MINISTERUL EDUCAŢIEI, CULTURII ŞI CERCETĂRII

INFORMATICĂ

Manual pentru clasa a 10-a

ANATOL GREMALSCHI • IURIE MOCANU • ION SPINEI
LUDMILA GREMALSCHI

Ştiinţa, 2020

CZU 004 (075.3)
G 80

Manualul şcolar a fost realizat în conformitate cu prevederile Curriculumului la disciplină, aprobat prin Ordinul Ministerului Educaţiei, Culturii şi Cercetării nr. 906 din 17 iulie 2019. Manualul a fost aprobat prin Ordinul Ministerului Educaţiei, Culturii şi Cercetării nr. 1048 din 28.09.2020, urmare a evaluării calităţii metodico-ştiinţifice.

Editat din sursele financiare ale Fondului Special pentru Manuale.

Comisia de evaluare:
Coordonator: *Svetlana Brînză*, grad didactic superior, LT „N.M. Spătaru", metodist DGETS, Chişinău;
Gheorghe Chistruga, grad didactic superior, LT „M. Eminescu", Drochia;
Natalia Schiţco, grad didactic superior, LT „Socrate", Chişinău;
Violina Bargan, grad didactic superior, LT „Gh. Asachi", Chişinău;
Ecaterina Adam, grad didactic unu, LCI „Prometeu-Prim", Chişinău

Denumirea instituției de învățământ _____

Manualul a fost folosit: _____

Anul de folosire	Numele, prenumele elevului	Anul de studii	Aspectul manualului	
			la primire	la returnare

Dirigintele verifică dacă numele, prenumele elevului sunt scrise corect.
Elevii nu vor face niciun fel de însemnări în manual.
Aspectul manualului (la primire şi la returnare) se va aprecia cu unul dintre următorii termeni: *nou, bun, satisfăcător, nesatisfăcător.*

Responsabil de ediţie: Larisa Dohotaru
Redactor: Mariana Belenciuc
Corector: Maria Cornesco
Redactor tehnic: Nina Duduciuc
Machetare computerizată: Vitalie Ichim
Copertă: Romeo Şveţ

ÎNTREPRINDEREA
EDITORIAL-POLIGRAFICĂ
ȘTIINȚA

str. Academiei, nr. 3; MD-2028, Chişinău, Republica Moldova
tel.: (+373 22) 73-96-16; fax: (+373 22) 73-96-27
e-mail: prini_stiinta@yahoo.com;
www.editurastiinta.md

Descrierea CIP a Camerei Naţionale a Cărţii
Informatică: Manual pentru clasa a 10-a / Anatol Gremalschi, Iurie Mocanu, Ion Spinei, Ludmila Gremalschi; Comisia de evaluare: Svetlana Brînză (coordonator) [et al.]; Ministerul Educaţiei, Culturii şi Cercetării. – Ch.: Î.E.P. *Ştiinţa*, 2020 (Tipogr. „Bons Offices"). – 212 p.: fig., tab.
Proprietate a Ministerului Educaţiei, Culturii şi Cercetării.

ISBN 978-9975-85-234-0

Tiparul executat la
Casa Editorial-Poligrafică „Bons Offices"
str. Feredeului, 4/6

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 gândirii algoritmice și formarea culturii informaționale. Realizarea acestui obiectiv presupune extinderea capacităților fiecărei persoane de a elabora algoritmi pentru rezolvarea problemelor ce apar 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, sunt utilizate limbajele de programare PASCAL și C++.

Limbajul PASCAL, numit astfel în cinstea matematicianului și filosofului francez Blaise Pascal, a cunoscut o răspândire mondială, fiind cel mai potrivit pentru predarea și învățarea informaticii în școli. Reprezentând un instrument destinat formării și dezvoltării gândirii algoritmice, limbajul PASCAL include toate conceptele de bază utilizate în sistemele informatice moderne: variabile, constante, tipuri de date, instrucțiuni simple și instrucțiuni compuse.

Limbajul C++ este foarte popular în industria produselor-program (*software*) și este recomandat pentru studiere elevilor ce doresc să urmeze o carieră profesională în sectorul tehnologiei informației și comunicațiilor.

Alegerea limbajului de programare ce va fi studiat este la decizia fiecărei instituții de învățământ, manualul oferind posibilitatea studierii atât a unuia, cât și a ambelor limbaje.

În manual, fiecare porțiune de materie teoretică este urmată de un șir de exemple, exerciții și întrebări de control. Se consideră că fiecare elev în parte va introduce și va lansa în execuție programele incluse în manual, va răspunde la întrebări și, când e cazul, el însuși va scrie și va depana programele PASCAL/C++, necesare pentru rezolvarea problemelor propuse.

Toate programele incluse în manual au fost testate și depanate în mediile de programare PASCAL/C++. Evident, elevii și profesorii pot utiliza și alte produse-program, destinate instruirii asistate de calculator.

Ca și în cazul altor discipline școlare, în studierea informaticii un rol aparte revine formării și evoluării competențelor de a învăța să înveți, de dezvoltare a gândirii creative. În acest scop, manualul conține și materii opționale, care vor fi studiate de elevi prin metodele învățării constructiviste, adică prin acțiune și aplicații practice, în bază de proiecte, cu ajutorul mijloacelor moderne de instruire asistată de calculator.

Fiind o parte indispensabilă a culturii moderne, informatica are un rol decisiv în dezvoltarea umană și deschide perspective promițătoare pentru fiecare dintre 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

Recapitulare

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 clasele de gimnaziu am studiat executanții **Cangurul** și **Furnica**, elaborați în scopuri didactice pentru școlile din țara noastră, diverși executanți din mediile grafic-interactive de programare de tip *Logo*, *Scratch*, *Robo* etc. Amintim că executantul **Cangurul** poate îndeplini comenzile PAS, ROTIRE, SALT, iar executantul **Furnica** – comenzile SUS, JOS, DREAPTA, STÂNGA.

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 secvenței 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 clasele de gimnaziu am elaborat 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, STÂNGA, apel de procedură și instrucțiunile compuse REPETĂ, CÂT, DACĂ.

Fiind elaborați în scopuri pur didactice, executanții **Cangurul** și **Furnica** nu permit efectuarea unor prelucrări complexe ale informației. Evident, 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. Amintim 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
```

Repere istorice:
1955 – FORTRAN (*FORmula TRANslation*)
1960 – ALGOL (*ALGOritmic Language*)
1960 – COBOL (*COmmon Business Oriented Language*)
1971 – PASCAL (*Blaise PASCAL*)
1972 – C
1980 – C++
1995 – Java

Întrucât elaborarea 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 cu ajutorul unor limbaje speciale, denumite limbaje de programare de nivel înalt. Pe parcursul dezvoltării informaticii au fost elaborate mai multe limbaje de programare, în prezent numărul lor fiind de circa opt mii. Totuși doar un număr mic dintre ele sunt utilizate pe scară largă, cele mai răspândite pentru instruire fiind LOGO, BASIC, PASCAL, C, C++, Java etc. Toate aceste limbaje conțin mijloace pentru descrierea și apelarea subalgoritmilor și instrucțiuni pentru programarea algoritmilor liniari, algoritmilor repetitivi și a algoritmilor cu ramificări.

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

PASCAL

```
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.
```

C++

```
// Programul Exemplu
#include <iostream>
using namespace std;
int main()
{
float a, b, x;
cin>>a>>b;
if (a!=0)
{
x=-b/a;
cout<<"Ecuatia are o singura
       radacina"<<endl;
cout<<x<<endl;
}
if ((a==0)&&(b==0))
   cout<<"Ecuatia are
          o multime infinita
          de radacini"<<endl;
if ((a==0)&&(b!=0)) cout<<
 "Ecuatia nu are sens"<<endl;
return 0;
}
```

Comparând fragmentul de program scris în limbajul calculatorului cu programul `Exemplu`, ne convingem că utilizarea unui limbaj de nivel înalt, în cazul exemplului de mai sus – a limbajului PASCAL/C++, 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 cu ajutorul 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 cu ajutorul schemelor logice;

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

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*. Pentru editarea și depanarea programelor, au fost elaborate programe de aplicații speciale, denumite *medii de dezvoltare a programelor*.

Întrebări şi exerciţii

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 sunt mijloacele principale de reprezentare a algoritmilor?

❹ Prin ce se deosebește modul de comandă prin programe de modul de comandă manuală?

❺ Care este diferența dintre algoritmi și programe? Argumentați răspunsul dvs.

❻ Numiți etapele principale de rezolvare a unei probleme cu ajutorul calculatorului.

❼ Care este diferența dintre un limbaj de programare de nivel înalt și limbajul calculatorului?

❽ **DESCOPERĂ SINGUR SAU ÎMPREUNĂ CU COLEGII!** Căutați pe Internet și aflați:
- Opiniile persoanelor marcante din domeniul dezvoltării umane, din lumea afacerilor, din sectorul tehnologiei informației și comunicațiilor referitoare la studierea programării în școli.
- Ce limbaje de programare sunt studiate în școlile din întreaga lume.
- Cotele procentuale ale limbajelor de programare studiate în școlile din țările cu tradiții în domeniu.
- Cotele procentuale ale limbajelor de programare utilizate în industria produselor-program (*software*).
- Popularitatea limbajelor de programare conform indicelui TIOBE. Abrevierea TIOBE este formată din primele litere ale renumitei comedii a lui Oscar Wild *The Importance Of Being Earnest* [Ce înseamnă să fii onest].

❾ **STUDIU DE CAZ (Individual sau în echipă).** Selectați două dintre cele mai răspândite limbaje de programare, unul utilizat în școli pentru instruire și altul utilizat în industria de produse-program. Comparați aceste limbaje după următoarele criterii: lizibilitatea, simplitatea, sugestivitatea, intuitivitatea.

Capitolul 1

VOCABULARUL ȘI SINTAXA LIMBAJULUI PASCAL/C++

1.1. Inițiere în limbajul PASCAL/C++

Pascal

Vom examina următorul program scris în Pascal:

```
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 stângă a paginii nu fac parte din programul prezentat. Ele servesc doar pentru referirea liniilor în explicațiile ce urmează.

Linia 1. Cuvântul **`program`** este un cuvânt rezervat al limbajului, iar `P1` este un cuvânt 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. Cuvântul rezervat **`var`** (*variable* – variabilă) descrie variabilele `x`, `y`, `z` și `s`, utilizate în program. Cuvântul `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. Cuvântul 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. Cuvântul `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 sunt tastate în aceeași linie și despărțite de unul sau mai multe spații. După tastarea ultimului număr se acționează tasta <ENTER>. Numerele citite sunt depuse în variabilele x, y, z. Cuvântul `readln` (*read line* – citire și trecere la linie nouă) reprezintă apelul unui subalgoritm-standard. Argumentele subalgoritmului constituie numele variabilelor în care sunt 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. Cuvântul rezervat **`end`** indică sfârșitul părții executabile, iar punctul – sfârșitul 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, tipurile de date, subprogramele etc. folosite 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 sunt instalate mediile de dezvoltare a programelor Turbo PASCAL sau Free PASCAL.

Vom examina programul P1, scris în limbajul C++:

```
1  // Programul P1
2  #include <iostream>
3  using namespace std;
4  // Suma numerelor intregi x, y, z
5  int main()
6  {
7  int x, y, z, s;
8  cout<<"Introduceti numerele intregi x, y, z:"<<endl;
9  cin>>x>>y>>z;
10 s=x+y+z;
11 cout<<"Suma numerelor introduse: ";
12 cout<<s<<endl;
13 return 0;
14 }
```

Numerele *1*, *2*, *3*, ..., *14* din partea stângă a paginii nu fac parte din programul prezentat. Ele servesc doar pentru referirea liniilor în explicațiile ce urmează.

Linia 1. Comentariul scris pe un singur rând începe cu simbolul "//". Comentariul nu influențează în niciun fel derularea programului și este destinat exclusiv utilizatorului.

Linia 2. Directiva `#include`. Această directivă inserează în textul programului `P1` textul ce se conține în fișierul `iostream`. Fișierul respectiv conține descrierile subprogramelor care efectuează introducerea și extragerea datelor. În general, programele C++ pot utiliza nu doar funcții standard, dar și funcții scrise de alți informaticieni. Aceste funcții pot fi inserate în programul în curs de elaborare prin indicarea în directivele `#include` a denumirilor de fișiere ce le conțin. De obicei, în procesul de elaborare și depanare a programelor, mediul de dezvoltare nu afișează textele incluse prin aceste directive.

Linia 3. Cuvintele-cheie **`using namespace`** din această linie indică faptul că programul în curs de elaborare trebuie să folosească spațiul de nume `std`. Acest spațiu reprezintă o descriere a idendificatorilor utilizați în bibliotecile ce conțin funcțiile standard ale limbajului C++.

Linia 4. Este un text explicativ, un comentariu.

Linia 5. Descrierea funcției principale `main`.

Această funcție trebuie să apară în mod obligatoriu în orice program C++. Ea este punctul din care începe rularea programului, indiferent de locul ei în codul sursă.

Linia 6. Acolada `{` indică începutul corpului funcției principale `main`.

Linia 7. Cuvântul-cheie **`int`** (întreg) indică tipul variabilelor `x`, `y`, `z` și `s`. Prin urmare, în calitate de valori ale acestor variabile pot fi doar numerele întregi.

Linia 8. Afișarea textul dintre ghilimele pe ecran. Ghilimelele nu fac parte din textul care va fi afișat. Cuvântul `cout` (*console output* – ieșire consolă) este denumirea fluxului (torentului) de date transmise de program dispozitivului-standard de ieșire. Simbolul `<<` este un operator de transfer a informației. Subprogramul ce realizează extragerea datelor este descris în fișierul `iostream` din biblioteca standard C++, iar denumirea `cout` aparține spațiului de nume `std`. Orice instrucțiune C++ se termină prin caracterul `;` (punct și virgulă).

Cuvântul `endl` (*end line* – sfârșit linie) este un manipulator ce realizează, după afișarea mesajului, trecerea la un rând nou.

Linia 9. Citirea de la dispozitivul-standard de intrare, în mod obișnuit tastatura, a valorilor variabilelor `x`, `y`, `z`. Cuvântul `cin` (*console input* – intrare consolă) este denumirea fluxului (torentului) de date transmise programului de dispozitivul-standard de intrare, iar simbolul `>>` este un operator de transfer a informației. Numerele de citit trebuie tastate pe o singură linie și separate prin unul sau mai multe spații. După tastarea ultimului număr se acționează tasta <ENTER>.

Linia 10. Instrucțiunea de atribuire. Variabila `s` primește valoarea `x+y+z`.

Linia 11. Afișarea la dispozitivul-standard de ieșire a mesajului

```
Suma numerelor introduse:.
```

Linia 12. Afișarea valorii variabilei `s` la dispozitivul-standard de ieșire.

Linia 13. Instrucțiunea **`return 0;`** are ca efect terminarea execuției funcției `main`. Instrucțiunea **`return`** se folosește pentru a comunica sistemului de operare

codul ce caracterizează modul în care a fost executat programul, `0` semnificând faptul că execuția programului s-a încheiat fără erori. Este cel mai obișnuit mod de a încheia un program C++.

Linia 14. Acolada `}` indică sfârșitul corpului funcției `main`.

În general, un program în limbajul C++ este alcătuit din următoarele componente:

- **directive** (opțional);
- **declarații globale** (opțional), în care se descriu tipurile utilizatorului, variabilele, constantele etc. folosite în program;
- **funcțiile utilizatorului** (opțional), elaborate de el pentru a implementa anumiți subalgoritmi;
- **funcția principală `main`** (obligatoriu).

Editarea, compilarea, lansarea în execuție și depanarea programelor C++ se realizează cu ajutorul unor aplicații speciale, denumite medii de dezvoltare a programelor, cele mai frecvent utilizate fiind **Code::Blocks** și **Dev-Cpp**.

Pascal C++

Mediile de dezvoltare a programelor oferă utilizatorilor următoarele facilități:

- introducerea și editarea programelor;
- păstrarea programelor în fișiere distincte;
- deschiderea, editarea și salvarea fișierelor ce conțin programele în curs de elaborare;
- depistarea erorilor sintactice;
- compilarea și lansarea în execuție a programelor în curs de elaborare;
- depanarea programelor.

Modalitățile de utilizare a mediilor de dezvoltare a programelor PASCAL și C++ sunt descrise în anexe.

Întrebări şi exerciţii

❶ Introduceți și lansați în execuție programul `P1`.

❷ Care sunt părțile componente ale unui program PASCAL/C++?

❸ Introduceți și lansați în execuție următorul program:

PASCAL	C++

`Program P2;`
`{ Afisarea constantei predefinite MaxInt }`
`begin`
`  writeln('MaxInt=', MaxInt);`
`end.`
	`// Programul P2`
	`#include <iostream>`
	`#include <limits>`
	`using namespace std;`
	`// Afisarea constantei predefinite INT_MAX`
	`int main()`
	`{`
	`cout<<"INT_MAX="<<INT_MAX<<endl;`
	`return 0;`
	`}`

❹ Indicați antetul, partea declarativă şi partea executabilă ale programului prezentat ca exemplu la această temă. Explicați destinația fiecărei linii a programului în studiu.

❺ **APLICĂ!** Modificați programul `P1` (propus ca model în paragraful studiat) în aşa mod, ca el să afişeze pe ecran textul:

```
Cobori în jos, luceafăr blând,
Alunecând pe-o rază,
Pătrunde-n codru şi în gând,
Norocu-mi luminează.
        (Luceafărul, Mihai Eminescu)
```

❻ **APLICĂ!** Modificați programul `P1` (propus ca model în paragraful studiat) în aşa mod, ca el să calculeze suma numerelor *x*, *y* introduse de la tastatură. Dați la execuție programul elaborat.

❼ **OBSERVĂ!** Salvați în calculator programul elaborat în itemul 6. Observați ce extensie are fişierul sursă.

❽ **DESCOPERĂ SINGUR SAU ÎMPREUNĂ CU COLEGII!** Căutați pe Internet şi aflați:

- Ce limbaje de programare există şi cum au evoluat acestea. Completați tabelul:

Generația	Limbajele de programare

- De la care dintre limbajele de programare frecvent utilizate au derivat alte noi limbaje?
- Identificați cinci limbaje de programare cel mai des folosite.
- De unde provine denumirea limbajului de programare pe care îl studiați?

1.2. Metalimbajul BNF

Un limbaj de programare se definește prin sintaxa şi 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*.

Evident, sintaxa oricărui limbaj de programare poate fi descrisă cu ajutorul unui limbaj de comunicare între oameni, de exemplu româna, engleza, franceza etc. S-a considerat însă că o astfel de descriere este voluminoasă şi neunivocă. Pentru o descriere concisă şi exactă a sintaxei limbajelor de programare, s-au elaborat limbajele speciale, denumite *metalimbaje*. Cel mai răspândit 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 şi în programele PASCAL/C++;

• **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` sunt simboluri terminale, iar <*Cifră*>, <*Literă*> – simboluri neterminale.

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

Prin **formulă metalingvistică** vom înțelege o construcție formată din două părți, stânga și dreapta, separate prin simbolurile "::=" ce au semnificația de *"egal prin definiție"*. În partea stângă 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 dintre 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` *4)* `x4`

2) `a1` *5)* `d0`

3) `c3` *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` sunt 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` sunt conforme cu această definiție, dar `2a` – nu.

În cazul în care alternativele de definire a unui neterminal sunt 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` sunt 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 sau C++.

Întrebări şi exerciţii

❶ Explicați termenii *sintaxă* și *semantică*.

❷ Care este destinația unui metalimbaj?

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

❹ **APLICĂ!** 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 dintre secvențele ce urmează sunt conforme definiției unității lexicale <*Număr*>?

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

Care dintre secvențele ce urmează sunt conforme definiției unității lexicale <*Expresie aritmetică*>?

a)	`0+1`	*f)*	`–13`	*k)*	`21+00000`
b)	`1+0–3`	*g)*	`21+–16`	*l)*	`39+00001`
c)	`0+0+4`	*h)*	`–21–16`	*m)*	`00001–00001`
d)	`1+1–9`	*i)*	`68–13`	*n)*	`379–486`
e)	`6+6+21`	*j)*	`42+650`	*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 dintre secvențele ce urmează sunt 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:`

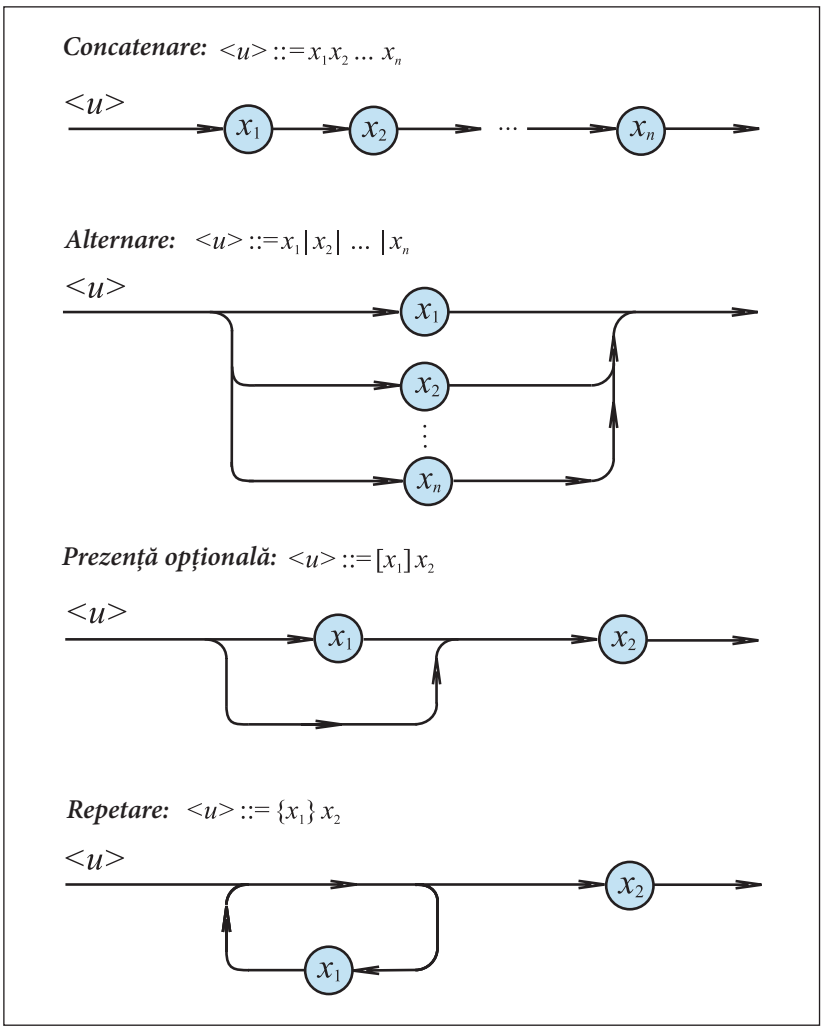


Fig. 1.1. Reprezentarea formulelor BNF prin diagrame sintactice

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*.

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

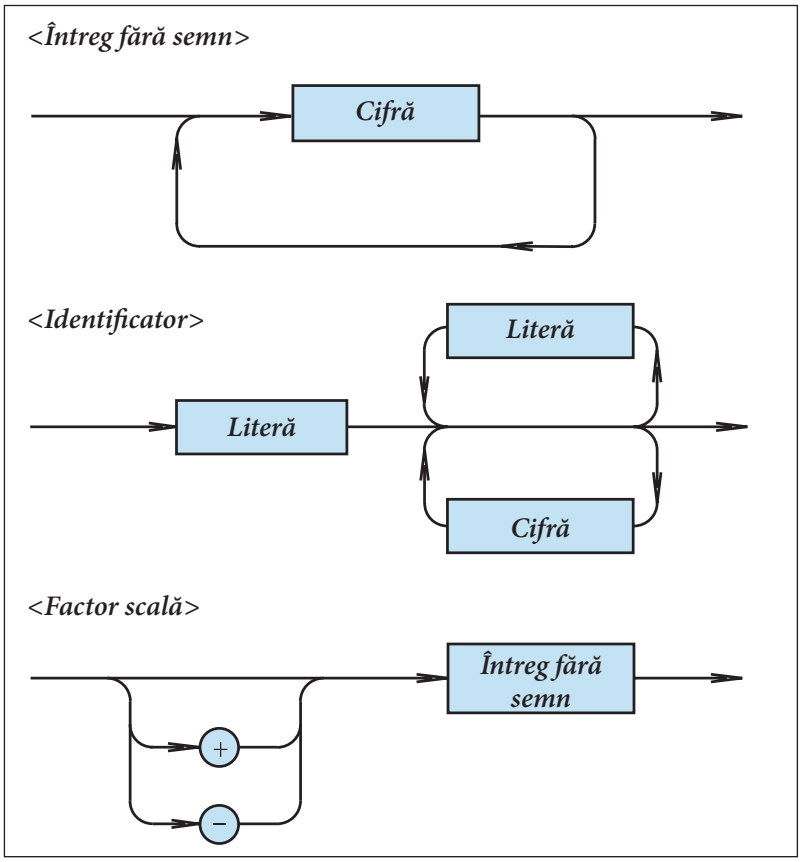


Fig. 1.2. Diagramele sintactice <*Întreg fără semn*>, <*Identificator*> şi <*Factor scală*>

Întrebări şi exerciţii

❶ Care este destinația diagramelor sintactice?

❷ Cum se reprezintă simbolurile terminale şi simbolurile neterminale pe diagramele sintactice?

❸ Cum se reprezintă formulele BNF pe diagramele sintactice?

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* sunt prezentate diagramele sintactice care definesc unitatea gramaticală <*Număr fracționar*>. Determinați care dintre secvențele ce urmează sunt conforme acestor diagrame:

a)	`0.1`	*f)*	`.538`
b)	`+0.1`	*g)*	`721.386.`
c)	`–0.0`	*h)*	`–421`
d)	`9.000`	*i)*	`247.532`
e)	`–538.`	*j)*	`+109.000`

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

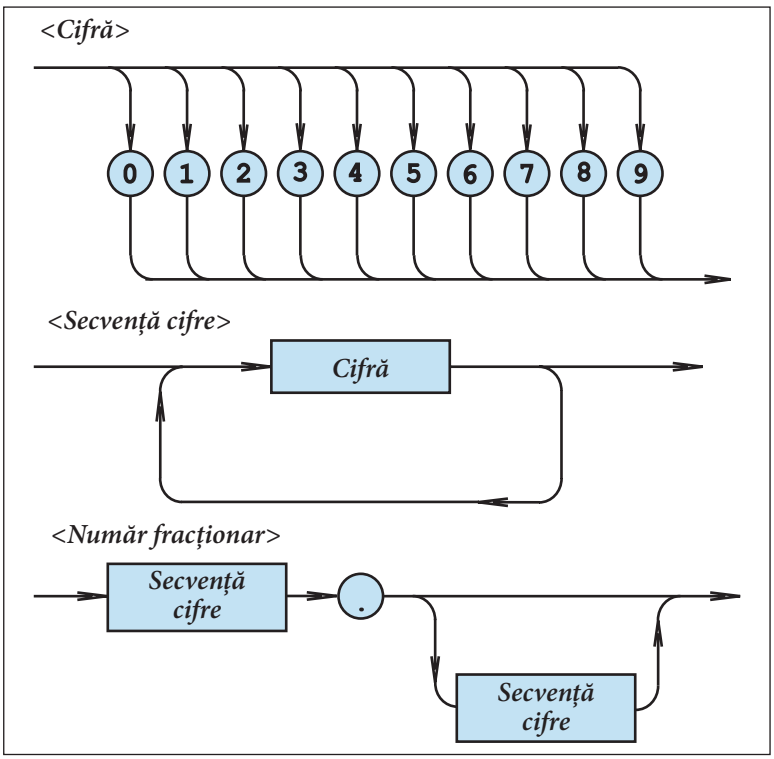


Fig. 1.3. Diagramele sintactice <*Cifră*>, <*Secvență cifre*> și <*Număr fracționar*>

1.4. Alfabetul limbajului

Alfabetul limbajului PASCAL / C++ 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, sfârșit 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, ș, ț ale alfabetului român.

Fiind destinat informaticienilor profesioniști, limbajul C++ utilizează unele caractere ce nu se întâlnesc în limbajul PASCAL standard, de exemplu: _ (*underscore*, linia de jos), ~ (tilda), ^ (caret), \ (*backslash*, bară oblică inversată) etc.

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** unui limbaj de programare.

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 sunt formate din unul sau două caractere:

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

{	acoladă din stânga	<=	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]

C++			
+	plus	<	mai mic
–	minus	>	mai mare
*	asterisc	[	paranteză pătrată din stânga
/	bară	]	paranteză pătrată din dreapta
==	egal	(	paranteză rotundă din stânga
,	virgulă	)	paranteză rotundă din dreapta
:	două puncte	;	punct și virgulă
.	punct	″	ghilimele
&	ampersand	$	dolar
{	acoladă din stânga	<=	mai mic sau egal
}	acoladă din dreapta	>=	mai mare sau egal
#	număr	=	atribuire
%	procent	!=	neegal
\n	linie nouă	\b	backspace
\″	ghilimele	\′	apostrof

Menționăm că, atât în limbajul PASCAL, cât și în C++, în cazul simbolurilor speciale alcătuite din două caractere, de exemplu <= sau >=, între ele nu trebuie să apară niciun spațiu intermediar.

Cuvintele-cheie sunt formate din două sau mai multe litere. Prezentăm în continuare lista acestor cuvinte:

PASCAL			
and	și	**nil**	zero
array	tablou	**not**	nu
begin	început	**of**	din
case	caz	**or**	sau
const	constante	**packed**	împachetat

div	câtul împărțirii	**procedure**	procedură
do	execută	**program**	program
down to	în descreștere la	**record**	înregistrare
else	altfel	**repeat**	repetare
end	sfârșit	**set**	mulțime
file	fișier	**then**	atunci
for	pentru	**to**	la
function	funcție	**type**	tip
goto	treci la	**until**	până ce
if	dacă	**var**	variabile
in	în	**while**	cât
label	etichetă	**with**	cu
mod	restul împărțirii	**delete**	a șterge

C++			
auto	auto	**int**	întreg
break	salt	**long**	lung
case	caz	**near**	apropiat
char	caracter	**new**	nou
const	constante	**private**	privat
continue	continuă	**return**	întoarce
default	predefinit	**short**	scurt
delete	a șterge	**signed**	cu semn
do	execută	**sizeof**	dimensiune
double	dublu	**static**	static
else	altfel	**struct**	structură
enum	enumerare	**switch**	comutator
extern	extern	**typedef**	definire tip
float	real	**union**	uniune
for	pentru	**unsigned**	fără semn
goto	salt	**void**	vid
if	dacă	**volatile**	volatil
inline	în linie	**while**	cât timp

Atât în limbajul PASCAL, cât și în C++ cuvintele-cheie sunt rezervate și nu pot fi folosite în alt scop decât cel dat prin definiția limbajului.

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

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

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

<*Cuvânt-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 sunt în același timp și elemente ale vocabularului PASCAL.

În limbajul PASCAL, pentru a evita confuziile, aceste simboluri, ca elemente ale vocabularului, pot fi redate prin simbolurile echivalente (*, *), (. și respectiv .).

<*Simbol special*> ::= + | - | * | / | = | < | > |] | [| , | (|)
| : | ; | ″ | . | & | { | } | $ | # | <= | >= |
<> | != | % |<*Cuvânt-cheie*>

<*Cuvânt-cheie*>::= **auto** | **asm** | **bool** | **break** |**case** | **catch** | **char** | **const** | **continue** | **class** | **default** | **delete** | **do** | **double** | **elsa** | **enum** | **extern** | **float** | **for** | **friend** | **goto** | **if** | **inline** | **int** | **long** | **namespace** | **new** | **operator** | **private** | **public** | **protected** | **plate** | **register** | **return** | **short** | **signed** | **sizeof** | **static** | **struct** | **switch** | **typedef** | **union** | **unsigned** | **using** | **void** | **volatile** | **virtual**| **while**

Important!

- Simbolurile {, }, [și] utilizate în notația BNF sunt în același timp și elemente ale vocabularului C++.
- În limbajul C++ toate cuvintele rezervate se scriu numai cu litere mici.
- În funcție de compilatorul utilizat, acesta poate conține și alte cuvinte-cheie specifice.

Întrebări şi exerciţii

❶ Memorați cuvintele-cheie ale limbajului PASCAL/C++.

❷ Care este diferența dintre caractere şi simboluri?

❸ Desenați diagramele sintactice pentru unitățile lexicale <*Simbol special*> şi <*Cuvânt-cheie*>.

❹ **DESCOPERĂ SINGUR SAU ÎMPREUNĂ CU COLEGII!** Găsiți pe Internet răspuns la următoarele întrebări:
- Câte simboluri speciale conține limbajul de programare studiat?
- Câte cuvinte-cheie conține limbajul de programare studiat?
- Alegeți oricare alt limbaj de programare şi identificați diferențele dintre numărul de simboluri speciale şi cuvinte-cheie. Cum credeți, influențează oare numărul acestora asupra eficienței proceselor de elaborare şi de execuție a programelor?

1.5.2. Identificatori

Identificatorii sunt unități lexicale care desemnează variabile, constante, funcții, programe etc. Un identificator începe cu o literă, care poate fi urmată de orice combinație de litere şi cifre. Lungimea identificatorilor nu este limitată, dar sunt 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`	*8)*	`listaelevilor`
2)	`y`	*9)*	`listatelefoanelor`
3)	`z`	*10)*	`registru`
4)	`x1`	*11)*	`adresa`
5)	`y10`	*12)*	`adresadomiciliu`
6)	`z01b`	*13)*	`casa10`
7)	`lista`	*14)*	`anul2020`

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

1)	`ListaElevilor`	*3)*	`AdresaDomiciliu`
2)	`ListaTelefoanelor`	*4)*	`BugetulAnului2020`

Menționăm că în construcțiile de bază ale limbajelor PASCAL și C++ nu se utilizează literele ă, â, î, ș, ț ale alfabetului român. Prin urmare, în scrierea identificatorilor semnele diacritice respective vor fi evitate.

Exemple:

1) `Suprafata`

2) `Numar`

3) `NumarElevi`

4) `Patrat`

5) `SirDeCaractere`

6) `NumarIncercari`

N O T E

În construcțiile gramaticale ale limbajului PASCAL, cu excepția șirurilor de caractere, literele mari și mici se consideră echivalente. Prin urmare, în PASCAL, identificatorii `ANM`, `AnM`, `Anm`, `aNM` și `Anm` sunt identici.

Limbajul C++ este sensibil la registru, adică în el se face diferența dintre literele mici și literele mari. Prin urmare, în C++, identificatorii `ANM`, `AnM`, `Anm`, `aNM` și `AnM` sunt diferiți.

În limbajul C++ și în unele implementări ale limbajului PASCAL identificatorii pot conține și chiar începe cu caracterul "_" (*underscore*, linia de jos).

Întrebări şi exerciţii

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

❷ **ANALIZEAZĂ!** Care dintre secvențele ce urmează sunt 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`

n) `B-1`

o) `abc`

p) `Dreptunghi`

q) `iI`

r) `I1j`

s) `Luni`

t) `Luna`

u) `20.07.2020`

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

❸ **ANALIZEAZĂ!** Găsiți perechile de identificatori echivalenți (PASCAL):

a) `x101`

b) `ya15`

c) `radacinaX1`

d) `radacinaX2`

e) `triunghi`

f) `cerc`

g)	`sirdecaractere`	*n)*	`RegistruClasa10`
h)	`registruclasa10`	*o)*	`zilE`
i)	`COTIDIAN`	*p)*	`X101`
j)	`ZILE`	*q)*	`RegistruClasa10`
k)	`CERCURI`	*r)*	`radaciniX1X2`
l)	`SirDeCaractere`	*s)*	`RADACINAX1`
m)	`Triunghiuri`	*t)*	`yA101`

❹ Care este destinația identificatorilor?

❺ **APLICĂ!** Propune nume de identificatori pentru memorarea datelor din tabelul de mai jos.

Date	Identificator
Înălțimea unei persoane	
Vârsta în ani a unei persoane	
Lungimea razei unui cerc	
Perimetrul unui dreptunghi	

❻ Pentru a găsi soluțiile x_1, x_2 ale ecuației pătrate $ax^2 + bx + c = 0$, mai întâi se calculează discriminantul d. Propuneți câteva 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* (p. 26) sunt 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.

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`	`4563906.734`

În scrierea numerelor reale se poate utiliza şi 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.

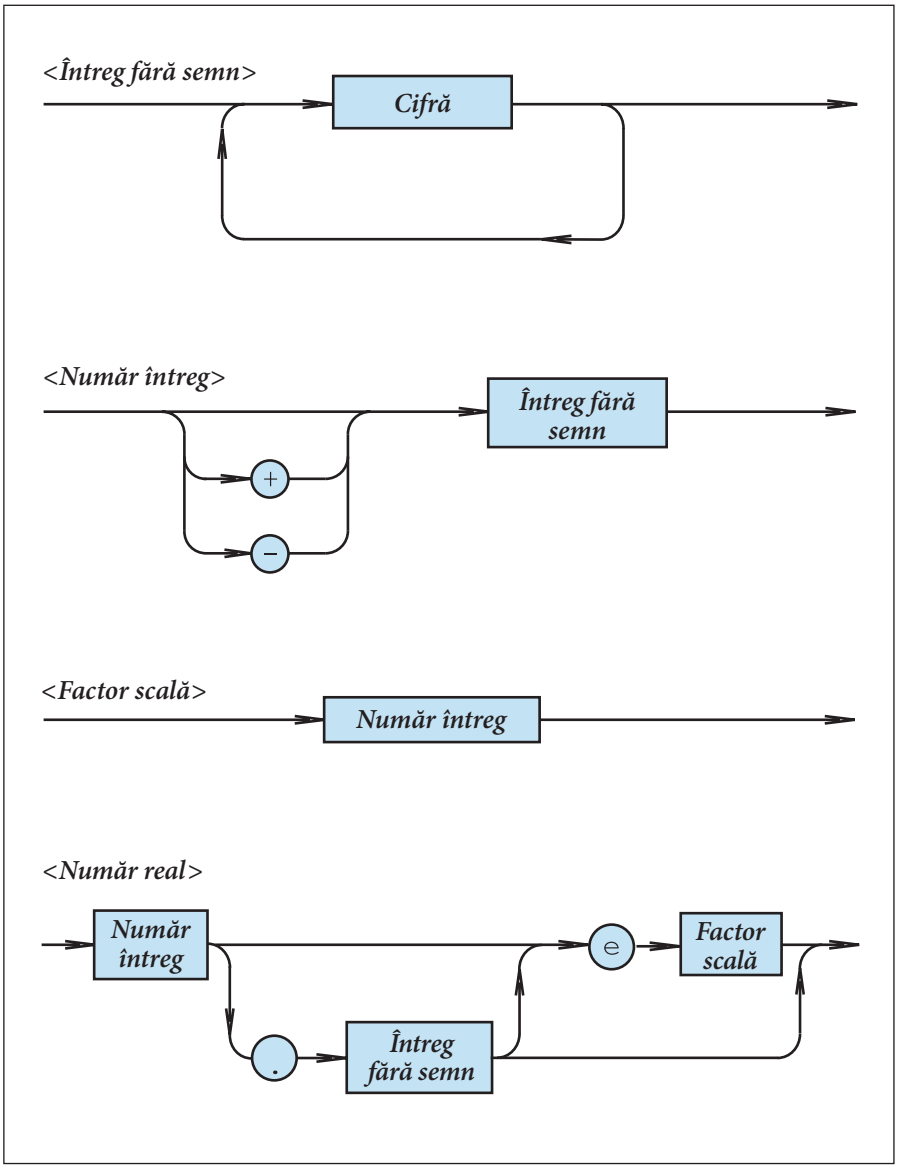


Fig. 1.4. Diagramele sintactice <*Număr întreg*> şi <*Număr real*>

Exemple:

	Forma uzuală	*Notația în PASCAL/C++*
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 $8{,}12 \cdot 10^{-5}$.

N O T Ă

În limbajul C++, la scrierea numerelor reale, partea întreagă sau partea fracționară pot lipsi, dar nu concomitent. De exemplu, ".021" și "318." sunt numere scrise corect.

Întrebări şi exerciţii

❶ **ANALIZEAZĂ!** Care dintre secvențele de caractere ce urmează sunt 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 şi aceeaşi valoare.

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

❸ **ANALIZEAZĂ ŞI APLICĂ!** Care dintre secvențele de mai jos sunt 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 şi aceeaşi 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 **ANALIZEAZĂ!** Indicați pe diagramele sintactice din *figura 1.4* drumurile care corespund numerelor:

a)	`-418`	*e)*	`2.514e+5`	*i)*	`+19.511e+2`
b)	`1961.0`	*f)*	`-1951.12`	*j)*	`+0001`
c)	`2514E+2`	*g)*	`32.014`	*k)*	`-614.85e-3`
d)	`281.3`	*h)*	`591.328`	*l)*	`2013e-4`

1.5.4. Şiruri de caractere

Pascal

Şirurile de caractere sunt 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 unități lexicale ale limbajului PASCAL, în şirurile de caractere pot fi utilizate și literele ă, â, î, ș, ț ale alfabetului român. În acest scop, e necesar ca pe calculatorul la care lucrați să fie instalate **programele-pilot** ce asigură introducerea, afișarea și imprimarea literelor date.

Exemple:

1) `'Șir de caractere'`

2) `'Limba engleză'`

3) `'Suprafață'`

4) `'Număr încercări'`

Unitatea lexicală <*Șir de caractere*> se definește cu ajutorul următoarelor formule BNF:

<*Șir de caractere*> ::= ' <*Element șir*> {<*Element șir*>} '

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

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

C++

Şirurile de caractere sunt şiruri de caractere imprimabile, delimitate de ghilimele. Șirul format dintr-un singur caracter poate fi delimitat de apostrof. Accentuăm că în cazul şirurilor de caractere literele mari și mici apar drept caractere distincte.

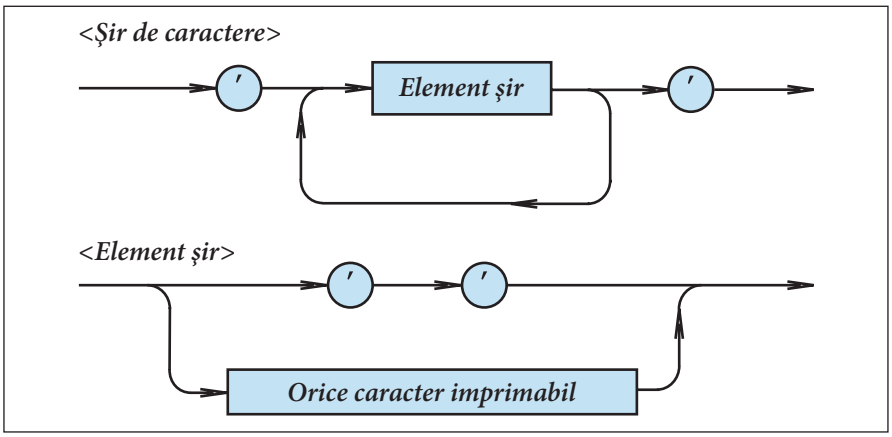


Fig. 1.5. Diagrama sintactică <*Şir de caractere*>, PASCAL

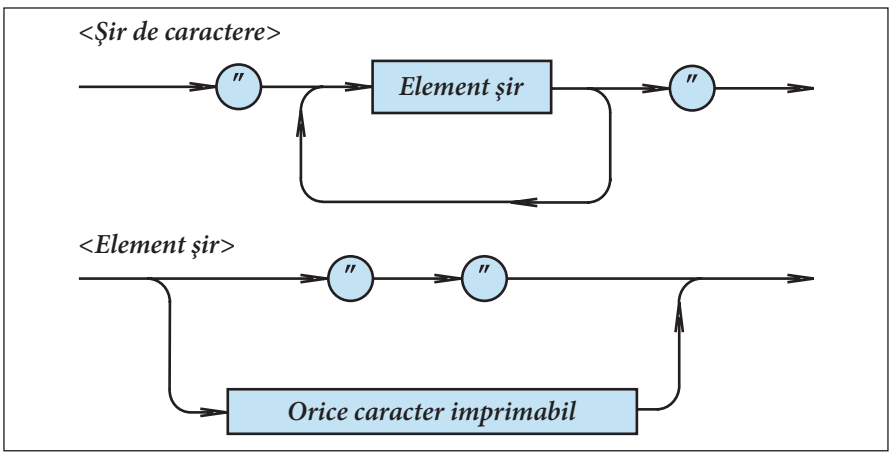


Fig. 1.5.* Diagrama sintactică <*Şir de caractere*>, C++

Exemple:

1) `"Variabila x"`

2) `"Calculul aproximativ"`

3) `"Apostrof \' "`

4) `'B'`

În limbajul C++, în şirurile de caractere pot fi utilizate şi literele ă, â, î, ş, ţ ale alfabetului român. În acest scop, va fi necesar a utiliza biblioteci speciale ce asigură introducerea, afişarea şi imprimarea literelor în studiu.

