apel funcție*> ::= <*Nume funcție*> [<*Listă parametri actuali*>]

65. <*Constantă fără semn*> ::= <*Număr fără semn*> | <*Șir de caractere*> | <*Identificator*> |
nil

66. <*Constructor mulțime*> ::= **(.** [<*Specificare element*> { **,** <*Specificare element*> }] **.)**

67. <*Specificare element*> ::= <*Expresie*> [**..** <*Expresie*>]

Notă. Simbolurile terminale **[** și **]** din formulele (17), (55) și (66) sînt redate prin simbolurile echivalente, respectiv **(.** și **.)**.

Anexa 3. Compilarea și depanarea programelor PASCAL

După scrierea unui program PASCAL se efectuează editarea, compilarea și depanarea lui.

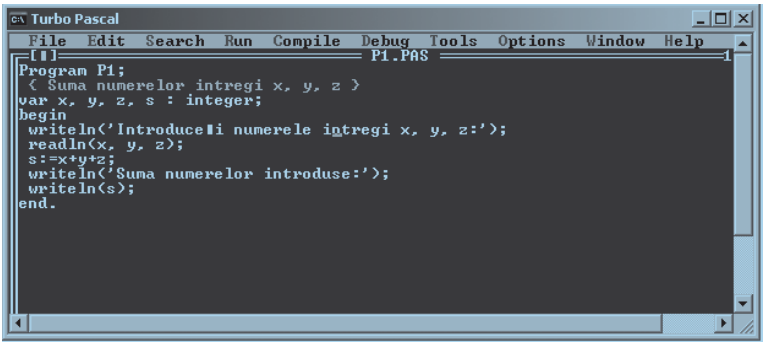
Editarea constă în introducerea programului în calculator. Programele introduse pot fi păstrate în fișiere text cu extensia ".pas".

Compilarea reprezintă procesul de traducere automată a programului scris în limbajul PASCAL într-un program scris în limbajul calculatorului. După compilare, programul în limbaj-mașină poate fi lansat în execuție sau păstrat într-un fișier executabil cu extensia ".exe".

Depanarea reprezintă procesul de depistare a greșelilor sintactice și semantice din programul PASCAL și corectarea lor.

De obicei, aceste operații se efectuează cu ajutorul unor programe speciale, denumite *medii integrate de dezvoltare* (IDE – *Integrated Development Environment*).

În cazul mediilor integrate de dezvoltare, comunicarea om–calculator se realizează cu ajutorul interfețelor grafice, care afișează pe ecran ferestre de aplicație, ferestre de dialog, ferestre de navigare, ferestre de explorare și ferestre de document.



În general, ferestrele mediilor de dezvoltare a programelor conțin elementele grafice standard, studiate în clasele precedente: bara de meniuri, meniuri, comenzi, butoane, cursoare, casete etc.

Prezentăm în continuare comenzile frecvent utilizate din componența meniurilor mediilor de programare Turbo PASCAL și Free PASCAL, instalate în majoritatea laboratoarelor de informatică din școlile Republicii Moldova.

Meniul `File` (Fișier)

`New` **(Nou)** – creează o fereastră nouă în care utilizatorul poate introduce și edita un text, de obicei un program PASCAL.

`Open...` **(Deschide...)** – citește fișierul specificat de utilizator și afișează conținutul acestuia într-o fereastră nouă. În continuare, conținutul respectiv poate fi supus prelucrărilor dorite.

Save (**Salvează**) – salvează conținutul ferestrei curente în fișierul deschis anterior cu ajutorul comenzii **Open**. Dacă însă fereastra curentă a fost creată cu ajutorul comenzii **New**, se va crea un fișier nou, iar utilizatorul va fi invitat să propună o denumire pentru fișierul nou-creat.

Save as... (**Salvează ca...**) – salvează conținutul ferestrei curente într-un fișier nou. Utilizatorul este invitat să propună o denumire pentru fișierul nou-creat.

Print (**Imprimă**) – tipărește conținutul ferestrei curente la imprimantă.

Exit (**Ieșire**) – ieșirea din mediul integrat de dezvoltare a programelor. Înainte de a ieși din program, aplicația va propune utilizatorului să salveze conținutul ferestrelor respective.

Meniul Edit (Editare)

Cut (**Decupează**) – decuparea fragmentului selectat de text. Fragmentul decupat este depus în memoria-tampon.

Copy (**Copie**) – copierea fragmentului selectat de text. Fragmentul copiat este depus în memoria-tampon.

Paste (**Lipește**) – fragmentul de text din memoria-tampon este inserat în locul în care se află cursorul.

Clear (**Șterge**) – ștergerea fragmentului selectat de text fără a-l depune însă în memoria-tampon.

Meniul Search (Căutare)

Find (**Găsește**) – caută, pornind de la poziția curentă a cursorului, fragmentul indicat de text.

Replace (**Înlocuiește**) – înlocuiește fragmentul selectat de text cu fragmentul indicat de utilizator.

Meniul Run (Rulare)

Run (**Derulează**) – compilează și lansează în execuție programul PASCAL din fereastra curentă. Dacă programul conține erori sintactice, se va afișa un mesaj care indică tipul și locul erorii.

Meniul Compile (Compilare)

Compile (**Compilează**) – compilează programul PASCAL din fereastra curentă, fără a-l lansa însă în execuție.

Build (**Construiește**) – compilează programul PASCAL din fereastra curentă și memorează rezultatul compilării într-un fișier executabil.

Meniul **Windows** conține comenzi ce asigură aranjarea și comutarea între ferestre, iar meniul **Help** – comenzile ce permit accesul la manualul de asistență.

Acest manual este proprietatea Ministerului Educației al Republicii Moldova

Gimnaziul/Liceul _____
Manualul nr. _____

Anul de folosire	Numele de familie și prenumele elevului	Anul școlar	Aspectul manualului	
			la primire	la restituire
1.				
2.				
3.				
4.				
5.				

- Dirigintele trebuie să controleze dacă numele elevului este scris corect.
- Elevul nu trebuie să facă niciun fel de însemnări în manual.
- Aspectul manualului (la primire și la restituire) se va aprecia folosind termenii: *nou, bun, satisfăcător, nesatisfăcător*.

Imprimare la Tipografia „BALACRON" SRL,
str. Calea Ieșilor, 10;
MD-2069, Chișinău, Republica Moldova
Comanda nr. 662

9 789975 850131