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 **ANALIZEAZĂ!** Indicați pe diagramele sintactice din *figura 1.5* (*PASCAL*) şi, respectiv, din *figura 1.5** (*C++*), drumurile care corespund următoarelor şiruri de caractere:

	PASCAL		C++
a)	`'variabila z'`	*a)*	`"variabila z"`
b)	`''''`	*b)*	`"\'"`
c)	`'Caracterele "x", "y"'`	*c)*	`"Caracterele 'x', 'y' "`
d)	`'UNITATI LEXICALE'`	*d)*	`"UNITATI LEXICALE"`

❷ **ANALIZEAZĂ!** Care dintre secvențele ce urmează sunt conforme definiției unității lexicale <*Şir de caractere*>:

	PASCAL		C++
a)	`'Numar intreg'`	*a)*	`"Numar intreg"`
b)	`'Sfarsitul programului'`	*b)*	`"Sfarsitul programului"`
c)	`'APOSTROF'`	*c)*	`"APOSTROF"`
d)	`"x"`	*d)*	`"'x'"`
e)	`'functie'`	*e)*	`"functie"`
f)	`'Anul 1997'`	*f)*	`"Anul 1997"`
g)	`'Radacina patrata'`	*g)*	`"Radacina patrata"`
h)	`'Anul '97'`	*h)*	`"Anul \'97"`
i)	`'Lista telefoanelor'`	*i)*	`"Lista telefoanelor"`
j)	`'''`	*j)*	`'''`

❸ **DESCOPERĂ!** Analizați exemplul de mai jos, dați la execuție programul, observați rezultatul execuției.

PASCAL	C++

`Program P;`
`begin`
`  writeln ('Buna ziua!');`
`  writeln ('Astazi este o zi minunata!');`
`end.`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`  cout<<"Buna ziua!"<<endl;`
	`  cout<<"Astazi este o zi minunata!";`
	`return 0;`
	`}`

❹ **APLICĂ!** Modificați programul din exercițiul 3 astfel, încât acesta să afișeze pe ecran următorul șir de caractere:

```
Acesta este un apostrof'
Imi place sa programez!
```

1.5.5. Etichete

În vederea realizării unui salt la o anumită instrucțiune, acea instrucțiune trebuie într-un fel marcată. Acest lucru se face cu ajutorul etichetelor.

Pascal

În limbajul PASCAL, etichetele sunt numere întregi fără semn din domeniul 0, 1, ..., 9999.

`1` `100` `999` `582` `1004`

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

<*Etichetă*> ::= <*Întreg fără semn*>

C++

În limbajul C++, o etichetă constă dintr-un identificator urmat de simbolul ":" (două puncte). Etichetele au propriul spațiu de nume (același identificator poate fi folosit pentru o etichetă și pentru o variabilă fără să interfereze), au ca domeniu de vizibilitate întregul corp al funcției în care apar și sunt recunoscute numai de instrucțiunea de salt `goto`. În orice alt context, o instrucțiune etichetată este executată fără să se țină seama de prezența etichetei.

`Suma1:` `Calcul:` `Afisare:` `Citire:` `Produs:`

Formula BNF care definește unitatea lexicală în studiu are forma:

<*Etichetă*> ::= <*Identificator*><:>

Exemplu:

```
#include <iostream>
using namespace std;
int main()
{
        int a;
        cout<<"a="; cin>>a;
        if (a>0) goto pozitiv;
        if (a<0) goto negativ;
        if (a==0) goto zero;
pozitiv:
       cout<<"numarul e pozitiv";
       goto Final;
```

```
negativ:
        cout<<"numarul e negativ";
        goto Final;
zero:
       cout<<"numarul este egal cu zero";
Final:
       cout<<"\nProgram finisat";
return 0;
 }
```

1.5.6. Directive

Pascal

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

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

Efectiv, directivele sunt 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.

C++

Prelucrarea programului C++ înaintea etapei de compilare se numește **preprocesare**, iar programul ce efectuează astfel de prelucrări se numește **preprocesor**. De obicei, preprocesorul este o parte componentă a compilatorului.

În limbajul C++ informațiile referitoare la modul cum trebuie de prelucrat textul programului înainte de compilarea acestuia se comunică preprocesorului cu ajutorul **directivelor de preprocesare**. Acestea sunt recunoscute de preprocesor prin prezența caracterului # (diez).

Directiva `#include` se folosește pentru a include în textul programului o copie a conținutului unui alt fișier. Această directivă are forma

`#include` <*Nume fișier*>

sau

`#include` *"Nume fișier"*

Aceste două forme diferă doar prin modul în care preprocesorul caută fișierul specificat în directivă.

În primul caz, fișierul este căutat în directoarele-standard. În al doilea caz, fișierul respectiv este căutat mai întâi în directorul curent, și dacă nu este găsit, în directoarele-standard.

Mai mult decât atât, forma a doua permite și indicarea căii de acces către fișierul propriu-zis, fapt ce exclude căutările ulterioare în directoarele-standard.

Accentuăm faptul că directivele de preprocesare nu sunt instrucțiuni și, în consecință, ele nu se termină cu caracterul "`;`" (punct și virgulă).

Exemple:

`#include <stdio>` – cere preprocesorului să includă în program conținutul fișierului `stdio` din directoarele-standard.

`#include "Matrice"` – cere preprocesorului să includă în program conținutul fișierului `Matrice`. Mai întâi, preprocesorul caută acest fișier în directorul curent. Dacă nu-l găsește în directorul curent, preprocesorul va continua căutarea în directoarele-standard.

`#include "C:\ProgrameleMele\Vectori.cpp"` – cere preprocesorului să includă în program conținutul fișierului `Vectori.cpp` din directorul `C:\ProgrameleMele`. Dacă fișierul nu va fi găsit în acest director, el nu va mai fi căutat și în alte directoare, iar preprocesorul va afișa un mesaj de eroare.

În ansamblu, limbajul de programare C++ conține peste zece directive de preprocesare. Aceste directive se studiază în cursurile avansate de informatică.

1.6. Separatori

Orice program PASCAL sau C++ constă din lexeme și separatori. Separatorii folosiți în limbaj sunt spațiul, sfârșitul de linie (retur de car) și comentariul.

Exemple:

	PASCAL		C++
1)	`x div y`	*1)*	`x/y`
2)	`not x`	*2)*	`!x`
3)	`a or b`	*3)*	`a || b`

4)	`begin`
	`  writeln(x);`
	`  writeln(y);`
	`end`
		4)	`{`
			`cout<<x<<endl;`
			`cout<<y<<endl;`
			`}`

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

Menționăm că atât în PASCAL, cât și în C++, simbolurile speciale compuse din două caractere <=, >= etc., cuvintele-cheie, identificatorii, numerele ș.a.m.d. sunt unități lexicale ale programului. Prin urmare, între caracterele componente ale unităților lexicale nu se pot introduce spații sau retururi de car.

Exemple:

	PASCAL		C++	
	Corect	*Incorect*	*Corect*	*Incorect*
1)	`CitireDisc`	`Citire Disc`	`CitireDisc`	`Citire Disc`
2)	`Program`	`Pro gram`	`typedef`	`type def`
3)	`:=`	`: =`	`==`	`= =`
4)	`..`	`. .`	`!=`	`! =`
5)	`345`	`3 45`	`345`	`3 45`
6)	`downto`	`down to`	`while`	`whi le`
7)	`begin`	`be gin`	`endl`	`end l`

Comentariile sunt texte care pot fi introduse în programul sursă, dar nu sunt luate în considerare de compilator și deci nu au efect în timpul executării programului. Comentariile sunt utile pentru o mai bună înțelegere a programului de către persoanele care-l citesc.

Pascal

În limbajul PASCAL comentariile se inserează în program prin secvențe de caractere incluse între acoladele `{` și `}` sau între `(*` și `*)`.

Exemple:

1)
```
{ Introducerea datelor initiale }
```

2)
```
(* Numerele introduse de la tastatura trebuie
sa fie mai mici ca 1000 *)
```

C++

În limbajul C++ comentariile se inserează în program prin secvențe de caractere incluse între `/*` și `*/`. Un astfel de comentariu poate fi scris pe mai multe linii.

În cazul în care dorim să scriem un comentariu scurt ce ocupă doar o singură linie, atunci înaintea acestuia se vor înscrie caracterele `//`.

Exemple:

1)
```
// Introducerea datelor initiale
```

2)
```
/* Numerele introduse de la tastatura trebuie
sa fie mai mici ca 1000 */
```

Accentuăm că utilizarea rațională a comentariilor, spațiilor și a returilor 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 | stanga`

<*Instrucţiune*> ::= <*Comandă*>(<*Număr*>)

<*Program*> ::= **`inceput`** {< *Instrucţiune* >;} **`sfarsit`**

Indicaţi programele corecte sintactic:

a) **`inceput`** `sus(1); dreapta(4); jos(0); stanga(00);` **`sfarsit`**

b) **`inceput`** `sus(1); dreapta(73); jos(0); stanga(00+23);` **`sfarsit`**

c) **`inceput`** `jos(30); dreapta(45); sus(980);` **`sfarsit`**

d) **`inceput`** `stanga(21); jos(50); dreapta(45); sus(980);` **`sfarsit`**

e) **`inceput`** `stanga(3); jos(13); stanga(21)` **`sfarsit`**`;` **`sfarsit`**

f) **`inceput`** `jos(73); dreapta(5); sus(71) stanga(13);` **`sfarsit`**

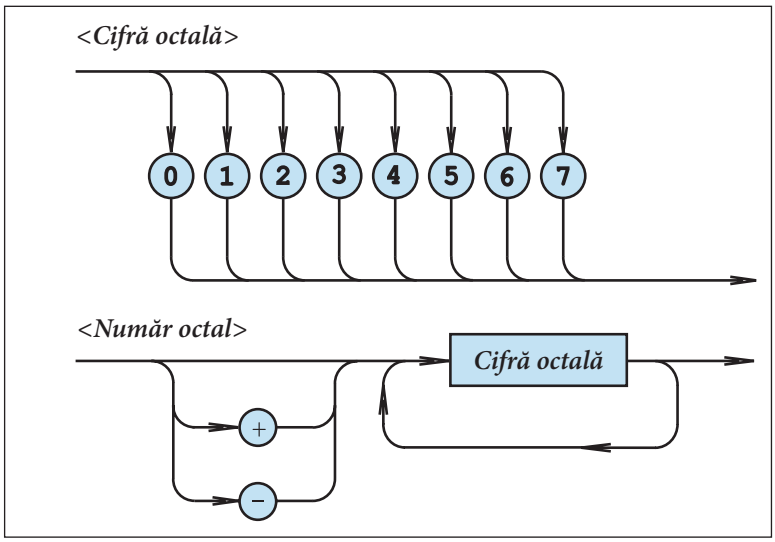
g) **`inceput`** `sus(1); dreapta(-4); jos(0); stanga(10950);` **`sfarsit`**

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

3. În figura de la pagina 36 sunt prezentate diagramele sintactice care definesc unitatea gramaticală <*Număr octal*>.

Determinaţi care dintre secvenţele ce urmează sunt 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. Scrieţi formulele metalingvistice ce corespund diagramelor sintactice din itemul 3.

5. În limbajele PASCAL și C++ 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. Compuneţi cel puţin zece identificatori care ar reflecta specificul problemelor din fizică, matematică, chimie, prelucrarea textelor și imaginilor.

7. Transcrieți din forma uzuală în notația PASCAL (sau C++) următoarele numere:

a)	3,14;	*f)*	–984,52;	*k)*	$-3628{,}297 \cdot 10^{12}$;
b)	265;	*g)*	–523;	*l)*	–38,00001;
c)	23,4635;	*h)*	+28;	*m)*	$35728{,}345452 \cdot 10^{-8}$;
d)	+0,000001;	*i)*	+28000000;	*n)*	24815;
e)	$6{,}1532 \cdot 10^{-5}$;	*j)*	$614{,}45 \cdot 10^{-12}$;	*o)*	–296,0020001.

8. Transcrieți din notația PASCAL/C++ în forma uzuală următoarele numere:

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 limbajelor PASCAL și C++ include următoarele unități lexicale: simboluri speciale, cuvinte-cheie, identificatori, numere, șiruri de caractere, etichete și directive. Indicați unitățile lexicale din programul ce urmează:

PASCAL

```
Program TA1;
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.
```

C++

```
#include <iostream>
using namespace std;
int main ()
{
float a, b, x;
cin>>a>>b;
if (a!=0)
 {
 x=-b/a;
 cout<<"Ecuatia are o singura radacina"<<endl;
 cout<<x;
 }
if ((a==0) && (b==0))
 {
  cout<<"Ecuatia are o multime infinita de radacini"<<endl;}
if ((a==0) && (b!=0))
 {
  cout<<"Ecuatia nu are sens"<<endl;
 }
return 0;
}
```

Exemplu (*PASCAL*): **Program** – cuvânt-cheie; TA1 – identificator; ; – simbol special; **var** – cuvânt-cheie ș.a.m.d.

Exemplu (*C++*): #include – directivă; < și > – simboluri speciale; **iostream** – nume de fișier; **using namespace** std – instrucțiune; ; – simbol special; **int** – cuvânt-cheie ș.a.m.d.

10. (PASCAL) Indicați antetul, partea declarativă și partea executabilă ale programului TA1 din itemul 9.

(C++) Indicați directivele, partea declarativă și funcția principală ale programului prezentat în itemul 9.

11. Programul de mai jos conține erori. Identificați erorile, le corectați și dați la execuție programul.

PASCAL

```
(* Sfarsit de toamna - Vasile
Alecsandri }
Program P 11;
begin
write ("Frunzele cad, zbor in aer"),
write ('si de crengi se dezlipesc');
write ln;
writeln ('Ca frumoasele iluzii dintr-un suflet omenesc. ');
end
```

C++

```
*/ Sfarsit de toamna - Vasile Alecsandri //
#include <stream>
int main
{
cout << "Frunzele cad, zbor in aer",
cout >> " si de crengi se dezlipesc, ";
cout << " endl;"
cout << 'Ca frumoasele iluzii dintr-un suflet omenesc. ' << endl;
return 0;
)
```

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** sunt constituite din cifre, litere, semne, numere, șiruri de caractere ș.a.m.d.

Într-un limbaj cod-calculator datele sunt 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, limbajele de programare 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.

Pe parcursul studierii limbajelor de programare, veți cunoaște și utiliza principalele tipuri de date predefinite (standard), precum și veți învăța să creați tipuri proprii.

Mai jos sunt prezentate câteva tipuri de date predefinite (standard):

PASCAL	C++
`char`	**`char`**
`integer`	**`int`**
`real`	**`float`**
`boolean`	**`bool`**

Tipul de date `integer/int`. Pentru a utiliza eficient memoria și a satisface necesitățile unei multitudini de aplicații, există mai multe tipuri de întregi și reali, (extensii) ce diferă prin memoria alocată și deci prin domeniul de valori.

De exemplu, în versiunile limbajelor de programare, compilatoarele cărora suportă doar întregi pe doi octeți, adică 16 biți (de exemplu, Borland C++, Turbo Pascal 7.0), *tipul de date* `integer/int` va include mulțimea numerelor întregi:

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

Menționăm că există compilatoare care alocă pentru întregi 4 octeți, adică 32 de biți (de exemplu, DELPHI, OBJFPC, Linux). Respectiv, va fi mai mare și mulțimea de valori admise de tipul de date în cauză.

Cu numerele întregi pot fi efectuate următoarele operații:

Operația	PASCAL	C++
Adunarea	+	+
Scăderea	–	–
Înmulțirea	*	*
Restul împărțirii	**mod**	%
Câtul împărțirii	**div**	/

Tipul de date `real`/**`float`** include o submulțime a numerelor reale, operațiile +, –, *, / (împărțirea) etc.

Ca și în cazul numerelor întregi, volumul de memorie alocat unei valori reale depinde de compilatorul utilizat. În limbajul de programare PASCAL/C++, volumul de memorie ocupat de o dată de un anumit tip, în octeți, poate fi aflat cu ajutorul operatorului `sizeof`.

Nu toate operațiile unui tip de date sunt admisibile pe alt tip de date.

De exemplu, operațiile **`div`** și **`mod`** ale tipului de date `integer` din limbajul PASCAL și, respectiv, operația % a limbajului C++ sunt inadmisibile pentru tipul de date `real`/**`float`**.

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

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 fiece moment concret fiecare variabilă are o valoare curentă (de exemplu, 105 ori –36) sau 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.

Forma generală pentru declararea unei variabile este:

PASCAL

`var` <*Identificatori*> **`:`** <*Tip de date*>`;`

C++

<*Tip de date*> <*Identificatori*>`;`

În aceste declarații, <*Tip de date*> poate fi orice tip de date standard sau definit de utilizator, iar <*Identificatori*> este o listă ce conține cel puțin un identificator. În cazul mai multor identificatori, ei vor fi separați prin virgulă.

Exemplu:

PASCAL

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

C++

```
int x, y;
float z;
```

În procesul derulării programului, variabilele `x` și `y` pot lua în calitate de valori numere

întregi, iar variabila `z` – numere reale. Mulțimile de valori ale tipurilor de date din declarațiile de mai sus vor fi studiate în paragrafele următoare.

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/int`, iar `10.0` este o constantă de tip `real/float`.

Pentru a face programele mai lizibile, constantele pot avea denumiri simbolice. Forma generală pentru declararea unei constante este:

PASCAL

```
const <Identificator> = <Valoare>;
```

C++

```
const <Tip de date> <Identificator> = <Valoare>;
```

Exemple:

PASCAL

```
const g = 9;
      pi = 3.14;
```

C++

```
const int g = 9;
const float pi = 3.14;
```

Evident, valorile constantelor `g` și `pi` nu pot fi modificate pe parcursul derulării programului.

Conceptul de dată realizat în limbajul PASCAL/C++ 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, predefinite, a căror definiție se consideră cunoscută, de exemplu în PASCAL: `integer` și `real`, iar în C++: `int`, `float`;

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

Întrebări şi exerciţii

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

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

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

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

❺ **OBSERVĂ!** Determinați tipul variabilelor `r`, `s`, `t`, `x`, `y` şi `z` din declarația ce urmează:

PASCAL

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

C++

```
int r,y;
float s,z;
bool t,x;
```

❻ **APLICĂ!** Scrieți o declarație care ar defini *a*, *b* şi *c* ca variabile întregi, iar *p* şi *q* ca variabile reale.

❼ **OBSERVĂ!** 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`

❽ **OBSERVĂ ŞI APLICĂ!** Analizați programul propus mai jos. Identificați mărimile utilizate şi determinați tipul acestora.

PASCAL

```
Program Ex8;
var z: real;
    a, b: integer;
begin
  a:=2;
  b:=17;
  z:=(a+b)*2;
  writeln('z=', z);
end.
```

C++

```
// Programul Ex8
#include <iostream>
using namespace std;
int main()
{
 float z;
 int a,b;
  a=2;
  b=17;
  z=(a+b)*2;
  cout<< "z="<<z<<endl;
  return 0;
}
```

Dați la execuție programul. Explicați rezultatele afişate pe ecran. Modificați programul astfel, încât acesta să calculeze aria unui dreptunghi cu laturile `a` şi `b`.

❾ **DESCOPERĂ!**

a) Caută pe Internet şi află ce tipuri de date standard are limbajul de programare pe care îl studiezi. Precizează domeniul de valori al fiecărui tip de date standard.

b) Caută pe Internet şi află care tip de date al limbajului de programare pe care îl studiezi poartă numele unui renumit savant? De ce anume acest tip de date a fost aşa denumit?

c) Dați la execuție programul de mai jos. Determinați câți octeți alocă compilatorul pentru valorile fiecărui tip de date.

PASCAL

```
Program Ex9c;
var   a: integer;
      b: real;
      c: boolean;
      d: char;
begin
  WriteLn (SizeOf (a));
  WriteLn (SizeOf (integer));
  WriteLn (SizeOf (real));
  WriteLn (SizeOf (char));
  WriteLn (SizeOf (boolean));
end.
```

C++

```
// Programul Ex9c
#include <iostream>
using namespace std;
int a;
float b;
bool c;
char d;
int main()
{
 cout<<sizeof(a)<<endl;
 cout<<sizeof(int)<<endl;
 cout<<sizeof(float)<<endl;
```

```
 cout<<sizeof(char)<<endl;
 cout<<sizeof(bool);
 return 0;
}
```

2.2. Tipul de date `integer/int`

Pascal

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 personal, 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 sunt: `+`, `-`, `*`, **`mod`**, **`div`** etc. 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.
```

Evident, 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 târziu (*tabelul 3.2*, p. 103).

În limbajul C++, mulțimea de valori ale tipului de date de bază **int** este formată din numerele întregi care pot fi reprezentate pe calculatorul-gazdă. Totodată, menționăm că în limbajul C++ există așa-numiții calificatori sau modificatori, care se pot aplica tipurilor de bază: `short`, `long`, `signed`, `unsigned`. Prin aplicarea acestor calificatori se va schimba domeniul de valori al tipului, care diferă prin numărul de octeți necesari pentru memorarea mărimilor respective, tipul mărimii (cu semn sau fără semn).

Datele de tip `signed` pot fi pozitive sau negative, iar cele de tip `unsigned` sunt întotdeauna pozitive. Implicit, toate tipurile de date sunt `signed`. Calificatorul `long` va extinde mulțimea de valori, iar `short` o va reduce. În consecință, se va mări sau se va micșora volumul de memorie necesar pentru stocarea valorilor respective.

Pentru a specifica numere întregi de diferite tipuri, se pot utiliza sufixe ale modificatorilor:

Sufixul	Tipul modificatorului
`u` sau `U`	`unsigned`
`l` sau `L`	`long`
`ll` sau `LL`	`long long`

Exemple:

```
48          // int
48u         // unsigned int
48l         // long
48ul        // unsigned long
48lu        // unsigned long
```

Valoarea maximă a tipului de date **`int`** este dependentă de compilator și de sistemul de operare al calculatorului-gazdă. De exemplu, în cazul compilatoarelor sub MS-DOS, care alocă cel mult 2 octeți (16 biți), tipul de date `int` avea domeniul de valori `[-32768, 32767]`, iar pentru compilatoarele sub UNIX (Linux) se alocă 4 octeți (32 de biți), astfel `int` are domeniul de valori `[–2147483648, 2147483647]`.

Valoarea maximă a tipului **`int`** este dată de constanta predefinită `INT_MAX`, iar cea minimă de constanta predefinită `INT_MIN`. Aceste constante simbolice, precum și limitele inferioară și superioară ale intervalului de valori pentru diverse tipuri de date, sunt definite în librăria (headerul) `<limits>`.

Programul ce urmează afișează pe ecran valoarea constantei `INT_MAX`.

```
// Programul P2
// Afisarea constantei INT_MAX
#include <iostream>
#include <limits>
using namespace std;
int main()
{
cout<<"INT_MAX="<<INT_MAX<<endl;
return 0;
}
```

Operațiile care se pot face cu valorile întregi sunt: +, –, *, %, / etc.

Operația % (modulo) are sens numai dacă ambii operanzi sunt de tip întreg, iar rezultatul ei este restul întreg de la împărțire. Iată câteva exemple:

```
11 % 3 = 2
30 % 10 = 0
276 % 10 = 6
```

Operația / aplicată asupra numerelor de tip întreg va realiza împărțirea întreagă, iar rezultatul operației / va fi câtul împărțirii întregi. Iată câteva exemple:

```
4 / 2 = 2
18 / 4 = 4
0 / 4 = 0
```

Rezultatele acestor operații pe numere întregi pot fi vizualizate cu ajutorul programului ce urmează.

```
// Programul P3
// Operatii cu date de tipul int
#include <iostream>
using namespace std;
int main()
{
int x, y, z;
cout<<"Introduceti numerele intregi x, y:";
cin>>x>>y;
cout<<"x="<<x<<endl;
cout<<"y="<<y<<endl;
z=x+y; cout<<"x+y="<<z<<endl;
z=x-y; cout<<"x-y="<<z<<endl;
z=x*y; cout<<"x*y="<<z<<endl;
z=x%y; cout<<"x%y="<<z<<endl;
z=x/y; cout<<"x/y="<<z<<endl;
return 0;
}
```

Evident, rezultatele operațiilor +, –, * cu valori întregi trebuie să aparțină mulțimii de valori ale tipului de date `int`. Dacă programatorul nu acordă atenția cuvenită acestei reguli, iar valoarea conținută de o variabilă depășește limitele impuse de tipul de date folosit, se produce așa-numitul *over-flow* (depășirea valorii reale), fapt ce poate cauza erori aparent inexplicabile.

Prioritățile operațiilor +, –, *, /, % vor fi studiate în Capitolul 3 (vezi *tabelul 3.2*, p. 103).

Menționăm că numerele întregi pot fi date în oricare din bazele 10, 8 și 16:

- În baza 10, de exemplu 176, –540. Evident, numerele zecimale pot conține cifrele 0, 1, 2, 3, 4, 5, 6, 7, 8, și 9.
- În baza 8, de exemplu 015, 062. Numerele octale pot conține cifrele 0, 1, 2, 3, 4, 5, 6, 7 și încep întotdeauna cu 0.
- În baza 16, de exemplu 0x15, 0x6f, 0xff. Numerele hexazecimale pot conține cifrele 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F și încep întotdeauna cu 0x.

Întrebări şi exerciţii

❶ Care este mulțimea de valori ale tipului de date *numere întregi*? Ce operații se pot face cu aceste valori?

❷ Când apar erori de depășire? Cum se depistează aceste erori?

❸ **OBSERVĂ!** Dați la execuție programele de mai sus. Explicați rezultatele afișate pe ecran.
Sarcinile a) şi b) pot fi repartizate elevilor pentru activitate individuală sau în perechi.

a)

PASCAL

```
Program Ex3a;
{ Operatii cu date de tipul integer }
var x, y, z : integer;
    m : real;
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.
```

C++

```
// Programul Ex3a
// Operatii cu date de tipul int
#include <iostream>
using namespace std;

int main()
{
int x, y, z;
float m;
cout<<"Introduceti numerele intregi x, y:";
cin>>x>>y;
cout<<"x="<<x<<endl;
cout<<"y="<<y<<endl;
z=x+y; cout<<"x+y="<<z<<endl;
z=x-y; cout<<"x-y="<<z<<endl;
z=x*y; cout<<"x*y="<<z<<endl;
z=x%y; cout<<"x%y="<<z<<endl;
z=x/y; cout<<"x/y="<<z<<endl;
return 0;
}
```

b)

PASCAL

```
Program Ex3b;
{ Program cu eroare }
var x, y, z : integer;
    m : real;
begin
 writeln('Introduceti numerele intregi x, y:');
  readln(x, y);
 writeln('Introduceti un numar real m:');
  readln (m);
  z:=m mod y;
  writeln('m mod y=', z);
  z:=m div y;
  writeln('m div y=', z);
end.
```

C++

```
// Programul Ex3b
// Program cu eroare
#include <iostream>
using namespace std;
int main()
{
int x, y, z;
float m;
cout<<"Introduceti numerele intregi x, y:";
cin>>x>>y;
cout<<"Introduceti numarul real m"; cin>>m;
z=m/y; cout<<"m/y="<<z<<endl;
z=m%y; cout<<"m%y="<<z<<endl;
return 0;
}
```

❹ **DESCOPERĂ!** Activitatea poate fi organizată în grup, fiecare dintre grupuri având una dintre sarcinile propuse mai jos.

1) Elaborați un program ce ar afișa valorile constantelor:

PASCAL: `MaxInt, MaxLongInt`

C++: `INT_MAX, SHRT_MAX, UINT_MAX, LONG_MAX`

Completați tabelul:

Constanta	Valoarea constantei
MaxInt	
...	

2) Consultând literatura de specialitate de pe Internet, dar şi utilizând `sizeof`, determinați volumul de memorie alocat pentru valorile de tip date întregi şi derivatele acestora:

PASCAL: `Byte`, `Word`, `Integer`, `ShortInt`, `LongInt`.
C++: **`int`**, **`unsigned int`**, **`long int`**, **`unsigned long int`**, **`short int`**, **`unsigned short int`**.

Completați tabelul:

Nume tip	Volumul de memorie în biți	Domeniul de valori
Byte		
...		

❺ **STUDIU DE CAZ!** Se consideră programele:

PASCAL	C++

Program P6;
{ Eroare de depasire }
var x : integer;
begin
x:=-2*MaxInt;
writeln(x);
end.
	// Programul P6
	// Eroare de depasire
	#include <iostream>
	#include <limits>
	using namespace std;
	int main()
	{
	int x;
	x=-2*INT_MAX;
	cout<<x;
	return 0;
	}

Program P7;
{ Eroare de depasire }
var x, y : integer;
begin
x:=-MaxInt;
y:=x-10;
writeln(y);
end.
	// Programul P7
	// Eroare de depasire
	#include <iostream>
	#include <limits>
	using namespace std;
	int main()
	{
	int x,y;
	x=-INT_MAX;
	y=x-10;
	cout<<y;
	return 0;
	}

Dați la execuție programele. Analizați erorile de depășire.
Când sunt semnalate erorile de depășire: la compilare sau la execuție?
Dați exemple de valori ale variabilelor `x` și `y` pentru care nu vor apare erori de depășire.

2.3. Tipul de date `real`/`float`

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

În tabelele de mai jos sunt prezentate cele mai utilizate tipuri de date reale și extensiile lor.

PASCAL			
Denumire tip	**Octeți**	**Domeniul de valori**	**Precizia**
`single`	4	$-3{,}4\cdot10^{38}$ ... $3{,}4\cdot10^{38}$	6 cifre zecimale
`real`	6	$-1{,}7\cdot10^{38}$ ... $+1{,}7\cdot10^{38}$	11-12 cifre zecimale
`double`	8	$-1{,}7\cdot10^{308}$ ... $1{,}7\cdot10^{308}$	15-16 cifre zecimale
`extended`	10	$-3{,}4\cdot10^{4932}$ ... $3{,}4\cdot10^{4932}$	18-19 cifre zecimale

C++			
Denumire tip	**Octeți**	**Domeniul de valori**	**Precizia**
`float`	4	$-3{,}4\cdot10^{38}$ ... $3{,}4\cdot10^{38}$	6-7 cifre zecimale
`double`	8	$-1{,}7\cdot10^{308}$ ... $+1{,}7\cdot10^{308}$	15-16 cifre zecimale
`long double`	10	$-3{,}4\cdot10^{4932}$ ... $3{,}4\cdot10^{4932}$	18-19 cifre zecimale

Reamintim că lungimea zonei de memorie ocupate de o dată de un anumit tip (pe câți octeți este memorată data) poate fi aflată cu ajutorul operatorului `sizeof`.

NOTĂ

La fel ca și în cazul tipurilor de date întregi, volumul de memorie alocat pentru stocarea datelor de tip real depinde de compilator și sistemul de operare al calculatorului-gazdă. Evident, vor fi diferite și domeniile de valori ale tipurilor respective de date.

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.

PASCAL

```
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);
```

C++

```
// Programul P8
// Date de tip float
#include <iostream>
using namespace std;
int main()
{
float x,y,z;
x=1.1;
```

```
  writeln('y=', y);
  writeln('z=', z);
end.
```

```
y=-6.14e8;
z=90.3e-29;
cout<<"x="<<x<<endl;
cout<<"y="<<y<<endl;
cout<<"z="<<z;
return 0;
}
```

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

Operațiile care se pot face cu valorile reale sunt +, –, *, / (împărțirea) etc.

Operațiile asupra valorilor reale sunt în general aproximative din cauza erorilor de rotunjire. Evident, rezultatele operațiilor în studiu trebuie să aparțină domeniului de valori ale tipului de date de numere reale. În caz contrar, apar erori de depășire.

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

PASCAL

```
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.
```

C++

```
// Programul P9
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
double x, y, z;
cout<<"Introduceti numerele
reale x si y:";
cin>>x>>y;
cout<<"x="<<x<<endl;
cout<<"y="<<y<<endl;
cout<<scientific<<
      setprecision(10);
z=x+y; cout<<"x+y="<<z<<endl;
z=x-y; cout<<"x-y="<< z<<endl;
z=x*y; cout<<"x*y="<<z<<endl;
z=x/y; cout<<"x/y="<<z;
return 0;
}
```

În *tabelul 2.1* sunt prezentate datele afișate de programele date 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* sunt 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 sunt aproximative. În cazul valorilor x = y = $1{,}7 \cdot 10^{38}$ (linia 5), pentru versiunea Turbo PASCAL 7.0, 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	Rezultate afișate pentru	
			$x + y$	$x - y$
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.	$1{,}7 \cdot 10^{38}$	$1{,}7 \cdot 10^{38}$	`PASCAL: depasire` `C++ :3.4000000000e+39`	`0.0000000000E+00`
6.	$3{,}1 \cdot 10^{-39}$	$3{,}0 \cdot 10^{-39}$	`6.1000000000E-39`	`0.0000000000E+00`

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

Prioritățile operațiilor `+`, `–`, `*`, `/` vor fi studiate în Capitolul 3.

Întrebări şi exerciţii

❶ Cum se scriu numerele reale în limbajul PASCAL/C++?

❷ Determinați domeniul de valori ale tipului de date `real` din versiunea PASCAL, respectiv `double` din versiunea C++ cu care lucrați. Care este precizia acestor numere?

❸ Ce operații se pot face cu tipul de date `real` ale limbajului PASCAL, respectiv cu tipul de date `double` ale limbajului C++? Sunt oare exacte aceste operații?

❹ **APLICĂ ŞI OBSERVĂ!** Elaborați un program pentru determinarea sumei şi diferenței 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 sunt cauzele erorilor de calcul cu tipul de date `real/float`?

2.4. Tipul de date `boolean/bool`

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

PASCAL	C++

`Program P10;`
`{ Date de tip boolean }`
`var x : boolean;`
`begin`
`x:=false;`
`writeln(x);`
`x:=true;`
`writeln(x);`
`end.`
	`// Programul P10`
	`#include <iostream>`
	`using namespace std;`
	`// Date de tip boolean`
	`int main()`
	`{`
	`bool x;`
	`x=false;`
	`cout<<x<<endl;`
	`x=true;`
	`cout<<x;`
	`return 0;`
	`}`

Operațiile predefinite ale tipului de date `boolean` sunt:

Operația predefinită	Notația	
	PASCAL	C++
Negația (inversia logică, operația logică *NU*)	`not`	!
Conjuncția (produsul logic, operația logică *ȘI*)	`and`	&&
Disjuncția (suma logică, operația logică *SAU*)	`or`	||

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

x	not x !x
false	true
true	false

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

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

Fig. 2.1. Tabelele de adevăr ale operaţiilor logice negația, conjuncția și disjuncția

Proprietățile operațiilor logice negația, conjuncția și disjuncția pot fi studiate cu ajutorul următoarelor programe:

PASCAL

```
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.
```

C++

```
// Programul P11
#include <iostream>
/* Operatii cu date de tip
   boolean */
using namespace std;
int main()
{
bool x, y, z;
x=false; y=false;
cout<<"x="<<x<<" y="<<y<<endl;
z=!x;
cout<<"!x="<<z<<endl;
z=x && y;
cout<<"x && y="<<z<<endl;
z=x || y;
cout<<"x || y="<<z<<endl;
cout<<endl;
x=false; y=true;
cout<<"x="<<x<<" y="<<y<<endl;
z=!x;
cout<<"!x="<<z<<endl;
z=x && y;
cout<<"x && y="<<z<<endl;
z=x || y;
cout<<"x || y="<<z<<endl;
cout<<endl;
x=true; y=false;
cout<<"x="<<x<<" y="<<y<<endl;
z=!x;
cout<<"!x="<<z<<endl;
z=x && y;
cout<<"x && y="<<z<<endl;
z=x || y;
cout<<"x || y="<<z<<endl;
cout<<endl;
x=true; y=true;
cout<<"x="<<x<<" y="<<y<<endl;
z=!x;
cout<<"!x="<<z<<endl;
z=x && y;
cout<<"x && y="<<z<<endl;
z=x || y;
cout<<"x || y="<<z<<endl;
return 0;
}
```

N O T E

În limbajul **Pascal**, spre deosebire de variabilele de tip întreg sau real, valorile curente ale variabilelor booleene nu pot fi citite de la tastatură cu ajutorul procedurilor-standard de citire. Din acest motiv, în programul prezentat mai sus valorile curente ale variabilelor `x` și `y` sunt date prin atribuire.

În limbajul **C++**, tipurile boolean, caracter, întreg, real, *enumerare* sunt tipuri aritmetice, deoarece valorile lor pot fi interpretate ca numere întregi. Astfel, într-un program C++ valorile curente ale variabilelor booleene pot fi citite de la tastatură cu ajutorul procedurilor-standard de citire **doar ca valori întregi**, adică `0` în loc de `false` și 1 în loc de `true`. De asemenea, dacă o valoare booleană este atribuită unei variabile de tip întreg, `true` devine 1 și `false` devine 0. Dacă o valoare întreagă este atribuită unei variabile de tip boolean, 0 devine `false` și orice valoare diferită de zero devine `true`.

Prioritățile operațiilor logice negația, conjuncția și disjuncția vor fi studiate în Capitolul 3.

Întrebări şi exerciţii

❶ Numiți mulțimea de valori şi operațiile cu tipul de date `boolean` (`bool`).

❷ Memorați tabelele de adevăr ale operațiilor logice.

❸ **APLICĂ!** Elaborați programe care:
a) afişează pe ecran tabelul de adevăr al operației logice negația;
b) calculează valorile funcției logice $z = x \,\&\, y$ pentru toate valorile posibile ale argumentelor x, y;
c) afişează valorile funcției logice $z = x \vee y$.

❹ **OBSERVĂ!** Dați la execuție programele prezentate ca model în acest paragraf şi observați cum se afişează valorile tipului de date `boolean`. Ce valoare se asociază cu `true` şi ce valoare se asociază cu `false`?

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.

N O T Ă

Reamintim, pentru a afișa însuși caracterul apostrof, folosiți:
În *PASCAL:* dublarea apostrofului, reprezentându-l prin `''''`.
În *C++:* caracterul \ (*backslash*) urmat de apostrof, reprezentat prin `'\''`.

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

PASCAL	C++

`Program P12;`
`  { Date de tip char }`
`var x : char;`
	`// Programul P12`
	`#include <iostream>`
	`using namespace std;`

```
begin
  x:='A';
  writeln(x);
  x:='+';
  writeln(x);
  x:='''';
  writeln(x);
end.
```

```
// Date de tip char
int main()
{
  char x;
  x='A';
  cout<<x<<endl;
  x='+';
  cout<<x<<endl;
  x='\'';
  cout<<x<<endl;
return 0;
}
```

Valorile curente ale unei variabile de tip `char` pot fi citite de la tastatură cu ajutorul procedurilor-standard de citire. Pentru exemplificare, prezentăm mai jos programe care citesc de la tastatură și afișează pe ecran valori de tipul `char`.

PASCAL

```
Program P13;
 { Citirea si afisarea carac-
terelor }
var x : char;
begin
  readln(x); writeln(x);
  readln(x); writeln(x);
  readln(x); writeln(x);
end.
```

C++

```
// Programul P13
#include <iostream>
using namespace std;
/* Citirea si afisarea carac-
terelor */
int main()
{
  char x;
  cin>>x;  cout<<x<<endl;
  cin>>x;  cout<<x<<endl;
  cin>>x;  cout<<x<<endl;
  return 0;
}
```

Caracterele respective se introduc de la tastatură și se afișează pe ecran fără apostrofurile care le încadrează în textul programului propriu-zis.

De regulă, caracterele unui limbaj de programare sunt ordonate conform tabelului de cod *ASCII* (vezi paragraful 1.4).

Mai jos vom prezenta modalitățile de afișare a numărului de ordine al oricărui caracter din mulțimea de valori ale tipului `char`.

	PASCAL		**C++**
1)	`ord('A')= 65`	*1)*	`int('A')=65`
2)	`ord('B')= 66`	*2)*	`int('B')=66`
3)	`ord('C')= 67`	*3)*	`int('C')=67`

ș.a.m.d.

Programele ce urmează afișează pe ecran numărul de ordine a patru caractere citite de la tastatură.

PASCAL	C++

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.
	// Programul P14
	#include <iostream>
	using namespace std;
	int main()
	{
	char x;
	int i;
	cin>>x; i=**int**(x);
	cout<<i<<endl;
	cin>>x; i=**int**(x);
	cout<<i<<endl;
	cin>>x; i=**int**(x);
	cout<<i<<endl;
	cin>>x; i=**int**(x);
	cout<<i;
	return 0;
	}

Pentru a returna caracterul care corespunde numărului de ordine indicat în PASCAL, este folosită funcția `chr`, iar în C++ este folosit convertorul de tip `char`.

Exemple:

PASCAL	C++
1) **chr**(65)='A'	*1)* **char**(65)='A'
2) **ord**(66)='B'	*2)* **char**(66)='B'
3) **ord**(67)='C'	*3)* **char**(67)='C'

Programele ce urmează afișează pe ecran caracterele ce corespund numerelor de ordine citite de la tastatură.

PASCAL	C++

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.
	// Programul P15
	#include <iostream>
	using namespace std;
	int main()
	{
	int i;
	char x;
	cin>>i; x=char(i);
	cout<<x<<endl;
	cin>>i; x=char(i);
	cout<<x<<endl;
	cin>>i; x=char(i);
	cout<<x<<endl;
	cin>>i; x=char(i);
	cout<<x<<endl;
	return 0;
	}

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 şirurilor de caractere.

Întrebări şi exerciţii

❶ Care este mulțimea de valori ale tipului de date `char`?

❷ Cum este ordonată mulțimea de valori ale tipului `char`?

❸ **APLICĂ!** Elaborați programe şi determinați numerele de ordine ale următoarelor caractere:
- cifrele zecimale;
- literele mari ale alfabetului englez;
- semnele de punctuație;
- operatorii aritmetici şi logici;
- caracterele de control şi editare;
- literele alfabetului român (dacă sunt implementate pe calculatorul dvs.).

❹ **APLICĂ!** Elaborați programe şi determinați caracterele care corespund următoarelor numere de ordine:

77	109	79	111	42	56	91	123

❺ **STUDIU DE CAZ!** Elaborați un program care afişează pe ecran setul de caractere al calculatorului cu care lucrați.

2.6. Tipuri de date *enumerare*

Tipurile de date studiate în paragrafele 2.2, 2.3, 2.4 şi 2.5 sunt tipuri predefinite, cunoscute oricărui program PASCAL sau C++. În completare la tipurile predefinite, programatorul poate defini şi utiliza tipuri proprii de date, în particular, tipuri *enumerare*.

Un tip ***enumerare*** include o mulțime ordonată de valori specificate prin identificatori.

În limbajul PASCAL, un tip de date *enumerare* şi mulțimea lui de valori se definește utilizând cuvântul-cheie **`type`** (tip), iar în limbajul C++ prin cuvântul-cheie **`enum`**.

Exemple:

PASCAL

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

C++

```
enum Culoare {Galben, Verde,
     Albastru, Violet};
enum Studii {Elementare=1,
     Medii, Superioare};
enum 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 treilea – numărul doi ş.a.m.d. În limbajul C++, numărul de ordine al unui identificator din lista de enumerare poate fi modificat, indicându-i valoarea de început. Acest lucru a fost utilizat

la definirea tipului *enumerare* `Studii`. Astfel, numărul de ordine al identificatorului `Elementare` va fi 1, pentru `Medii` va fi 2, iar pentru `Superioare` va fi 3.

Analizați exemplele ce urmează pentru a vedea cum putem afla numărul de ordine al unei valori a tipului *enumerare* definit.

Exemple:

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

C++		
1)	`cout<<Galben;`	`// 0`
2)	`cout<<Verde;`	`// 1`
3)	`cout<<Albastru;`	`// 2`
4)	`cout<<Violet;`	`// 3`
5)	`cout<<Elementare;`	`// 1`
6)	`cout<<Medii;`	`// 2`

NOTĂ
În limbajul C++ tipurile *enumerare* sunt tipuri întregi, iar identificatorii tipului *enumerare* se pot folosi la fel ca variabilele întregi.

Programul ce urmează afișează pe ecran numerele de ordine ale valorilor tipurilor de date `Studii` și `Raspuns`.

PASCAL

```
Program P16;
  { Tipul de date Studii si
Raspuns}
type Studii = (Elementare=1,
Medii, Superioare);
Raspuns = (Nu, Da);
var i : integer; {numar de
ordine}
begin
     i:=ord(Elementare); wri-
teln(i);
  i:=ord(Medii); writeln(i);
     i:=ord(Superioare); wri-
teln(i);
  i:=ord(Nu); writeln(i);
  i:=ord(Da); writeln(i);
end.
```

C++

```
// Programul P16
/*Tipul de date Studii si
Raspuns */
#include <iostream>
using namespace std;
int main()
{
 enum Studii {Elementare=1,
Medii, Superioare};
 enum Raspuns {Nu, Da};
 int i;
 i=Elementare; cout<<i<<endl;
 i=Medii; cout<<i<<endl;
 i=Superioare; cout<<i<<endl;
 i=Nu; cout<<i<<endl;
 i=Da; cout<<i<<endl;
 return 0;
 }
```

Variabilele de tip *enumerare* pot lua numai valori din lista de enumerare a tipului de date cu care sunt asociate.

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

PASCAL	C++

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.
	// Programul P17
	// Variabile de tip enumerare
	#include <iostream>
	using namespace std;
	int main()
	{
	enum Culoare {Galben, Verde, Albastru, Violet};
	enum Raspuns {Nu, Da};
	Culoare x; /*variabila de tip Culoare */
	Raspuns y; /*variabila de tip Raspuns */
	int i; /*numar de ordine*/
	x=Albastru;
	i=x; cout<<i<<endl;
	y=Nu;
	i=y; cout<<i<<endl;
	return 0;
	}

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

De exemplu, declarațiile

PASCAL	C++
type Studii = (Elementare, Medii, Superioare); Grade = (Inferioare, Superioare)	**enum** Studii {Elementare, Medii, Superioare}; **enum** Grade {Inferioare, Superioare}

sunt incorecte, întrucât 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 de citire și scriere. Totuși utilizarea tipurilor de date în studiu permite elaborarea unor programe lizibile, simple și eficiente.

Întrebări şi exerciţii

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

	PASCAL		C++
1)	`type Continente = (Europa, Asia, Africa, AmericaDeNord, AmericaDeSud, Australia, Antarctida);`	1)	`enum Continente {Europa, Asia, Africa, AmericaDeNord, AmericaDeSud, Australia, Antarctida};`
2)	`type Sex = (Masculin, Feminin);`	2)	`enum Sex {Masculin, Feminin};`
3)	`type PuncteCardinale = (Nord, Sud, Est, Vest);`	3)	`enum PuncteCardinale {Nord, Sud, Est, Vest};`
4)	`type Etaje = (Unu, Doi, Trei, Patru, Cinci);`	4)	`enum Etaje {Unu, Doi, Trei, Patru, Cinci};`

❹ **STUDIU DE CAZ!** Dați la execuție programul de mai jos. Ce afișează pe ecran acest program? Analizați de ce se afișează anume aceste rezultate? Numiți tipul fiecărei variabile din program.

PASCAL	C++

`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.`
	`// Programul P18`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`enum Litere {A, B, C, D, E, F, G};`
	`Litere x;`
	`char y;`
	`int i;`
	`x=A; i=x; cout<<i<<endl;`
	`y='A'; i=int(y); cout<<i;`
	`return 0;`
	`}`

❺ Se consideră declarațiile:

PASCAL	C++

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

Care dintre instrucțiunile ce urmează sunt corecte?

	PASCAL		C++
1)	`x:=Verde`	1)	`x=Verde`
2)	`y:=Negru`	2)	`y=Negru`
3)	`z:=Alb`	3)	`z=Alb`
4)	`x:=Gri`	4)	`x=Gri`

5)	`y:=Gri`	5)	`y=Gri`
6)	`z:=Violet`	6)	`z=Violet`
7)	`x:=Albastru`	7)	`x=Albastru`
8)	`y:=Azuriu`	8)	`y=Azuriu`

❻ **STUDIU DE CAZ!** Inserați înainte de cuvântul-cheie **`end`** (respectiv înainte de **`return`** pentru varianta C++) al programului `P17` una dintre liniile ce urmează:

	PASCAL		C++
1)	`readln(x);`	1)	`cin>>x;`
2)	`writeln(x);`	2)	`cout<<x;`

Explicați mesajele afișate pe ecran în procesul compilării programului modificat.

2.7. Tipuri de date *subdomeniu* (PASCAL)

NOTĂ
Tipul de date *subdomeniu* este definit în limbajul PASCAL, dar nu are analog în C++. Respectiv, acest paragraf se va studia doar de elevii ce învață a programa în limbajul PASCAL.

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 cuvântul-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` sunt 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` sunt 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` sunt subdomenii ale tipului *enumerare* `T1`. Tipurile de bază ale tipurilor de date *subdomeniu* din exemplele în studiu sunt:

	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 cuvântului-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('Sfarsit');
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` sunt 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 şi exerciţii

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 declarațiile ce urmează:

```
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 moştenit de la tipul de bază.

❹ Care dintre următoarele definiții sunt 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 declanşa 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 **STUDIU DE CAZ!** Se consideră programul `P20`. După lansarea în execuție, utilizatorul introduce `x=1`, `y=2`. Evident, `x-y=-1`. Întrucât 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.`

2.7.* Tipul `void` (C++)

NOTĂ
Tipul de date `void` nu este definit în limbajul PASCAL.
Respectiv, acest paragraf va fi studiat doar de elevii ce învață a programa în limbajul C++.

Tipul `void` este un tip de date special, pentru care mulțimea valorilor este vidă.

Acest tip se utilizează atunci când este necesar să specificăm absența oricărei valori. De exemplu, poate fi utilizat pentru a specifica tipul unei funcții care nu returnează niciun rezultat (rezultatul este de tip `void`) sau care nu are argumente formale (lista argumentelor formale este de tip `void`). Celelalte utilizări ale tipului `void` privesc în special variabilele de tip `pointer` și vor fi precizate la momentul oportun.

Exemplu:

```
void mesaj()
{
  cout << "Eu sunt o functie!";
}
```

2.8. Generalităţi despre tipurile ordinale de date

Tipurile de date `integer`, `boolean`, `char`, *enumerare* și *subdomeniu* din PASCAL, respectiv **`int`**, **`bool`**, **`char`** și **`enum`** din C++, 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` din PASCAL, respectiv **`int`** din C++, este însuși numărul considerat;

2) valorile de adevăr `false` și `true` ale tipului `boolean`/**`bool`** 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, de obicei *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 sunt numerotate prin 0, 1, 2, ... ș.a.m.d., dacă nu se specifică un alt mod de numerotare;

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

Numărul de ordine al unei valori de tip ordinal poate fi aflat și afișat pe ecran.

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

PASCAL

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

C++

```
// Programul P22
#include <iostream>
/* Numerele de ordine ale
valorilor de tip ordinal */
using namespace std;
int main()
{
enum T1 {A, B, C, D, E, F,
         G, H};
cout<<int(-32)<<endl;   // -32
cout<<int(true)<<endl;  // 1
cout<<int('A')<<endl;   // 65
cout<<A<<endl;          // 0
cout<<B<<endl;          // 1
return 0;
}
```

NOTĂ

Operatorul **`int`** din programul C++ de mai sus se utilizează pentru conversia valorii unei variabile de un anumit tip într-un alt tip de date. Un astfel de operator se numește **operator de conversie**.

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

Operații relaționale	PASCAL	C++
Mai mic	<	<
Mai mic sau egal	<=	<=
Egal	=	==
Mai mare sau egal	>=	>=
Mai mare	>	>
Diferit	<>	!=

Rezultatul unei operații relaționale este de tip `boolean`, adică poate obține una dintre valorile `false` sau `true`, în funcție de numerele de ordine ale valorilor operanzilor.

Exemple:

PASCAL

Fie

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

Rezultatul operației `Verde<Violet` este `true`, deoarece `ord(Verde)=1`, `ord(Violet)=3` și 1 este mai mic ca 3.

Rezultatul operației `Galben>Violet` este `false`, deoarece `ord(Galben)= 0`, `ord(Violet)=3`, iar 0 nu este mai mare ca 3.

C++

Fie

```
enum Culoare {Galben, Verde, Albastru, Violet};
```

Rezultatul operației `Verde<Violet` este `true`, deoarece numărul de ordine al valorii `Verde` este `1`, iar pentru `Violet` este 3 și 1 este mai mic ca 3.

Rezultatul operației `Galben>Violet` este `false`, deoarece numărul de ordine al valorii `Galben` este `0`, pentru `Violet` este 3, iar 0 nu este mai mare ca 3.

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

PASCAL

```
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.
```

C++

```
// Programul P23
/* Operatii relationale asupra
   valorilor de tip ordinal */
#include <iostream>
using namespace std;
int main()
{
enum Culoare {Galben, Verde,
     Albastru, Violet};
cout<<(Verde<Violet)<<endl;
//true
cout<<(Verde<=Violet)<<endl
//true
cout<<(Verde==Violet)<<endl;
//false
cout<<(Verde>=Violet)<<endl;
//false
cout<<(Verde>Violet)<<endl;
//false
cout<<(Verde!=Violet)<<endl;
//true
return 0;
}
```

Pentru valorile tipurilor ordinale de date pot fi aflați predecesorii și succesorii acestora. *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 $i+1$.

NOTE

În limbajul PASCAL, pentru determinarea succesorilor și predecesorilor există funcțiile predefinite `succ` și `pred`;
În limbajul C++ nu există funcții predefinite pentru determinarea directă a succesorilor și predecesorilor. În consecință, se va recurge la operații echivalente ce vor avea același efect, utilizând **operatori de conversie**. În limbajul C++, în expresii aritmetice orice dată de tip enumerare este tratată ca un număr întreg, conversia către **`int`** fiind implicită, însă conversia unui întreg către un tip enumerare trebuie cerută explicit.

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

	PASCAL		C++
1)	`pred(Verde) = Galben`	*1)*	`Verde-1 == Galben`
2)	`succ(Verde) = Albastru`	*2)*	`Verde+1 == Albastru`
3)	`pred(Albastru) = Verde`	*3)*	`Albastru-1 == Verde`
4)	`succ(Albastru) = Violet`	*4)*	`Albastru+1 == Violet`

Evident, valoarea cea mai mică nu are predecesor, iar valoarea cea mai mare nu are succesor. Programele ce urmează afișează pe ecran predecesorii și succesorii valorilor ordinale `'B'`, `0` și `'0'`.

PASCAL

```
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.
```

C++

```
// Programul P24
/* Determinarea predecesorilor si succesorilor valorilor
ordinale */
 #include <iostream>
using namespace std;
int main()
{
cout<<char('B'-1)<<endl;    // 'A'
cout<<char('B'+1)<<endl;    // 'C'
cout<<0-1<<endl;            // -1
cout<<0+1<<endl;            // 1
cout<<char ('0'-1)<<endl;   // '/'
cout<<char('0'+1)<<endl;    // '1'
return 0;
}
```

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

NOTĂ
Subliniem faptul că tipul de date `real/float` nu este un tip ordinal.
Prin urmare, pentru valorile reale nu pot fi determinate numărul de ordine, predecesorul și succesorul. Nerespectarea acestei reguli va declanșa erori.

Întrebări şi exerciţii

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

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

❸ **ANALIZEAZĂ!** Ce vor afișa pe ecran programele ce urmează?

PASCAL

```
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.
```

C++

```
// Programul P25
#include <iostream>
using namespace std;
int main()
{
enum Zi {L, Ma, Mi, J, V, S, D}
z1,z2;
z1=Ma;
cout<<z1<<endl;
cout<<z1-1<<endl;
cout<<z1+1<<endl;
z1=Mi; z2=V;
cout<<(z1<z2)<<endl;
cout<<(z1>z2)<<endl;
cout<<(z1!=z2)<<endl;
return 0;
}
```

❹ **OBSERVĂ!** Excludeți din programele de mai jos linia care conține o eroare.

PASCAL

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

C++

```
// Programul P26
#include <iostream>
#include <limits>
using namespace std;
// Eroare
int main()
{
int i;
i=INT_MAX;
cout<<i-1<<endl;
cout<<i+1<<endl;
return 0;
}
```

Ce rezultate vor fi afişate pe ecran după execuția programelor modificate?

❺ Comentați programul PASCAL 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 afişate pe ecran după execuția programului modificat?

2.9. Definirea tipurilor de date

Limbajele de programare oferă utilizatorului tipurile predefinite de date întregi, reale, valori booleene, caractere etc. Dacă e necesar, programatorul poate defini tipuri proprii de date, de exemplu, *enumerare*.

Pascal

Denumirea unui tip de date şi mulțimea lui de valori, în PASCAL, 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 sunt 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';
```

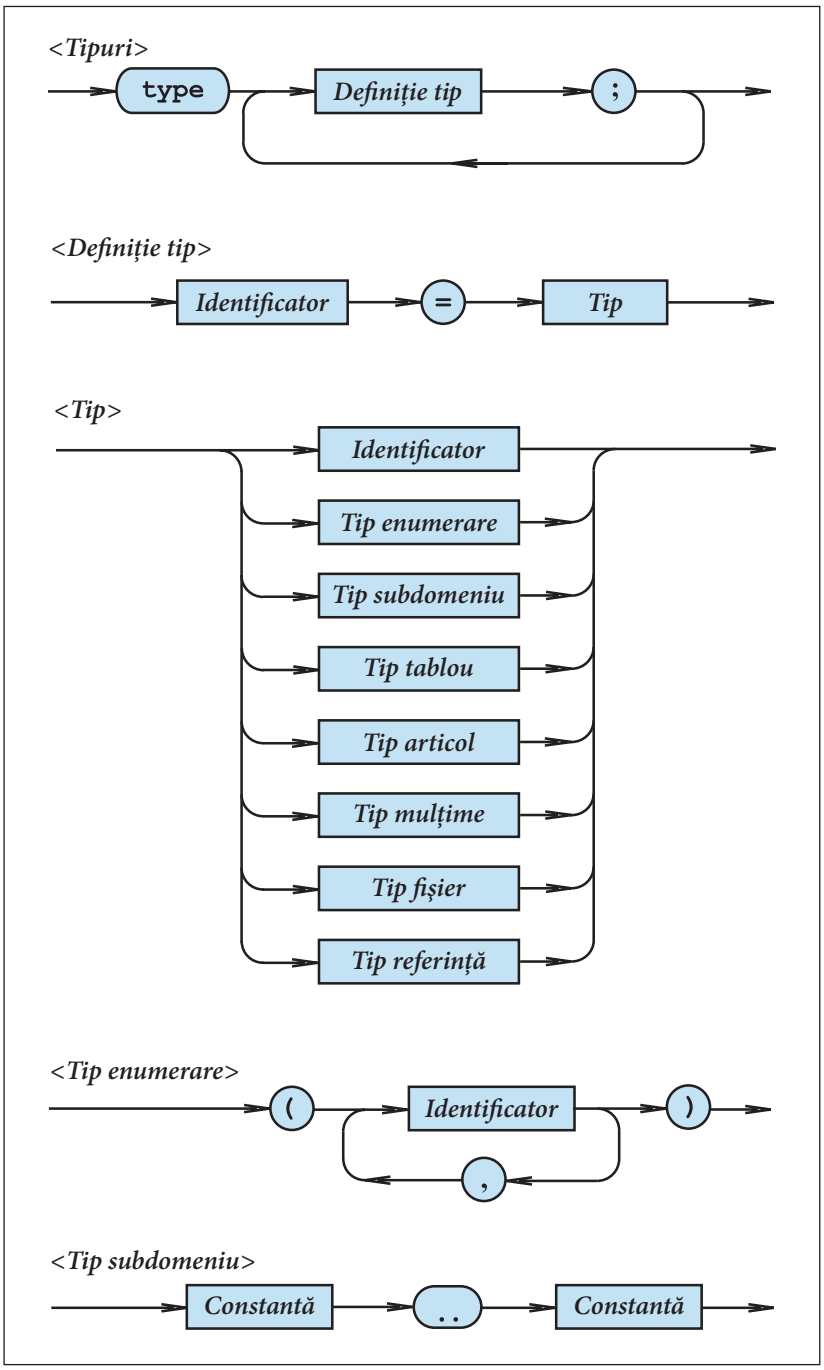


Fig. 2.2. Diagramele sintactice pentru definirea tipurilor de date, PASCAL

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

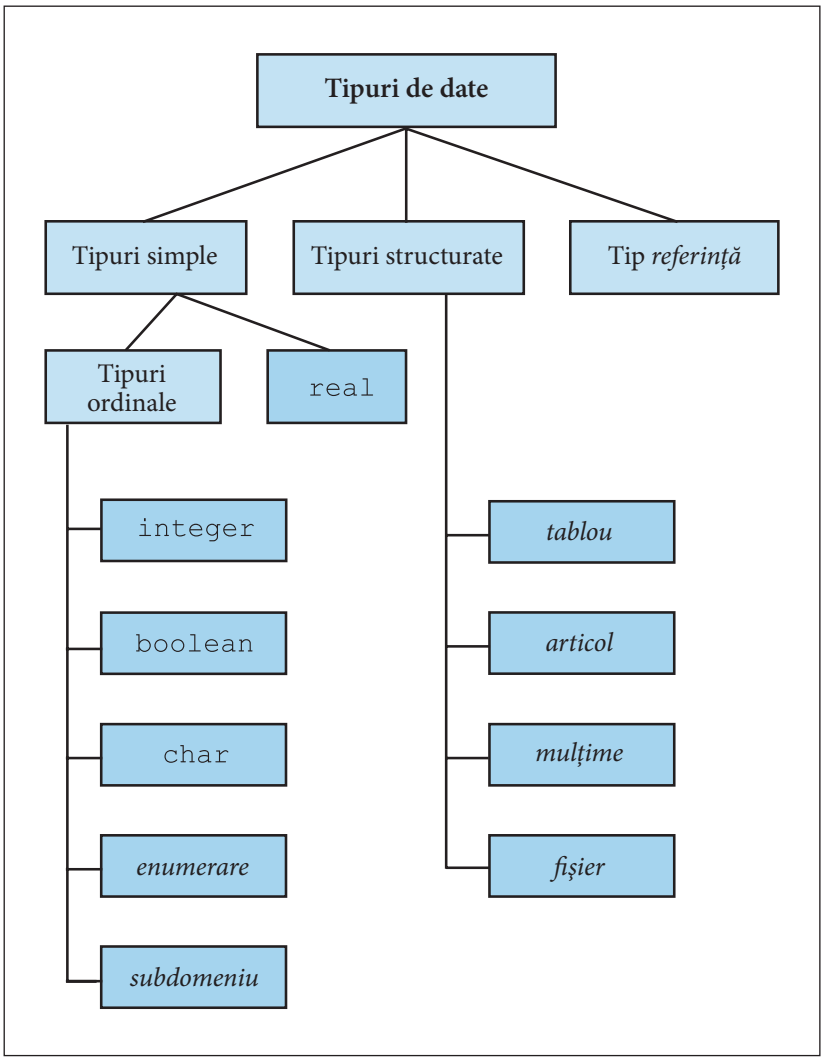


Fig. 2.3. Clasificarea tipurilor de date, PASCAL

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

Două tipuri sunt **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` sunt identice.

Două tipuri se consideră identice și atunci când se definesc cu nume diferite, dar aceste nume sunt echivalente prin tranzitivitate.

De exemplu, fie

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

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

Două tipuri sunt **compatibile** atunci când este adevărată măcar una dintre afirmațiile:

1) cele două tipuri sunt identice;
2) un tip este un subdomeniu al celuilalt tip;
3) ambele tipuri sunt 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` sunt compatibile. Tipurile `Zi` și `Culoare` sunt tipuri incompatibile. Prin urmare, se admit operațiile relaționale:

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

etc. și nu se admit operațiile de tipul:

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

ș.a.m.d.

În completare la tipurile de date definite explicit de utilizator cu ajutorul cuvântului-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.

Fiind destinat informaticienilor profesioniști, limbajul C++ oferă programatorilor mai multe posibilități de utilizare și de creare a unor tipuri noi de date.

Astfel, cu ajutorul cuvântului-cheie **`typedef`**, tipurilor existente li se pot asocia denumiri noi (alias-uri):

`typedef` <*Nume tip*> <*Nume tip nou*>;

Exemple:

1)
```
typedef int Intreg;
Intreg x1, i, t1, t2;
```

2)
```
typedef float Real;
Real a, b, c, x, delta, x1, x2;
```

Conform exemplelor de mai sus, identificatorul `Intreg` devine denumirea a doua a tipului de date **`int`**, iar identificatorul `Real` – denumirea a doua a tipului de date **`float`**.

Prin urmare, deși variabilele `x1`, `i`, `t1` și `t2` sunt declarate ca fiind de tipul `Intreg`, ele, de fapt, vor fi de tipul **`int`**. Într-un mod similar, deși variabilele `a`, `b`, `c`, `x`, `delta`, `x1` și `x2` sunt declarate ca fiind de tipul `Real`, ele, de fapt, vor fi de tipul **`float`**.

În practica cotidiană, alias-urile se utilizează pentru a face programele mari mai lizibile.

Tipurile de date *enumerare* se definesc cu ajutorul cuvântului-cheie **`enum`**:

`enum` <*Nume tip*> {<*Identificator*> {, <*Identificator*>};

În construcția gramaticală de mai sus, unitatea gramaticală {<*Identificator*> {, <*Identificator*>} reprezintă mulțimea de valori a tipului de date <*Nume tip*>.

Exemple:

1)
```
enum ZideLucru {Luni, Marti, Miercuri, Joi, Vineri};
ZideLucru Z;
```

2)
```
enum GenDeMuzica {Blues, Etno, Folk, Disco, HipHop,
                  Rock, Pop, Jazz};
GenDeMuzica G;
```

În exemplele de mai sus sunt definite două tipuri de date *enumerare*: `ZiDeLucru` și `GenDeMuzica`. Evident, variabila `Z` poate lua valorile `Luni`, ..., `Vineri`, iar variabila `G` – valorile `Blues`, ..., `Jazz`.

În limbajul C++ puteți găsi mai multe moduri de clasificare a tipurilor de date, în simple și structurate, în predefinite (tipuri standard) și definite de utilizator, în fundamentale și derivate. Cel mai utilizat mod de clasificare a tipurilor de date este prezentat în *figura 2.3**.

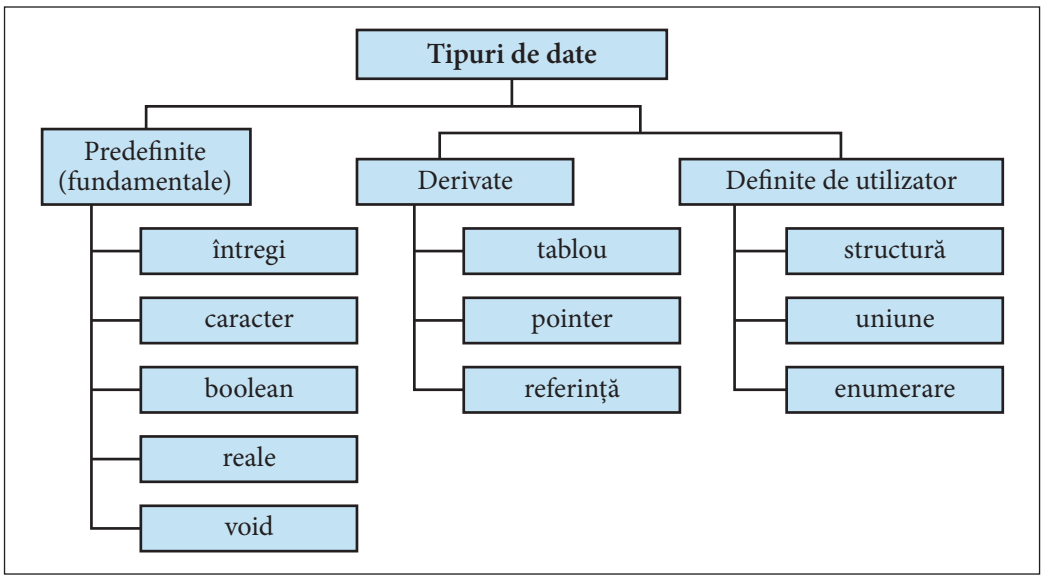


*Fig. 2.3.** Clasificarea tipurilor de date, limbajul C++

Următoarele tipuri predefinite sunt fundamentale: tipul **void**, tipul logic **bool**, tipurile aritmetice întregi **char** și **int** și tipurile aritmetice în virgulă mobilă **float** și **double**. Toate celelalte tipuri (tablouri, funcții, pointeri, referințe, structuri, uniuni) sunt tipuri derivate ce pleacă de la aceste tipuri fundamentale. Tipurile derivate și cele definite de utilizator vor fi studiate mai târziu.

În anumite construcții variabilele și constantele trebuie să fie de tipuri identice. Două tipuri sunt **identice**, dacă ele au fost definite cu același nume de tip.

De exemplu, fie

```
typedef int T4;
typedef int T5;
```

sau

```
typedef int T4, T5;
```

Aici tipurile **int**, T4 și T5 sunt identice.

Două tipuri se consideră identice și atunci când se definesc cu nume diferite, dar aceste nume sunt echivalente prin tranzitivitate.

De exemplu, fie

```
typedef double T6;
typedef T6 T7;
typedef T7 T8;
```

Aici double, T6, T7 și T8 sunt tipuri identice.

În general, limbajul C++ oferă programatorului o flexibilitate foarte mare în utilizarea tipurilor de date, fapt ce explică popularitatea sa în industria produselor program (*software*). Această particularitate se reflectă cel mai mult în manipularea tipurilor (lucru inadmisibil în Pascal). În schimb, limbajul PASCAL este mai indicat pentru utilizarea în scopuri didactice, limitarea intenționată a gradului de libertate în manipularea tipurilor de date, contribuind la respectarea principiului "de la simplu la compus".

Cea mai evidentă flexibilitate se atestă în cazul tipului **char** (caracter). Acest tip este de fapt un tip întreg, reprezentat pe un octet, asemănător cu tipul byte din PASCAL. Astfel, secvența următoare de cod este perfect legală, acest tip de "artificii" de programare fiind des întâlnit:

```
int i;
char c;

c='A';
i=c;
cout<<"Codul ASCII al caracterului "<<c<<" este "<<i<<endl;
i='A'+1;
cout<<"si urmatorul caracter are codul "<<i;
```

În urma rulării pe calculator a acestei secvențe, pe ecran se va afișa:

```
Codul ASCII al caracterului A este 65
si urmatorul caracter are codul 66
```

Compilatoarele C++ verifică compatibilitățile de tip în următoarele cazuri:
– la atribuire;
– la transmiterea de parametri;
– la calcularea valorilor expresiilor.

Consecințele verificării depind de specificația limbajului și, eventual, de particularitățile sistemului de operare al calculatorului-gazdă.

În completare la tipurile de date definite explicit de utilizator cu ajutorul cuvintelor-cheie **`typedef`** și **`enum`**, într-un program C++ pot fi definite și tipuri anonime de date (fără denumire).

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

Exemple:

1)
```
enum ZiDeOdihna {Sambata, Duminica}; // tip explicit
ZiDeOdihna Zi;
```

2)
```
enum {Sambata, Duminica} Zi; // tip anonim
```

În primul exemplu, tipul de date *enumerare* `ZiDeOdihna` este definit explicit. Ulterior, acest tip este folosit în declarația variabilei `Zi`. În al doilea exemplu, tipul variabilei `Zi` este anonim (nu are denumire), fiind definit implicit în declararea acesteia.

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

Întrebări şi exerciţii

❶ **OBSERVĂ!** Se consideră programele:

PASCAL

```
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;
```

C++

```
// Programul P28
#include <iostream>
using namespace std;
int main();
{
enum T3 {A, B, C, D, E, F, G, H};
typedef T3 T4;
typedef int T5;
typedef double T6;
typedef char T7;
typedef bool T8;
T5 j;
T3 m;
T4 n;
double p;
T6 q;
char r;
T7 s;
bool t;
```

```
    z : T8;
    y : real;
begin
  { calcule ce utilizeaza }
  { variabilele in studiu }
  writeln('Sfarsit');
end.
```

```
T8 z;
double y;
// calcule ce utilizeaza
// variabilele in studiu
cout<<"Sfarsit";
return 0;
}
```

Precizați tipurile de date ale programului PASCAL/C++. Ce valori poate lua fiecare dintre variabilele din acest program? Care tipuri sunt compatibile?

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

❸ **ANALIZEAZĂ!** Precizați tipurile anonime de date din următorul program:

PASCAL

```
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('Sfarsit')
end.
```

C++

```
// Programul P29
#include <iostream>
using namespace std;
int main()
{
typedef int T1;
enum {Rosu, Galben,Verde}
Semafor;
enum {Iunie, Iulie, August} v;
enum Numere {Unu, Doi,Trei,
Patru, Cinci};
T1 i;
Numere n;
// calcule ce utilizeaza
// variabilele in studiu
cout<<"Sfarsit";
return 0;
}
```

Ce valori poate lua fiecare variabilă din programul PASCAL/C++?

❹ Se consideră declarațiile:

PASCAL

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

C++

```
typedef bool T1;
typedef T1 T2;
typedef T2 T3;
typedef T3 T4;
T4 x;
```

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

❺ În care cazuri două tipuri de date sunt identice? Dați exemple.

❻ În care cazuri două tipuri de date sunt compatibile? Dați exemple.

❼ Se consideră declarațiile:

PASCAL	C++

`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';`
	`typedef int D1;`
	`typedef D1 D2;`
	`typedef double D3;`
	`typedef D3 D4;`
	`enum D6 {A, B, C, D, E, F, G, H};`
	`typedef D6 D5;`

Precizați tipurile identice şi tipurile compatibile de date.

2.10. Declaraţii de variabile

Cunoaștem deja că fiecare variabilă care apare într-un program în mod obligatoriu se asociază cu un anumit tip de date.

Pentru aceasta se utilizează următoarele construcții gramaticale:

PASCAL

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

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

C++

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

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

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

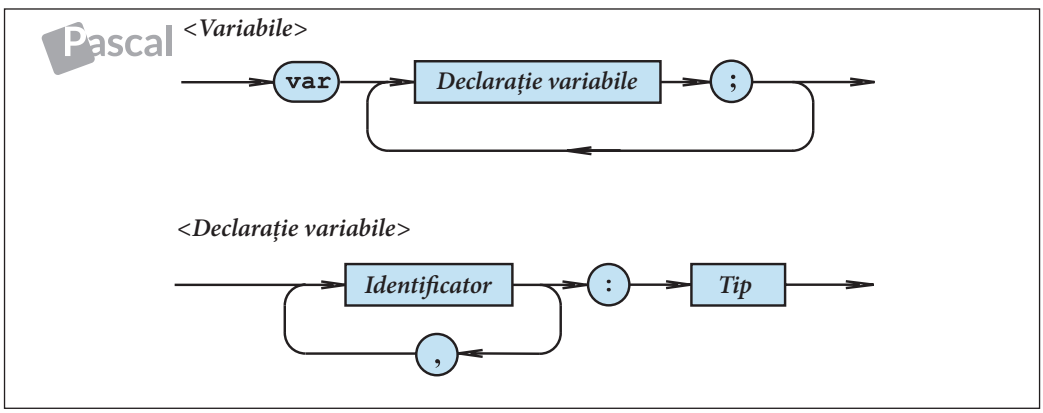


Fig. 2.4. Diagramele sintactice <*Variabile*> și <*Declarație variabile*>, PASCAL

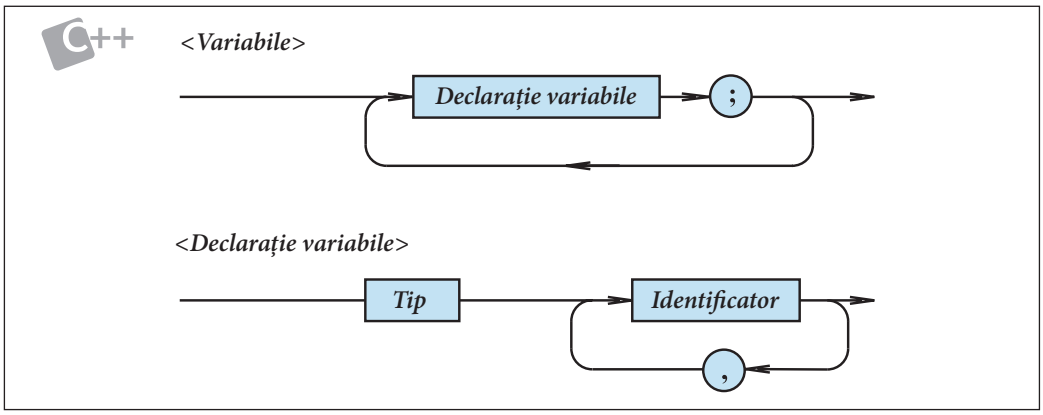


*Fig. 2.4.** Diagramele sintactice <*Variabile*>, <*Declarație variabile*>, C++

Declarațiile de variabile pot utiliza tipuri predefinite de date (întregi, reale, caractere, booleene etc.) și tipuri definite de utilizator (enumerare, tablou etc.).

Exemple:

	PASCAL		C++

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

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;`
		2)	`enum T1 {A, B, C, D, E, F, G, H};`
			`double x, y, z;`
			`char r, s;`
			`int i, j;`
			`T1 p;`

3)	`type Zi = (L, Ma, Mi, J, V, S, D);`
	`var x, y : real;`
	`z : Zi;`
	`zl : L..V;`
	`m, n : 1..10;`
		3)	`enum Zi {L, Ma, Mi, J, V, S, D};`
			`double x, y;`
			`Zi z;`
			`enum {A,B,C} z1;`
			`enum {True, False} m, n;`

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 şi exerciţii

❶ **OBSERVĂ!** Determinaţi tipul variabilelor din următorul program:

PASCAL

```
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 in care
    se utilizeaza variabilele
    in studiu}
  writeln('Sfarsit');
end.
```

C++

```
// Programul P30
#include <iostream>
using namespace std;
int main()
{
typedef int T1;
enum Studii {Elementare,
Medii, Superioare};
typedef double T2;
enum Culoare {Galben, Verde,
Albastru, Violet};
float x;
T1 y;
int i;
T2 j;
bool p;
Culoare c,q;
Studii s;
int r;
// calcule in care se utilizeaza
// variabilele in studiu
cout<<"Sfarsit";
return 0;
}
```

Precizaţi valorile pe care le poate lua fiecare variabilă şi operaţiile tipului corespunzător de date.

❷ Indicaţi pe diagramele sintactice din *figura 2.4* drumurile care corespund declaraţiilor de variabile din programul prezentat în exerciţiul 1.

❸ **APLICĂ!** Daţi la execuţie programul şi explicaţi mesajele ce vor fi afişate pe ecran pe parcursul compilării următorului program. Modificaţi programul astfel, încât să vă afişeze rezultatul: 3.

PASCAL

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

C++

```
// Programul P31
#include <iostream>
using namespace std;
int main()
{
int i, j;
i=1; j=2; k=i+j;
cout<<"k="<<k<<endl;
return 0;
}
```

❹ Cum se declară variabilele unui tip anonim de date?

2.11. Definiţii 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, limbajele de programare PASCAL și C++ permit 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 construcțiilor gramaticale:

PASCAL

<*Constante*> ::= **const** <*Definiție constantă*>; { <*Definiție constantă*>;}
<*Definiție constantă*> ::= <*Identificator*> = <*Constantă*>
<*Constantă*> ::= [+ | -] <*Număr fără semn*> | [+ | -] <*Nume de constantă*> | <*Șir de caractere*>

C++

<*Constante*> ::= **const** <*Definiție constante*>;
<*Definiție constante*> ::= <*Tip*> <*Identificator*> = <*Constantă*> {, <*Identificator*> = <*Constantă*>}
<*Constantă*> ::= [+ | -] <*Număr fără semn*> | [+ | -] <*Nume de constantă*> | <*Șir de caractere*>

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

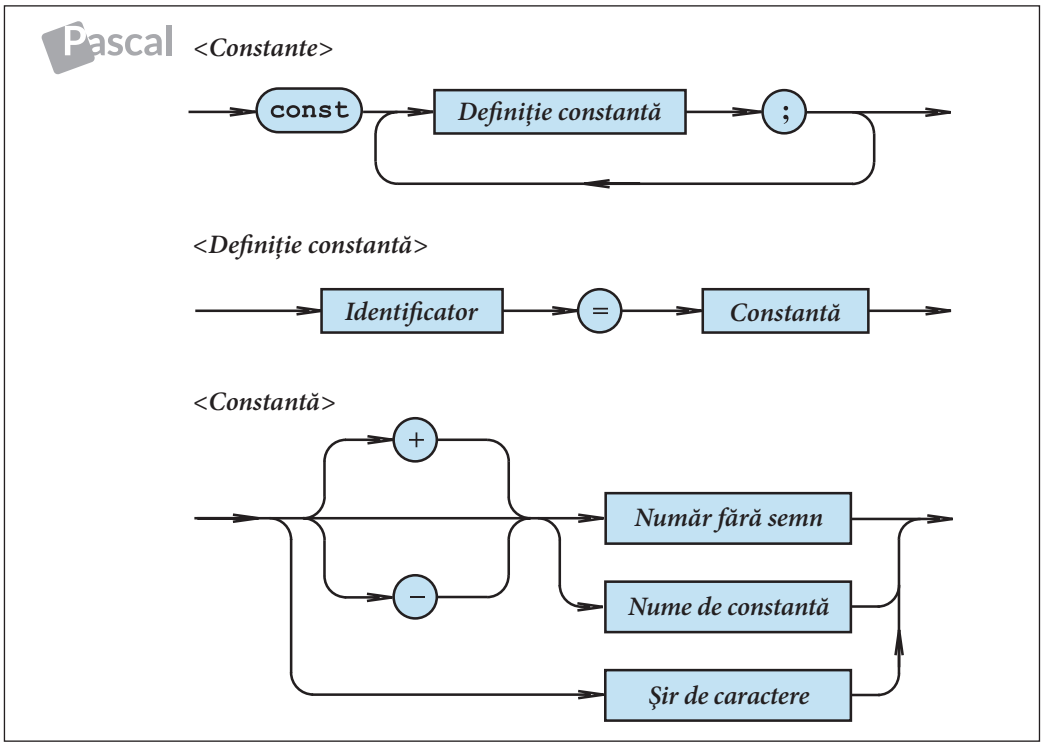


Fig. 2.5. Diagramele sintactice <*Constante*>, <*Definiţie constantă*> şi <*Constantă*>, PASCAL

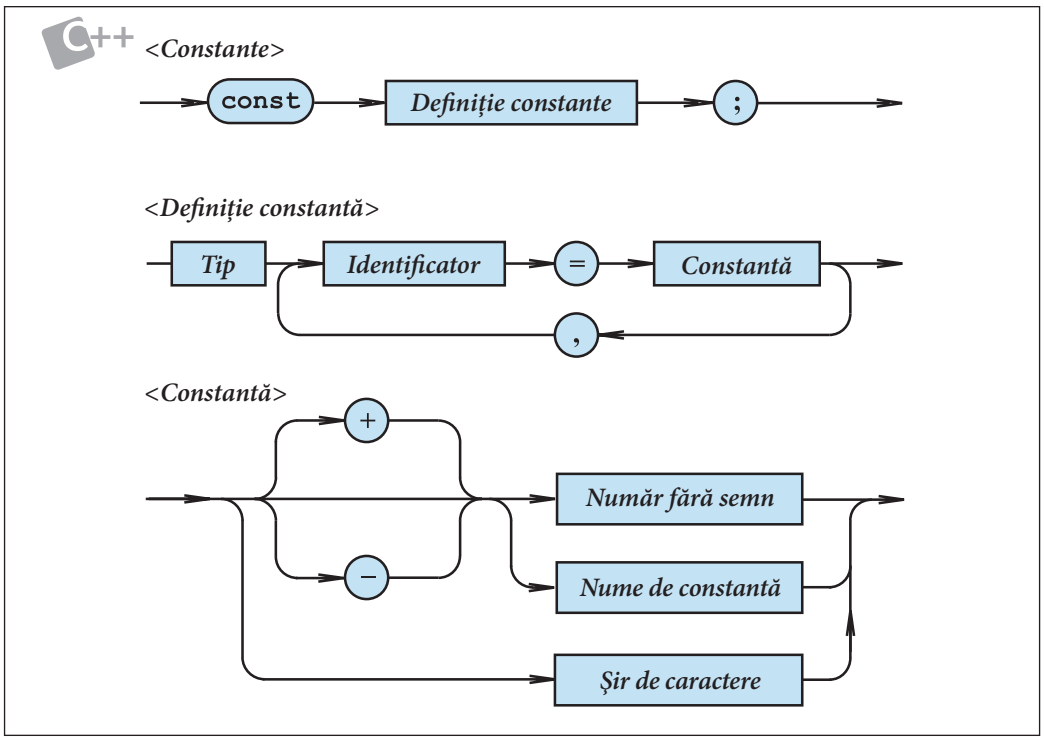


*Fig. 2.5.** Diagramele sintactice <*Constante*>, <*Definiție constante*> și <*Constantă*>, C++

Exemple:

	PASCAL		C++

1)	`const a = 10;`
	`b = 9.81;`
	`c = '*';`
	`t = 'TEXT';`
		1)	`const int a = 10;`
			`float b = 9.81;`
			`char c = '*';`
			`string t = "TEXT";`

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

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

În exemplele de mai sus tipul constantelor este:

`a, NumarCaractere, LungimePagina, n, m` – *întregi*;
`b, Pmax, Pmin` – *reale*;
`c` – *char*;
`t, S` – *șir de caractere*.

În programele de mai jos sunt definite constantele `Nmax`, `Nmin`, `Pi`, `Separator`, `Indicator` și `Mesaj`. Valorile acestor constante se afișează pe ecran.

PASCAL

```
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.
```

C++

```
// Propgramul P32
#include <iostream>
using namespace std;
int main()
{
const int Nmax = 40;
const int Nmin = -Nmax;
const double Pi = 3.14;
const char Separator = '/';
const bool Indicator = true;
const string Mesaj = "
     Verificati imprimanta";
cout<<Nmax<<endl;
cout<<Nmin<<endl;
cout<<Pi<<endl;
cout<<Separator<<endl;
cout<<Indicator<<endl;
cout<<Mesaj<<endl;
// calcule in care se
// utilizeaza constantele
// in studiu
return 0;
}
```

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 programele ce urmează, 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`.

PASCAL

```
Program P33;
 { Lungimea si aria cercului }
const Pi = 3.14;
var L, S , r : real;
```

C++

```
// Programul P33
// Lungimea si aria cercului
#include <iostream>
using namespace std;
```

```
begin
  writeln('Introduceti raza r=');
  readln(r);
  L:=2*Pi*r;
  writeln('Lungimea L=', L);
  S:=Pi*r*r;
  writeln('Aria S=', S);
end.
```

```
int main()
{
const double Pi = 3.14;
float L, S , r;
cout<<"Introduceti raza r=";
cin>>r;
L=2*Pi*r;
cout<<"Lungimea L="<<L<<endl;
S=Pi*r*r;
cout<<"Aria S="<<S;
return 0;
}
```

Dacă utilizatorul are nevoie de rezultate mai exacte, se modifică numai linia programului în care se indică valoarea constantei:

PASCAL

```
const Pi = 3.141592654;
```

C++

```
const double Pi = 3.141592654;
```

Evident, restul programului rămâne neschimbat.

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

Întrebări şi exerciţii

❶ Determinați tipul următoarelor constante:

	PASCAL		C++

1)	`const a = 29.1;`
	`b = TRUE;`
	`c = 18;`
		1)	`const double a = 29.1;`
			`const bool b = TRUE;`
			`const int c = 18;`

2)	`const d = -16.82e-14;`
	`f = -d;`
	`i = 15;`
	`j = '15';`
	`n = '-d';`
		2)	`const double d = -16.82e-14;`
			`const double f = -d;`
			`const int i = 15;`
			`const string j = "15";`
			`const string n = "-d";`

3)	`const t = 'F';`
	`q = '1';`
	`c = '18';`
		3)	`const char t = 'F';`
			`const char q = '1';`
			`const string c = "18";`

4)	`const x = 65;`
	`q = FALSE;`
	`y = -x;`
	`m = 'PAUZA';`
		4)	`const int x = 65;`
			`const bool q = FALSE;`
			`const int y = -x;`
			`const string m = "PAUZA";`

❷ **APLICĂ!** 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 **OBSERVĂ!** Programele de mai jos conțin erori. Identificați erorile, explicați cauza apariției acestora, modificați programul ca să lucreze fără erori.

PASCAL

```
Program P34;
 { Erori }
const Nmax = 40;
   Nmin = -Nmax;
   Pi = 3.14;
   Separator = '\';
   Indicator = TRUE;
begin
  readln(Nmax);
  writeln(Nmax);
  writeln(Nmin);
  Nmax:=10;
  writeln(Nmax);
  writeln(Pi);
  writeln(Separator);
  writeln(Indicator);
end.
```

C++

```
// Programul P34
#include <iostream>
using namespace std;
// Erori
int main()
{
const int Nmax = 40;
const int Nmin = -Nmax;
const double Pi = 3.14;
const char Separator = '/';
const bool Indicator = true;
cin>>Nmax;
cout<<Nmax<<endl;
cout<<Nmin<<endl;
Nmax=10;
cout<<Nmax<<endl;
cout<<Pi<<endl;
cout<<Separator<<endl;
cout<<Indicator<<endl;
return 0;
}
```

Explicați mesajele afișate pe ecran pe parcursul compilării programelor modificate.

❺ Se consideră programele:

PASCAL

```
Program P35;
const t = '1';
      s = -t;
begin
  writeln(s);
end.
```

C++

```
// Programul P35
#include <iostream>
using namespace std;
int main()
{
const char t = '1';
const char s = -t;
cout<<s;
return 0;
}
```

Ce mesaje vor fi afișate pe ecran în procesul compilării?

❻ **ANALIZEAZĂ!** Ce rezultate se vor afişa după execuţia programelor ce urmează?

PASCAL

```
Program P36;
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.
```

C++

```
// Programul P36
#include <iostream>
using namespace std;
int main()
{
const char i = 1;
const int j = i;
const int k = j;
int x, y, z;
x=i+1; cout<<x<<endl;
y=j+2; cout<<y<<endl;
z=k+3; cout<<z;
return 0;
}
```

❼ Se consideră programele:

PASCAL

```
Program P37;
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.
```

C++

```
// Programul P37
#include <iostream>
using namespace std;
int main()
{
const char a = 1;
const int b = 2;
const int c = 3;
const int d = 4;
int i, j, k;
i=a+1; cout<<i<<endl;
j=b+1; cout<<j<<endl;
k=c+1; cout<<k<<endl;
d=a+1; cout<<d;
return 0;
}
```

Explicaţi mesajele afişate pe ecran.

❽ Excludeţi din programele ce urmează linia care conţine o eroare.

PASCAL

```
Program P38;
const a = 1;
var i, j : integer;
begin
```

C++

```
// Programul P38
#include <iostream>
using namespace std;
```

```
  writeln('Introduceti i=');
  readln(i); j:=i+a;
  writeln(j);
  writeln('Introduceti a=');
  readln(a);
  j:=i+a;
  writeln(j);
end.
```

```
int main()
{
const int a = 1;
int i, j;
cout<<"Introduceti i=";
cin>>i; j=i+a;
cout<<j;
cout<<"Introduceti a=";
cin>>a;
j=i+a;
cout<<j;
return 0;
}
```

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

Test de autoevaluare nr. 2

1. Explicaţi semnificaţia termenului *tip de date*. Numiţi cel puţin două tipuri de date și daţi câteva exemple de astfel de date.

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

3. Indicaţi tipul datelor din programul de mai jos.

PASCAL

```
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.
```

C++

```
// Programul TA2
#include <iostream>
using namespace std;
// Tipuri de date simple
int main()
{
int i, j;
double a, b, c;
char s;
bool p;
i=5; j=i+9;
cout<<i<<endl;
cout<<j<<endl;
a=1.0; b=1.0e-01; c=-2.001;
cout<<a<<endl;
cout<<b<<endl;
cout<<c<<endl;
s='A'; cout<<s<<endl;
p=true; cout<<p;
return 0;
}
```

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

4. Cum se definește tipul de date întregi în limbajul studiat și care este mulțimea lui de valori? Ce operații se pot face cu aceste valori?

5. Se consideră programele:

PASCAL	C++

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.
	// Programul TA3
	#include <iostream>
	using namespace std;
	int main()
	{
	int x, y, z;
	cout<<"Introduceti numerele intregi x, y:";
	cin>>x>>y;
	z=x*y; cout<<"x*y="<<z;
	return 0;
	}

Dați exemple de valori ale variabilelor x și y 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? Sunt oare exacte aceste operații?

7. Se consideră programele:

PASCAL	C++

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.
	// Programul TA4
	#include <iostream>
	using namespace std;
	int main()
	{
	double x, y, z;
	cout<<"Introduceti numerele reale x, y:";
	cin>>x>>y;
	z=x*y; cout<<"x*y="<<z;
	return 0;
	}

Dați exemple de valori ale variabilelor x și y pentru care apar erori de depășire.

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

9. Alcătuiţi tabelele de adevăr ale operațiilor logice:

PASCAL: **not**, **and** și **or**;
C++: !, && și ||.

10. Analizați programele ce urmează. Ce mesaje vor fi afișate în urma rulării acestora pe calculator?

PASCAL

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

C++

```
// Programul TA5
#include <iostream>
using namespace std;
int main()
{
bool p, q, r;
cout<<"Introduceti valorile
       logice p, q:";
cin>>p>>q;
r=p && q;
cout<<q;
return 0;
}
```

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

PASCAL

1)
```
type FunctiaOcupata = (Muncitor=10, SefDeEchipa, Maistru,
SefDeSantier, Director);
```

2)
```
type StareaCivila = (Casatorit, Necasatorit);
```

C++

1)
```
enum FunctiaOcupata {Muncitor=10, SefDeEchipa, Maistru,
SefDeSantier, Director};
```

2)
```
enum StareaCivila {Casatorit, Necasatorit};
```

15. (PASCAL) 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:

PASCAL	C++

`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;`
	`typedef char T1;`
	`typedef int T2;`
	`typedef char T3;`
	`enum T4 {Alfa, Beta, Gama, Delta};`
	`T1 p;`
	`T2 q;`
	`T3 r;`
	`char s;`
	`int t;`
	`T4 u;`

(PASCAL) Numiți tipul de bază al fiecărui tip de date *subdomeniu*.

17. Numiţi tipurile ordinale de date. Care sunt proprietăţile lor comune?

18. Ce vor afișa pe ecran programele ce urmează?

PASCAL	C++

`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.`
	`// Programul TA6`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`enum Culoare {Galben, Verde, Albastru, Violet};`
	`cout<<char('Z'-1)<<endl;`
	`cout<<char('D'+1)<<endl;`
	`cout<<-5-1<<endl;`
	`cout<<9+1<<endl;`
	`cout<<Verde<<endl;`
	`cout<<Violet;`
	`return 0;`
	`}`

19. Ce vor afișa pe ecran programele ce urmează?

PASCAL	C++

`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));`
	`// Programul TA7`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`enum Nivel {A, B, C, D, E, F, G};`

```
  n:=C; m:=E;
  writeln(n<m);
  writeln(n>m);
  writeln(n<>m);
end.
```

```
Nivel n, m;
n=B; cout<<n<<endl;
cout<<n-1<<endl;
cout<<n+1<<endl;
n=C; m=E;
cout<<(n<m)<<endl;
cout<<(n>m)<<endl;
cout<<(n!=m);
return 0;
}
```

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

21. Se consideră declaraţiile:

PASCAL	C++

`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);`
	`typedef int T1;`
	`typedef T1 T2;`
	`typedef int T3;`
	`typedef T3 T4;`
	`typedef int T5;`
	`enum T6 {A, B, C, D, E, F, G, H};`
	`typedef T6 T7;`
	`typedef T6 T8;`
	`typedef char T9;`
	`typedef char T10;`
	`int x;`
	`enum {Alfa, Beta, Gama, Delta} y;`

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

22. Determinaţi tipul constantelor:

PASCAL	C++

`const Alfa = 5;`
`Beta = 12.485;`
`Indicator = true;`
`Mesaj = 'Eroare de execu-tie';`
`Semn = '+';`
`Inscris = '12.485';`
	`const int Alfa = 5;`
	`const float Beta = 12.485;`
	`const bool Indicator = true;`
	`const string Mesaj = "Eroare de executie";`
	`const char Semn = '+';`
	`const string Inscris = "12.485";`

23. Într-un algoritm se utilizează variabilele întregi i, j, variabilele reale x, y, constantele 3,14 și 9,8. Scrieți în limbajul de programare cercetat declarațiile pentru variabile și constantele în studiu.

Capitolul 3

INSTRUCŢIUNI

3.1. Conceptul de acţiune

Conform **conceptului de acțiune** realizat în limbajul studiat, 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. Evident, în urma acestor acțiuni valorile variabilelor pot fi schimbate, pe când cele ale constantelor – nu.

Acțiunile 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 sunt:

– instrucțiunea de atribuire;

– instrucțiunea apel de procedură (doar în limbajul PASCAL);

– instrucțiunea de salt necondiționat;

– instrucțiunea de efect nul (sau vidă).

Instrucțiunile structurate sunt construite din alte instrucțiuni. Instrucțiunile structurate sunt:

– instrucțiunea compusă;

– instrucțiunile condiționale (în PASCAL: **`if`**, **`case`**; în C++: **`if`**, **`switch`**);

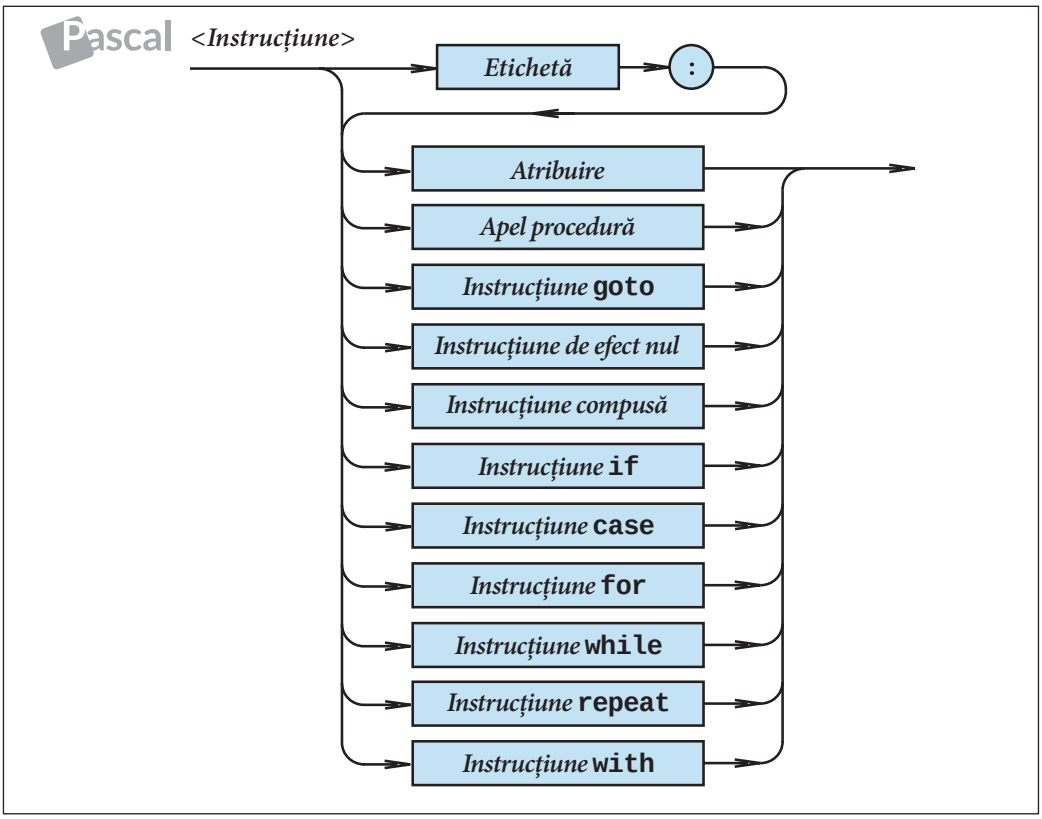
– instrucțiunile iterative (în PASCAL: **`for`**, **`while`**, **`repeat`**; în C++: **`for`**, **`while`**, **`do while`**);

– instrucțiunea **`with`** (doar în PASCAL).

Instrucțiunile condiționale se utilizează pentru programarea algoritmilor cu ramificări, iar instrucțiunile iterative – pentru programarea algoritmilor repetitivi. Amintim că algoritmii repetitivi sunt folosiți 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`**.

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



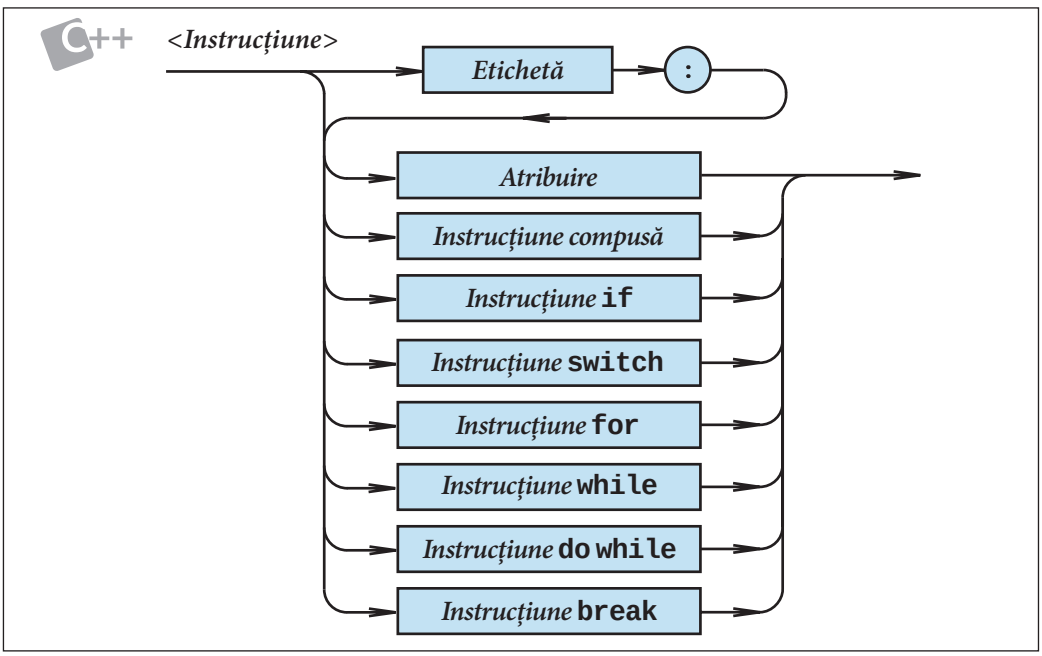


Fig. 3.1. Diagrama sintactică <*Instrucţiune*>

Într-un program PASCAL / C++ 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).

3.2. Expresii

În limbajele PASCAL și C++ formulele pentru calculul unor valori se reprezintă prin **expresii**. Acestea sunt formate din operanzi (constante, variabile, apeluri de funcții) și operatori (simbolurile operațiilor). Operatorii se clasifică după cum urmează:

PASCAL

<*Operator multiplicativ*> ::= `*` | `/` | **`div`** | **`mod`** | **`and`**
<*Operator aditiv*> ::= `+` | `-` | **`or`**
<*Operator relațional*> ::= `<` | `<=` | `=` | `>=` | `>` | `<>` | **`in`**

C++

<*Operator multiplicativ*> ::= `*` | `/` | `%` | `&&`
<*Operator aditiv*> ::= `+` | `-` | `||`
<*Operator relațional*> ::= `<` | `<=` | `==` | `>=` | `>` | `!=`

În componența expresiilor intră factori, termeni și expresii simple.
Factorul poate fi o variabilă, o constantă fără semn, apelul unei funcții etc. Mai exact:

<*Factor*> ::= <*Variabilă*> | <*Constantă fără semn*> | <*Apel funcție*> | <*Negație*> <*Factor*> | (<*Expresie*>) | <*Constructor mulțime*>

Exemple:

	PASCAL		C++
1)	`15`	*1)*	`15`
2)	`x`	*2)*	`x`
3)	`sin(x)`	*3)*	`sin(x)`
4)	**`not`** `p`	*4)*	`!p`

Un **termen** are forma:

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

Exemple:

	PASCAL		C++
1)	`15`	*1)*	`15`
2)	`x`	*2)*	`x`
3)	`x*y*z`	*3)*	`x*y*z`
4)	`p` **`and`** `q`	*4)*	`p && q`

Prin **expresie simplă** se înțelege:

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

Exemple:

	PASCAL		C++
1)	`+15`	*1)*	`+15`
2)	`p or q`	*2)*	`p || q`
3)	`sin(x)/3+cos(x)`	*3)*	`sin(x)/3+cos(x)`

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

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

Exemple:

	PASCAL		C++
1)	`-15`	*1)*	`-15`
2)	`15*x+sin(x)/3<>11`	*2)*	`15*x+sin(x)/3 != 11`
3)	`x+6>z-3.0`	*3)*	`x+6>z-3.0`

Fie variabilele `a` și `b` de tip întreg, `c` și `d` de tip logic (boolean), iar `x` și `y` de tip real. Valorile acestor variabile sunt `a=2`, `b=8`, `c=true`, `d=false`, `x=-2.5` și `y=1.4`.

Operatori în PASCAL		
Operator	**Descriere**	**Exemple**
Operatori unari		
+ (semn)	Indică un număr pozitiv.	`+2`, `+0.25`, `+1999`
– (semn)	Indică un număr negativ.	`-2`, `-0.25`, `-1999`
`not`	Negație (operator logic)	**`not`**`(c)` → rezultat `false`
Operatori binari		
+	Adunarea	`a+b` → rezultat 10 `a+x` → rezultat –0,5
–	Scăderea	`a-b` → rezultat –6 `a-x` → rezultat 4,5
*	Înmulțirea	`a*b` → rezultat 16 `a*y` → rezultat 2,8
/	Împărțirea	`a/b` → rezultat 0,25 `b/a` → rezultat 4,0
`div`	Câtul împărțirii a două numere întregi	`a` **`div`** `b` → rezultat 0 `b` **`div`** `a` → rezultat 4
`mod`	Restul împărțirii a două numere întregi	`a` **`mod`** `b` → rezultat 8 `b` **`mod`** `a` → rezultat 0

<	Verifică dacă primul operator este mai mic decât al doilea.	a<b → rezultat true b<a → rezultat false
<=	Verifică dacă primul operator este mai mic decât sau egal cu al doilea.	a<=b → rezultat true b<=a → rezultat false
>	Verifică dacă primul operator este mai mare decât al doilea.	a>b → rezultat false b>a → rezultat true
>=	Verifică dacă primul operator este mai mare decât sau egal cu al doilea.	a>=b → rezultat false b>=a → rezultat true
=	Verifică dacă primul operator este egal cu al doilea.	a=b → rezultat false a=x → rezultat false
<>	Verifică dacă primul operator este diferit de al doilea.	a<>b → rezultat true b<>y → rezultat true
and	Produs logic. Dacă ambii operanzi sunt true, atunci rezultatul este true.	c and d → rezultat false c and c → rezultat true d and d → rezultat false
or	Suma logică. Dacă cel puțin un operand este true, atunci rezultatul este true.	c or d → rezultat true c or c → rezultat true d or d → rezultat false

Operatori în C++		
Operator	**Descriere**	**Exemple**
Operatori unari		
+ (semn)	Indică un număr pozitiv.	+2, +0, 25, +1999
– (semn)	Indică un număr negativ.	–2, –0, 25, –1999
!	Negație logică	!(c) → rezultat false
++	Incrementare. Există două moduri de incrementare a unei variabile x: • Postincrementare: x++. Incrementarea se face după evaluarea expresiei în care apare ++. • Preincrementare: ++x. Incrementarea se face înainte de evaluarea expresiei în care apare ++.	++a+b → rezultat 11. Valoarea lui a este mărită înainte de evaluarea expresiei. (a++)+b → rezultat 10. Valoarea lui a este mărită după evaluarea expresiei.
––	Decrementare. Operația de decrementare a variabilei x poate fi: • Postdecrementare: x--. Decrementarea se face după evaluarea expresiei în care apare --. • Predecrementare: --x. Decrementarea se face înainte de evaluarea expresiei în care apare --.	––a+b → rezultat 9. Valoarea lui a este micșorată înainte de evaluarea expresiei. (a––)+b → rezultat 10. Valoarea lui a este micșorată după evaluarea expresiei.

Operatori binari		
+	Adunarea	a+b → rezultat 10 a+x → rezultat –0,5
–	Scăderea	a–b → rezultat –6 a–x → rezultat 4,5
*	Înmulțirea	a*b → rezultat 16 a*y → rezultat 2,8
/	Împărțirea. Dacă cel puțin un operand este real, atunci rezultatul returnat va fi rezultatul împărțirii "cu virgulă".	a/x → rezultat –0,8 y/x → rezultat –0,56
	Câtul împărțirii. Dacă ambii operanzi sunt întregi, atunci rezultatul returnat va fi câtul întreg de la împărțirea lor.	a/b → rezultat 0 b/a → rezultat 4
%	Modulo. Restul împărțirii a două numere întregi	a % b → rezultat 8 b % a → rezultat 0
<	Verifică dacă primul operator este mai mic decât al doilea.	a<b → rezultat true b<a → rezultat false
<=	Verifică dacă primul operator este mai mic decât sau egal cu al doilea.	a<=b → rezultat true b<=a → rezultat false
>	Verifică dacă primul operator este mai mare decât al doilea.	a>b → rezultat false b>a → rezultat true
>=	Verifică dacă primul operator este mai mare decât sau egal cu al doilea.	a>=b → rezultat false b>=a → rezultat true
==	Verifică dacă primul operator este egal cu al doilea.	a==b → rezultat false a==x → rezultat false
!=	Verifică dacă primul operator este diferit de al doilea.	a!=b → rezultat true b!=y → rezultat true
&&	Produs logic. Dacă ambii operanzi sunt true, atunci rezultatul este true.	c && d → rezultat false c && c → rezultat true d && d → rezultat false
||	Suma logică. Dacă cel puțin un operand este true, atunci rezultatul este true.	c || d → rezultat true c || c → rezultat true d || d → rezultat false
Operatori ternari		
?	Operatorul ? are forma: <exp1> ? <exp2>:<exp3>, unde <exp1> este o expresie de tip logic. Când <exp1> este adevărată, rezultatul operației ? este <exp2>, iar când este falsă, rezultatul operației ? este <exp3>. <exp1> și <exp1> trebuie să aibă rezultate de același tip, sau de tipuri compatibile.	a % 2 == 0? "par" : "impar" → rezultat "par" x < 0 ? x*2 : x-2 → rezultat x *2=–5 a < b ? b : a → rezultat 8 a < b ? a : b → rezultat 2

Tipul rezultatelor evaluării expresiilor se determină în funcție de tipul operanzilor și operatorii respectivi. Aceste reguli vor fi studiate în paragrafele următoare.

Diagramele sintactice ale operatorilor și expresiilor sunt prezentate în *figurile 3.2* și *3.3*.

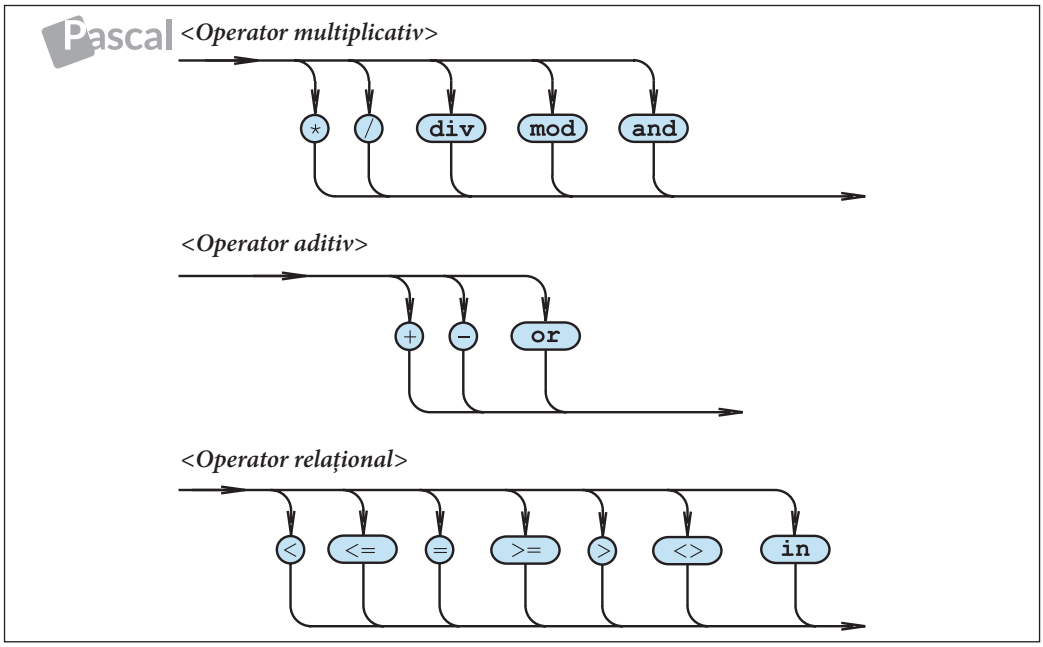


Fig. 3.2. Diagramele sintactice pentru operatori, PASCAL

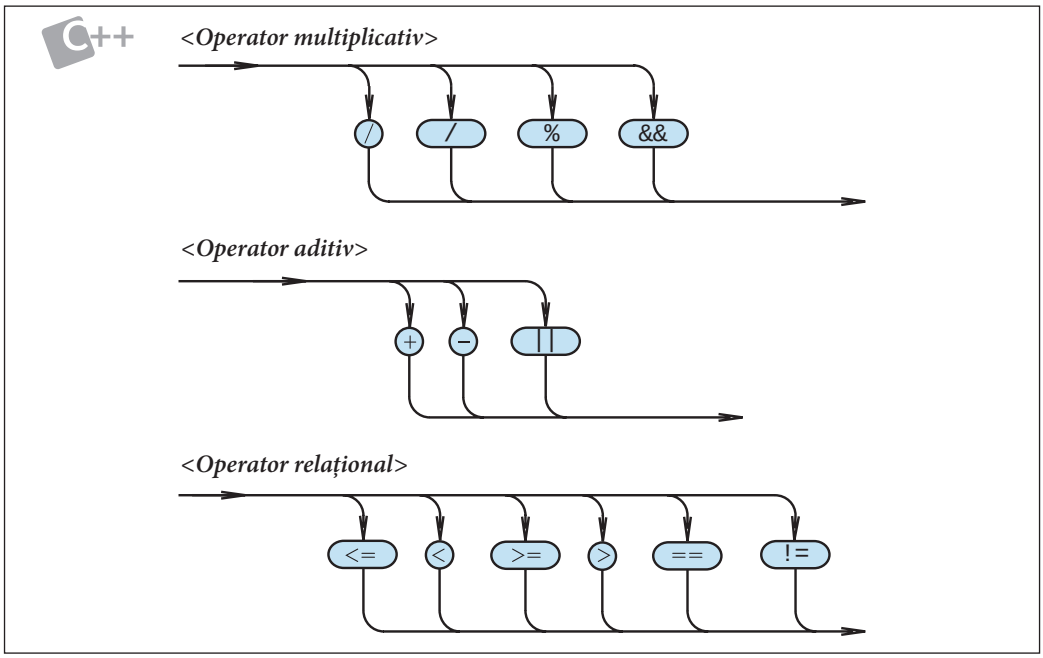


*Fig. 3.2**. Diagramele sintactice pentru operatori, C++

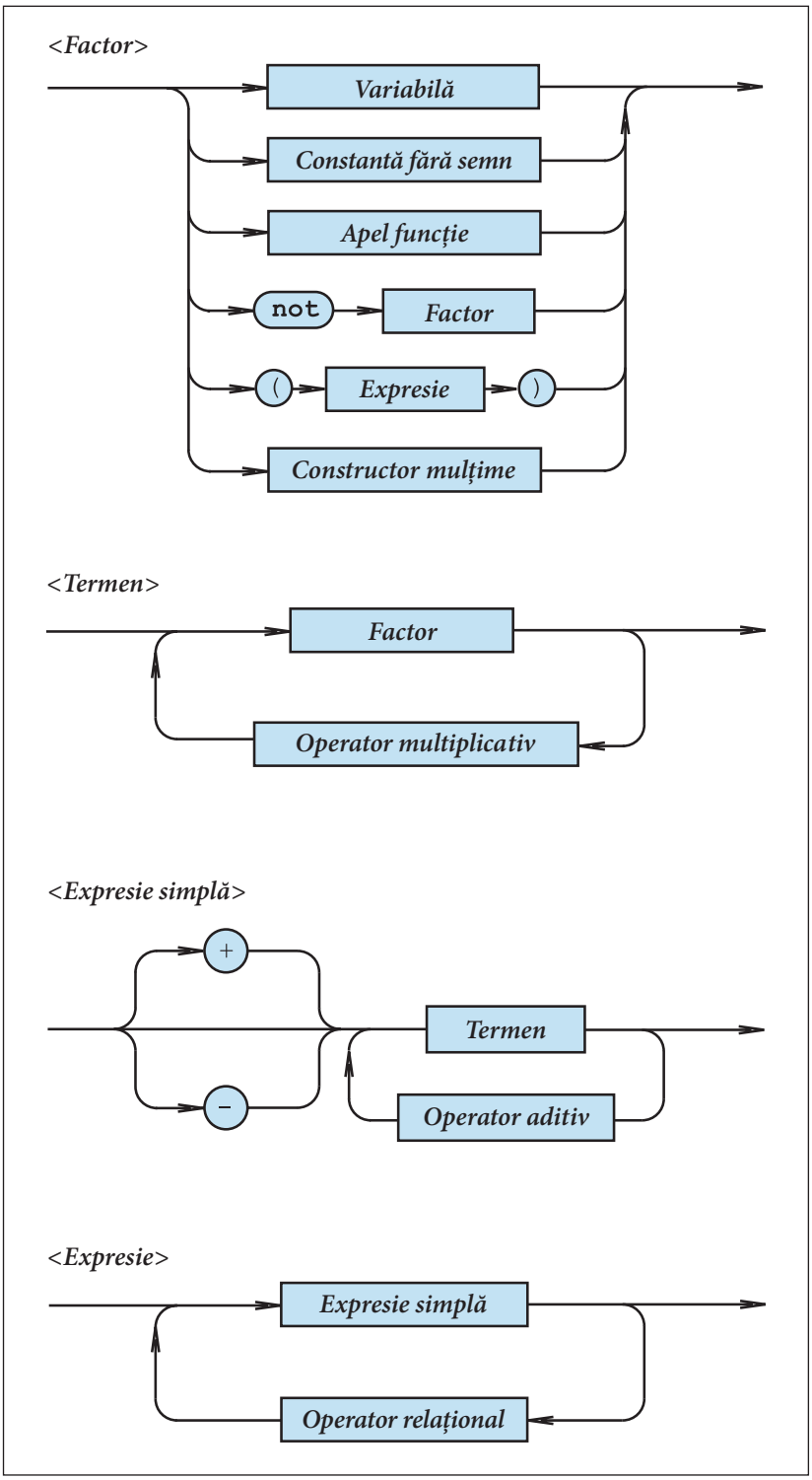


Fig. 3.3. Diagramele sintactice pentru definirea expresiilor

Din diagramele sintactice pentru definirea expresiilor se observă că ele sunt definite recursiv:

- o expresie este formată dintr-un operand sau dintr-un operator aplicat operanzilor săi;
- un operand este format dintr-un literal, un identificator de variabilă sau dintr-o expresie scrisă între paranteze rotunde.

De exemplu, expresia `a*(b+1)` este formată din operatorul de multiplicare `*` aplicat operanzilor `a` și `(b+1)`, primul operand este un identificator, al doilea este expresia `b+1`. La rândul ei, această expresie este formată din operatorul de adunare + aplicat unui identificator și unui literal.

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 în matematică	Notația în PASCAL	Notația în C++
1)	$\frac{a+b}{c+d}$	`(a+b)/(c+d)`	`(a+b)/(c+d)`
2)	$\frac{-b+\sin(b-c)}{2a}$	`(-b+sin(b-c))/(2*a)`	`(-b+sin(b-c))/(2*a)`
3)	$-\frac{1}{xy}$	`-1/(x*y)`	`-1/(x*y)`
4)	$(p<q)\ \&\ (r>s)$	`(p<q)and(r>s)`	`(p<q)&&(r>s)`
5)	$\overline{x \vee y}$	`not (x or y)`	`!(x || y)`
6)	$\frac{1}{a+b}>\frac{1}{c+d}$	`1/(a+b)>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*, p. 99). Limbajele de programare conțin un set de **funcții predefinite**, cunoscute oricărui program. Aceste funcții sunt prezentate în *tabelul 3.1*.

Tabelul 3.1

Funcțiile predefinite ale limbajelor PASCAL și C++

Denumirea funcției	Notația în PASCAL	Notația în C++
valoarea absolută $\|x\|$	`abs(x)`	`abs(x)` sau `fabs(x)`
sinus sin x	`sin(x)`	`sin(x)`
cosinus cos x	`cos(x)`	`cos(x)`
arctangenta arctg x	`arctan(x)`	`atan(x)`
pătratul lui x x^2	`sqr(x)`	`pow(x,2)`
puterea lui x x^n	–	`pow(x,n)`
puterea lui 2 2^x	–	`exp2(x)`

rădăcina pătrată $\sqrt{x}$	`sqrt(x)`	`sqrt(x)`
puterea numărului e e^x	`exp(x)`	`exp(x)`
logaritmul natural ln x	`ln(x)`	`log(x)`
logaritm în baza 10 din x $log_{10}x$	–	`log10(x)`
rotunjirea lui x	`round(x)`	`round(x)`
trunchierea lui x	`trunc(x)`	`trunc(x)`
rotunjește pe x la cel mai apropiat întreg mai mare decât x	–	`ceil(x)`
trunchiază pe x la cel mai apropiat număr întreg mai mic decât el	–	`floor(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 sfârșitului de fișier	`eof(f)`	`EOF`
testarea sfârșitului de linie	`eoln(f)`	–

NOTĂ

Pentru ca funcțiile matematice să funcționeze, în programul C++ trebuie inclusă biblioteca `cmath`.

Întrebări şi exerciţii

❶ **APLICĂ!** Scrieți conform regulilor limbajului de programare studiat următoarele expresii:

a) a^2+b^2;

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

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

d) $v_0t + \frac{at^2}{2}$;

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) $x_1x_2 \vee x_3x_4$;

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

l) $|x| < 3$;

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

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

❷ **OBSERVĂ!** Indicați pe diagramele sintactice din *figura 3.3* drumurile care corespund expresiilor:

a)
```
x
```

b)
```
3.14
```

c)
```
sin(x)
```

d)
```
PASCAL:   not q
C++:      !q
```

e)
```
(+3.14)
```

f)
```
x>2.85
```

g)
```
PASCAL:   sqr(b)-4*a*c>0
C++:      pow(b,2)-4*a*c>0
```

h)
```
PASCAL:   (a>b)and(c>d)
C++:      (a>b)&&(c>d)
```

❸ **APLICĂ!** Transpuneți expresiile în notații obișnuite:

a)
```
PASCAL:  sqr(a)+sqr(b)
C++:  pow(a,2)+pow(b,2)
```

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

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

d)
```
exp(x+y)
```

e)
```
cos(ALFA-BETA)
```

f)
```
PASCAL:   sqr(a+b)/(a-b)
C++:      pow(a+b,2)/(a-b)
```

g)
```
PASCAL:   (x>0) or (q<p)
C++:      (x>0) || (q<p)
```

h)
```
PASCAL: not(x and y)
C++:  !(x && y)
```

❹ **OBSERVĂ!** Care dintre expresiile ce urmează sunt greșite pentru limbajul studiat? 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)
```
a!=b || c>d
```

i)
```
!!! p
```

j)
```
Q && ! p
```

k)
```
not q and p
```

l)
```
a+-b
```

m)
```
sin(-x)
```

n)
```
sin-x
```

o)
```
cos(x+y)
```

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

q)
```
sqrt(-y)
```

r)
```
!q && p
```

s)
```
x>0 or q<p
```

t)
```
EXP(x)
```

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 sunt cele obișnuite în matematică:

– operațiile se efectuează conform priorității operatorilor;
– în cazul priorităților egale, operațiile se efectuează de la stânga spre dreapta;
– mai întâi se calculează expresiile dintre paranteze.

Prioritățile operatorilor sunt indicate în *tabelul 3.2*.

Tabelul 3.2

Prioritățile operatorilor limbajelor PASCAL și C++

Categorie	Operatori		Prioritate
	PASCAL	C++	
Operatori unari	+, -(semn), **not**, @	+, - (semn), !, &, - -, ++	prima (cea mai mare)
Operatori multiplicativi	*, /, **div**, **mod**, **and**	*, /, %	a doua
Operatori aditivi	+, -, **or**	+, -	a treia
Operatori relaționali	<, <=, =, >=, >, <>, **in**	*a*) <, <=, >=, > *b*) ==, !=	a patra și a cincea
Operatori logici	-	*a*) && *b*) ||	a șasea și a șaptea (cea mai mică)

Exemplu:

Fie x și y numere întregi, iar m și n numere reale, având valorile $x = 2$, $y = 6$, $m = 5$ și $n = 1{,}5$. 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) **PASCAL:** `x/y = 2/6 = 0.3333`.

*5)** **C++:** `x/y = 2/6 = 0`. În cazul operanzilor de tip întreg, se realizează împărțirea întreagă, iar rezultatul operației / este câtul împărțirii întregi.

6) **PASCAL:** `m/n = 5/1.5 = 3.3333`.

*6)** **C++:** `m/n = 5/1,5 = 3.3333`. În cazul operanzilor de tip real (`float`, `double`, `long double`), se realizează împărțirea zecimală, iar rezultatul operației / este rezultatul acestei împărțiri, "cu virgulă".

7) **PASCAL:** `x+y/(x–y) = 2+6/(2–6) = 2+6/(–4) = 2+(–1.5) = 0.5;`

*7)** **C++:** `x+y/(x–y) = 2+6/(2–6) = 2+6/(–4) = 2+(–1) = 1;`

8) `n+x/(m–y) = 1,5+2/(5–6) = 1.5+2/(–1) = 1.5-2 = -0.5;`

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

10) **PASCAL:** `(x+y<15)`**`and`**`(x>3) = (2+6<15)`**`and`**`(2>3) = (8<15` **`and`** `(2>3) = = true` **`and`** `false = false`.

*10)** **C++:** `(x+y<15)&&(x>3) =(2+6<15)&&(2>3) = (8<15)&&(2>3) = true && false = false`.

Se observă că părțile componente ale unei expresii (*fig. 3.3*, p. 99) 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 astfel:

PASCAL	C++
`writeln(`*<Expresie>*`);`	`cout << `*<Expresie>*` << endl;`

Programul de mai jos afișează pe ecran rezultatele evaluării expresiilor `x*y+z` și `x+y<z-1.0`. Valorile curente ale variabilelor `x`, `y` și `z` sunt citite de la tastatură.

PASCAL

```
Program P39;
 { 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.
```

C++

```
// Programul P39
#include <iostream>
using namespace std;
// Evaluarea expresiilor
int main()
{
double x, y, z;
cout<<"Introduceti numerele
reale x, y, z:\n";
cin>>x>>y>>z;
cout<<x*y+z<<endl;
cout<<(x+y<z-1.0);
return 0;
}
```

Întrebări și exerciții

❶ **APLICĂ!** 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)
```
PASCAL:  (x>y) or (6*x>y+z)
C++:     (x>y) || (6*x>y+z)
```

i)
```
PASCAL: not(x+y+z>0)
C++:  !(x+y+z>0)
```

j)
```
PASCAL: not(x+y>0) and not(z<0)
C++:   !(x+y>0) && !(z<0)
```

❷ Care sunt regulile de evaluare a unei expresii în limbajul de programare studiat?

❸ Indicați prioritatea fiecărui operator al limbajului studiat.

❹ Precizați ordinea în care se calculează componentele unei expresii în limbajul studiat.

❺ **ELABOREAZĂ!** Creați 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.

❻ Ce valoare vor returna expresiile:

PASCAL

```
a) sqr(2)+2*sgr(2)
b) (1+sqrt(4))/3
c) trunc(sqrt(20))
d) trunc(27%4) + round(27.3) / 4
e) trunc( -47%5 -
   sqrt(2))/round(19.3)/3
```

C++

```
a) pow(2,2)+pow(2,3)
b) 1+sqrt(4))/3
c) floor(sqrt(20))
d) floor(27%4) + ceil(27.3) / 4
e) floor( -47%5 -
   sqrt(2))/ceil(19.3)/3
```

3.4. Tipul expresiilor

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 limbajele PASCAL și C++, **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 (p. 106) conține tipul rezultatelor furnizate de funcțiile predefinite ale limbajului PASCAL/C++.

Tabelul 3.3

Tipul rezultatelor furnizate de operatori

PASCAL		
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`

C++		
Operator	**Tipul operanzilor**	**Tipul rezultatului**
+, –, *	**int**	**int**
	unul **int**, altul **double**	**double**
++, --	tipuri identice sau compatibile	identic cu tipul operanzilor
/	**int**	**int**
	double	**double**
%	**int**	**int**
!, &&, ||	**bool**	**bool**
<, <=, ==, >=, >, !=	tipuri identice	**bool**
	tipuri compatibile	**bool**
	unul **int**, altul **double**	**bool**

Tabelul 3.4

Tipul rezultatelor furnizate de funcțiile predefinite

PASCAL		
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`

C++		
Funcția	**Tipul argumentului**	**Tipul rezultatului**
`abs(x)` sau `fabs(x)`	**`int`** sau **`double`**	**`double`**
`sin(x)`	**`int`** sau **`double`**	**`double`**
`cos(x)`	**`int`** sau **`double`**	**`double`**
`atan(x)`	**`int`** sau **`double`**	**`double`**
`pow(x,2)`	**`int`** sau **`double`**	**`double`**
`pow(x,n)`	**`int`** sau **`double`** dacă `x<0`, atunci `n` trebuie să fie întreg	**`double`**
`sqrt(x)`	**`int`** sau **`double`**	**`double`**
`exp(x)`	**`int`** sau **`double`**	**`double`**

`log(x)`	**`double, x>0`**	**`double`**
`round(x)`	**`double`**	**`double`**
`trunc(x)`	**`double`**	**`double`**
`log10(x)`	**`double`**	**`double`**
`ceil(x)`	**`double`**	**`double`**
`floor(x)`	**`double`**	**`double`**
`EOF`	fișier	**`bool`**

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

În limbajul C++, operatorul `/` va furniza un rezultat întreg, dacă ambii operanzi sunt întregi și un rezultat real, dacă cel puțin un operand este de tip real, iar operatorii relaționali – numai rezultate de tip **`bool`**.

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:

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

unde `i` este de tip `integer/int`, iar `x` și `y` sunt de tip `real/double` .

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

1)	`6*i`	rezultat de tip `integer/int`;
2)	`x/y`	rezultat de tip `real/double`;
3)	`sin(x/y)`	rezultat de tip `real/double`;
4)	`6*i < sin(x/y)`	rezultat de tip `boolean/bool`.

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

În funcție de tipul expresiei, distingem:

– expresii aritmetice (`integer` sau `real` în PASCAL; **`int`** sau **`double`** în C++);

– expresii ordinale (`integer`, `boolean`, `char`, *enumerare* în PASCAL; **`int`**, **`bool`**, **`char`**, *enumerare* în C++);

– expresii booleene (`boolean` în PASCAL, **`bool`** în C++).

Întrebări şi exerciţii

❶ Prin ce metodă se află tipul unei expresii?

❷ **OBSERVĂ!** În prezența declarațiilor

PASCAL	C++

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

aflați tipul următoarelor expresii:

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

❸ **OBSERVĂ ŞI APLICĂ!** Tipul unei expresii poate fi aflat din forma textuală a rezultatelor afişate pe ecran de instrucțiunea

în PASCAL	în C++
`writeln(`<*Expresie*>`)`	`cout<<`*Expresie*`<<endl;`

Exemple:

	Rezultatul afişat pe ecran	Tipul expresiei în PASCAL	Tipul expresiei în C++
1)	`100`	`integer`	`int`
2)	`1.0000000000E+02`	`real`	`double`
3)	`A`	`char`	`char`

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

	PASCAL		C++
a)	`1+1.0`	*a)*	`1+1.0`
b)	`1/1+1`	*b)*	`1/1+1`
c)	`9*3 mod 4`	*c)*	`9*3 % 4`
d)	`4*x>9*y`	*d)*	`4*x>9*y`
e)	`chr(65)`	*e)*	`char(65)`

f)	`not(x>y)`	*f)*	`!(x>y)`
g)	`pred(9)>succ(7)`	*g)*	`9 > 7`
h)	`15 div ord(3)`	*h)*	`15 / 3`
i)	`trunc(x)+round(6*y)`	*i)*	`trunc(x)+round(6*y)`
j)	`sqr(3)-sqrt(16)`	*j)*	`pow(3,2)-sqrt(16)`

Se consideră că variabilele x şi y sunt reale.

❹ **OBSERVĂ!** Se consideră declarațiile:

PASCAL	C++

`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;`
	`typedef int T1;`
	`typedef int T2;`
	`typedef char T3;`
	`enum T4 {A, B, C, D, E, F, G, H};`
	`T1 i;`
	`T2 j;`
	`T3 k;`
	`char m;`
	`T4 n;`
	`T4 p;`
	`bool q;`

Aflați tipul următoarelor expresii:

	PASCAL		C++
a)	`i-j`	*a)*	`i-j`
b)	`i div j`	*b)*	`i / j`
c)	`6.3*i`	*c)*	`6.3*i`
d)	`cos(3*i-6*j)`	*d)*	`cos(3*i-6*j)`
e)	`4*i>5*j`	*e)*	`4*i>5*j`
f)	`k<m`	*f)*	`k<m`
g)	`k<>m`	*g)*	`k!=m`
h)	`chr(i)`	*h)*	`char(i)`
i)	`ord(k)`	*i)*	`int(k)`
j)	`ord(m)`	*j)*	`int(m)`
k)	`n>p`	*k)*	`n>p`
l)	`ord(n)`	*l)*	`int(n)`
m)	`succ(n)`	*m)*	`n+1`
n)	`pred(p)`	*n)*	`p-1`
o)	`ord(p)`	*o)*	`int(p)`
p)	`ord(k)>ord(m)`	*p)*	`int(k)> int(m)`
q)	`(i>j) and q`	*q)*	`(i>j) && q`

	PASCAL		C++
r)	`not(i+j>0) or q`	*r)*	`!(i+j>0) || q`

❺ Ce tip va avea rezultatul expresiilor:

	PASCAL		C++
a)	`sqrt(2)+2*sgr(2)`	*a)*	`pow(2.5,2)`
b)	`(1+sqr(4))*3`	*b)*	`(1+pow(4,2))*3`
c)	`trunc(sqrt(20))`	*c)*	`floor(sqrt(20))`
d)	`trunc(27%4) + round(27.8)`	*d)*	`floor(27%4) + ceil(27.8)`
e)	`(trunc(-47%3) - round(19.5))*2`	*e)*	`(floor(-47%5) - ceil(19.5))*2`

❻ **ANALIZEAZĂ!** Se consideră numărul real x. Care dintre expresiile ce urmează au valoarea `true`, dacă se cunoaşte că x ia valori din intervalul (–2, 2)? Indicați ordinea efectuării operațiilor.

	PASCAL		C++
a)	`x*x-4<=0`	*a)*	`x*x-4<=0`
b)	`4-x*x>0`	*b)*	`4-x*x>0`
c)	`(2<x) and (x<-2)`	*c)*	`(2<x)&&(x<-2)`
d)	`(x-2)*(x+2)>0`	*d)*	`(x-2)*(x+2)>0`

❼ Se consideră numerele întregi $x = 8$ şi $y = 6$. Care dintre expresiile ce urmează au valoarea `false`? Indicați ordinea efectuării operațiilor.

	PASCAL		C++
a)	`3*x-4*y= 0`	*a)*	`3*x-4*y==0`
b)	`(x+y)/2 > x mod y+1`	*b)*	`(x+y)/2 > x%y+1`
c)	`not (x/2+2 = y)`	*c)*	`!(x/2+2 == y)`
d)	`x-y+3 <> 0`	*d)*	`x-y+3 != 0`

❽ Se consideră numărul real x. Care dintre expresiile ce urmează au valoarea `true`, dacă se cunoaşte că x ia valori din intervalul (5, 8)?

	PASCAL		C++
a)	`(x<=8) or (x>5)`	*a)*	`(x<=8) || (x>5)`
b)	`(x>8) or (x<=5)`	*b)*	`(x>8) || (x<=5)`
c)	`(x<=8) and (x>5)`	*c)*	`(x<=8) && (x>5)`
d)	`(x<8) and (x>=5)`	*d)*	`(x<8) && (x>=5)`

3.5. Conversii de tip în limbajul C++

La evaluarea expresiilor, în limbajul C++ pot apare conversii de tip implicite sau explicite.

Conversia implicită se realizează automat. La o atribuire *v = Expresie* tipul membrului drept se convertește la tipul membrului stâng, care este și tipul rezultatului.

Cu unele situații de conversii de tip ne-am întâlnit la studierea paragrafului 2.9.

Exemple de conversii implicite:

```
int i;
char c='c';
float f;
char c1, c2;
c1=100;  /* 100 e convertit implicit la char, c1 va primi
            valoarea caracterului cu codul 100, adica 'd'  */
i=c;     /*  i va fi 99 (codul ASCII al caracterului 'c')  */
i=2.9;   /* 2.9 e convertit implicit la int, deci i va fi 2   */
c2=c1;   /* c2 va fi 'd'  */
f='A';   /* f va fi 65.0 ( codul ASCII al caracterului 'A')  */
i=30000+c;     /* expresia se calculeaza in domeniul int, i
                  va fi 30099  */
```

ATENȚIE! Se pot produce:

- pierderea preciziei (**double** –> **float** –> **long int**);
- pierderea unor biți semnificativi (**long** –>**int**);
- nedeterminări.

Operatorul de ***conversie explicită*** (`cast`) se utilizează când se dorește ca tipul unui operand (expresie) să fie altul decât cel implicit. Operatorul are forma:

(<*Tip*>) <*Expresie*> sau <*Tip*> (<*Expresie*>)

Aceste construcții se mai numesc expresii `cast`.

Prima formă este forma tradițională, iar cea de a doua una specifică. În forma specifică, numele de tip de date predefinit sau definit de utilizator este folosit ca o "funcție de conversie", având ca parametru tipul ce trebuie convertit.

Având în vedere faptul că ambele moduri tratează uniform toate tipurile de date, se recomandă utilizarea formei tradiționale.

Exemple de conversii explicite:

```
float r;
long l;
int i, x=7, y=3, z=-2;
float a=2.1,b=3.0, c=-1.5, m;
double d;
r=5/2;          /* impartirea se face in domeniul int, deci
                   r va fi 2 */
r=(float)5/2;   /* r va fi 2.5, pentru ca primul operand
                   este convertit la tipul float, deci calculul
                   se face in domeniul real  */
```

```
r=(float)(5/2);        /* r va fi 2, pentru ca rezultatul
                       impartirii intregi este convertit la
                       tipul float, deci calculul se face in
                       domeniul int  */

l=(long)('A' + 1.0);     // l=66
i=(int)(b*b-4*a*c);      // i=21
d=(double)(x+y)/z;       // d=-5.0
m=x*y/z;                 // m=-10
m=(float)x*y/z;         // m=-10.5
m=x/(float)2;           // m=3.5
```

3.6. Instrucţiunea de atribuire

Pascal

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 stângă.

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 sunt compatibile din punctul de vedere al atribuirii. În caz contrar, se va declanșa o eroare.

Variabila și rezultatul evaluării expresiei sunt **compatibile din punctul de vedere al atribuirii**, dacă este adevărată una dintre afirmațiile:

1) variabila și rezultatul evaluării sunt de tipuri identice;
2) tipul rezultatului este un subdomeniu al tipului variabilei;
3) ambele tipuri sunt 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 considerăm următorul program:

```
Program P40;
 { 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:

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

Evident, în programul `P39` instrucțiunile de tipul

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

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

C++

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 stângă.

Exemple:

1) `x=1`

2) `y=x+3`

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

4) `p= !q`

5) `q=(a<b) || (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 sunt compatibile din punctul de vedere al atribuirii. În caz contrar, se va declanșa o eroare.

Exemple:
Se consideră declarațiile:

```
int x, y;
float m, n;
```

și atribuirile:

`x=2;`	Variabilei `x` i se atribuie valoarea 2.
`y=x+2;`	Variabilei `y` i se atribuie valoarea 2 + 2 = 4.
`m=x/y;`	Variabilei `m` i se atribuie valoarea 0, întrucât în procesul evaluării expresiei din partea dreaptă a atribuirii are loc împărțirea întreagă 2 / 4.
`n=m*2+x-5;`	Variabilei `n` i se atribuie valoarea 0 × 2 + 2–5 = –3.
`x=y+12;`	Variabilei `x` i se atribuie valoarea 4 + 12 = 16.
`x=(m>n);`	Variabilei `x` i se atribuie valoarea (0 > –3), adică `true`.

În limbajul C++ se utilizează și operatorii de atribuire compusă +=, –=, /=, *=, %=, care permit o scriere mai compactă a programelor. Aceste atribuiri au forma:

<Variabilă> <Operator de atribuire compusă> = <Expresie>

Exemple:
Se consideră declarațiile:

```
int x=6, y=3;
```

și atribuirile compuse:

`x+=12;`	Echivalent cu `x=x+12`. Lui `x` i se atribuie valoarea 18.
`y/=5;`	Echivalent cu `y=y/5`. Întrucât operanzii sunt de tip **`int`**, se realizează împărțirea întreagă. Variabilei `x` i se atribuie valoarea 0.
`x%=y;`	Echivalent cu `x=x%y`. Variabilei `x` i se atribuie valoarea 0.
`y*=2;`	Echivalent cu `y=y*2`. Variabilei `y` i se atribuie valoarea 3.

Menționăm că expresia din partea dreaptă a operației de atribuire la fel poate fi o operație de atribuire, adică se pot scrie operații de atribuire înlănțuite.

Exemple:
Se consideră declarațiile:

```
int a=8, b=2, c=7, j=9, k=11;
```

și atribuirile compuse:

`a=b*=2;`	Echivalent cu `a=b*2`. Variabilelor `a` și `b` li se atribuie valoarea 4.
`a=b=a*b;`	Echivalent cu `a=b=8*2`. Variabilelor `a` și `b` li se atribuie valoarea 16.
`k%=j-=4;`	Mai întâi se execută atribuirea `j-=4`. În consecință, variabilei `j` i se va atribui valoarea 5. Această valoare este utilizată pentru a efectua atribuirea `k%=j`, deci variabilei `k` i se va atribui valoarea 1.
`a=b=c=10;`	Fiecăreia dintre variabilele `a`, `b`, `c` i se va atribui valoarea 10.

Din această categorie mai fac parte și operatorii unari de incrementare (++) și decrementare (––).

Întrebări şi exerciţii

❶ Cum se execută o instrucțiune de atribuire?

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

❸ Se consideră declarațiile:

PASCAL

```
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;
```

C++

```
enum Zi {L, Ma, Mi, J, V, S, D};
enum Culoare {Galben, Verde,
     Albastru, Violet};
int i, j, k;
Zi z;
Culoare c;
double x;
```

Care dintre instrucțiunile ce urmează sunt corecte?

	PASCAL		C++
a)	`i:=12`	*a)*	`i=12`
b)	`j:=ord(i)`	*b)*	`j=int(i)`
c)	`x:=ord(z)+1`	*c)*	`x=int(z)+1`
d)	`k:=ord(x)+2`	*d)*	`k=int(x)+2`
e)	`c:=i+4`	*e)*	`c=i+4`
f)	`c:=Verde`	*f)*	`c=Verde`
g)	`z:=D`	*g)*	`z=D`
h)	`c:=Pred(Galben)`	*h)*	`c=Galben-1`
i)	`x:=Succ(z)`	*i)*	`x=z+1`
j)	`i:=Succ(c)`	*j)*	`i=c+1`

❹ Precizați pentru care valori ale variabilei `j` programele ce urmează vor derula fără erori?

PASCAL

```
Program P41;
var i : -10..+10;
    j : integer;
begin
  write('j=');
  readln(j);
  i:=j+15;
  writeln('i=', i);
end.
```

C++

```
// Programul P41
#include <iostream>
using namespace std;
int main()
{
short int i;
int j;
cout<<"j=";  cin>>j;
i=j+15;   cout<<"i="<<i;
return 0;
}
```

❺ **OBSERVĂ ŞI DESCOPERĂ!** Se consideră următorul fragment de program PASCAL/C++, în care variabilele x, y şi z reprezintă mărimi întregi.

PASCAL	C++
`x:=y+z; z:=x-z; y:=z; z:=x-y;`	`x=y+z; z=x-z; y=z; z=x-y;`

După execuția acestui fragment de program, unele dintre variabilele *x*, *y*, *z* ar putea să-şi modifice valorile. Selectați din lista de mai jos varianta ce conține toate variabilele, valorile cărora vor rămâne fără schimbări:

a) x; *d)* x şi y;
b) y; *e)* x şi z;
c) z; *f)* x, y şi z.

3.7. Instrucțiunea *apel de procedură* în limbajul PASCAL

NOTĂ
În limbajul C++ există doar subprograme de tip funcție. Pentru proceduri se folosește o formă particulară a funcțiilor, care va fi studiată în clasa următoare. Astfel, paragraful dat va fi studiat doar de elevii ce învață limbajul PASCAL.

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` ş.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 în studiu este

<*Apel procedură*> ::= <*Nume procedură*> [<*Listă parametri actuali*>]
<*Nume procedură*> ::= <*Identificator*>
<*Listă parametri actuali*> ::= (<*Parametru actual*> {, <*Parametru actual*>})

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

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

Exemple:

1)
```
Exit
```

2)
```
writeln(x+y, sin(x))
```

3)
```
writeln(2*x)
```

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

Întrebări şi exerciţii

❶ Care este destinația instrucțiunii *apel de procedură*?

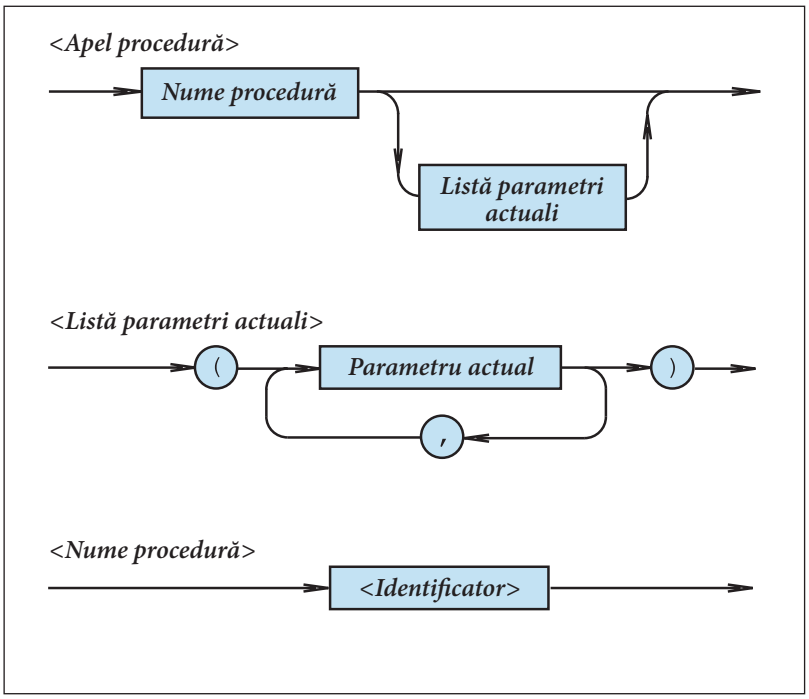


Fig. 3.4. Diagramele sintactice ale instrucţiunii *apel de procedură*

❷ **OBSERVĂ!** Se consideră instrucțiunile:

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 şi parametrii propriu-zişi.

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

3.8. Afișarea informației alfanumerice

Pascal

În versiunile uzuale ale limbajelor PASCAL şi C++, ecranul vizualizatorului este desemnat ca dispozitiv-standard de ieşire.

În limbajul PASCAL, de regulă, ecranul este împărțit în zone convenționale, numite zone-caracter. De obicei, aceste zone formează 25 de linii – câte 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(`x_1`, `x_2`, …, `x_n`)`

este echivalent cu

`write(`x_1`); write(`x_2`); …; write(`x_n`)`

Parametrii actuali dintr-un apel `write` sau `writeln` se numesc ***parametri de ieșire***. Aceștia pot avea una dintre 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 sunt 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 sunt necesare mai puțin de w caractere, atunci forma externă a valorii lui e va fi completată cu spații la stânga până la w caractere (*fig. 3.5*).

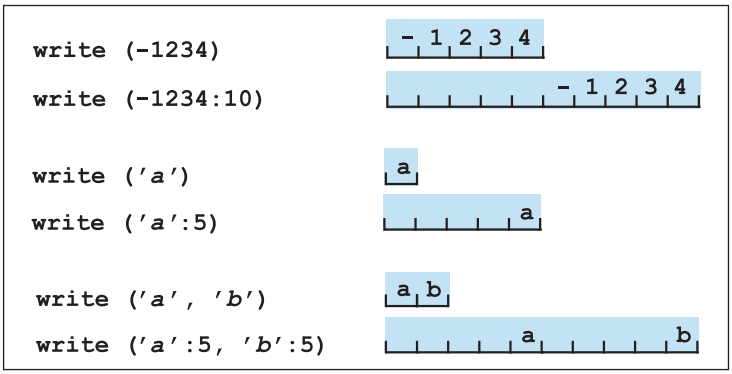


Fig. 3.5. Semnificaţia specificatorului de format w

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*).

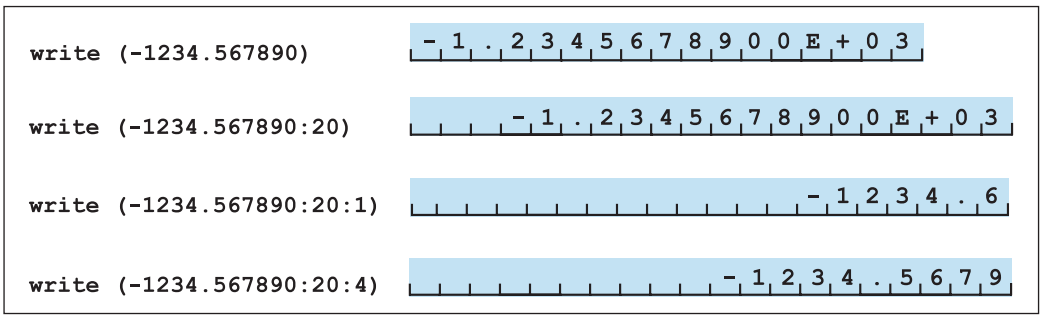


Fig. 3.6. Semnificaţia specificatorului de format f

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)
```

În limbajul C++, afișarea numerelor sau șirurilor de caractere pe ecran (consolă) se face cu ajutorul instrucțiunii:

`cout` << <*Expresie*>

Exemple:

```
1) cout<<x;
2) cout<<x<<y<<z;
3) cout<<"Aria cercului="<<(Pi*r*r);
4) cout<<x<<endl;
5) cout<<"Dati un numar intreg \n";
```

Pentru a afișa informația alfanumerică pe un anumit număr de caractere și setarea dimensiunii fiecărei valori de afișat, se va utiliza opțiunea `setw`, care se mai numește *manipulator*:

```
cout<<setw(n);
```

Numărul de poziții (coloane) stabilite pentru afișare se numește *câmp*. Data afișată va fi aliniată la dreapta, iar pozițiile câmpului rămase astfel libere vor fi umplute cu spații. Parametrul `n` este o expresie întreagă numită *specificație a dimensiunii câmpului*.

Pentru a putea folosi manipulatorul `setw`, în programul C++ trebuie inclusă biblioteca `iomanip`, lucru care se face cu ajutorul directivei `#include`.

Exemplu:

În prezența declarației

```
double x=3;
```

instrucțiunea

```
cout<<setw(10)<<x;
```

va afișa pe 10 poziții valoarea variabilei x.

Întrucât variabila x are valoarea 3, pentru afișarea acesteia este nevoie de doar o singură poziție. În consecință, cele nouă poziții din stânga vor fi completate cu spațiu.

Semnificația specificatorului n al manipulatorului setw este prezentată în *figura 3.5**.

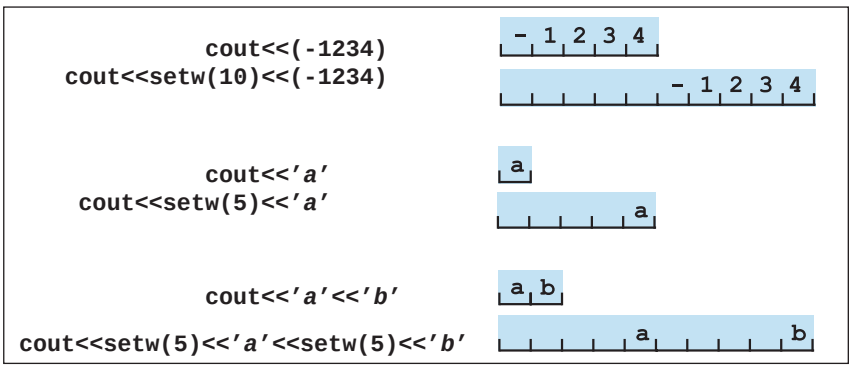


*Fig. 3.5**. Semnificația specificatorului de câmp al manipulatorului setw

În cazul afișării datelor reale, pot fi utilizate funcții speciale de format, care stabilesc o anumită precizie de afișare, de exemplu numărul de cifre după virgulă la un număr de tip **double**. În acest scop, în program pot fi incluse următoarele instrucțiuni:

`cout.setf(ios::fixed);`	Numerele reale trebuie afișate în formă zecimală.
`cout.setf(ios::showpoint);`	Punctul zecimal va fi afișat întotdeauna, chiar și în cazul numerelor întregi.
`cout.precision(n);`	Indică numărul de cifre zecimale n ce trebuie afișate.

Prima instrucțiune ne permite să folosim funcția precision doar pentru partea fracționară (de după punct), altfel ar fi luat în considerare tot numărul.

A doua instrucțiune afișează punctul zecimal de fiecare dată, chiar și pentru numere întregi.

A treia instrucțiune setează precizia numărului la n zecimale. Se fac rotunjiri! Argumentul n trebuie să fie un număr întreg pozitiv sau o expresie de tip int.

După aceste instrucțiuni, funcția cout va afișa numerele reale în noul format.

La specificarea dimensiunii câmpului pentru numerele reale, trebuie să ținem cont de faptul că punctul zecimal ocupă și el o poziție. De exemplu, valoarea 3.14 ocupă nu 3, ci 4 poziții.

Exemple:

```
double pi = 3.14159;
cout<<pi<<endl;          // se va afisa 3.14159
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<pi;                 // se va afisa 3.14
```

Instrucțiunile de format utilizate în exemplul prezentat mai sus pot fi scrise într-o linie astfel:

```
cout<<setiosflags(ios::fixed)<<setiosflags(ios::showpoint)<<
setprecision(2)<<pi;
```

sau în formă scurtă:

```
cout<<fixed<<showpoint<<setprecision(2)<<pi;
```

Pentru a afișa numărul în notație științifică (cu factor scală), se vor folosi instrucțiunile:

```
cout.setf(ios::scientific);  /* asigura ca numerele reale
                                vor fi tiparite in forma sti-
                                intifica (cu factor scala) */
 cout.setf(ios::showpoint);
 cout.precision(2);
 cout<<pi;                  //  se va afisa 3.14e+000
```

sau

```
cout<<setiosflags(ios::scientific)<<setiosflags(ios::showpoint)<<
setprecision(2)<<pi;
```

sau într-o formă și mai scurtă:

```
cout<<scientific<<showpoint<<setprecision(2)<<pi;
```

Exemple:

```
 double x = 0.0000123;
 cout.setf(ios::scientific);
 cout.setf(ios::showpoint);
 cout.precision(3);
 cout << x;                // se va afisa 1.230e-005
```

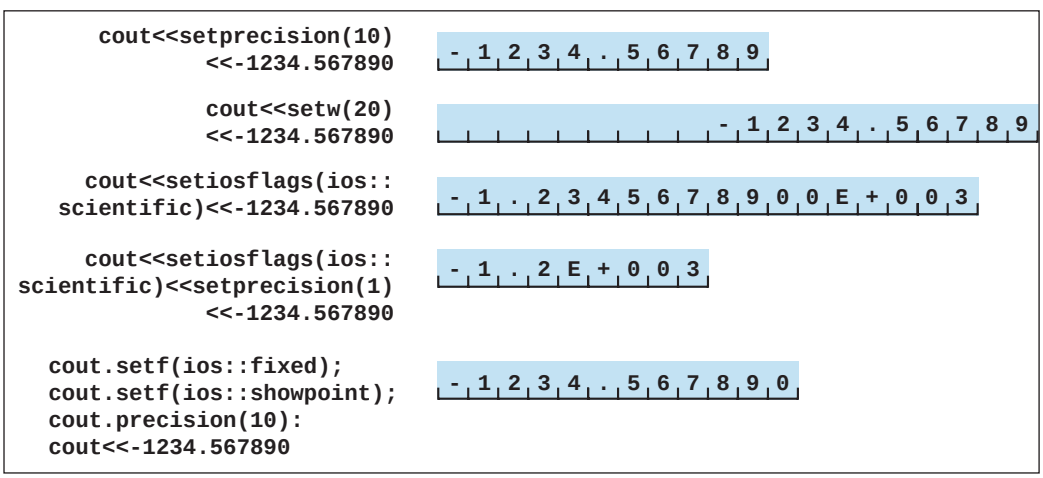


*Fig. 3.6**. Semnificația specificatorilor de format pentru numere reale

Un alt manipulator folosit foarte des este `endl`, care asigură trecerea cursorului la linie nouă.

Exemplu:

```
cout<<"Dati doua numere"<<endl;
```

Această instrucțiune va afișa pe ecran textul `Dati doua numere`, apoi va trece la linie nouă.

Pentru a trece la linie nouă, putem folosi și specificatorul `"\n"` (*escape*). Acesta este echivalentul lui `endl`. Caracterul *escape* trebuie introdus în textul dintre ghilimele. El nu va fi afișat, ci va forța cursorul să treacă pe linia imediat următoare:

Exemple:

1) `cout<<"Suma numerelor introduse este \n";`
2) `cout<<s<<endl;`
3) `cout<<"A fost odata \n ca niciodata \n";`
4) `cout<<"s="<<s<<endl;`
5) `cout<<"\n x="<<x<<"\n y="<<y<<" \n z="<<z<<endl;`

Întrebări şi exerciţii

❶ Care este destinația specificatorului de format?

❷ (PASCAL) Cum se numesc parametrii actuali ai unui apel de procedură `write` sau `writeln`?
(C++) Cum se numesc parametrii manipulatorului `setw`?

❸ **APLICĂ!** Precizați formatul datelor afișate pe ecran de programele ce urmează:

PASCAL

```
Program P42;
 { 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.
```

C++

```
// Programul P42
#include <iostream>
#include <iomanip>
using namespace std;
//Afisarea datelor de tip int
int main()
{
int i;
i=-1234;
cout<<i<<endl;
cout<<setw(1)<<i<<endl;
cout<<setw(8)<<i<<endl;
cout<<i<<i<<endl;
cout<<setw(8)<<i<<setw(8)<<i;
cout<<i<<i<<i<<endl;
cout<<setw(8)<<i<<setw(8)<<i;
return 0;
}
```

PASCAL	C++

Program P43;
{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.
	// Programul P43
	#include <iostream>
	#include <iomanip>
	using namespace std;
	// Afisarea datelor de tip real
	int main()
	{
	double x;
	x=-1234.567890;
	cout<<x<<endl;
	cout<<setw(20)<<x<<endl;
	cout<<setw(20)<<setprecision(1)<<x
	  <<endl;
	cout<<setw(20)<<setprecision(2)<<x
	  <<endl;
	cout<<setw(20)<<setprecision(4)<<x
	  <<endl;
	cout<<x<<x<<x<<endl;
	cout<<setw(20)<<x<<setw(20)<<x<<
	  setw(20)<<x<<endl;
	cout<<setw(20)<<setprecision(4)<<x
	  <<setw(20)<<setprecision(4)<<x
	  <<endl;
	cout<<fixed<<showpoint<<setpreci-
	  sion(2)<<x<<endl;
	cout<<scientific<<showpoint<<
	  setprecision(2)<<x<<endl;
	return 0;
	}

Program P44;
 { 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.
	// Programul P44
	#include <iostream>
	#include <iomanip>
	using namespace std;
	// Afisarea datelor de tip bool
	int main()
	{
	bool p;
	p=false;
	cout<<p<<endl;
	cout<<setw(10)<<p<<endl;
	cout<<p<<p<<endl;
	cout<<setw(10)<<p<<setw(10)
	  <<p<<endl;
	return 0;
	}

```
Program P45;
  {Afisarea sirurilor de
  caractere }
begin
  writeln('abc');
  writeln('abc':10);
  writeln('abc', 'abc');
  writeln('abc':10,
    'abc':10);
end.
```

```
// Programul P45
#include <iostream>
#include <iomanip>
using namespace std;
// Afisarea sirurilor de caractere
int main()
{
cout<<"abc"<<endl;
cout<<setw(10)<<"abc \n";
cout<<"abc"<<"abc \n";
cout<<setw(10)<<"abc"<<setw(10)
  <<"abc \n";
return 0;
}
```

❹ **APLICĂ!** 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
```

3.9. Citirea datelor de la tastatură

În mod obișnuit, tastatura este desemnată ca **dispozitiv-standard de intrare**.

Pascal

În PASCAL, 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(`x_1`, `x_2`, …, `x_n`)`

este echivalent cu

`read(`x_1`); read(`x_2`); …; read(`x_n`)`

Datele numerice introduse de la tastatură trebuie separate prin spații sau caractere sfârșit de linie. Spațiile dinaintea unei valori numerice sunt 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 P46;
 { 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 sunt citite de la tastatură valorile variabilelor `i`, `j`, `x` și `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, sunt convertite în valori reale. De exemplu, în cazul programului `P46` 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 `P47`:

```
Program P47;
 { 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` și `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);`

În C++, citirea datelor de la tastatură se realizează prin instrucțiunea

`cin >>` <*Variabilă*> `;`

unde <*Variabilă*> poate să indice variabile întregi, reale, caractere sau șiruri de caractere.

Dacă variabila `x` este de tip **`int`** sau **`double`**, atunci este citit întregul șir de caractere care reprezintă valoarea întreagă sau reală. Dacă `x` este de tip **`char`**, instrucțiunea citește un singur caracter.

Instrucțiunea

`cin >>` x_1 `>>` x_2 `>> ... >>` x_n `;`

este echivalentă cu

`cin >>` x_1`; cin >>` x_2`; ...; cin >>` x_n`;`

Datele numerice introduse de la tastatură trebuie separate prin spații sau caractere sfârșit de linie. Spațiile dinaintea unei valori numerice sunt ignorate. Când se citesc variabile de tip `char` (caracter), caracterele albe sunt ignorate.

Pentru a înlesni introducerea datelor, se recomandă ca instrucțiunile de citire a datelor să fie precedate de afișarea unor mesaje sugestive.

Exemple:

1)
```
cout<<"Dati doua numere: "; cin>>x>>y;
```

2)
```
cout<<"Dati un numar intreg: "; cin>>i;
```

3)
```
cout<<"x= "; cin>>x;
```

4)
```
cout<<"Raspundeti D sau N: "; cin>>c;
```

Întrebări şi exerciţii

❶ Cum se separă datele numerice ce se introduc de la tastatură?

❷ (PASCAL) Care este diferența dintre procedurile `read` și `readln`?

❸ **DESCOPERĂ!** Se consideră programele:

PASCAL	C++

`Program P48;`
`var i : integer;`
`c : char;`
`x : real;`
`begin`
`readln(i);`
`readln(c);`
	`// Programul P48`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`int i;`
	`char c;`
	`double x;`
	`cin>>i;`

```
  readln(x);
  writeln('i=', i);
  writeln('c=', c);
  writeln('x=', x);
  readln;
end.
```

```
cin>>c;
cin>>x;
cout<<"i="<<i<<endl;
cout<<"c="<<c<<endl;
cout<<"x="<<x<<endl;
return 0;
}
```

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
```

❹ **APLICĂ!** Scrieți un program care va citi de la tastatură un caracter sau o cifră și va afișa pe ecran următoarele:

a)
```
*
**
***
****
*****
```

b)
```
############
########
####
#
```

c)
```
00      00
 0    0  0   0
0       0      0
0              0
 0           0
   0      0
     0
```

❺ De la tastatură se citesc două mărimi de unghiuri exprimate în grade, minute și, respectiv, secunde: g_1, m_1, s_1; g_2, m_2, s_2. Scrieți un program ce calculează și afișează pe ecran suma mărimilor celor două unghiuri, exprimată la fel în grade, minute și secunde.

❻ De la tastatură se citesc coordonatele a două puncte: x_1, y_1; x_2, y_2. Scrieți un program care calculează și afișează pe ecran lungimea segmentului cu extremitățile în aceste două puncte și coordonatele mijlocului acestuia.

❼ Un produs are prețul unitar de x lei, la care se aplică TVA de t procente. Scrieți un program ce afișează pe ecran prețul de vânzare. Valorile x și t se citesc de la tastatură.

❽ Maria și Petru intenționează să prepare un chec cu mere. Pentru aceasta, ei au nevoie de x grame de făină, y grame de zahăr, p ouă, m kg de mere, z ml de lapte. Prețul unui kg de făină este px lei, al unui kg de zahăr py lei; kilogramul de mere costă pm lei, litrul de lapte costă pz lei, ouăle costă pp lei/bucata. Scrieți un program care calculează și afișează pe ecran prețul checului.

3.10. Instrucţiunea 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. Întrucât instrucțiunile unui program sunt despărțite între ele prin delimitatorul "`;`", prezența instrucțiunii de efect nul este marcată de apariția acestui delimitator.

De exemplu, în textul

PASCAL	C++
`x:=4;;;; y:=x+1`	`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.

3.11. Instrucțiunea `if`

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

PASCAL

<*Instrucțiune* **`if`**> ::= **`if`** <*Expresie booleană*> **`then`** <*Instrucțiune 1*>
[**`else`** <*Instrucțiune 2*>]

C++

<*Instrucțiune* **`if`**> ::= **`if`** **`(`**<*Expresie booleană*>**`)`** <*Instrucțiune 1*>;
[**`else`** <*Instrucțiune 2*>]

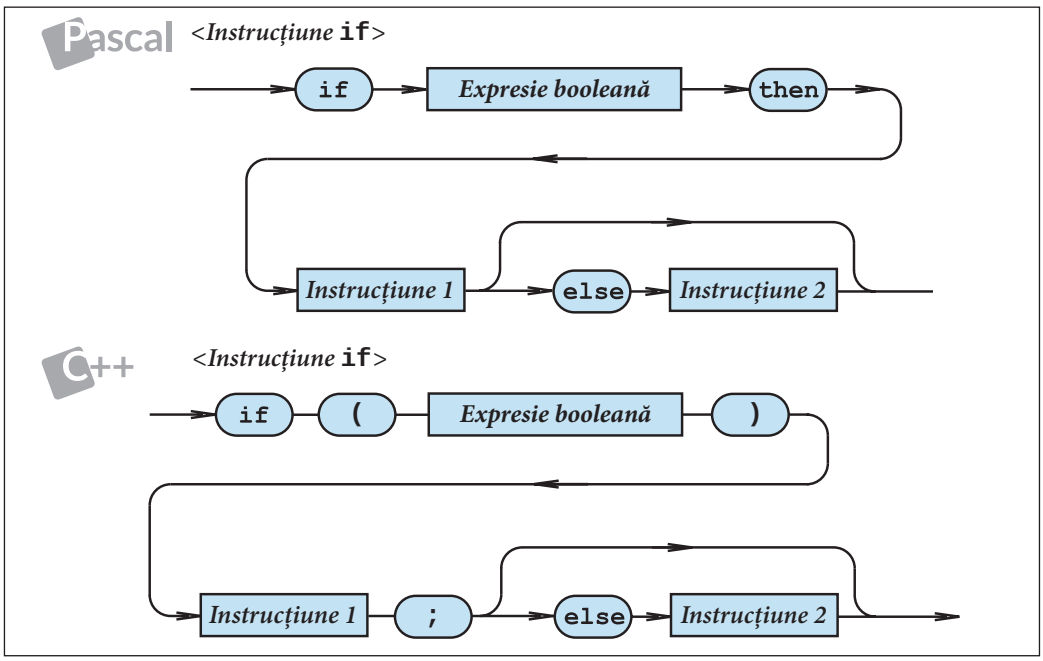


Fig. 3.7. Diagrama sintactică <*Instrucțiune* **`if`**>

Diagrama sintactică a instrucțiunii în studiu este prezentată în *figura 3.7* (p. 129). Expresia booleană din componența instrucțiunii **`if`** se numește **condiție**.

Execuția instrucțiunii **`if`** începe prin evaluarea condiției. Dacă rezultatul evaluării este `true`, atunci se execută *Instrucțiunea 1* , iar dacă condiția are valoarea `false`, atunci: fie că se execută *Instrucțiunea 2* situată după cuvântul-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.

PASCAL

```
Program P49;
{ 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.
```

C++

```
// Programul P49
#include <iostream>
using namespace std;
/* Determinarea maximului a
doua numere  */
int main()
{
float x, y, max;
cout<<"Dati doua numere: \n";
cout<<"x="; cin>>x;
cout<<"y="; cin>>y;
if (x>=y) max=x; else max=y;
cout<<"max="<<max;
return 0;
}
```

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.

PASCAL

```
Program P50;
 { Conversia cifrelor romane }
var i : integer; c : char;
begin
  i:=0;
  writeln('Introduceti una
    dintre 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;
```

C++

```
// Programul P50
#include <iostream>
using namespace std;
// Conversia cifrelor romane
int main()
{
int i; char c;
i=0;
cout<<"Introduceti una dintre
  cifrele\n";
cout<<"romane I, V, X, L, C,
  D, M \n";
cin>>c;
if (c=='I') i=1;
if (c=='V') i=5;
```

```
  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.
```

```
if (c=='X') i=10;
if (c=='L') i=50;
if (c=='C') i=100;
if (c=='D') i=500;
if (c=='M') i=1000;
if (i==0) cout<<c<<" nu este
  o cifra romana"; else cout<<i;
return 0;
}
```

Întrebări şi exerciţii

❶ Care este destinația instrucțiunii **if**?

❷ **OBSERVĂ ŞI APLICĂ!** Ce valori va lua variabila x după executarea fiecăreia dintre instrucțiunile ce urmează? Se consideră că a=18, b=-15 şi p=true.

	PASCAL		C++
a)	`if a>b then x:=1 else x:=4`	*a)*	`if (a>b) x=1; else x=4`
b)	`if a<b then x:=15 else x:=-21`	*b)*	`if (a<b) x=15; else x=-21`
c)	`if p then x:=32 else x:=638`	*c)*	`if (p) x=32; else x=638`
d)	`if not p then x:=0 else x:=1`	*d)*	`if (!p) x=0; else x=1`
e)	`if (a<b) and p then x:=-1 else x:=1`	*e)*	`if ((a<b)&& p) x=-1; else x=1`
f)	`if (a>b) or p then x:=-6 else x:=-5`	*f)*	`if ((a>b)|| p) x=-6; else x=-5`
g)	`if not (a>b) then x:=19 else x:=-2`	*g)*	`if (!(a>b)) x=19; else x=-2`
h)	`if (a=b) or p then x:=89 else x:=-15`	*h)*	`if ((a==b)||p) x=89; else x=-15`

❸ **APLICĂ!** Elaborați un program care calculează valorile uneia dintre funcțiile ce urmează:

a) $y=\begin{cases}2x, x\geq 0;\\ \dfrac{x}{2}, x<0;\end{cases}$ *b)* $y=\begin{cases}x+3, x>5;\\ x-3, x\leq 5;\end{cases}$ *c)* $z=\begin{cases}\sqrt{x+y}, \text{ dacă } c>0;\\ x\cdot y, \text{ dacă } c=0;\\ \dfrac{1}{x-y}, \text{ dacă } c<0;\end{cases}$

d) $y=\begin{cases}x, x\geq 3;\\ x+4, x<3;\end{cases}$ *e)* $y=\begin{cases}x, |x|>5;\\ 2x, |x|\leq 5;\end{cases}$ *f)* $y=\begin{cases}x^3-6x, \text{ dacă } x<-12;\\ \sqrt{x^4+12}, \text{ dacă } -12\leq x<-5;\\ 2x+12, \text{ dacă } -5\leq x<2;\\ 14, \text{ dacă } x\geq 2.\end{cases}$

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

PASCAL	C++

`Program P51;`
`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.`
	`// Programul P51`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`float x, y;`
	`cout<<"x="; cin>>x;`
	`if (x>4) y=x+6; else y=x-3;`
	`cout<<"y="<<y<<endl;`
	`return 0;`
	`}`

❹ Ce rezultate vor afişa programele ce urmează?

PASCAL	C++

`Program P52;`
`var x, y : real;`
`begin`
`  write('x='); readln(x);`
`  y:=x;`
`  if x>0 then; y:=2*x;`
`  writeln('y=', y);`
`  readln;`
`end.`
	`// Programul P52`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`float x, y;`
	`cout<<"x="; cin>>x;`
	`y=x;`
	`if (x>0); y=2*x;`
	`cout<<"y="<<y;`
	`return 0;`
	`}`

❺ Comentaţi mesajele afişate pe ecran pe parcursul compilării programului P53:

PASCAL	C++

`Program P53;`
`var x, y : real;`
`begin`
`  write('x=');`
`  readln(x);`
`  if x>4 then y:=2*sqr(x)+6;`
`  else y:=x*x*x-3;`
`  writeln('y=', y); readln;`
`end.`
	`// Programul P53`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`float x, y;`
	`cout<<"x="; cin>>x;`
	`if (x>4) y=2*pow(x,2)+6;`
	`  else y=x*x*x-3;`
	`cout<<"y="<<y;`
	`return 0;`
	`}`

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

❼ Se consideră numărul natural nenul N ce reprezintă nota unui elev. Scrieţi un program ce afişează şirul de caractere *"Promovat"* dacă $N \geq 5$ şi *"Nepromovat"* în caz contrar.

❽ Denumirile zilelor săptămânii sunt codificate după cum urmează: 1 – *Luni*, 2 – *Marţi*, …, 7 – *Duminică*. Scrieţi un program ce citeşte de la tastatură numărul

întreg n, $1 \le n \le 7$, şi afişează pe ecran denumirea zilei ce corespunde acestuia. Rescrieţi programul utilizând tipul de date *enumerare*, mulţimea de valori a căruia este {*Luni*, *Marţi*, …, *Duminică*}.

❾ Scrieţi un program care calculează soluţiile ecuaţiei $ax^2+bx+c = 0$. Numerele reale a, b şi c se citesc de la tastatură.

❿ Se consideră trei segmente de dreaptă cu lungimile, respectiv, a, b şi c. Scrieţi un program care verifică dacă din aceste segmente poate fi construit un triunghi. În caz afirmativ, programul va afişa tipul triunghiului (echilateral, isoscel sau scalen) şi aria acestuia. Pentru calcularea ariei triunghiului astfel format, se va utiliza formula lui Heron. Numerele reale a, b şi c se citesc de la tastatură.

3.12. Instrucţiunea de ramificare multiplă

În paragraful precedent am făcut cunoştinţă cu una dintre instrucţiunile de ramificare. Am observat că instrucţiunea **`if`** ne permite să creăm fluxuri de execuţie cu cel mult două ramificaţii. În cazurile în care apare necesitatea de creare a unor fluxuri de execuţie cu mai mult de două ramificări, se utilizează instrucţiunile de ramificare multiplă.

Instrucţiunea de ramificare multiplă conţine obligatoriu o expresie numită **selector** şi o listă de instrucţiuni. Fiecare instrucţiune este prefixată de una sau mai multe constante de caz. Sintaxa instrucţiunii în studiu este:

PASCAL

<*Instrucţiune* **`case`**> ::= **`case`** <*Expresie*> **`of`** [<*Caz*>{**`;`** <*Caz*>}] [**`;`**] **`end`**
<*Caz*> ::= <*Constantă*> {**`,`** <*Constantă*>} **`:`** <*Instrucţiune*>

C++

<*Instrucţiune* **`switch`**> ::= **`switch`** (<*Expresie*>) { <*Caz*>[{**`;`** <*Caz*>}]**`;`**
[**`default`**: <*Instrucţiune*> }
<*Caz*> ::= **`case`** <*Constantă*> **`:`**{**`case`** <*Constantă*>:} <*Instrucţiune*>**`;`** **`break;`**

Diagramele sintactice corespunzătoare sunt prezentate în *figura 3.8*.

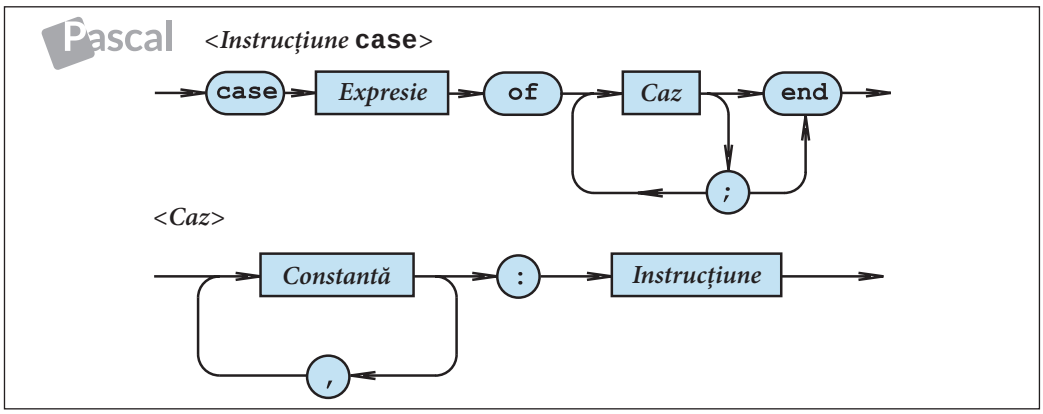


Fig. 3.8. Diagrama sintactică <*Instrucţiune* **`case`**>, PASCAL

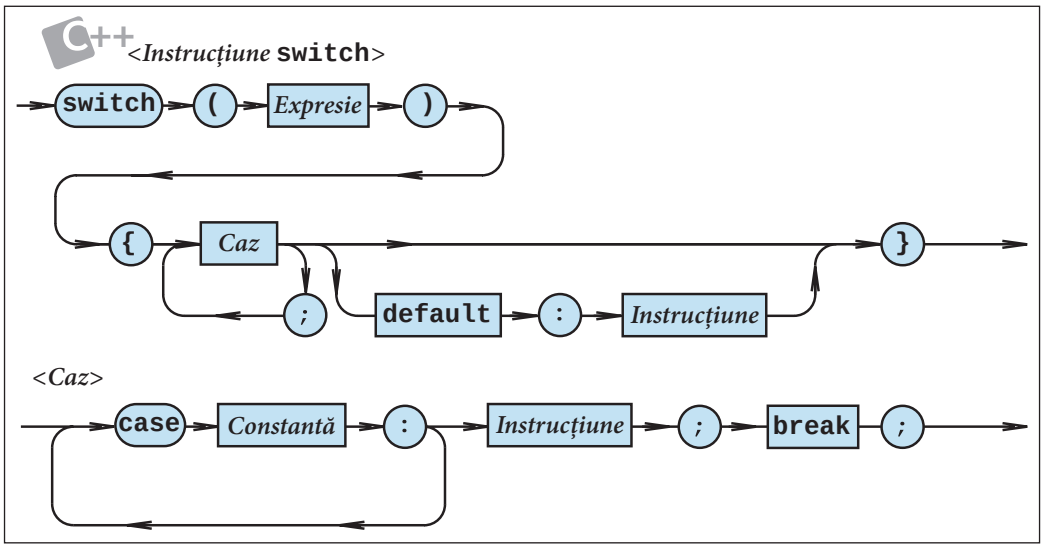


*Fig. 3.8**. Diagrama sintactică <*Instrucţiune* **switch**>, C++

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:

	PASCAL		C++
	`var i : integer; c : char; a, b, y : real;`		`int i; char c; float a, b, y;`

1)	`case i of`
	`  0, 2, 4, 6, 8 :`
	`writeln('Cifra para');`
	`  1, 3, 5, 7, 9 :`
	`writeln('Cifra impara');`
	`end;`
		1)	`switch (i)`
			`{`
			`case 0: case 2: case 4:`
			`case 6:case 8: cout<<"Cifra para"; break;`
			`case 1: case 3: case 5: case 7:`
			`case 9: cout<<"Cifra impara";`
			`}`

2)	`case c of`
	`  '+' : y:=a+b;`
	`  '-' : y:=a-b;`
	`  '*' : y:=a*b;`
	`  '/' : y:=a/b;`
	`end;`
		2)	`switch (c)`
			`{`
			`case '+' : y=a+b; break;`
			`case '-' : y=a-b; break;`
			`case '*' : y=a*b; break;`
			`case '/' : y=a/b; break;`
			`}`

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

În programul ce urmează instrucțiunea de ramificare multiplă este utilizată pentru conversia cifrelor romane în numere zecimale.

PASCAL	C++

Program P54;
{ Conversia cifrelor romane }
var i : integer; c : char;
begin
i:=0;
writeln('*Introduceti una dintre 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.
	// Programul P54
	#include <iostream>
	using namespace std;
	// Conversia cifrelor romane
	int main()
	{
	int i; **char** c;
	i=0;
	cout<<"Introduceti una din cifrele ";
	cout<<" romane I, V, X, L, C, D, M \n";
	cin>>c;
	switch (c)
	{
	case 'I' : i=1; **break**;
	case 'V' : i=5; **break**;
	case 'X' : i=10; **break**;
	case 'L' : i=50; **break**;
	case 'C' : i=100; **break**;
	case 'D' : i=500; **break**;
	case 'M' : i=1000; **break**;
	}
	if (i==0) cout<<c<<" nu este o cifra romana"; **else** cout<<i;
	return 0;
	}

Subliniem faptul că în unele implementări ale limbajului PASCAL/C++ sintaxa și semantica instrucțiunii **case/switch** poate fi extinsă. Astfel, pentru programele scrise în PASCAL/C++, lista cazurilor poate să includă o instrucțiune precedată de cuvântul-cheie **else** (în unele versiuni **otherwise**)**/default**.

Exemplu:

PASCAL	C++

Program P55;
{ 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);
	// Programul P55
	/* Simularea unui calculator de buzunar */
	#include <iostream>
	using namespace std;
	int main()
	{
	double a, b;
	char c;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;

```
  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.
```

```
cout<<"Cod operatie "; cin>>c;
switch (c)
{
case '+' : cout<<"a+b="<<a+b;
break;
case '-' : cout<<"a-b="<<a-b;
  break;
case '*' : cout<<"a*b="<<a*b;
  break;
case '/' : cout<<"a/b="<<a/b;
  break;
default: cout<<"Cod operatie
  necunoscut";
}
return 0;
}
```

Întrebări şi exerciţii

❶ Indicați pe diagramele sintactice din *figura 3.8* (respectiv, *fig. 3.8**) drumurile care corespund instrucțiunilor de ramificare multiplă din programele prezentate în acest paragraf.

❷ Cum se execută instrucțiunea de ramificare multiplă? De ce tip trebuie să fie selectorul?

❸ Ce fel de constante pot fi utilizate ca constante de caz?

❹ Înlocuiți instrucțiunea de ramificare multiplă din programele prezentate în acest paragraf cu o secvență echivalentă de instrucțiuni **if**.

❺ **APLICĂ!** Utilizând instrucțiunea de ramificare multiplă, scrieți un program care transformă numerele zecimale 1, 5, 10, 50, 100, 500, 1000, citite de la tastatură, în cifre romane.

❻ **ANALIZEAZĂ!** Ce va afişa pe ecran în urma execuției programelor ce urmează?

PASCAL

```
Program P56;
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.
```

C++

```
// Programul P56
#include <iostream>
using namespace std;
int main()
{
enum Semnal {Rosu, Galben, Verde};
Semnal s;
int x;
s=Verde; x=s-1;
switch ((Semnal)x)
{
case Rosu  : cout<<"STOP"; break;
case Galben : cout<<"ATENTIE";
  break;
case Verde : cout<<"START"; break;
}
return 0;
}
```

❼ **OBSERVĂ!** Comentați programele:

PASCAL	C++

Program P57;
{ 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.
	// Programul P57
	#include <iostream>
	using namespace std;
	// Eroare
	int main()
	{
	double x;
	cout<<"x="; cin>>x;
	switch (x)
	{
	case 0: **case** 2: **case** 4: **case** 6:
	case 8 : cout<<"Cifra para";
	break;
	case 1: **case** 3: **case** 5: **case** 7:
	case 9 : cout<<"Cifra impara";
	break;
	}
	return 0;
	}

Program P58;
{ 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.
	// Programul P58
	#include <iostream>
	using namespace std;
	// Eroare
	int main()
	{
	enum Dis {A,B,C,D} x;
	int i;
	cout<<"i="; cin>>i;
	x=(Dis)i;
	switch (x)
	{
	case A : cout<<"unu"; **break;**
	case B : cout<<"doi"; **break;**
	case C : cout<<"trei"; **break;**
	case D : cout<<"patru"; **break;**
	case E : cout<<"cinci"; **break;**
	}
	return 0;
	}

Program P59;
{ Eroare }
type Semnal = (Rosu, Galben, Verde);
Culoare = (Albastru, Portocaliu);
var s : Semnal;
c : Culoare;
	// Programul P59
	#include <iostream>
	using namespace std;
	// Eroare
	int main()
	{
	enum Semnal {Rosu, Galben, Verde};
	enum Culoare {Albastru, Portocaliu};

```
begin
  { ... }
  case s of
   Rosu     : writeln
('STOP');
   Galben   : writeln
('ATENTIE');
   Verde    : writeln
('START');
   Albastru : writeln
('PAUZA');
  end;
  { ... }
end.
```

```
Semnal s;  Culoare c;
// ...
switch (s)
{
case Rosu : cout<<"STOP"; break;
case Galben : cout<<"ATENTIE";
  break;
case Verde: cout<<"START"; break;
case Albastru : cout<<"PAUZA";
}
  // ...
return 0;
}
```

❽ **PROGRAMEAZĂ!** Zilele săptămânii *luni*, *marți*, *miercuri*, …, *duminică* sunt notate prin numerele *1*, *2*, *3*, …, *7*. Scrieți un program care citeşte de la tastatură numărul *x* şi afişează pe ecran:
a) denumirea zilei ce corespunde numărului *x*;
b) mesajul "Zi de şcoală", dacă numărului *x* îi corespunde o zi de şcoală şi mesajul "Zi liberă", dacă numărului *x* îi corespunde o zi de odihnă.

❾ **CREEAZĂ!** Scrieți un program care execută repetitiv următoarele operații:
1) Citeşte de la tastatură numerele reale *x* şi *y*.
2) Afişează pe ecran un meniu care conține comenzile ce specifică operațiile care trebuie efectuate asupra numerelor *x* şi *y*.
3) Citeşte de la tastatură numărul întreg *c* care indică comanda din meniu, selectată de utilizator.
4) În funcție de comanda *c*, calculează şi afişează pe ecran numărul real *r* – rezultatul efectuării operației respective asupra numerelor *x* şi *y*.
5) Procesul repetitiv continuă până la introducerea de către utilizator a comenzii de ieşire din program.

Meniul ce trebuie afişat pe ecran are forma:

```
CALCULEAZA:
1. Suma
2. Diferenta
3. Produsul
4. Raportul
5. Iesire
```

Evident, numărul întreg *c* introdus de utilizator poate lua doar valorile 1, 2, 3, …, 5. Dacă utilizatorul introduce o altă valoare, se afişează mesajul de eroare "Comandă greşită" şi procesul repetitiv continuă.

3.13. Instrucţiunea `for`

Instrucțiunea **`for`** indică execuția repetată a unei instrucțiuni în funcție de valoarea unei variabile de control. Sintaxa instrucțiunii în studiu este:

PASCAL

<*Instrucțiune* **for**> ::= **for** <*Variabilă*> **:=** <*Expresie1*> <*Pas*> <*Expresie 2*>
do <*Instrucțiune*>
<*Pas*> ::= **to** | **downto**

C++

<*Instrucțiune* **for**> ::= **for** (<*Variabila*>=<*Expresie 1*>; <*Variabila*><*Operator*>
<*Expresie 2*> ; <*Expresie 3*>) <*Instrucțiune*>
<Operator>::= < | <= | > | >=

Diagramele sintactice sunt prezentate în *figura 3.9*.

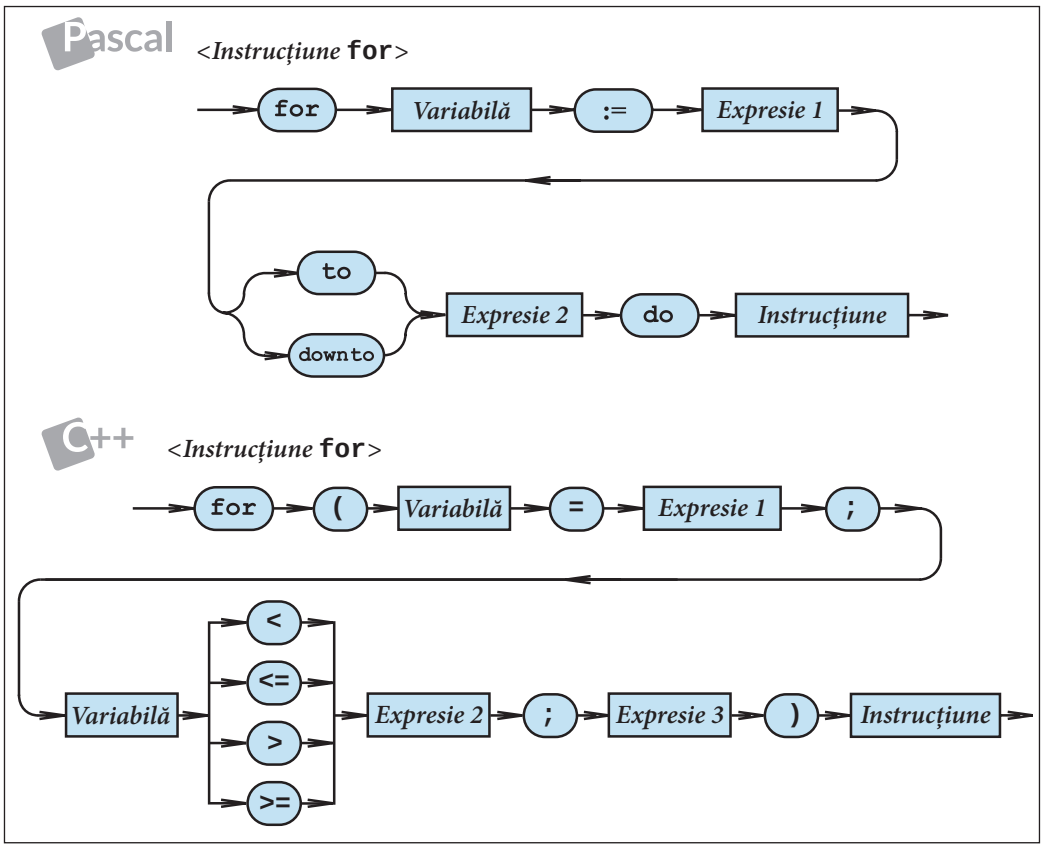


Fig. 3.9. Diagrama sintactică <*Instrucțiunea* **for**>

Variabila situată după cuvântul-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. *Expresia 1* indică valoarea inițială, iar *Expresia 2* – valoarea finală a variabilei de control.

Instrucțiunea încorporată în ciclul **for** se execută pentru fiecare valoare din domeniul determinat de valoarea inițială și de valoarea finală.

Procesul de execuție al instrucțiunii **for** poate fi descris astfel:

Pascal

Pasul 1. Se calculează valoarea *Expresiilor 1* și *2*. Aceste valori sunt calculate doar o singură dată, la începutul execuției ciclului.

Pasul 2. Se compară valorile inițială (*Expresia 1*) și finală (*Expresia 2*) ale domeniului de repetiție. Dacă valoarea finală este mai mare decât valoarea inițială, instrucțiunea încorporată în ciclul **for** nu se execută niciodată, iar procesul de execuție a instrucțiunii **for** se încheie.

Pasul 3. Contorului i se atribuie valoarea inițială (valoarea *Expresiei 1*).

Pasul 4. Este executată instrucțiunea încorporată în ciclul **for**. Aceasta poate fi o instrucțiune simplă sau compusă.

Pasul 5. Se modifică valoarea contorului (în cazul **to** se trece la succesorul, iar în cazul `downto` – la predecesorul valorii curente a contorului).

Pasul 6. Dacă valoarea modificată a contorului este mai mare ca valoarea finală (valoarea *Expresiei 2*), execuția instrucțiunii **for** se încheie, în caz contrar se trece la *Pasul 4*.

C++

Pasul 1. Se calculează valoarea *Expresiei 1*, care, în calitate de valoare inițială, este atribuită contorului *Variabilă*.

Pasul 2. Este calculată valoarea *Expresiei 2*.

Pasul 3. Este calculată valoarea expresiei booleene ce indică condiția de repetiție.

Pasul 4. Dacă valoarea curentă a expresii booleene este `false`, execuția instrucțiunii **for** se încheie.

Pasul 5. Dacă însă valoarea expresiei booleene este `true`, se execută instrucțiunea încorporată în ciclul **for**. Aceasta poate fi o instrucțiune simplă sau compusă.

Pasul 6. Se calculează *Expresia 3*, prin care indică cu cât se va mări sau se va micșora valoarea curentă a contorului și se trece la *Pasul 2*.

În limbajul C++, oricare dintre cele trei expresii din antetul instrucțiunii **for** poate lipsi, fiind obligatorie doar prezența celor două simboluri "**;**" (punct și virgulă). În cazul în care lipsește *Expresia 2*, se consideră că condiția de reluare a iterației este în mod implicit satisfăcută. În acest caz, dacă nu este prevăzută ieșirea forțată (prin salt) din ciclu, execuția instrucțiunii **for** va continua la nesfârșit.

Exemplu:

PASCAL	C++

Program P60;
{ Instructiunea **for** }
var i : integer;
c : char;
begin
for i:=0 **to** 9 **do** write(i:2);
writeln;
	// Programul P60
	#include <iostream>
	#include <iomanip>
	using namespace std;
	// Instructiunea for
	int main()
	{
	int i; **char** c;

```
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.
```

```
for (i=0; i<=9; i++)
cout<<setw(2)<<i;
cout<<endl;
for (i=9; i>=0; i--)
cout<<setw(2)<<i;
cout<<endl;
for (c='A'; c<='Z'; c++)
cout<<setw(2)<<c;
cout<<endl;
for (c='Z'; c>='A'; c--)
cout<<setw(2)<<c;
cout<<endl;
return 0;
}
```

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 *citire* în care apare variabila de control.

La ieșirea din instrucțiunea **for** valoarea variabilei de control nu este definită, în afara cazului când 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 în continuare trei programe care calculează, respectiv, $n!$, x^n și suma $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+...+\frac{1}{n}$.

PASCAL

```
Program P61;
 { 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.
```

C++

```
// Programul P61
// Calcularea factorialului
#include <iostream>
using namespace std;
int main()
{
unsigned short int n, i, f;
cout<<"n="; cin>>n;
f=1;
for (i=1; i<=n; i++) f=f*i;
cout<<"n!="<<f<<endl;
return 0;
}
```

```
Program P62;
 { 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.
```

```
// Programul P62
 /*  Calcularea lui x la
puterea n */
#include <iostream>
using namespace std;
int main()
{
float x, y;
unsigned short int n, i;
cout<<"x="; cin>>x;
cout<<"n="; cin>>n;
y=1;
for (i=1; i<=n; i++) y=y*x;
cout<<"y="<<y<<endl;
return 0;
}
```

```
Program P63;
 { 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.
```

```
// Programul P63
/* Calcularea sumei 1 + 1/2 +
+ 1/3 + ... + 1/n  */
#include <iostream>
using namespace std;
int main()
{
unsigned short int n, i;
double s;
cout<<"n="; cin>>n;
s=0;
for (i=1; i<=n; i++)
s=s+(double)1/i;
cout<<"s="<<s<<endl;
return 0;
}
```

Întrebări şi exerciţii

❶ Indicați pe diagrama sintactică din *figura 3.9* drumurile care corespund instrucțiunilor **for** din programul P60, varianta PASCAL sau C++, după caz.

❷ Cum se execută o instrucțiune **for**?

❸ **ANALIZEAZĂ!** Ce vor afişa pe ecran programele ce urmează?

PASCAL	C++

`Program P64;`
`type Zi = (L, Ma, Mi, J, V, S, D);`
`var z : Zi;`
`begin`
`  for z:=L to S do`
	`Program P64`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`enum Zi {L,Ma,Mi,J,V,S,D};`

```
writeln(ord(z));
  readln;
  for z:=D downto Ma do
writeln(ord(z));
  readln;
end.
```

```
int z;
for (z=(int)L; z<=(int)V; z++)
cout<<z; cout<<endl;
for (z=(int)D; z>=(int)Ma; z--)
cout<<z; cout<<endl;
return 0;
}
```

❹ Se consideră declarațiile:

PASCAL	C++

`var i, j, n : integer;`
`x, y : real;`
`c : char;`
	`int i,j,n;`
	`double x,y;`
	`char c;`

Care dintre instrucțiunile ce urmează sunt corecte?

	PASCAL		C++
a)	`for i:=-5 to 5 do j:=i+3`	*a)*	`for (i=-5; i<=5; i++) j=i+3`
b)	`for i:=-5 to 5 do i:=j+3`	*b)*	`for (i=-5; i<=5; i++) i=j+3`
c)	`for j:=-5 to 5 do i:=j+3`	*c)*	`for (j=-5; i<=5; i++) i=j+3`
d)	`for i:=1 to n do y:=y/i`	*d)*	`for (i=1; i<=n; i++) y=y/i`
e)	`for x:=1 to n do y:=y/x`	*e)*	`for (x=1; x<=n; x++) y=y/x`
f)	`for c:='A' to 'Z' do writeln(ord(c))`	*f)*	`for (c='A'; c<='Z'; c++) cout<<(int)c`
g)	`for c:='Z' downto 'A' do writeln(ord(c))`	*g)*	`if (c='Z'; c>='A'; c--) cout<<c`
h)	`for i:=-5 downto -10 do readln(i)`	*h)*	`if (i=-5; i>=-10; i--) cin>>i`
i)	`for i:=ord('A') to ord('A')+ 9 do writeln(i)`	*i)*	`for (i=(int)'A'; i<=(int) 'A'+9; i++) cout<<i`
j)	`for c:='0' to '9' do writeln(c, ord(c):3)`	*j)*	`for (c='0'; c<='9'; c++) cout<<c<<setw(3)<<(int)c`
k)	`for j:=i/2 to i/2+10 do writeln(j)`	*k)*	`for (j=i/2; j<=i/2+10; j++) cout<<j`

❺ **OBSERVĂ!** Se consideră declarațiile:

PASCAL	C++
`var i, m, n : integer;`	`int i, m, n;`

De câte ori se vor executa instrucțiunile de afişare a informațiilor din componența instrucțiunilor **for** ce urmează:

PASCAL	C++
`for i:=m to n do writeln(i);` `for i:=m to n do writeln(2*i);`	`for (i=m; i<=n; i++) cout<<i;` `for (i=m; i<=n; i++) cout<<2*i;`

dacă:

a) `m=1, n=5;`

b) `m=3, n=5;`

c) `m=3, n=3;`

d) `m=5, n=3.`

❻ **APLICĂ!** Elaborați 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 $4 \cdot 8 \cdot 12 \cdot 16 \cdot \ldots$;

e) $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \ldots$.

Exemplu: Pentru $n = 3$ avem $1 + 3 + 5 = 9$; $1 \cdot 3 \cdot 5 = 15$.

❽ Scrieți un program care citește de la tastatură n numere întregi și afișează pe ecran:
a) câte dintre numerele citite de la tastatură sunt pare;
b) suma numerelor pozitive citite de la tastatură;
c) suma tuturor numerelor citite de la tastatură;
d) media aritmetică a numerelor citite de la tastatură.

❾ Scrieți un program care citește de la tastatură n caractere și afișează pe ecran numărul de apariții în mulțimea de caractere introduse ale caracterelor `A` și `B`.

❿ Fiind pasionat de pomicultură, Mihai a sădit mai mulți pomi. În primul an de roadă, unul dintre pomii sădiți de Mihai a rodit M mere. În fiecare dintre următorii ani, roada a crescut cu P procente. Scrieți un program care calculează numărul de mere din roada anului N. Numerele M, P și N se citesc de la tastatură.

⓫ În parcul amenajat cu sprijinul elevilor voluntari ai liceului din localitate a fost instalat un balansoar destinat copiilor mici. În parc se joacă patru copii. Scrieți un program care determină dacă doi dintre aceşti copii se pot aşeza pe balansoar în aşa mod încât acesta să fie în echilibru. Greutatea fiecăruia dintre copii se citeşte de la tastatură.

3.14. Instrucţiunea compusă

Instrucțiunea în studiu are următoarea sintaxă:

PASCAL

<*Instrucțiune compusă*> ::= **`begin`** <*Instrucțiune*> {**`;`** <*Instrucțiune*>} **end**

C++

<*Instrucțiune compusă*> ::= **`{`** <*Instrucțiune*> **`;`**{ <*Instrucțiune*> **`;`**} **`}`**

Diagrama sintactică este prezentată în *figura 3.10*.

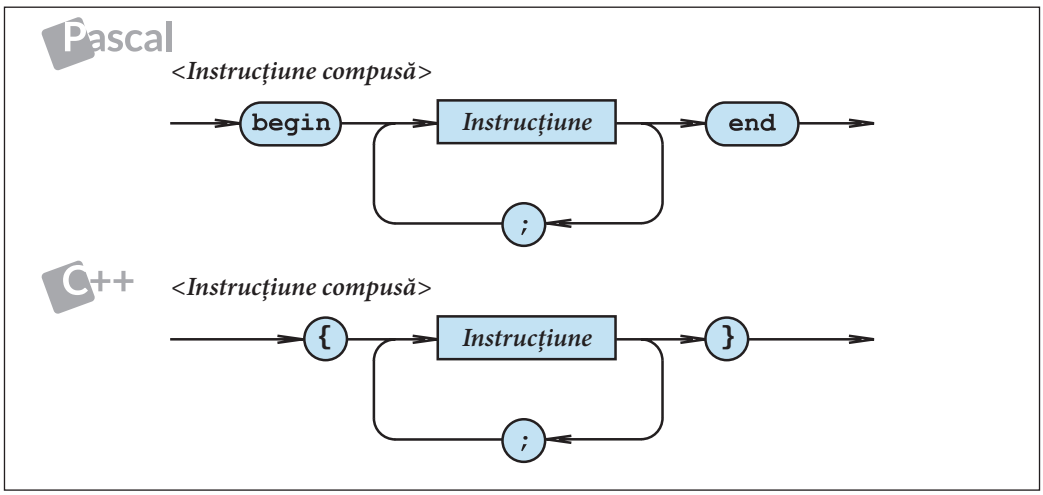


Fig. 3.10. Diagrama sintactică <*Instrucţiune compusă*>

	PASCAL		C++

1)	`begin`
	`  a:=x+12;`
	`  p:=q and r;`
	`  writeln(p)`
	`end;`
		1)	`{`
			`a=x+12;`
			`p=q && r;`
			`cout<<p<<endl;`
			`}`

2)	`begin`
	`  write('x=');`
	`  readln(x)`
	`end;`
		2)	`{`
			`cout<<"x=";`
			`cin>>x;`
			`}`

În limbajul PASCAL, cuvintele-cheie **`begin`** și **`end`**, respectiv caracterele { și } în limbajul C++ 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`**, **`for`**, **`case/switch`** etc.).

Exemple:

PASCAL

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;
```

C++

1)
```
if (a>0) {x=a+b; y=a*b ;}
    else {x=a-b; y=a/b;}
```

2)
```
switch (c)
{
case '+' : y=a+b; cout<<"Adunarea \n"; break;
case '-' : y=a-b; cout<<"Scaderea \n"; break;
case '*' : y=a*b; cout<<"Inmultirea \n"; break;
case '/' : y=a/b; cout<<"Impartirea \n"; break;
}
```

3)
```
for (i=1; i<=n; i++)
{
 cout<<"x=";
 cin>>x;
 s=s+x;
}
```

De menționat că partea executabilă a oricărui program este o instrucțiune compusă, adică o secvență de instrucțiuni încadrată între "parantezele" **begin** și **end** în PASCAL sau { și } în C++.

Pentru a face programele mai lizibile, instrucțiunile acestuia se scriu strict una sub alta, iar instrucțiunile dintre "paranteze" – cu câteva poziții mai la dreapta. Dacă instrucțiunea compusă este inclusă în componența altor instrucțiuni (**if**, **for** , **case**/**switch** etc.), instrucțiunile acestuia se scriu deplasate la dreapta.

Pentru exemplificare, prezentăm un program în PASCAL și altul în C++, în care se calculează media aritmetică a *n* numere, citite de la tastatură.

PASCAL

```
Program P65;
 { 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.
```

C++

```
// Programul P65
// Media aritmetica a n numere
#include <iostream>
using namespace std;
int main()
{
  double x, Suma, Media;
  int i, n;
  cout<<"n="; cin>>n;
  Suma=0;
  cout<<"Dati "<<n<<" numere:"<<endl;
  for (i=1; i<=n; i++)
  {
  cout<<"x="; cin>>x;
  Suma=Suma+x;
  };
  if (n>0)
  {
  Media=Suma/n;
  cout<<"Media= "<<Media<<endl;
  }
  else cout<<"Media= *****";
  return 0;
}
```

Întrebări şi exerciţii

❶ Care este destinația instrucțiunii compuse?

❷ Indicați pe diagrama sintactică din *figura 3.10* drumurile care corespund instrucțiunii compuse din programul P65 varianta PASCAL sau C++, după caz.

❸ **APLICĂ!** Elaborați un program care citeşte de la tastatură n numere şi afişează pe ecran:
a) suma şi media aritmetică a numerelor citite;
b) suma şi media aritmetică a numerelor pozitive;
c) suma şi media aritmetică a numerelor negative.

❹ Elaborați un program care citeşte de la tastatură n caractere şi afişează 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>. Sunt admise cifrele zecimale 0, 1, 2, ..., 9 şi literele mari A, B, C, ..., Z ale alfabetului latin.

❺ Scrieți un program care citeşte de la tastatură numărul întreg n, $1 \leq n \leq 9$, şi afişează pe ecran "figurile" formate din cifrele 1, 2, 3, …, n, după cum urmează:

a)
```
1
12
123
1234
...
1234...n
```

b)
```
123456...n
...
1234
123
12
1
```

c)
```
1
22
333
4444
...
nnn...n
```

d)
```
    1
   222
  33333
   ...
nnnnnnn
   ...
  33333
   222
    1
```

3.15. Instrucţiunea `while`

Ca oricare alte limbaje de programare, limbajele PASCAL şi C++ dispun de mai multe instrucțiuni destinate controlării execuției repetate a altor instrucțiuni, prin alte cuvinte, de organizare a ciclurilor. Cu una dintre astfel de instrucțiuni am făcut cunoştință în paragrafele precedente, şi anume, cu instrucțiunea **`for`**.

Reamintim că instrucțiunea **`for`** execută repetat o altă instrucțiune, simplă sau compusă, de un anumit număr de ori cunoscut la momentul scrierii programului. Există însă şi situații în care numărul de repetări nu este cunoscut la momentul scrierii programului, de exemplu când se cere repetarea calculelor până nu se ajunge la respectarea unei anumite condiții sau atingerea unui anumit scop. Astfel de situații apar în cazul scrierii programelor destinate obținerii unui profit maxim, minimizării unor eventuale costuri, prognozării veniturilor sau pierderilor unei companii, simulării proceselor de răspândire a unor epidemii etc.

Instrucțiunile repetitive, care permit organizarea ciclurilor cu un număr de repetări apriori necunoscut, ce depinde de valorile curente ale altor variabile din componența programului, se clasifică în:

- instrucțiuni repetitive cu test inițial;
- instrucțiuni repetitive cu test final.

În acest paragraf vom studia instrucțiunea repetitivă cu test inițial. Această instrucțiune are drept simbol cuvântul-cheie **`while`**. Ea conține o expresie booleană care controlează execuția repetată a altei instrucțiuni. Sintaxa instrucțiunii în studiu este:

PASCAL

<*Instrucțiune* **`while`**> ::= **`while`** <*Expresie booleană*> **`do`** <*Instrucțiune*>

C++

<*Instrucțiune* **`while`**> ::= **`while`** (<*Expresie booleană*>) <*Instrucțiune*>

Diagrama sintactică este prezentată în *figura 3.11*.

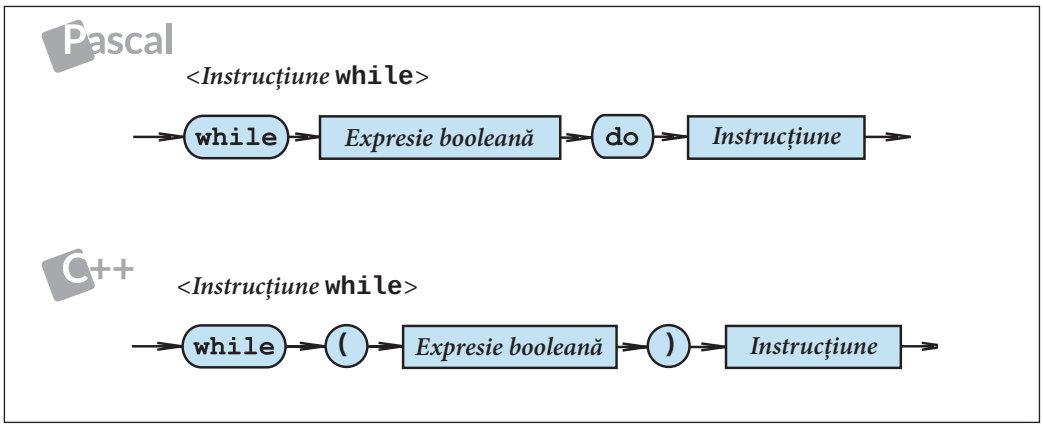


Fig. 3.11. Diagrama sintactică <*Instrucţiune* **`while`**>

Exemple:

	PASCAL		C++
1)	`while x>0 do x:=x-1;`	1)	`while (x>0) x=x-1;`

2)	`while x<3.14 do`
	`  begin`
	`  x:=x+0.001;`
	`  writeln(sin(x));`
	`  end;`
		2)	`while (x<3.14)`
			`  {`
			`  x=x+0.001;`
			`  cout<<sin(x);`
			`  }`

3)	`while p do`
	`  begin`
	`  x:=x+0.001;`
	`  y:=10*x;`
	`  p:=y<1000;`
	`  end;`
		3)	`while (p)`
			`  {`
			`  x=x+0.001;`
			`  y=10*x;`
			`  p=y<1000;`
			`  }`

Instrucțiunea simplă sau compusă din componența instrucțiunii **`while`** se execută repetat atâta timp, cât valoarea expresiei booleene este `true`. Dacă expresia booleană ia valoarea `false`, instrucțiunea respectivă nu se mai execută. Se recomandă ca expresia booleană să fie cât mai simplă, deoarece ea este evaluată la fiecare iterație.

În mod obișnuit, instrucțiunea **while** se utilizează pentru organizarea calculelor repetitive cu variabile de control de tip real.

În programele 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.

PASCAL

```
Program P66;
 { 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:10, y:20);
      x:=x+deltaX;
    end;
  readln;
end.
```

C++

```
// Programul P66
// Tabelul functiei y=2*x
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
double x, y, x1, x2, deltaX;
cout<<"x1="; cin>>x1;
cout<<"x2="; cin>>x2;
cout<<"deltaX="; cin>>deltaX;
cout<<setw(10)<<'x'<<setw(20)
<<'y';
cout<<endl;
x=x1;
while (x<=x2)
  {
   y=2*x;
   cout<<setw(10)<<x<<setw(20)
          <<y<<endl;
   x=x+deltaX;
  }
return 0;
}
```

Instrucțiunea **while** este deosebit de utilă în situația în care numărul de execuții repetate ale unei secvențe de instrucțiuni e dificil de evaluat.

Pentru exemplificare, prezentăm programul P67, care afișează pe ecran media aritmetică a numerelor pozitive citite de la tastatură.

PASCAL

```
Program P67;
{Media aritmetica a numere-
lor pozitive citite de la
tastatura}
var x, suma : real;
    n : integer;
begin
  n:=0;
  suma:=0;
```

C++

```
// Programul P67
#include <iostream>
/* Media aritmetica a nume-
relor pozitive citite de la
tastatura */
#include <iomanip>
using namespace std;
int main()
{
```

```
  writeln('Dati numere pozi-
tive:');
  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.
```

```
double x, suma;
int n;
n=0;
suma=0;
cout<<"Dati numere pozitive:
\n";
cin>>x;
while (x>0)
 {
   n=n+1;
   suma=suma+x;
   cin>>x;
  };
cout<<"Ati introdus "<<n<<
"numere pozitive.\n";
if (n>0) cout<<"media="<<su-
ma/n<<endl;
else cout<<"media=*****";
return 0;
}
```

Se observă că numărul de execuții repetate ale instrucțiunii compuse din componența instrucțiunii **while** nu poate fi calculat din timp. Execuția instrucțiunii **while** se termină când utilizatorul introduce un număr $x \leq 0$.

Întrebări şi exerciţii

❶ Cum se execută o instrucțiune **while**?

❷ Indicați pe diagramele sintactice din *figura 3.11* drumurile care corespund instrucțiunilor **while** din programele P66 şi P67, variantele PASCAL sau C++, după caz.

❸ **APLICĂ!** Utilizând instrucțiunea **while**, scrieți un program care afişează 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 **APLICĂ!** Utilizatorul introduce de la tastatură numere întregi pozitive, separate prin acționarea tastei <ENTER>. Sfârşitul secvenței de numere e indicat prin introducerea numărului 0. Scrieți un program care afişează pe ecran:
a) suma şi media aritmetică a numerelor pare;
b) suma şi media aritmetică a numerelor impare.

❺ **OBSERVĂ ŞI APLICĂ!** 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: $y = \begin{cases} x+1, & x > 8; \\ x-2, & x \le 8. \end{cases}$

PASCAL

```
Program P68;
 { Tabelul functiei }
var x, y, x1, x2, deltaX :
real;
begin
  write('x1='); readln(x1);
  write('x2='); readln(x2);
  write('deltaX='); readln(del-
taX);
  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.
```

C++

```
// Programul P68
// Tabelul functiei
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
double x, y, x1, x2, deltaX;
cout<<"x1="; cin>>x1;
cout<<"x2="; cin>>x2;
cout<<"deltaX="; cin>>deltaX;
cout<<setw(10)<<'x'<<setw(20)
<<'y';
cout<<endl;
x=x1;
while (x<=x2)
 {
    if (x>8) y=x+1; else y=x-2;
    cout<<setw(10)<<x<<setw(20)
<<y
    <<endl;
    x=x+deltaX;
 }
return 0;
}
```

❻ **DESCOPERĂ!** Instrucțiunea repetitivă

PASCAL

```
for i:=i1 to i2 do
              writeln(ord(i))
```

C++

```
for (i=i1; i<=i2; i++) cout<<i;
```

este echivalentă cu secvența de instrucțiuni

PASCAL	C++

`i:=i1;`
`while i<=i2 do`
`  begin`
`  writeln(ord(i));`
`  i:=succ(i);`
`  end.`
	`i=i1;`
	`while (i<=i2)`
	`  {`
	`  cout<<i;`
	`  i:=i+1;`
	`  }`

Scrieți o secvență echivalentă pentru instrucțiunea repetitivă

PASCAL	C++
`for i:=i1 downto i2 do` `writeln(ord(i))`	`for (i=i1; i>=i2, i--) cout<<i;`

❼ Se consideră declarațiile:

PASCAL	C++
`var x1, x2, deltaX : real;` `i, n : integer;`	`double x1, x2, deltaX;` `int i, n;`

Care dintre secvențele de instrucțiuni ce urmează sunt echivalente?

	PASCAL		C++

1)	`x:=x1;`
	`while x<=x2 do`
	`begin`
	`  writeln(x);`
	`  x:=x+deltaX;`
	`end;`
		1)	`x=x1;`
			`while (x<=x2)`
			`  {`
			`  cout<<x<<endl;`
			`  x=x+deltaX;`
			`};`

2)	`n:=trunc((x2-x1)/deltaX)+1;`
	`x:=x1;`
	`for i:=1 to n do`
	`  begin`
	`  writeln(x);`
	`  x:=x+deltaX;`
	`  end;`
		2)	`n=trunc((x2-x1)/deltaX)+1;`
			`x=x1;`
			`for (i=1; i<=n; i++)`
			`  {`
			`  cout<<x<<endl;`
			`  x=x+deltaX;`
			`  };`

3)	`n:=round((x2-x1)/deltaX)+1;`
	`x:=x1;`
	`  for i:=1 to n do`
	`  begin`
	`  writeln(x);`
	`  x:=x+deltaX;`
	`  end;`
		3)	`n=round((x2-x1)/deltaX)+1;`
			`x=x1;`
			`  for (i=1; i<=n; i++)`
			`  {`
			`  cout<<x<<endl;`
			`  x=x+deltaX;`
			`  };`

Argumentați răspunsul.

❽ **REZOLVĂ!** Utilizând instrucțiunea **while**, scrieți un program care citește de la tastatură numărul natural nenul N și afișează pe ecran:

a) ultima cifră a numărului N;
b) prima cifră a numărului N;
c) suma cifrelor numărului N;
d) produsul cifrelor numărului N;
e) numărul de apariții ale cifrei c în scrierea numărului N. Cifra c se citește de la tastatură;
f) cifra cea mai mare ce se conține în scrierea numărului N;
g) cifra cea mai mică ce se conține în scrierea numărului N;
h) numărul M, format prin scrierea în ordine inversă a cifrelor numărului N. De exemplu, pentru $N = 12345$, $M = 54321$;
i) numărul M, format prin schimbarea locului primei și al ultimei cifre. De exemplu, pentru $N = 12345$, $M = 52341$;
j) mesajul "Este palindrom" dacă N este un palindrom sau "Nu este palindrom" în caz contrar. Amintim că un număr este palindrom dacă, fiind citit de la stânga la dreapta sau de la dreapta la stânga, rămâne neschimbat. De exemplu, pentru $N = 1234321$ se va afișa "Este palindrom".

ȘTIAI CĂ?
Cel mai lung palindrom din lume este un text scris în finlandeză de Teemu Paavolainen în 1992. Acesta are 49 935 de caractere. (sursa: ro.wikipedia.org)

❾ Scrieți un program care:
a) pentru început, citește de la tastatură un caracter, pe care îl vom nota prin c;
b) în continuare, citește de la tastatură, unul câte unul, un număr apriori necunoscut de caractere;
c) oprește procesul de citire a caracterelor imediat cum utilizatorul tastează caracterul "!";
d) afișează pe ecran numărul de apariții ale caracterului c în secvența de caractere citite anterior de la tastatură.

❿ O bancă comercială oferă clienților depozite cu o dobândă anuală de p procente. Scrieți un program care calculează și afișează pe ecran:
a) suma de bani pe care o va deține clientul după x ani, dacă inițial el a depus S_i lei. Valorile p, x și S_i se citesc de la tastatură.
b) peste câți ani un client care inițial a depus S_i lei va avea pe cont S_f lei? Valorile p, S_i și S_f se vor citi de la tastatură.

3.16. Instrucțiunea repetitivă cu test final

În instrucțiunea repetitivă cu test final, verificarea condiției de încheiere a ciclului se face după execuția instrucțiunilor simple sau compuse din componența acesteia.

În PASCAL, instrucțiunea repetitivă cu test final este simbolizată prin cuvintele-cheie **repeat** ... **until**, iar în C++ – prin cuvintele-cheie **do** ... **while**.

Sintaxa acestor instrucțiuni este:

PASCAL

< *Instrucțiune* **repeat**> ::=
repeat <*Instrucțiune*> { **;** <*Instrucțiune*>} **until** <*Expresie booleană*>

C++

< *Instrucțiune* **do...while**> ::=
do {<*Instrucțiune*> **;** { <*Instrucțiune*> **;** } } **while** (<*Expresie booleană*>)

Diagramele sintactice sunt prezentate în *figura 3.12.*

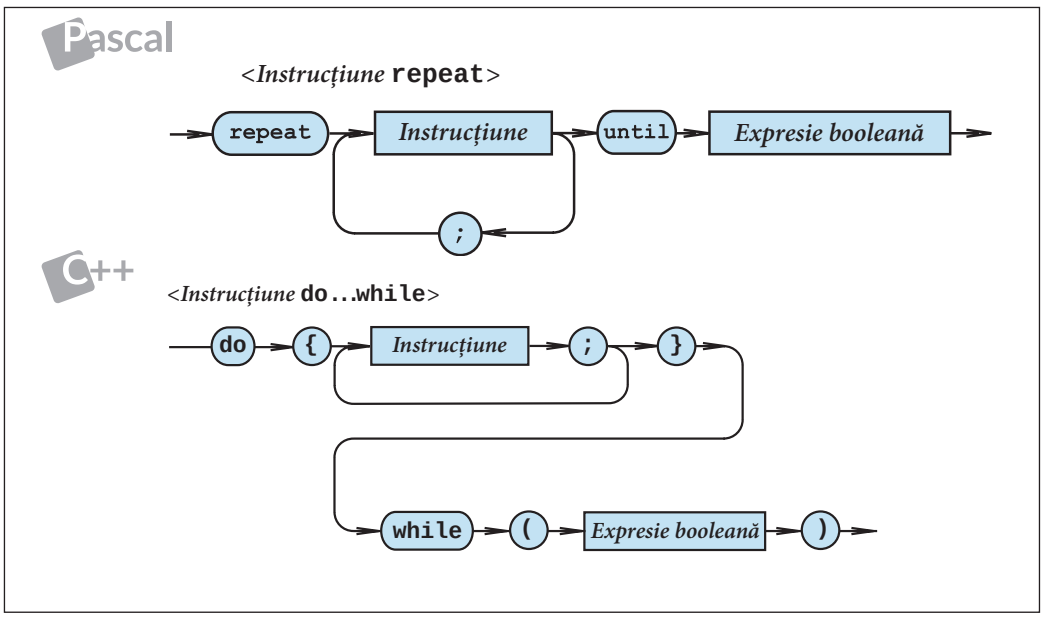


Fig. 3.12. Diagramele sintactice ale instrucțiunilor repetitive cu test final

	PASCAL		C++
1)	`repeat x:=x-1 until x<0;`	1)	`do x=x-1; while (x>=0);`

2)	`repeat`
	`  y:=y+delta;`
	`  writeln(y)`
	`until y>20.5;`
		2)	`do`
			` { y=y+delta;`
			`  cout<<y<<endl; }`
			`while  (y<=20.5);`

3)	`repeat`
	`  readln(i);`
	`  writeln(odd(i))`
	`until i=0;`
		3)	`do`
			`{  cin>>i;`
			` (i%2==0)? cout<<"True \n" :`
			`  cout<<"False \n";`
			` }`
			`while (i!=0);`

Instrucțiunile simple sau compuse din componența unei instrucțiuni cu test final se execută repetat atât timp cât expresia booleană este falsă în cazul limbajului PASCAL și adevărată în cazul limbajului C++. Când această expresie devine adevărată (PASCAL), respectiv, falsă (C++), se trece la instrucțiunea următoare.

Evident, instrucțiunile simple sau compuse din componența unei instrucțiuni cu test final vor fi executate cel puțin o singură dată, deoarece evaluarea expresiei logice are loc după execuția acestora.

În mod obișnuit, instrucțiunea repetitivă cu test final se utilizează în locul instrucțiunii repetitive cu test inițial atunci când evaluarea expresiei care controlează repetiția se face după executarea secvenței de instrucțiuni de repetat.

Programele ce urmează afișează pe ecran paritatea numerelor întregi citite de la tastatură.

PASCAL

```
Program P69;
 { 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.
```

C++

```
// Programul P69
/* Paritatea numerelor citite de la tastatura */
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
 int i;
 cout<<"Dati numere intregi:";
 do
  {
   cin>>i;
  if (i%2!=0) cout<<setw(6)<<i<<" numar impar\n";
   else  cout<<setw(6)<<i<<"numar par\n";
  }
 while (i!=0);
return 0;
}
```

Execuția instrucțiunii repetitive se termină când utilizatorul introduce `i=0`.

Instrucțiunea repetitivă 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 $y = \sqrt{x}$. Desigur, valorile negative ale variabilei x sunt inadmisibile.

PASCAL

```
Program P70;
 { 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.
```

C++

```
// Programul P70
/* Calcularea radacinii
    patrate */
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
 double x, y;
 do
  {
    cout<<"Introduceti
     numarul nenegativ  x=";
    cin>>x;
   }
```

```
 while (x<0);
 y=sqrt(x);
 cout<<"Radacina patrata
y="<<y;
 return 0;
}
```

Imediat după lansarea în execuție a acestor programe, utilizatorul este invitat să introducă un număr mai mare sau egal cu zero. Dacă, din greșeală, utilizatorul va introduce un număr negativ, instrucțiunile de citire și de scriere din componența instrucțiunilor repetitive vor fi executate din nou. Procesul repetitiv va continua până când utilizatorul va introduce un număr corect.

Din exemplele studiate se observă că instrucțiunea repetitivă cu test final este utilă în situația în care numărul de executări repetate ale unei secvențe de instrucțiuni este dificil de evaluat.

Întrebări şi exerciţii

❶ Cum se execută instrucțiunea repetitivă cu test final?

❷ Indicați pe diagramele sintactice din *figura 3.12* drumurile care corespund instrucțiunilor repetitive cu test final din programele acestui paragraf.

❸ **OBSERVĂ!** Se consideră instrucțiunile:

	PASCAL		C++

a)	`repeat`
	`  <Instructiune 1>;`
	`  <Instructiune 2>;`
	`  ...`
	`  <Instructiune n>;`
	`until p`
		a)	`do`
			`{ <Instructiune 1>;`
			`  <Instructiune 2>;`
			`  ...`
			`  <Instructiune n>;`
			`}`
			`while (!p)`

b)	`while not p do`
	`  begin`
	`  <Instructiune 1>;`
	`  <Instructiune 2>;`
	`  ...`
	`  <Instructiune n>;`
	`  end.`
		b)	`while (!p)`
			`{`
			`  <Instructiune 1>;`
			`  <Instructiune 2>;`
			`  ...`
			`  <Instructiune n>;`
			`}`

Sunt oare echivalente aceste instrucțiuni? Argumentați răspunsul.

❹ **APLICĂ!** 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>. Sunt admise cifrele zecimale 0, 1, 2, ..., 9 și caracterul * care indică sfârșitul 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 repetitive cu test final.

❻ **OBSERVĂ!** Este oare echivalentă instrucțiunea

PASCAL	C++
`for i:=i1 to i2 do writeln(ord(i))`	`for (i=i1; i<=i2; i++) cout<<i;`

cu secvența de instrucțiuni ce urmează?

PASCAL	C++

`i:=i1;`
`repeat`
`  writeln(ord(i));`
`  i:=succ(i) ;`
`until i>i2;`
	`i=i1;`
	`do`
	`{`
	`  cout<<i;`
	`  i=i+1;`
	`}`
	`while (i<=i2);`

Argumentați răspunsul.

❼ **APLICĂ!** 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 repetitive cu test final.

a) $y = 2x;$ *c)* $y = x - 4;$

b) $y = \frac{x}{3} + 9;$ *d)* $y = \frac{x}{8} - 6.$

❽ **APLICĂ!** Elaborați un program care citește de la 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>. Sunt admise literele mari și mici ale alfabetului latin și caracterul * care indică sfârșitul secvenței.

❾ Din raționamente de securitate, o bancă schimbă săptămânal codul seifului principal. Un cod se consideră sigur dacă el nu conține cifra 0, iar numărul cifrelor pare din componența acestuia este mai mare decât numărul cifrelor impare. Scrieți un program care verifică dacă codul curent corespunde cerințelor de securitate ale băncii. Cifrele codului se citesc consecutiv de la tastatură, una câte una, până la introducerea cifrei 0, care doar indică sfârșitul codului, fără a face parte din acesta.

3.17. Instrucţiunea `goto`

Instrucțiunile unui program sunt 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.

Pascal

În limbajul de programare PASCAL, instrucțiunea de salt necondiționat are sintaxa:

<*Instrucțiune* **`goto`**> ::= **`goto`** <*Etichetă*>

Amintim că în limbajul PASCAL eticheta este un număr întreg fără semn care prefixează o instrucțiune a programului (*fig. 3.1*, p. 93).

Etichetele unui program sunt listate în partea declarativă a programului după cuvântul-cheie **`label`**. Sintaxa acestei declarații este:

<*Etichete*> ::= **`label`** <*Etichetă*> {**`,`** <*Etichetă*>}**`;`**

Diagramele sintactice ale unităților gramaticale propuse sunt prezentate în *figura 3.13*. De reținut că declararea prin **`label`** a etichetelor este obligatorie.

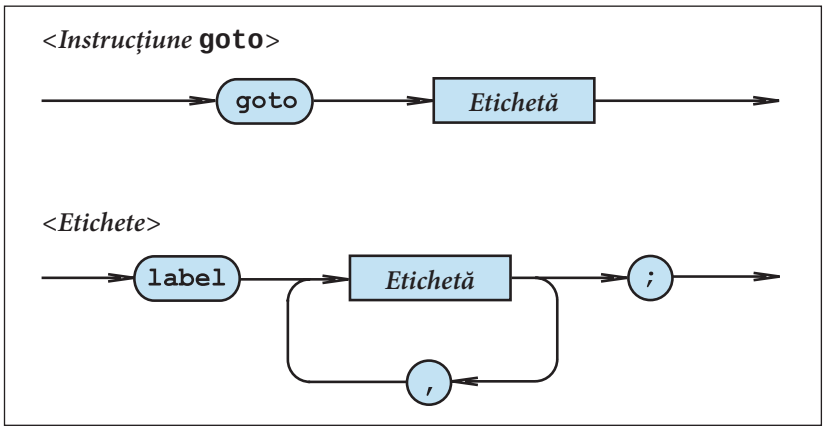


Fig. 3.13. Diagramele sintactice <*Instrucţiune* **`goto`**> şi <*Etichete*>, PASCAL

Execuția instrucțiunii **`goto`** are ca efect transferul controlului la instrucțiunea prefixată de eticheta respectivă.

Pentru exemplificare, prezentăm programul `P71` 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 P71;
 { Executia instructiunii goto }
label 1, 2;
```

```
var x, y : real;
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 cuvântului-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 sunt executate în ordinea în care sunt scrise. Prin urmare, instrucțiunile **goto** încalcă concordanța dintre textul programului și ordinea de excuție a instrucțiunilor. Acest fapt complică elaborarea, verificarea și depanarea programelor. În consecință, folosirea instrucțiunii **goto** nu este recomandată.

De exemplu, programul P71 poate fi refăcut după cum urmează:

```
Program P72;
 { Excluderea instructiunii goto din programul P71 }
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.

C++

În limbajul C++ instrucțiunea de salt necondiționat are sintaxa:

<*Instrucțiune* **goto**> ::= **goto** <*Etichetă*>**;**

Amintim că în limbajul C++ eticheta este un identificator urmat de simbolul două puncte ":", care prefixează o instrucțiune a programului (*fig. 3.13**).

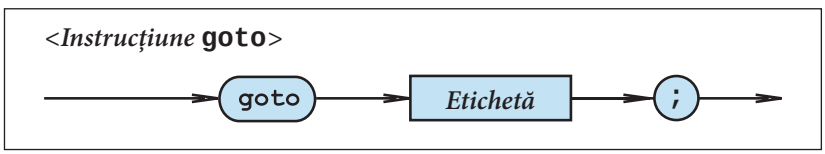


*Fig. 3.13**. Diagramele sintactice <*Instrucțiune* **goto**>, C++

O etichetă nu poate să apară de mai multe ori în corpul aceleiași funcții, domeniul de vizibilitate al unei etichete fiind restrâns la corpul funcției în care apare. Aceste etichete sunt utilizate numai de instrucțiunea **goto**, în orice alt context o instrucțiune etichetată este executată fără să se țină seama de prezența etichetei.

Pentru exemplificare, prezentăm programul care calculează valoarea funcției

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

Valorile argumentului x se citesc de la tastatură.

```
// Programul P71
#include <iostream>
using namespace std;
int main()
{
  double x, y;
  cout<<"x="; cin>>x;
  if (x>=0) goto A1;
  y=2*x;
  cout<<"x<0, y="<<y;
  goto A2;
A1: y=x;
    cout<<"x>=0, y="<<y;
A2:
return 0;
}
```

Dacă utilizatorul tastează o valoare $x \geq 0$, se va executa instrucțiunea **goto** `A1` și controlul va trece la instrucțiunea de atribuire

```
A1: y=x;
```

După aceasta se vor executa instrucțiunile

```
    cout<<"x>=0, y="<<y;
A2: return 0;
```

Dacă utilizatorul tastează o valoare $x<0$, se execută instrucțiunile

```
y= 2*x;
cout<<"x<0 , y="<<y;
goto A2;
```

Ultima instrucțiune va transfera controlul instrucțiunii

```
A2: return 0;
```

Evident, orice etichetă trebuie să prefixeze o singură instrucțiune.

În lipsa lui **goto** instrucțiunile unui program sunt executate în ordinea în care sunt scrise. Prin urmare, instrucțiunile **goto** încalcă concordanța dintre textul programului și ordinea de excuție a instrucțiunilor. Acest fapt complică elaborarea, verificarea și depanarea programelor. În consecință, folosirea instrucțiunii **goto** nu este recomandată.

De exemplu, programul de mai sus poate fi rescris în felul următor:

```
// Programul P72
#include <iostream>
using namespace std;
int main()
{
double x, y;
cout<<"x=";
cin>>x;
if (x>=0)
   {
     y=x;
     cout<<"x>=0, y="<<y;
   }
      else
   {
     y=2*x;
     cout<<"x<0, y="<<y;
   }
return 0;
}
```

De regulă, instrucţiunea **goto** se utilizează în cazuri extraordinare, de exemplu pentru a mări viteza de derulare sau pentru a micşora lungimea unui program.

Întrebări şi exerciţii

❶ Care este destinația instrucțiunii **goto**?

❷ Indicați pe diagramele sintactice din *figura 3.13* (respectiv, *fig. 3.13**) drumurile care corespund etichetelor şi instrucțiunilor **goto** din programele prezentate în acest paragraf.

❸ **APLICĂ!** Transcrieți fără a utiliza instrucțiunea **goto**:

PASCAL	C++

Program P73;
{ Afisarea formulelor de salut }
label 1, 2, 3;
var i : 6..23;
begin
write('Cat e ora?'); readln (i);
if i>12 **then goto** 1;
writeln('Buna dimineata!');
goto 3;
	// Programul P73
	// Afisarea formulelor de salut
	#include <iostream>
	using namespace std;
	int main()
	{
	unsigned short int i;
	cout<<"Cat e ora? \n"; cin>>i;
	if (i>12) **goto** Timp1;
	cout<<"Buna dimineata!";
	goto Timp3;

```
1: if i>17 then goto 2;
  writeln('Buna ziua!');
  goto 3;
2: writeln('Buna seara!');
3: readln;
end.
```

```
Timp1: if (i>17) goto Timp2;
 cout<<"Buna ziua!";
 goto Timp3;
Timp2: cout<<"Buna seara!";
Timp3: return 0;
}
```

❹ **OBSERVĂ!** Comentați următorul program:

PASCAL	C++

`Program P74;`
`{ Eroare }`
`label 1;`
`var i : 1..5;`
`begin`
`  i:=1;`
`1: writeln(i);`
`  i:=i+1;`
`  goto 1;`
`end.`
	`// Programul P74`
	`// Eroare`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`unsigned short int i;`
	`i=1;`
	`E1: cout<<i;`
	`  i+=1;`
	`  goto E1;`
	`return 0;`
	`}`

❺ **OBSERVĂ!** Ce va afișa pe ecran programul care urmează?

PASCAL	C++

`Program P75;`
`{ Eroare }`
`label 1;`
`var x : real;`
`begin`
`  x:=0;`
`1: writeln(x);`
`  x:=x+1e-30;`
`  goto 1;`
`end.`
	`// Programul P75`
	`// Eroare`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`double i;`
	`x=0;`
	`E1: cout<<x;`
	`  x=x+1e-30;`
	`  goto E1;`
	`return 0;`
	`}`

Amintim că, în PASCAL, derularea unui program poate fi întreruptă prin acționarea tastelor <CTRL+C> sau <CTRL+BREAK> , iar în C++ prin simpla închidere a ferestrei **Run**.

❻ **ANALIZEAZĂ!** Comentați programul ce urmează:

PASCAL	C++

`Program P76;`
`{ Eroare }`
`label 1;`
`var i : integer;`
`begin`
`  i:=1;`
	`// Programul P76`
	`// Eroare`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`

```
  while i<=20 do
    begin
      writeln(i);
      1: i:=i+1;
    end;
  goto 1;
end.
```

```
int i;
i=1;
  while (i<=20)
   {
      cout<<i<<endl;
 A1:   i+=1;
   }
  goto A1;
  return 0;
}
```

❼ **DESCOPERĂ!** Activitatea poate fi organizată individual sau în grupuri, pe sarcini separate.
În limbajul C++ există şi alte instrucțiuni care permit realizarea salturilor, precum: **break**, **continue** şi **return**. Studiați modul în care se execută fiecare dintre aceste instrucțiuni. Cu ajutorul instrucțiunilor în cauză, transcrieți programul din exercițiul 3 fără a utiliza însă instrucțiunea **goto**.

3.18. Generalități despre structura unui program

Pascal

Un program PASCAL are următoarea structură:

<*Program*> ::= <*Antet program*>
<*Corp*>**.**

Antetul specifică numele programului şi 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 obişnuit, parametrii formali se folosesc pentru comunicarea programului cu mediul său. Aceştia vor fi studiați mai amănunțit în capitolele următoare.

Corpul unui program este compus din partea declarativă şi 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 sunt prezentate în *figura 3.14*.

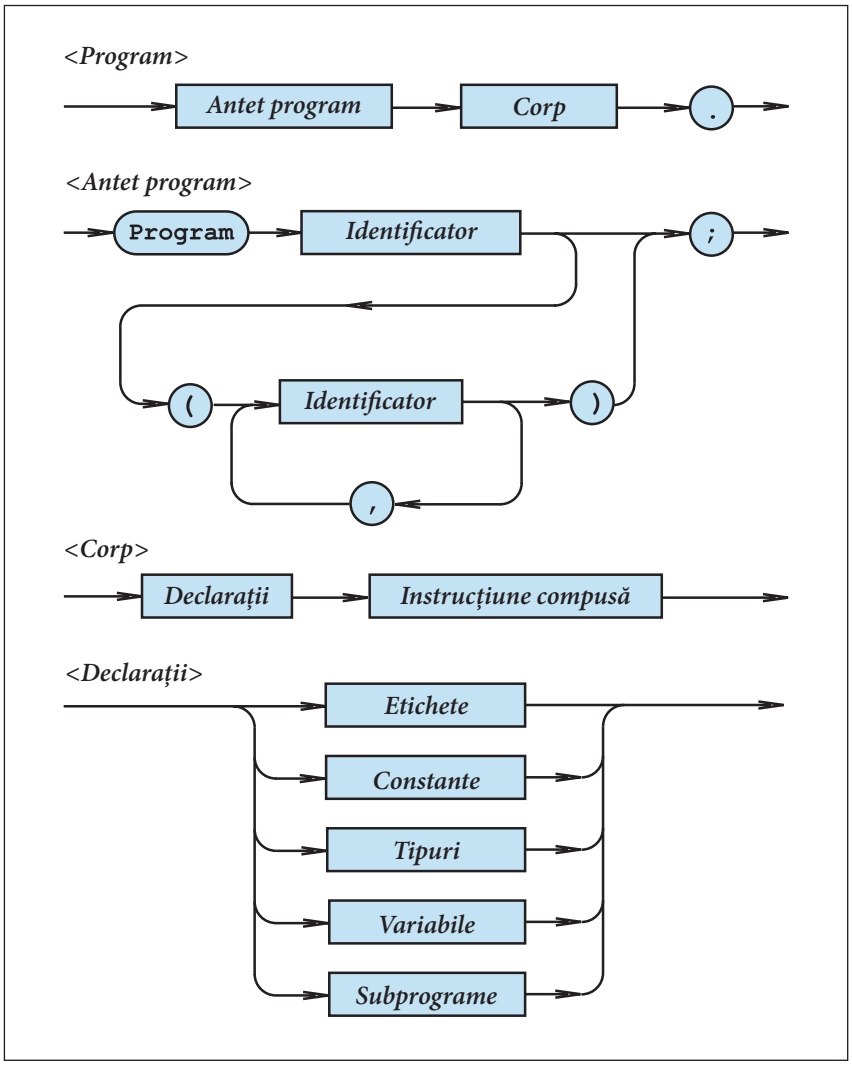


Fig. 3.14. Structura unui program PASCAL

Subliniem faptul că sfârșitul unui program PASCAL este indicat de simbolul "`.`" (punct).

Pentru exemplificare, prezentăm în continuare programul `P77`, care calculează lungimea arcului de cerc de α grade și aria sectorului respectiv.

```
Program P77;
 { 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.
```

Un program C++ are următoarele componente:

- directive;
- declarații globale, în care se definesc tipurile de date, se declară variabilele și constantele globale, folosite în program;
- funcțiile utilizatorului;
- funcția principală `main`.

Menționăm că un program scris în limbajul C++ este un ansamblu de funcții, fiecare dintre acestea efectuând o activitate bine definită. Funcția `main` este funcția principală. Anume cu ea începe execuția oricărui program C++ și prezența ei este obligatorie. Toate celelalte funcții sunt opționale, iar denumirile lor se dau de către programator. Funcțiile utilizatorului vor fi studiate mai târziu.

Forma funcției principale **`main`** este:

```
int main ()  // Antetul functiei
{            // Inceputul corpului functiei
  Declaratii locale;
  Instructiuni;
  return 0;
}          // Sfarsitul corpului functiei
```

Instrucțiunea **`return`** este utilizată pentru a încheia execuția unei funcții și a returna valoarea expresiei specificate în ea.

Pentru exemplificare, prezentăm în continuare programul care calculează lungimea arcului de cerc de α grade și aria sectorului respectiv.

```
// Programul P77
/* Lungimea arcului de cerc si aria sectorului respectiv */
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
  const double Pi=3.141592654;
  unsigned int alfa;
  double raza, lungimea, aria;
  cout<<"raza="; cin>>raza;
  if (raza<0) goto R1;
  cout<<"alfa="; cin>>alfa;
  lungimea=Pi*raza*alfa/180;
  cout<<"lungimea="<<lungimea<<endl;
  aria=Pi*pow(raza,2)*alfa/360;
  cout<<"aria="<<aria<<endl;
  goto R2;
R1: cout<<"Eroare: raza<0";
R2: return 0;
}
```

Întrebări şi exerciţii

❶ (PASCAL) Care este destinația antetului unui program? Cum se indică sfârşitul unui program?

(C++) Care este destinația funcției principale `main`?

❷ Indicați pe diagramele sintactice din *figura 3.14* drumurile care corespund unităților gramaticale ale programelor din acest paragraf.

❸ **APLICĂ!** Transcrieți programul P77 fără a utiliza instrucțiunea **goto**. Indicați părțile componente ale programului elaborat.

❹ (C++) Indicați directivele, funcția principală, începutul şi sfârşitul corpului funcției principale, declarațiile locale şi constantele din programul P77.

Test de autoevaluare nr. 3

1. Scrieți conform regulilor limbajului de programare PASCAL/C++:

a)	$(a + b) - 2ab$;	*d)*	$2\alpha\beta - 5\pi r$;
b)	$6a^2 + 15ab - 13b^2$;	*e)*	$\pi r^2 + \alpha\beta^2$;
c)	$(a + b)\,(a - b)$;	*f)*	$xy \vee xz$.

2. Transpuneți expresiile limbajului PASCAL/C++ în notații obișnuite:

	PASCAL		C++
a)	`sqr(a)+2/sqr(b)`	*a)*	`pow(a,2)+2/pow(b,2)`
b)	`2*a/(b+c)`	*b)*	`2*a/(b+c)`
c)	`15*sqrt(a/(a-b))`	*c)*	`15*sqrt(a/(a-b))`
d)	`not(x and y) or z`	*d)*	`!(x && y) || z`
e)	`sqr((a+b)/2)`	*e)*	`pow((a+b)/2,2)`
f)	`(x<>0) and (q<p)`	*f)*	`(x!=0) && (q<p)`

3. Care dintre expresiile ce urmează sunt greșite în limbajul studiat?

	PASCAL		C++
a)	`2*a+2*b`	*a)*	`2*a+2*b`
b)	`4*sinx+4*cosy`	*b)*	`4*sinx+4*cosy`
c)	`3*sqr(x)+3/sin(y)`	*c)*	`3*pow(x,2)+3/sin(y)`
d)	`a+2*-b`	*d)*	`a+2*-b`
e)	`not (q and p)`	*e)*	`! (q && p)`
f)	`2*(+x)+((-y))`	*f)*	`2*(+x)+((-y))`

4. Fie x=1, y=2 și z=3. Evaluaţi expresiile:

	PASCAL		C++
a)	`x+2*y+3*z`	*a)*	`x+2*y+3*z`
b)	`(1+x+y-2)*z`	*b)*	`(1+x+y-2)*z`
c)	`x*y+y*(-z)`	*c)*	`x*y+y*(-z)`
d)	`not(x+y+z>0)`	*d)*	`!(x+y+z>0)`
e)	`x*y<y+z`	*e)*	`x*y<y+z`
f)	`(x>y)or(2*x<y+z)`	*f)*	`(x>y)||(2*x<y+z)`

5. În prezenţa declaraţiilor:

PASCAL	C++

`var x : real;`
`i : integer;`
`p : boolean;`
`s : char;`
`Zi : (Luni, Marti, Miercuri, Joi, Vineri, Sambata, Duminica);`
	`double x;`
	`int i;`
	`bool p;`
	`char s;`
	`enum {Luni, Marti, Miercuri, Joi, Vineri, Sambata, Duminica} Zi;`

aflaţi tipul următoarelor expresii:

	PASCAL		C++
a)	`i mod 9`	*a)*	`i % 9`
b)	`i/9`	*b)*	`i/9`
c)	`i+x`	*c)*	`i+x`
d)	`ord(pred(Zi))`	*d)*	`Zi+1`
e)	`ord(Zi)+trunc(x)`	*e)*	`Zi+trunc(x)`
f)	`sqr(ord(s))`	*f)*	`pow(s,2)`
g)	`p or (x>i)`	*g)*	`p ||(x>i)`
h)	`chr(i+ord(p))`	*h)*	`char(i+p)`

6. Scrieți un program care afișează pe ecran valorile expresiei $15i(x+y)$. Valoarea variabilei întregi i și valorile variabilelor reale x, y se citesc de la tastatură.

7. Se consideră declarațiile:

PASCAL	C++

`type FunctiaOcupata = (Muncitor, SefDeEchipa, Maistru, SefDeSantier, Director);`
`StareaCivila = (Casatorit, Necasatorit);`
`var i : integer;`
`x : real;`
`f : FunctiaOcupata;`
`s : StareaCivila;`
	`enum FunctiaOcupata {Muncitor, SefDeEchipa, Maistru, SefDeSantier, Director};`
	`enum StareaCivila {Casatorit, Necasatorit};`
	`int i;`
	`double x;`
	`FunctiaOcupata f;`
	`StareaCivila s;`

Care dintre instrucțiunile de pe pagina ce urmează sunt corecte?

	PASCAL		C++
a)	`i:=ord(f)+15`	*a)*	`i=f+15`
b)	`f:=Casatorit`	*b)*	`f=Casatorit`
c)	`x:=ord(f)+1`	*c)*	`x=f+1`
d)	`i:=2*x-15`	*d)*	`i=2*x-15`
e)	`s:=pred(s)`	*e)*	`s=s-1`
f)	`f:=succ(SefDeEchipa)`	*f)*	`f=SefDeEchipa+1`

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/C++ 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. Utilizând instrucțiunea **`for`**, scrieți un program care calculează pentru primii *n* termeni:

$$s=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{n}$$

și produsul

$$p=\frac{1}{1}\cdot\frac{1}{2}\cdot\frac{1}{3}\cdot...\cdot\frac{1}{n}.$$

11. Utilizând 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 următoarelor formule metalingvistice:

<Cifră> ::= `0|1|2|3|4|5|6|7|8|9`
<Mesaj special> ::= `*{`*<Cifră>*`}#`

Utilizând instrucțiunea repetitivă cu test final, scrieți un program PASCAL/C++ care afișează pe ecran numărul de cifre din mesajele speciale. De exemplu, dacă utilizatorul tastează:

```
*104#<ENTER>
```

pe ecran se va afișa numărul 3.

Capitolul 4

MODULE LA ALEGERE

Modulele din acest capitol sunt opționale. Ce înseamnă acest lucru? Aveți posibilitatea să alegeți unul dintre ele și să-l studiați în ritm propriu, de sine stătător sau împreună cu colegii, în baza de exersare și elaborare de proiecte, cu ajutorul mijloacelor de instruire asistată de calculator. După ce ați ales un modúl, împreună cu profesorul de informatică veți decide ce programe de aplicații veți utiliza, veți avea grijă ca ele să fie licențiate sau cu distribuție liberă, vă veți asigura că ele sunt instalate pe calculatoarele din laboratoarele de informatică și pe cele personale, după caz.

4.1. Web design

În sens larg, prin *Web design* se înțelege elaborarea site-urilor Web, începând cu structura acestora și terminând cu conținuturile și forma în care ele vor fi oferite utilizatorului. Primul designer Web a fost Tim Berners-Lee, inventatorul unuia dintre cele mai răspândite servicii ale Internetului – WWW (*World Wide Web* – Pânza Mondială [de Păianjen]). Această "pânză" este formată din totalitatea site-urilor, documentelor și informațiilor legate între ele, ce pot fi accesate prin intermediul rețelei mondiale Internet.

În domeniul designului Web se utilizează următorii termeni:

Pagină Web – un fișier ce conține text simplu, neformatat, modul de afișare a componentelor căruia este indicat cu ajutorul mijloacelor unui limbaj de marcare, denumit HTML (*Hyper Text Markup Language* – Limbaj de marcare a hipertextului). Pe lângă informațiile referitoare la stilul de afișare a textului propriu-zis (caractere, paragrafe, liste, tabele etc.), limbajul HTML permite crearea elementelor de control (butoane, casete de text, liste derulante, contoare etc.) și a referințelor (linkurilor). Referințele (linkurile) pot duce atât la obiectele din interiorul paginii Web, cât și la diverse obiecte externe. Obiectele externe, cum ar fi imaginile, secvențele audio, secvențele video și chiar alte pagini Web, pot fi stocate atât pe calculatorul local, cât și pe oricare alt calculator, conectat la Internet.

Document Web – un set de pagini Web inter-referite, împreună cu obiectele lor externe, stocate pe calculatorul local.

Site Web – document Web, publicat pe Internet.

Procesul de studiere a Web designului va urma etapele de elaborare a documentelor Web:

1. Alegerea aplicațiilor de elaborare a documentelor Web. Acestea pot fi de uz general sau specializate.

În cazul aplicațiilor de uz general, cele mai utilizate sunt aplicațiile de oficiu, studiate în clasele gimnaziale: procesoarele de texte, procesoarele de calcul tabelar, aplicațiile de prezentări electronice. Aplicațiile în cauză permit salvarea documentelor în formatele Webului: HTML (un format text proiectat pentru a reprezenta paginile Web) și MHT (un format pentru documentele Web arhivate). Accentuăm faptul că aplicațiile de uz general sunt utile doar la etapele inițiale de studiere a Web designului, întrucât nu oferă mari posibilități de valorificare a tuturor instrumentelor de personalizare a paginilor Web, de inserare a elementelor multimedia, de asigurare a interactivității, de selectare a informațiilor furnizate utilizatorului în funcție de profilul acestuia.

În cazul aplicațiilor specializate, studierea cărora necesită mai mult timp și mai mult efort, elevii vor putea interveni la nivel de fiecare componentă a documentului Web, vor putea proiecta în detaliu structura acestora, vor avea posibilitatea să-și dezvolte creativitatea prin axarea propriilor activități pe aspectele artistice ale produselor digitale create de ei.

Menționăm faptul că marile companii de servicii Internet oferă utilizatorilor aplicații online de elaborare a documentelor Web, aplicații care sunt însoțite de sisteme de asistență și de manuale electronice, destinate studierii de sine stătătoare a proceselor de dezvoltare a documentelor Web.

2. Stabilirea grupului-țintă. În majoritatea absolută a cazurilor, documentele Web sunt postate pe Internet și pot fi accesate de un număr nedefinit de utilizatori. Aceștia pot avea cele mai diverse profiluri: categoria de vârstă (copii, adolescenți, tineri, maturi), nivelul de studii (primare, gimnaziale, liceale, tehnice profesionale, superioare), statutul ocupațional (elevi, studenți, angajați, neangajați, pensionari), domeniul ocupațional (agricultură, industrie, comerț, învățământ, sănătate, aplicarea legii, justiție, artă, administrația publică etc.), limbile preferate ș.a.m.d.

Evident, impactul unui document Web va fi cu atât mai puternic, cu cât conținuturile și stilurile de prezentare a acestuia vor corespunde profilului grupului-țintă pentru care el va fi elaborat. Prin urmare, designerul Web va stabili cu exactitate caracteristicile grupului-țintă căruia este destinat, în principal, documentul.

De exemplu, un document destinat elevilor din clasele primare va conține mai puțin text și mai multe imagini, corpul de literă al textului va fi suficient de mare pentru a nu crea un disconfort vizual copiilor, structura documentului va fi una cât mai simplă, fără referințe inutile, pe mai multe niveluri. Conținuturile propuse copiilor vor corespunde particularităților lor de vârstă, vor fi adaptate la modul lor de percepere și studiere a lumii prin observare și elemente de joacă.

Un document destinat elevilor din clasele liceale va avea o structură mai sobră, fără elemente grafice ce doar distrag atenția, un sistem logic de navigare, fără referințe ce formează cercuri vicioase.

3. Stabilirea scopului documentului Web. Documentele Web pot urmări diferite scopuri: de informare, de promovare, de exprimare artistică, de socializare, de prestare a anumitor servicii, de divertisment, de agrement, recreative etc.

Pornind de la scopul propus, designerul Web va alege conținuturile și stilurile de prezentare a acestora, evitând furnizarea de informații irelevante în raport cu obiectivul urmărit și aglomerarea documentului cu stiluri inutile, ce doar pur și simplu ilustrează capacitatea individuală a autorului de a mânui anumite instrumente informatice.

De exemplu, în cazul unui document Web de promovare a voluntariatului, accentul se va pune pe conținuturile ce pun în evidență rolul voluntariatului în viața comunității, motivația participanților de a se implica în astfel de activități, impactul activităților de voluntariat asupra stării de bine a comunității în ansamblu sau a anumitor categorii de cetățeni în particular. Conținuturile documentului Web (textuale, grafice, sonore și video) vor fi mobilizatoare, vor proteja identitatea beneficiarilor individuali, vor conține exemple de bune practici, vor sugera utilizatorilor eventualele domenii de extindere a activităților de voluntariat.

În cazul unui document Web ce promovează egalitatea de gen, accentul se va pune pe exemplele de succes, pe bunele practici și se va evita aglomerarea conținutului cu citate lungi din documentele juridice, destinate doar specialiștilor în drept. Nu se va recurge la un ton moralizator; realizările și neajunsurile care încă mai persistă în acest domeniu vor fi abordate într-o manieră echidistantă, din perspectiva ambelor genuri. Conținuturile vor fi orientate mai mult spre depășirea stereotipurilor, spre conștientizarea efectelor benefice pe care le are egalitatea de gen în dezvoltarea umană.

4. Elaborarea structurii documentului Web. Tradițional, în activitatea cotidiană, tot mai digitalizată pe zi ce trece, semnificația termenului *document* este considerată ca fiind foarte aproape de semnificația termenului *fișier*.

În cazul documentelor Web, această asociere poate crea mari confuzii, întrucât, în general, un document Web conține mai multe fișiere. Principalul dintre ele, cel în format HTML, este citit și afișat de programul de navigare (*browserul*), care, urmând referințele din acesta, afișează/rulează celelate fișiere. De oibicei, fișierele respective sunt în formatele text, grafic, audio și video. Evident, referințele din fișierul principal pot duce și la alte fișiere HTML, obținându-se astfel o structură pe multe niveluri.

În consecință, în cazul structurii unui document Web vom deosebi:

Structura paginii Web, afișată pe ecran de programul de navigare. Această structură este formată din texte, imagini și elemente de control: de navigare, de introducere a informațiilor, de rulare a secvențelor audio și video etc.

La elaborarea structurii paginii Web, afișate pe ecran, designerul va ține cont atât de specificul grupului-țintă și de scopul documentului, cât și de particularitățile echipamentelor cu care vor fi accesate paginile respective: staționare, mobile, cu ecran mare sau cu ecran mic, controlate prin tastatură și șoricel sau prin ecrane tactile, puterea de rezoluție a ecranului, capacitatea de transmisie a canalului de Internet ș.a.m.d.

Structura documentului Web, care constă din pagina principală, paginile subordonate și obiectele externe. Pagina principală se află pe nivelul superior al ierarhiei (nivelul 0), iar paginile subordonate – pe nivelurile 1, 2, 3 ș.a.m.d. Practic, nivelul unei pagini subordonate se determină prin numărul de clicuri, necesare pentru a ajunge la ea, pornind de la pagina principală.

Se recomandă ca în cazul proiectării documentelor Web destinate largului public să se respecte "regula de aur a Webului": orice pagină subordonată poate fi accesată prin cel mult trei clicuri.

De obicei, pentru a simplifica procesul de proiectare, structura documentului Web este redată printr-un desen format din mici dreptunghiuri ce reprezintă paginile Web și linii ce reprezintă trecerea de la o pagină la alta. Se recomandă ca desenul în cauză să nu conțină "cercuri vicioase", adică să poată fi asociat cu un arbore, ramurile căruia cresc în jos. La "rădăcina" arborelui se află pagina principală (nivelul 0), iar paginile subordonate sunt simbolizate prin "frunzele" acestuia.

Structura sistemului de fișiere ale documentului Web. Ca și oricare altă structură de fișiere, ea este formată din directoare ce conțin obiectele externe locale, cum ar fi fișierele text, grafice, audio, video și, desigur, fișierele HTML. De obicei, în cazul aplicațiilor de uz general, această structură este creată în mod automat în momentul salvării documentului Web și constă dintr-un singur director, numele căruia derivă din denumirea paginii HTML. În cazul aplicațiilor specializate, structura respectivă este una arborescentă, directoarele respective fiind create în mod automat sau de designerul Web. În majoritatea cazurilor, designerii Web preferă ca unul dintre aceste directoare să conțină fișierele grafice, altul – fișierele audio, al treilea – fișierele video ș.a.m.d.

5. Crearea paginilor Web. Această etapă include crearea fiecăreia dintre paginile Web din componența documentului Web și constă în inserarea și formatarea obiectelor din componența acestora. Într-o anumită măsură, acest proces este similar celuia de creare a documentelor tradiționale:

– stabilirea structurii ierarhice a paginii Web;
– inserarea textelor, listelor, tabelelor și formatarea acestora;
– inserarea și formatarea diagramelor, imaginilor, formulelor, secvențelor audio, secvențelor video, elementelor de navigare și de control;
– inserarea referințelor (linkurilor) către obiectele externe;
– așezarea în pagină.

Accentuăm faptul că obiectele ce vor fi inserate în paginile Web pot fi preluate din alte surse sau create de autorii documentului Web cu ajutorul unor aplicații specializate, de exemplu cu ajutorul programelor de creație și prelucrare a imaginilor cu rastru și a celor vectoriale, de prelucrări digitale audio și video. Evident, în astfel de cazuri, lucrul de creare a documentelor Web poate fi organizat în echipe, fiecare membru al echipei fiind specializat pe un anumit tip de obiecte.

6. Testarea documentului Web. Pentru început, documentul Web va fi testat local cu ajutorul unui program de navigare. În procesul testării se va verifica accesibilitatea tuturor

obiectelor din componența documentelor Web, corectitudinea referințelor, funcționarea instrumentelor de navigare, corectitudinea afișării imaginilor, fidelitatea redării secvențelor audio și video.

În cazul depistării unor erori, se va reveni la etapa de creare a paginilor Web și a obiectelor din componența acestora.

Accentuăm faptul că, pentru testare, aplicațiile specializate de elaborare a documentelor Web au mijloace speciale, fapt ce simplifică verificarea documentelor aflate în curs de definitivare.

7. Verificarea respectării dreptului de autor. În general, într-un document Web pot fi inserate și obiecte ce nu sunt create de autorii acestuia. În astfel de cazuri se va acorda o atenție deosebită respectării dreptului de autor. Înainte de a insera un obiect preluat dintr-o sursă de pe Internet sau dintr-o publicație tipărită sau multimedia, de exemplu dintr-o carte, de pe un disc audio sau video etc., designerii Web vor verifica dacă obiectul respectiv este protejat prin dreptul de autor și dacă da, prin ce tip de licență este protejat acesta (proprietară sau liberă). În funcție de tipul licenței, designerii Web vor concretiza cerințele referitoare la utilizarea obiectelor respective în propriile lor documente Web și, în caz de utilizare a acestora, vor indica sursele respective.

8. Verificarea respectării cerințelor referitoare la protecția datelor cu caracter personal și siguranța pe Internet. Întrucât majoritatea absolută a documentelor Web este destinată publicării pe Internet, în ele nu trebuie incluse date cu caracter personal, fie ele proprii, ale colegilor sau ale profesorilor. Evident, documentul Web nu trebuie să includă mijloace de solicitare și de colectare a datelor cu caracter personal de la utilizatorii ce vor accesa documentul respectiv.

Documentul Web nu va conține obiecte ce pot conține programe dăunătoare (viruși) și nici referințe (linkuri) la alte site-uri de pe Internet ce conțin astfel de programe.

În general, se va evita referirea site-urilor ce nu au indicate în mod explicit datele de ieșire: posesorul site-ului, autorii materialelor de pe site, data ultimei actualizări a materialelor respective, adresele și telefoanele companiilor ce gestionează site-ul la care se face referință.

9. Publicarea documentului Web pe Internet. Există mai multe modalități de publicare a documentelor Web, principalele dintre ele fiind:

– pe un site Web personal;

– pe site-ul Web al instituției de învățământ;

– pe site-ul Web al unei organizații obștești.

În funcție de statutul serverului pe care a fost postat documentul Web, derivă și cerințele înaintate față de conținutul și designul paginilor respective, responsabilitățile morale și materiale ce revin autorilor, posesorilor și deținătorilor de site-uri.

În cazul site-urilor Web personale, toate responsabilitățile revin posesorilor și deținătorilor acestora. Desigur, în cazul elevilor ce încă nu au atins vârsta majoratului, pentru crearea de site-uri personale se cere acordul părinților/reprezentanților legali ai acestora.

În cazul site-urilor instituțiilor de învățământ și ale asociațiilor obștești, responsabilitățile revin organizațiilor respective și autorilor de documente Web. Evident, în cazul instituțiilor de învățământ, aceste documente trebuie să corespundă misiunii și obiectivelor sistemului educațional, iar în cazul organizațiilor obștești – misiunii, scopurilor, obiectivelor și domeniilor de activitate ale acestora.

10. Menținerea și dezvoltarea documentului Web. În funcție de scopul urmărit, documentele Web pot fi postate pe Internet pe o durată relativ scurtă sau pe un termen mai lung.

În cazul unor postări de scurtă durată, documentele Web nu necesită actualizări și, după expirarea timpului preconizat, ar trebui retrase din spațiul virtual și arhivate. O astfel de abordare permite evitarea fenomenelor de "bruiaj informațional", "inflație informațională" și "saturație informațională".

În cazul postărilor de durată, documentele Web trebuie actualizate. Aceste actualizări pot fi făcute atât la solicitările utilizatorilor, cât și din proprie inițiativă. Din acest punct de vedere, se consideră ca fiind un element de bun ton includerea în documentele Web a unor mijloace automatizate ce ar permite utilizatorilor să semnaleze o eventuală eroare în conținuturile sau structura documentelor respective.

Dacă documentul Web a fost elaborat și publicat în cadrul unui serviciu online, actualizările se fac în timp real, înlocuindu-se și/sau modificându-se obiectele dorite. Dacă documentul Web a fost creat pe un calculator local și ulterior a fost postat pe Internet, desigur, actualizarea se face tot pe un calculator local, cu publicarea ulterioară a versiunii înnoite pe aceeași locație din spațiul virtual.

După cum s-a subliniat mai sus, se recomandă indicarea în mod explicit a datei ultimei actualizări pe fiecare dintre paginile Web, fapt ce-l va ajuta pe utilizator să estimeze noutatea informațiilor accesate.

Activităţile și produsele școlare recomandate

Exerciții de:

– identificare a obiectelor din componența unei pagini Web;
– identificare a obiectelor din componența unui document Web;
– reprezentare prin desen a structurii unei pagini Web;
– reprezentare prin desen a structurii unui document Web;
– elaborare a structurii unei pagini Web;
– elaborare a structurii unui document Web;
– elaborare a structurii de fișiere a unui document Web;
– diferențiere a etapelor de elaborare a unui document Web și de explicare a conținutului fiecărei etape;
– creare a documentelor Web cu ajutorul aplicațiilor de oficiu;
– creare a documentelor Web cu ajutorul aplicațiilor dedicate;
– creare a documentelor Web cu ajutorul aplicațiilor online;

– verificare a respectării drepturilor de autor (site-urile propuse de profesor și documentele Web elaborate în cadrul clasei);
– verificare a respectării regulilor de siguranță pe Internet și de protecție a datelor cu caracter personal (site-urile propuse de profesor și documentele Web elaborate în cadrul clasei);
– publicare a documentelor Web în rețeaua locală și pe Internet.

Studii de caz:
– Mesajele transmise de site-urile frecvent vizitate de elevi.
– Scopurile site-urilor frecvent vizitate de elevi.
– Structura, aspectul grafic, funcționalitatea și comoditatea de utilizare a site-urilor propuse de profesor și a celor frecvent vizitate de elevi.
– Relațiile dintre scopurile site-urilor frecvent vizitate de elevi și designul acestora.
– Relațiile dintre specificul grupurilor-țintă ale site-urilor frecvent vizitate de elevi și designul acestora.
– Actualitatea, originalitatea, relevanța, obiectivitatea, veridicitatea și corectitudinea informațiilor de pe site-urile frecvent vizitate de elevi.

Elaborarea site-urilor:
– Casa mea / Școala mea / Orașul meu / Satul natal.
– Clasa mea / Cercul meu școlar / Secția mea sportivă.
– Consiliul elevilor / Consiliul tineretului.
– Voluntariatul din școala mea, din satul/orașul meu.
– Librăria / Biblioteca / Casa de cultură / Salonul de muzică.
– Muzeul satului/orașului meu.
– Istoria satului/orașului meu.
– Oamenii remarcabili din satul/orașul meu.
– Arta în viața mea (pe domenii: muzica, arta plastică, arta decorativă).
– Sportul în viața mea.
– E minunat să fii sănătos.
– Săli de sport/de fitness.
– Saloane de înfrumusețare / Saloane de modă.
– Ateliere de cusut/de reparații (auto, calculatoare, electrocasnice, audio, video).
– Gospodăria țărănească.
– Magazine (mărfuri pentru copii, mărfuri școlare, mărfuri de uz casnic, îmbrăcăminte).

Proiectele care se referă la subiecte ample, de exemplu la orașe, sate, școli, vor fi elaborate în cadrul echipelor de elevi, fiecare membru al echipei specializându-se pe un anumit aspect, de exemplu pe cel istoric, demografic, economic, cultural, etnografic etc.

Tot în echipe vor fi elaborate și proiectele ce se caracterizează printr-un bogat conținut multimedia, specializarea membrilor echipelor realizându-se după genul materialelor respective, de exemplu grafică cu rastru, grafică vectorială, animații, foto, audio, video etc.

4.2. Grafica pe calculator

Grafica pe calculator, denumită foarte des și grafica digitală, este un domeniu al informaticii care se ocupă cu bazele teoretice și aspectele aplicative, legate de prelucrarea imaginilor cu ajutorul calculatoarelor. Cunoștințele teoretice și practice din acest domeniu foarte important al informaticii, ce are conexiuni strânse cu artele vizuale, este utilizat pentru sintetizarea, modificarea, stocarea și managementul imaginilor, precum și pentru prelucrarea informației vizuale obținute din realitatea înconjurătoare. Cunoașterea elementelor de bază ale graficii pe calculator este necesară inginerilor, oamenilor de știință, artiștilor plastici, designerilor, fotografilor, pictorilor de animație ș.a. De asemenea, pentru mulți oameni, grafica pe calculator a devenit o ocupație, o îndeletnicire plăcută în afara preocupărilor profesionale (un hobby).

În funcție de modul de reprezentare a imaginilor în memoria calculatorului, deosebim *grafica orientată pe puncte* și *grafica orientată pe obiecte*.

Grafica orientată pe puncte

Amintim că în grafica orientată pe puncte imaginile se reprezintă în memoria calculatorului prin împărțirea ei în microzone, denumite *puncte* sau *pixeli*. Descompunerea imaginii în pixeli se realizează cu ajutorul unui *rastru* (de la cuvântul latin *raster*, literalmente "greblă").

Rastrul reprezintă o suprafață plană, în general dreptunghiulară, pe care sunt trasate două seturi de linii paralele, perpendiculare între ele. Densitatea liniilor și, respectiv, densitatea pixelilor caracterizează puterea de rezoluție a echipamentelor pentru reproducerea sau formarea imaginilor.

În procesul digitalizării imaginii, pixelii sunt parcurși în ordinea în care se citesc: de la stânga la dreapta, de sus în jos. Fiecărui pixel i se pune în corespondență un număr binar, ce reprezintă într-o formă înțeleasă de calculator informațiile despre luminozitatea și culoarea acestuia.

Prin alte cuvinte, în formă digitală, în grafica orientată pe puncte imaginea este reprezentată printr-o secvență de numere binare, iar toate prelucrările acesteia se realizează prin operații asupra numerelor respective.

Secvențele de numere binare ce conțin informații despre luminozitatea și culoarea pixelilor se numesc *imagini digitale cu rastru* sau, mai simplu, *imagini cu rastru*.

Cel mai des, imaginile cu rastru se utilizează pentru prelucrarea informației vizuale obținute din realitatea înconjurătoare. De obicei, majoritatea echipamentelor destinate digitalizării imaginilor, cum ar fi camerele de luat vederi, camerele video, scanerele etc., furnizează la ieșire imagini cu rastru, mai exact fișiere în formate special elaborate pentru reprezentarea, memorarea și prelucrarea imaginilor cu rastru. De asemenea, imaginile cu rastru sunt utilizate în cazul afișării informațiilor vizuale pe ecranele dispozitivelor digitale și pentru tipărirea acestora.

Principalul avantaj al imaginilor cu rastru constă în fidelitatea lor, adică în precizia, exactitatea prezentării sau reproducerii realității. În calitate de neajunsuri vom menționa

volumul mare de memorie necesar pentru stocarea imaginilor cu rastru și înrăutățirea calității în cazul măririi acestora.

Grafica orientată pe obiecte

Pentru a asigura o calitate mai bună a imaginilor digitale în cazul redimensionării acestora și al reducerii volumului de memorie necesară pentru stocarea lor, în grafica orientată pe obiecte imaginile sunt formate din obiecte grafice simple: linii, pătrate, dreptunghiuri, circumferințe, elipse etc.

În calculator, fiecare obiect grafic din componența imaginii este codificat printr-un set de numere binare, denumit convențional *vector*. Un astfel de set de numere binare conține toate informațiile necesare pentru desenarea obiectului respectiv: coordonatele centrului și raza fiecărui cerc, coordonatele vârfului fiecărui dreptunghi etc. Evident, în vectorul ce caracterizează un obiect grafic se includ și informații despre culoarea și grosimea liniilor ce-l formează, culoarea de umplere, gradientele de culoare etc. Prelucrarea imaginilor vectoriale se realizează prin recalcularea coordonatelor și dimensiunilor fiecărui obiect grafic din componența acestora cu ajutorul formulelor din geometria analitică. Amintim că în geometria analitică figurile sunt definite cu ajutorul ecuațiilor sau inecuațiilor, iar transformarea acestora se face pur algebric. În acest scop, planul, în cazul imaginilor bidimensionale, și spațiul, în cazul imaginilor tridimensionale, sunt dotate cu sisteme de coordonate, de obicei carteziene.

Datorită dezvoltării aplicațiilor de grafică digitală, componentele de geometrie analitică pe care se bazează prelucrarea imaginilor vectoriale, adică ecuațiile, inecuațiile și algoritmii de rezolvare a acestora, sunt "invizibile" pentru utilizator, el operând doar cu termeni specifici designului grafic.

Imaginile vectoriale pot fi mărite și micșorate fără a înrăutăți calitatea lor, recalculând dimensiunile acestora conform funcțiilor matematice asociate fiecărui obiect grafic. Mai mult decât atât, cu ajutorul calculelor, obiectele respective pot fi animate, deplasate în spațiu, recolorate, transfigurate etc., ceea ce constituie avantaje deosebit de importante în crearea simulatoarelor digitale și a jocurilor de calculator.

Neajunsul imaginilor vectoriale constă în faptul că codificarea informațiilor vizuale complexe prin obiecte grafice simple, descrise cu ajutorul unor formule matematice, duce la scăderea fidelității prezentării sau reproducerii lumii reale. Desigur, scăderea fidelității nu este cauzată de formulele matematice propriu-zise, ci de complexitatea și numărul mare de obiecte ce ar fi necesare pentru a asigura nivelul dorit de fidelitate. Însă, odată cu creșterea performanțelor calculatoarelor moderne, complexitatea și numărul obiectelor grafice nu mai reprezintă un obstacol de neînvins, fapt ce poate fi ușor observat în cazul jocurilor de calculator, imaginile din componența cărora devin tot mai complexe și mai naturale.

Conversia imaginilor

Imaginile cu rastru pot fi transformate în imagini vectoriale și invers, cele vectoriale – în imagini cu rastru. În plus, aplicațiile moderne de procesare a imaginilor permit crearea

și procesarea imaginilor digitale mixte, adică imagini ce conțin atât fragmente cu rastru, cât și fragmente vectoriale.

De obicei, transformarea imaginilor cu rastru în imagini vectoriale se face în scopul micșorării volumului ocupat de ele, al combinării lumii reale cu elemente din lumea virtuală (realitatea augmentată), al recunoașterii formelor. Transformarea imaginilor vectoriale în imagini cu rastru se face în vederea afișării acestora pe ecranele echipamentelor digitale și pentru a fi tipărite.

Procesarea imaginilor cu rastru

Pentru a forma și dezvolta competențele de procesare a imaginilor cu rastru, se recomandă parcurgerea următoarelor teme:

1. Recapitularea materiilor din clasele gimnaziale: pixel, rastru, imagine cu rastru, rezoluție, dimensiuni, model de culoare.

2. Spațiul de lucru al editorului grafic: panouri, meniuri, instrumente, rigle, riglete, ghidaje, personalizarea spațiului de lucru.

3. Instrumente pentru desen: peniță, pensulă, fundal, figuri/primitive grafice.

4. Gestionarea proprietăților instrumentelor de desen.

5. Instrumente pentru selecție și editare: selector, măști, foarfeci, pipetă, radieră, cuțit.

6. Instrumente de prelucrare a textelor: de scriere, de editare, de formatare.

7. Prelucrări elementare ale imaginilor cu rastru:

– desenarea/redesenarea imaginilor cu ajutorul instrumentelor standard;

– importul imaginilor din surse externe (camere de luat vederi, scanere);

– inserarea și formatarea textelor;

– adăugarea conturilor;

– reglarea nivelurilor de transparență și umbrire;

– aplicarea efectelor artistice (mozaic, sticlă udă, cristalizare, textură, estompare).

8. Operare cu obiectele grafice: crearea, editarea, clonarea, ordonarea, gruparea/de-gruparea/regruparea.

9. Operare cu straturi: adăugarea/editarea; selectarea, gruparea și legarea; mutarea, stivuirea și blocarea.

10. Prelucrări avansate a textelor: scalarea, rotirea, aplicarea efectelor geometrice, aplicarea efectelor artistice.

11. Aplicarea efectelor speciale personalizate: geometrice tridimensionale, de estompare, de filtrare, de contur, de textură, de claritate.

12. Stocarea și difuzarea imaginilor: arhivarea, pregătirea pentru tipărire, organizarea în albume locale și/sau pe Internet.

Procesarea imaginilor vectoriale

Competențele necesare pentru procesarea imaginilor vectoriale pot fi formate și dezvoltate studiind următoarele teme:

1. Noțiunile de bază ale graficii orientate pe obiecte: punct/nod, linii drepte, curbele Besier, rezoluție, dimensiuni, modele de culoare.

2. Spațiul de lucru al editorului grafic vectorial și personalizarea acestuia.

3. Instrumente pentru desen: trasator de linii, convertor de linii, patrulatere, poligoane, primitive grafice, constructor caroiaje.

4. Gestionarea proprietăților instrumentelor de desen.

5. Instrumente pentru selecție și editare: selector obiecte/noduri, foarfeci, pipetă, cuțit.

6. Prelucrarea textelor.

7. Crearea și editarea obiectelor.

8. Rasterizarea imaginilor vectoriale.

Temele propuse pentru studiere pot fi găsite în sistemele de asistență a editoarelor de imagini cu rastru și a celor de imagini vectoriale, în resursele educaționale deschise, postate pe Internet. În procesul creării, editării și difuzării imaginilor, o atenție deosebită se va acorda respectării regulilor de siguranță pe Internet, eticii digitale, dreptului de autor.

Activităţile și produsele școlare recomandate

Exerciții de:

– identificare a resurselor educaționale deschise pentru studierea graficii digitale;
– personalizare a spațiului de lucru al editoarelor grafice;
– identificare a elementelor și proprietăților imaginilor propuse de profesor și ale celor descărcate de pe Internet;
– identificare a obiectelor grafice din componența imaginilor și a proprietăților acestora;
– desenare/redesenare a imaginilor cu rastru și a celor vectoriale;
– aplicare a efectelor asupra imaginilor și a obiectelor din componența acestora;
– import și export al imaginilor;
– conversie a formatelor imaginilor digitale;
– vectorizare a imaginilor cu rastru;
– rasterizare a imaginilor vectoriale;
– creare a albumelor și arhivelor locale și pe Internet.

Studii de caz:

– Impactul artistic și social-economic al imaginilor digitale.
– Evoluția aparatelor fotografice digitale.
– Evoluția editoarelor grafice în distribuție liberă.
– Evoluția editoarelor grafice proprietare.
– Platforme Web pentru imagini digitale.
– Servicii Web pentru albume digitale.
– Mijloacele grafice de punere în evidență a relevanței și importanței ideilor și stărilor de spirit.
– Transformări artistice ale formelor în procesul creării și editării imaginilor.
– Valoarea artistică și estetică a imaginilor digitale frecvent întâlnite în viața cotidiană.
– Realismul și abstracționismul imaginilor digitale.

– “Furt” și “inspirație” în imaginile digitale.
– Protejarea dreptului de autor asupra imaginilor digitale.
– Grafica pe calculator și etica digitală.

Proiecte:
– Expoziții tematice de grafică digitală.
– Albume digitale: școala mea, localitatea mea, prietenii mei, evenimente festive, evenimente de divertisment, manifestări sportive.
– Texte artistice: citate remarcabile, citate cu evidențierea mesajului principal, mottouri.
– Profiluri de personalități ilustre.
– Arborele genealogic al uneia dintre personalitățile celebre, originare din localitatea natală.
– Pliante, postere tematice, afișe (drepturile copilului, activismul civic, voluntariatul, modul sănătos de viață, ecologia, temă liberă).
– Colecții digitale de indicatoare, semne (rutiere, de securitate a muncii, de avertizare, de informare).
– Planșe pentru diverse discipline școlare.

4.3. Fotografia digitală

Termenul de *fotografie* are o semnificație triplă. Prin el înțelegem: a) tehnica de creare a imaginilor sub acțiunea luminii; b) imaginea obținută prin această tehnică și c) o ramură a artei grafice care folosește tehnica respectivă.

Inițial tehnica fotografică se baza pe utilizarea unor materiale fotosensibile, însă, odată cu dezvoltarea tehnologiilor informației și comunicațiilor, imaginile sunt obținute cu ajutorul unor matrice, formate din celule fotosensibile, denumite *pixeli*.

Aparatele digitale moderne de fotografiat utilizează matrice ce conțin zeci și sute de milioane de pixeli și stochează imaginile respective în fișiere grafice. Prelucrările simple ale fotografiilor digitale sunt efectuate de calculatoarele încorporate în aparatele de fotografiat, însă prelucrările mai sofisticate se execută cu ajutorul editoarelor grafice ce rulează pe calculatoarele personale.

Menționăm faptul că fotografiile digitale pot fi obținute nu doar cu aparatele de fotografiat propriu-zise, dar și cu ajutorul camerelor de luat vederi, încorporate în telefoanele mobile și calculatoarele personale de tip tabletă. Indiscutabil, calitatea fotografiilor digitale depinde în primul rând de echipamentele utilizate, cea a fotografiilor realizate cu telefoanele mobile, cu calculatoarele de tip tabletă și cu aparatele de fotografiat destinate amatorilor fiind mai joasă decât a fotografiilor digitale realizate cu aparatele destinate profesioniștilor. Mai mult decât atât, în cazul artelor vizuale, unii fotografi profesioniști preferă să utilizeze aparatele clasice cu film, imaginile developate fiind ulterior scanate și prelucrate cu ajutorul editoarelor grafice.

Evident, fotografiile digitale pot fi afișate pe ecranele echipamentelor digitale, proiectate cu ajutorul aparatelor multimedia, tipărite și imprimate pe diverși purtători statici, cum ar fi hârtia, țesăturile, foliile, obiectele de ceramică etc.

Dezvoltarea competențelor de creare a fotografiilor digitale se bazează pe studierea următoarelor teme:

1. Noțiunile de bază ale fotografiei digitale: dimensiune, rezoluție, putere de rezoluție, modele de culoare, formate grafice.

2. Clasificarea și caracteristicile tehnice ale aparatelor digitale de fotografiat.

3. Structura și funcționarea aparatelor digitale de fotografiat.

4. Echipamente pentru aparatele digitale de fotografiat: obiective, filtre, blițuri, stative, instrumente fotometrice, accesorii.

5. Factorii de calitate: compoziția, focalizarea, profunzimea, expunerea.

6. Tehnici de fotografiere digitală: portrete și oameni, natură moartă, reportaj, arhitectură, peisaje, sport, animale, subiecte abstracte.

7. Produse-program pentru procesarea tehnică și artistică a fotografiilor digitale.

8. Tehnici de prelucrare a fotografiilor digitale. Transformări: de format, geometrice, coloristice, artistice.

9. Stocarea și difuzarea fotografiilor digitale: arhivarea, pregătirea pentru tipărire, organizarea în albume locale și/sau pe Internet.

Exerciții de:

– identificare a resurselor educaționale deschise pentru studierea fotografiei digitale;
– identificare a proprietăților fotografiilor digitale propuse de profesor și ale celor descărcate de pe Internet;
– identificare a părților componente și a controalelor aparatelor digitale de fotografiat;
– fotografiere și gestionare ulterioară a fișierelor cu ajutorul controalelor aparatului digital de fotografiat;
– calculare a cantității de informație în fotografiile digitale;
– calculare a rezoluțiilor posibile ale fotografiilor;
– redimensionare a fotografiilor digitale;
– decupare și montare a fragmentelor fotografiilor digitale;
– modificare a modelelor coloristice, a contrastului, a curbelor de intensitate a culorilor primare;
– determinare a corelației dintre dimensiunile de ecran și dimensiunile fizice ale fotografiei după tipar;
– aplicare a efectelor de vitraliu, cristalizare, vânt, ploaie, poster;
– tehnoredactare a fotografiilor digitale;
– prelucrare artistică a fotografiilor digitale: peisaj, portret individual, portret în grup, natură moartă, reportaje, călătorii, obiecte arhitecturale, competiții sportive, animale, artă fotografică abstractă;
– identificare a însemnelor ce declară dreptul de autor;

– explicare a regulilor ce vizează respectarea dreptului de autor;
– utilizare a licențelor pentru distribuție.

Studii de caz:
– Istoria fotografiei.
– Istoria fotografiei digitale.
– Impactul artistic și social-economic al fotografiilor digitale.
– Evoluția aparatelor fotografice digitale.
– Specificul aparatelor fotografice digitale reflex (*DSLR – Digital Single-Lens Reflex*).
– Factorii ce influențează calitatea fotografiilor digitale.
– "Furt" și "inspirație" în fotografierea digitală.
– Protejarea dreptului de autor asupra fotografiilor digitale.
– Fotografierea digitală și etica Internetului.
– Top zece cele mai reușite portrete fotografice.
– Animalele în natură.
– Cele mai exotice flori.
– Reportajele fotografice ce impresionează.
– Carnavalurile în imagini.

Proiecte:
– Reportaje fotografice de la evenimentele festive, concursurile școlare, evenimentele artistice, evenimentele de divertisment, activitățile civice, activitățile de voluntariat.
– Expoziții tematice de fotografii digitale.
– Albume digitale: viața școlii, viața clasei, de vacanță, satul/orașul meu, portrete, natură moartă, reportaje, călătorii, obiecte arhitecturale, peisaje, competiții sportive, animale, artă fotografică abstractă.
– Colecții de fotografii digitale pentru muzeul școlii, satului/orașului natal.
– Colecții de fotografii digitale didactice pentru diverse discipline școlare.
– Colecții de fotografii digitale pentru profilurile individuale de pe Internet.

Recomandăm elevilor să descarce de pe Internet cele mai potrivite pentru fiecare dintre ei resurse educaționale deschise din domeniul fotografiei digitale, manuale de utilizare a echipamentelor fotografice digitale și a produselor-program de grafică digitală. Elevii sunt încurajați să respecte cu strictețe regulile de siguranță pe Internet, etica digitală, dreptul de autor.

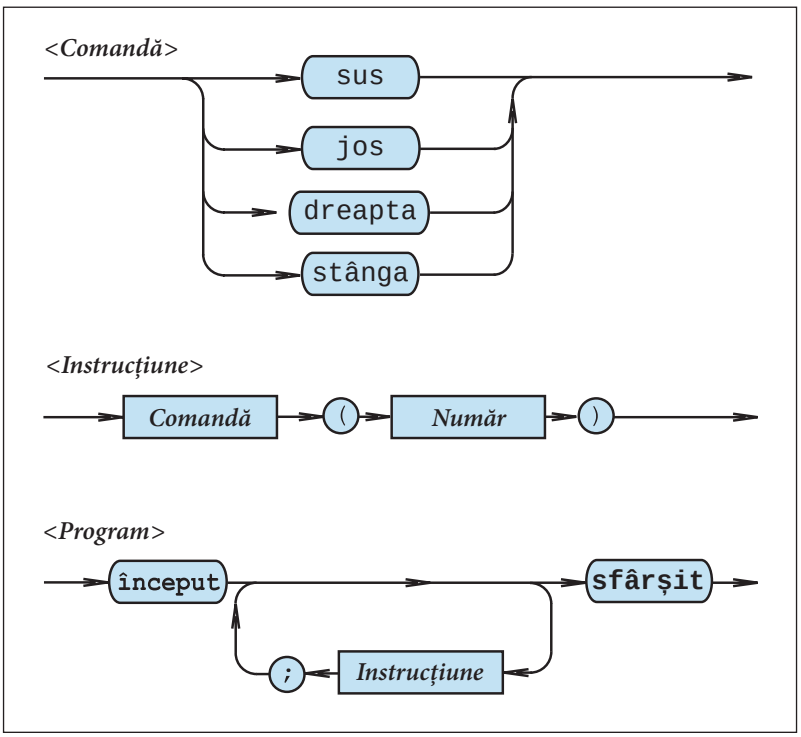
Răspunsuri la testele de autoevaluare

Testul nr. 1

Pascal

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`	*f)*	`-984.52`	*k)*	`-3628.297e12`
b)	`265`	*g)*	`-523`	*l)*	`-38.00001`
c)	`23.4635`	*h)*	`+28`	*m)*	`35728.345452e-8`
d)	`+0.000001`	*i)*	`+28000000`	*n)*	`24815`
e)	`6.1532e-5`	*j)*	`614.45e-12`	*o)*	`-296.0020001`

8.

a)	6124,485;	*f)*	$-0{,}03428 \cdot 10^{-8}$;	*k)*	2005;
b)	+18,315;	*g)*	232847,5213;	*l)*	$+23{,}08 \cdot 10^{-5}$;
c)	$-218{,}034 \cdot 10^{-3}$;	*h)*	$-0000012 \cdot 10^{+2}$;	*m)*	–17502;
d)	193526;	*i)*	18,45;	*n)*	+1;
e)	$1000{,}01 \cdot 10^{23}$;	*j)*	$623{,}495 \cdot 10^{-6}$;	*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.

11. Erorile pot fi depistate cu ajutorul compilatorului mediului de dezvoltare a programelor PASCAL.

C++

1. *a*, *c*, *d* – corect; *b*, *e*, *f*, *g* – greşit.

2. Coincide cu desenul din itemul 2 de la pagina 185, cazul limbajului PASCAL.

3. *a*, *c*, *e*, *f* , *h*, *i*, *m*, *o* – corect; *b*, *d*, *g*, *j*, *k*, *l*, *n* – greşit.

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`	*6)*	`DistantaParcursa`
2)	`x2`	*7)*	`ListaElevilor`
3)	`Delta`	*8)*	`Vocala`
4)	`UnghiulAlfa`	*9)*	`Pixel`
5)	`UnghiulBeta`	*10)*	`Culoare`

7.

a)	`3.14`	*f)*	`-984.52`	*k)*	`-3628.297e12`
b)	`265`	*g)*	`-523`	*l)*	`-38.00001`
c)	`23.4635`	*h)*	`+28`	*m)*	`35728.345452e-8`
d)	`+0.000001`	*i)*	`+28000000`	*n)*	`24815`
e)	`6.1532e-5`	*j)*	`614.45e-12`	*o)*	`-296.0020001`

8.

a)	6124,485;	*f)*	$-0{,}03428 \cdot 10^{-8}$;	*k)*	2005;
b)	+18,315;	*g)*	232847,5213;	*l)*	$+23{,}08 \cdot 10^{-5}$;
c)	$-218{,}034 \cdot 10^{-3}$;	*h)*	$-0000012 \cdot 10^{+2}$;	*m)*	–17502;
d)	193526;	*i)*	18,45;	*n)*	+1;
e)	$1000{,}01 \cdot 10^{23}$;	*j)*	$623{,}495 \cdot 10^{-6}$;	*o)*	$-46341{,}2 \cdot 10^{-6}$.

9. Directive: `#include`
Cuvinte-cheie: **`ussing`**, **`namespace`**, **`int`**, **`float`**, **`if`**, **`return`**
Simboluri speciale: `;`, `,`, `==`, `!=`, `&&`, `(`, `)`, `{`, `}`, `<`, `>`
Identificatori: `a`, `b`, `x`, `iostream`
Numere: `0`.
Şiruri de caractere: *`"Ecuatia are o singura radacina"`*,
`"Ecuatia are o multime infinita de radacini"`,
`"Ecuatia nu are sens"`.

10. *Linia 1* – directivă; *Linia 3* – funcţia principală; *Linia 5* – partea declarativă a funcţiei principale.

11. Erorile pot fi depistate cu ajutorul compilatorului din componenţa mediului de dezvoltare a programelor C++.

Testul nr. 2

Pascal

1. Prin *tip de date* se înţelege o mulţime de valori şi o mulţime de operaţii care pot fi efectuate cu valorile respective. Exemple de tipuri de date: `integer`, `real`, `char`. Valorile `1`, `2` şi `3` sunt de tip `integer`; valorile `1.0`, `2.0` şi `0.5e+07` sunt de tip `real`, iar valorile `'A'`, `'B'` şi `'+'` – de tip `char`.

2. Într-un program PASCAL mărimile sunt obiecte destinate pentru a reprezenta datele. Există două genuri de mărimi: *variabile* şi *constante*. Pe parcursul execuţiei 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. Mulţimea 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)`. Operaţiile care se pot face cu valorile întregi sunt: `+`, `-`, `*`, **`mod`**, **`div`** etc.

5. Erorile de depăşire vor apărea atunci când `x*y>MaxInt`. De exemplu, pentru `MaxInt=32767` (versiunea Turbo PASCAL 7.0) vor apărea erori de depăşire, dacă utilizatorul va tasta pentru `x` şi `y` valori mai mari ca `200`.

6. Mulţimea de valori ale tipului de date în studiu este formată din numerele reale care pot fi reprezentate pe calculatorul-gazdă al limbajului. Operaţiile care se pot face cu valorile reale sunt `+`, `-`, `*`, `/` (împărţirea) etc. Aceste operaţii sunt în general aproximative din cauza erorilor de rotunjire.

7. Erorile de depăşire vor apărea atunci când rezultatul operaţiei `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 $-1{,}7 \cdot 10^{38}$, ..., $+1{,}7 \cdot 10^{38}$. Prin urmare, erori de depăşire vor apărea atunci când utilizatorul va tasta pentru numerele `x` şi `y` valori mai mari, de exemplu 10^{20}.

8. Tipul de date `boolean` include valorile de adevăr `false` şi `true`. Operaţiile predefinite ale tipului de date `boolean` sunt: **`not`**, **`and`** şi **`or`**.

9. Vezi *figura 2.1*.

10. Instrucţiunea `readln(p, q)` este greşită, întrucât 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` sunt: `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 de 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* sunt: `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 sunt permise operațiile relaționale;
c) pentru tipurile ordinale de date există funcțiile predefinite `pred` (predecesor) și `succ` (succesor).

18. Pe linii separate se va afișa:

```
Y   E   -6   10   1   3
```

19. Pe linii separate se va afișa:

```
1   0   2   true   false   true
```

20. Două tipuri sunt identice, dacă ele au fost definite cu același nume de tip.
Două tipuri sunt compatibile atunci când este adevărată cel puțin una dintre următoarele afirmații:
a) cele două tipuri sunt identice;
b) un tip este un subdomeniu al celuilalt tip;
c) ambele tipuri sunt 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;
```

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: **`int`**, **`double`**, **`char`**. Valorile `1`, `2` și `3` sunt de tip întreg **`int`**; valorile `1.0`, `2.0` și `0.5e+07` sunt de tip **`real double`**, iar valorile `'A'`, `'B'` și `'+'` – de tip **`char`**.

2. Într-un program C++ mărimile sunt obiecte destinate pentru reprezentarea datelor. Există două genuri de mărimi: *variabile* și *constante*. Pe parcursul execuției programului, valoarea oricărei variabile poate fi schimbată, pe când valorile constantelor nu pot fi schimbate.

3. `i`, `j` – variabile de tip **`int`**; a, b, c – variabile de tip **`double`**; s – variabilă de tip **`char`**; p – variabilă de tip **`bool`**; 5, 9 – constante de tip **`int`**; `1.0`, `1.0e-01`, `-2.001` – constante de tip **`double`**; `'A'` – constantă de tip **`char`**; `true` – constantă de tip **`bool`**.

4. Mulțimea de valori ale tipului de date **`int`** este formată din numerele întregi care pot fi reprezentate pe calculatorul-gazdă al limbajului. Valoarea maximă poate fi referită prin constanta `INT_MAX`, cunoscută oricărui program C++. De obicei, valoarea minimă, admisă de tipul de date în studiu, este – (`INT_MAX+1`). Operațiile care se pot face cu valorile întregi sunt: +, -, *, %, / etc.

5. Erorile de depășire vor apărea atunci când `x*y> INT_MAX`. De exemplu, pentru compilatoarele care alocă 2 octeți (16 biți), `INT_MAX=32767`, erori de depășire vor apărea dacă utilizatorul va tasta pentru x și y valori mai mari ca `200`. Pentru compilatoarele care alocă 4 octeți (32 de biți), `INT_MAX=2147483647`, erori de depășire vor apărea dacă utilizatorul va tasta pentru x și y valori mai mari ca 46350.

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 sunt +, -, *, / (împărțirea) etc. Aceste operații sunt în general aproximative din cauza erorilor de rotunjire.

7. Erorile de depășire vor apărea atunci când rezultatul operației `x*y` nu se va încadra în domeniul de valori al tipului de date **`double`**. Pentru compilatoarele care alocă 8 octeți pentru tipul de date **`double`**, domeniul de valori al acestuia este $-1{,}7 \cdot 10^{308}$ ... $+1{,}7 \cdot 10^{308}$. Prin urmare, erori de depășire vor apărea atunci, când utilizatorul va tasta pentru numerele x și y valori mai mari, de exemplu 10^{160}.

8. Tipul de date **`bool`** include valorile de adevăr `false` și `true`. Operațiile predefinite ale tipului de date **`bool`** sunt: `!`, `&&` și `||`.

9. Vezi *figura 2.1*.

10. La executarea instrucțiunii `cin>>p>>q`, variabilelor de tip `boolean` li se pot atribui de la tastatură doar valorile numerice 0 (în loc de `false`) sau 1 (în loc de `true`).

11. Mulțimea valorilor tipului de date **`char`** include toate caracterele imprimabile, ordonate conform tabelului de cod *ASCII*. Cu aceste valori pot fi efectuate operațiile relaționale.

12.

```
//Programul RTA1
// Numerele de ordine ale cifrelor zecimale
#include <iostream>
using namespace std;
int main()
{
cout<<int('0')<<endl;
cout<<int('1')<<endl;
cout<<int('2')<<endl;
cout<<int('3')<<endl;
cout<<int('4')<<endl;
cout<<int('5')<<endl;
cout<<int('6')<<endl;
cout<<int('7')<<endl;
cout<<int('8')<<endl;
cout<<int('9')<<endl;
return 0;
}
```

13. Mulțimea de valori a oricărui tip de 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 ș.a.m.d. În limbajul C++ tipurile *enumerare* sunt tipuri întregi, iar identificatorii tipului *enumerare* se pot folosi la fel ca variabilele întregi. În limbajul C++ nu există funcții predefinite pentru determinarea directă a succesorului sau predecesorului. În expresii aritmetice, orice dată de tip *enumerare* este tratată ca un număr întreg, conversia către **`int`** fiind implicită, dar conversia unui întreg către un tip *enumerare* trebuie cerută explicit. Astfel, se va recurge la operații de modificare a tipului pentru a realiza trecerea la elementul succesor sau predecesor, utilizând așa-numiții operatori de conversie. De exemplu, fie `x` un întreg, prin `(char)x` (sau `char(x)`) vom modifica tipul lui `x` de la **`int`** la **`char`**.

14.

```
//Programul RTA2
//Numerele de ordine ale valorilor de tip enumerare
#include <iostream>
using namespace std;
int main()
{
 enum FunctiaOcupata {Muncitor, SefDeEchipa, Maistru,
  SefDeSantier, Director};
 enum StareaCivila {Casatorit, Necasatorit};
  cout<<Muncitor<<endl;
  cout<<SefDeEchipa<<endl;
  cout<<Maistru<<endl;
  cout<<SefDeSantier<<endl;
```

```
  cout<<Director<<endl;
  cout<<Casatorit<<endl;
  cout<<Necasatorit<<endl;
 return 0;
}
```

16. p – tip T1, care este un tip **char**. Ca valoare poate lua orice caracter imprimabil *ASCII*;
q – tip T2, care este un tip **int**. Ca valoare poate lua orice număr întreg care poate fi reprezentat pe calculatorul-gazdă al limbajului;
r – tip T3, care este un tip **char**. Ca valoare poate lua orice caracter imprimabil *ASCII*;
s – tip **char**. Ca valoare poate lua orice caracter imprimabil *ASCII*;
t – tip **int**. 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: **int**, **bool**, **char**, **enum**. Proprietățile comune:
a) fiecare valoare a unui tip ordinal are un număr de ordine, care poate fi aflat cu ajutorul operatorului de conversie `(int)x` sau `int(x)`, (x aparține unui tip ordinal);
b) asupra valorilor oricărui tip ordinal de date sunt permise operațiile relaționale.

18. Pe linii separate se va afișa:

```
Y   E   -6   10   1   3
```

19. Pe linii separate se va afișa:

```
1   0   2   1   0   1
```

20. Două tipuri sunt identice, dacă ele au fost definite cu același nume de tip sau atunci când sunt definite cu nume diferite, dar aceste nume sunt echivalente prin tranzitivitate.
Spre deosebire de limbajul PASCAL sau C, limbajul C++ nu are un concept de tipuri compatibile.
Totodată, compilatoarele verifică compatibilitățile de tip în următoarele cazuri:
– la atribuire;
– la transmiterea de parametri;
– la calcularea valorilor expresiilor.
Un tip de date este anonim, dacă el este definit implicit într-o declarație de variabile.

21. Tipuri identice: *a)* T1, T2, T3, T4, T5 și `int`; *b)* T6,T7 și T8; *c)* T9, T10 și **char**;
Tipuri anonime: (`Alfa`, `Beta`, `Gama`, `Delta`).

22. `Alfa` – **int**; `Beta` – **float**; `Indicator` – **bool**; `Mesaj` – șir de caractere; `Semn` – **char**; `Inscris` – șir de caractere.

23.

```
const float a = 3.14, b = 9.8;
int i, j;
float x, y;
```

Testul nr. 3

Pascal

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
    read(c);
    if (c<>'*') and (c<>'#') then n:=n+1;
  until c='#';
  writeln('Numarul de cifre n=', n);
readln;
end.
```

C++

1.

a) `(a+b)-2*a*b;`

b) `6*pow(a,2)+15*a*b-13*pow(b,2);`

c) `(a+b)*(a-b);`

d) `2*Alfa*Beta-5*Pi*r;`

e) `Pi*pow(r,2)+Alfa*pow(Beta,2);`

f) `x && y || x && 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*, *d*, *e*, *f* – corect; *b* – greșit.

4. *a*) 14; *b*) 6; *c*) –4; *d*) 0; *e*) 1; *f*) 1.

5.

a) **int**;
b) **int**;
c) **double**;
d) **int**;
e) **int**;
f) **int**;
g) **bool**;
h) **char**.

6.

```
// Programul RTA3
#include <iostream>
using namespace std;
int main()
{
int i;
double x, y;
cout<<"Dati i="; cin>>i;
cout<<"Dati x="; cin>>x;
cout<<"Dati y="; cin>>y;
cout<<15*i*(x+y);
return 0;
}
```

7. *a*, *c*, *d* – corect; *b*, *e*, *f* – greșit.

8.

```
// Programul RTA4
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
double x, y;
cout<<"x="; cin>>x;
```

```
if (x>15) {y=9*x+3*pow(x,2);} else {y=3*x-5*sqrt(x+28);}
cout<<"y="<<y<<endl;
return 0;
}
```

9.

```
// Programul RTA5
#include <iostream>
using namespace std;
int main()
{
int i;
cout<<"Dati valoarea monedei: ";
cin>>i;
switch (i)
{
    case 1 : cout<<"un ban"; break;
    case 5 : cout<<"cinci bani"; break;
    case 10 : cout<<"zece bani"; break;
    case 25 : cout<<"douazeci si cinci de bani"; break;
    case 50 : cout<<"cicncizeci de bani"; break;
    default : cout<<"valoare inadmisibila";
}
  return 0;
}
```

10.

```
// Programul RTA6
#include <iostream>
using namespace std;
int main()
{
int n, i;
double s, p;
cout<<"n="; cin>>n;
s=0; p=1;
for (i=1; i<=n; i++)
{s=s+(double)1/i; p=p*(double)1/i;}
cout<< "s="<<s<<endl;
cout<<"p="<<p;
return 0;
}
```

11.

```
// Programul RTA7
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
int main()
{
double y, x, x1, x2, deltaX;
cout<<"x1="; cin>>x1;
cout<<"x2="; cin>>x2;
cout<<"deltaX="; cin>>deltaX;
cout<<setw(10)<<'x'<<setw(10)<<'y'<<endl;
x=x1;
while (x<=x2)
{
    if (x>=4) {y=2*sqrt(x+6);} else {y=3-abs(x);}
    cout<<setw(10)<<x<<setw(10)<<y<<endl;
    x=x+deltaX;
}
return 0;
}
```

12.

```
// Programul RTA8
#include <iostream>
using namespace std;
int main()
{
char c;     // caracterul citit de la tastatura
int n;  // numarul de cifre in mesaj
n=0;
cout<<"Dati mesajul:"<<endl;
do
{cin>>c;
if ((c!='*')&&(c!='#')) { n=n+1;}
}
while (c!='#');
cout<<"Numarul de cifre n="<<n<<endl;
return 0;
}
```

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*> ::= +|-
|*|/|=|<|>|]|[|,|(|)|:|;|^|.|@|{|}|$|#|<=|>=|
<> | := |..| <*Cuvânt-cheie*> | <*Simbol echivalent*>

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

5. <*Cuvânt-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 sfârşit de linie neconţinând acolade*> *)

Notă. Simbolurile terminale { şi } din formula (17) sunt 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 fişier*> ::= [**packed**] **file of** <*Tip*>

20. <*Tip referinţă*> ::= ^<*Tip*>

21. <*Tip articol*> ::= [**packed**] **record** <*Listă câmpuri*> [**;**] **end**

22. <*Listă câmpuri*> ::= <*Parte fixă*> [**;** <*Parte variante*>] | <*Parte variante*>

23. <*Parte fixă*> ::= <*Secţiune articol*> { **;** <*Secţiune articol*> }

24. <*Secţiune articol*> ::= <*Nume câmp*> {**,** <*Nume câmp*> } **:** <*Tip*>

25. <*Parte variante*> ::= **case** [<*Identificator*> **:**] <*Tip*> **of** <*Variantă*> {**;** <*Variantă*> }

26. <*Variantă*> ::= <*Constantă*> {**,** <*Constantă*> } **:** ([<*Listă câmpuri*>] [**;**])

27. <*Funcţie*> ::= <*Antet funcţie*>**;** <*Corp*> | <*Antet funcţie*>**;** <*Directivă*> |
function <*Identificator*> **;** <*Corp*>

28. <*Antet funcţie*> ::= **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 funcţie*> | <*Antet procedură*>

33. <*Instrucţiune*> ::= [<*Etichetă*> **:**] <*Instrucţiune neetichetată*>

34. <*Instrucţiune neetichetată*> ::= <*Atribuire*> | <*Apel procedură*> |
<*Instrucţiune compusă*> |
<*Instrucţiune* **if**> | <*Instrucţiune* **case**> |
<*Instrucţiune* **while**> | <*Instrucţiune* **repeat**> |
<*Instrucţiune* **for**> | <*Instrucţiune* **with**> |
<*Instrucţiune* **goto**> | <*Instrucţiune de efect nul*>

35. <*Atribuire*> ::= <*Variabilă*> **:=** <*Expresie*> | <*Nume funcţie*> **:=** <*Expresie*>

36. <*Apel procedură*> ::= <*Nume procedură*> [<*Listă parametri actuali*> |
<*Listă parametri de ieşire*>]

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

38. <*Parametru actual* > ::= <*Expresie*> | <*Variabilă*> | <*Nume funcţie*> | <*Nume procedură* >

39. <*Nume funcţie*> ::= <*Identificator*>

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

41. <*Listă parametri de ieşire*> ::= **(**<*Parametru de ieşire*> { **,** <*Parametru de ieşire*> }**)**

42. <*Parametru de ieşire*> ::= <*Expresie*> [**:**<*Expresie*> [**:** <*Expresie*>]]

43. <*Instrucţiune compusă*> ::= **begin** <*Instrucţiune*> { **;** <*Instrucţiune*> } **end**

44. <*Instrucţiune* **if** > ::= **if** <*Expresie booleană*> **then** <*Instrucţiune*>
[**else** <*Instrucţiune*>]

45. <*Instrucţiune* **case**> ::= **case** <*Expresie*> **of** [<*Caz*>{**;** <*Caz*>}] [**;**] **end**

46. <*Caz*> ::= <*Constantă*> {**,** <*Constantă*>} **:** <*Instrucţiune*>

47. <*Instrucţiune* **while**> ::= **while** <*Expresie booleană*> **do** <*Instrucţiune*>

48. <*Instrucţiune* **repeat**> ::= **repeat** <*Instrucţiune*> {**;**<*Instrucţiune*>}
until <*Expresie booleană*>

49. <*Expresie booleană*> ::= <*Expresie*>

50. <*Instrucţiune* **for**> ::= **for** <*Variabilă*> **:=** <*Expresie*> <*Pas*> <*Expresie*>
do <*Instrucţiune*>

51. <*Pas*> ::= **to** | **downto**

52. <*Instrucţiune* **with**> ::= **with** <*Variabilă*> {**,**<*Variabilă*>} **do** <*Instrucţiune*>

53. <*Instrucţiune* **goto**> ::= **goto** <*Etichetă*>

54. <*Instrucţiune de efect nul*> ::=

55. <*Variabilă*> ::= <*Identificator*> | <*Variabilă*> **(.** <*Expresie*> { **,** <*Expresie*> } **.)** |
<*Variabilă*>**.**<*Nume câmp*> | <*Variabilă*>

56. <*Nume câmp*> ::= <*Identificator*>

57. <*Expresie*> ::= <*Expresie simplă*>{<*Operator relaţional*> <*Expresie simplă*>}

58. <*Operator relaţional*> ::= **<** | **<=** | **=** | **>=** | **>** | **<>** | **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 funcţie*> | **not** <*Factor*> |
(<*Expresie*>**)**|<*Constructor mulţime*>

64. <*Apel funcţie*> ::= <*Nume funcţie*> [<*Listă parametri actuali*>]

65. <*Constantă fără semn*> ::= <*Număr fără semn*> | <*Şir de caractere*> | <*Identificator*> | **nil**

66. <*Constructor mulţime*> ::= **(.** [<*Specificare element*> { **,** <*Specificare element*> }] **.)**

67. <*Specificare element*> ::= <*Expresie*> [**..** <*Expresie*>]

Notă. Simbolurile terminale [şi] din formulele (17), (55) şi (66) sunt redate prin simbolurile echivalente, respectiv (. şi .).

Anexa 3. Compilarea și depanarea programelor PASCAL

După scrierea unui program PASCAL se efectuează editarea, compilarea şi depanarea lui.

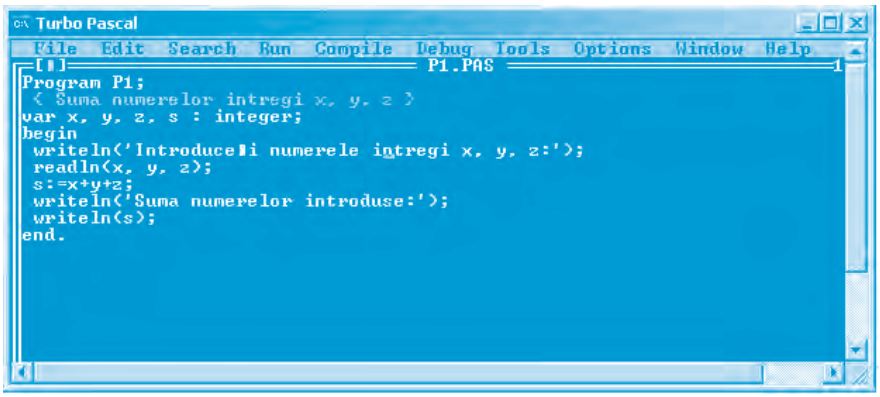
Editarea constă în introducerea programului în calculator. Programele introduse pot fi păstrate în fişiere text cu extensia ".pas".

Compilarea reprezintă procesul de traducere automată a programului scris în limbajul PASCAL într-un program scris în limbajul calculatorului. După compilare, programul în limbaj-maşină poate fi lansat în execuție sau păstrat într-un fişier executabil cu extensia ".exe".

Depanarea reprezintă procesul de depistare a greşelilor sintactice şi semantice din programul PASCAL şi 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 afişează pe ecran ferestre de aplicație, ferestre de dialog, ferestre de navigare, ferestre de explorare şi 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 şi Free PASCAL, instalate în majoritatea laboratoarelor de informatică din şcolile Republicii Moldova.

Meniul `File` (Fişier)

`New` (Nou) – creează o fereastră nouă în care utilizatorul poate introduce şi edita un text, de obicei un program PASCAL.

Open... **(Deschide...)** – citeşte fişierul specificat de utilizator şi afişează conținutul acestuia într-o fereastră nouă. În continuare, conținutul respectiv poate fi supus prelucrărilor dorite.

Save (**Salvează**) – salvează conținutul ferestrei curente în fişierul deschis anterior cu ajutorul comenzii **Open**. Dacă însă fereastra curentă a fost creată cu ajutorul comenzii **New**, se va crea un fişier nou, iar utilizatorul va fi invitat să propună o denumire pentru fişierul nou-creat.

Save as... (**Salvează ca...**) – salvează conținutul ferestrei curente într-un fişier nou. Utilizatorul este invitat să propună o denumire pentru fişierul nou-creat.

Print (**Imprimă**) – tipăreşte conținutul ferestrei curente la imprimantă.

Exit (**Ieşire**) – ieşirea din mediul integrat de dezvoltare a programelor. Înainte de a ieşi 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 **(Lipeşte)** – fragmentul de text din memoria-tampon este inserat în locul în care se află cursorul.

Clear **(Şterge)** – ştergerea fragmentului selectat de text fără a-l depune însă în memoria-tampon.

Meniul Search (Căutare)

Find **(Găseşte)** – caută, pornind de la poziția curentă a cursorului, fragmentul indicat de text.

Replace **(Înlocuieşte)** – înlocuieşte fragmentul selectat de text cu fragmentul indicat de utilizator.

Meniul Run (Rulare)

Run **(Derulează)** – compilează şi lansează în execuție programul PASCAL din fereastra curentă. Dacă programul conține erori sintactice, se va afişa un mesaj care indică tipul şi locul erorii.

Meniul Compile (Compilare)

Compile **(Compilează)** – compilează programul PASCAL din fereastra curentă, fără a-l lansa însă în execuție.

Build **(Construieşte)** – compilează programul PASCAL din fereastra curentă şi memorează rezultatul compilării într-un fişier executabil.

Meniul Windows conține comenzi ce asigură aranjarea şi comutarea între ferestre, iar meniul **Help** – comenzile ce permit accesul la manualul de asistență.

Anexa 4. Vocabularul limbajului C++

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*> ::= + | - | * | / | = | < | > |] | [| , | (|) | : | ; | ” | . | \ | { | } | $ | # | <= | >= | == | != | % | &

4. <*Cuvânt-cheie*>::= **auto** | **asm** | **bool** | **break** |**case** | **catch**| **char** | **const** | **continue** | **class** | **default** | **delete** | **do** | **double** | **else** | **enum** | **extern** | **float** | **for** | **friend** | **goto** | **if** | **inline** | **int** | **long** | **namespace** | **new** | **operator** | **private** | **public** | **protected** | **plate** | **register** | **return** | **short** | **signed** | **sizeof** | **static** | **struct** | **switch** | **typedef** | **union** | **unsigned** | **using** | **void** | **volatile** | **virtual** | **while**

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

6. <*Directivă*> ::= # <*Cuvânt-cheie*> <*Fișier sursă*>| # <*Cuvânt-cheie*> <*Identificator*> [<*Număr*>|<*Șir de caractere*>]

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

8. <*Semn*> ::= + | –

9. <*Număr întreg*> ::= [<*Semn*>] <*Întreg fără semn*>

10. <*Factor scală*> ::= <*Număr întreg*>

11. <*Număr real*> ::= <*Număr întreg*> e <*Factor scală*> |
<*Număr întreg*>.<*Întreg fără semn*> [e <*Factor scală*>]

12. <*Număr*> ::= <*Număr întreg*> | <*Număr real*>

13. <*Șir de caractere*> ::= “<*Element șir*> { <*Element șir*> } ”

14. <*Element șir*> ::= “ ”| <*Orice caracter imprimabil*>

15. <*Etichetă*> ::= <*Identificator*> <:>

16. <*Comentariu*> ::= // <*Orice secvență de caractere și sfârșit de linie* > | /* < *Orice secvență de caractere* >*/

Anexa 5. Compilarea și depanarea programelor C++

Programul este un fişier executabil binar. Fişierul text ce conține programul scris într-un limbaj de programare se numeşte ***fişier cod sursă***, sau simplu ***sursă***, de exemplu sursă C++, sursă Java etc. Procesul de scriere a unui fişier cod sursă se mai numeşte *scriere de cod sursă*, sau *scriere de cod*.

După scrierea unui program C++ se efectuează editarea, compilarea, editarea legăturilor şi execuția lui.

Editarea constă în introducerea programului în calculator. Programele introduse pot fi păstrate în fişiere text cu extensia *.cpp*.

Compilarea reprezintă procesul de traducere automată a programului scris în limbajul C++ într-un program scris în limbajul calculatorului (cod maşină). Procesul de compilare este realizat cu ajutorul unui compilator. În cazul limbajului C++, în prima fază a compilării este invocat *preprocesorul*. Acesta recunoaşte şi analizează mai întâi *directivele* de preprocesare. Ulterior se verifică codul sursă pentru a constata dacă acesta respectă sintaxa şi semantica limbajului. Dacă există erori, acestea sunt semnalate utilizatorului. Utilizatorul trebuie să corecteze erorile modificând programul sursă. Abia apoi codul sursă este tradus în codul maşină, binar, propriu calculatorului. Acest cod binar este numit *cod obiect* şi este memorat într-un alt fişier, numit *fişier obiect*. Fişierul obiect va avea, de obicei, acelaşi nume cu fişierul sursă şi extensia *.obj*.

Editarea legăturilor. După ce programul sursă a fost tradus în programul obiect, el va fi supus operației de editare a legăturilor. Scopul acestei operații este de a obține o formă finală a programului, în vederea execuției acestuia. Editorul "leagă" modulele obiect, execută referințele către funcțiile externe şi rutinele din biblioteci şi produce un cod executabil, memorat într-un alt fişier, numit *fişier executabil* cu acelaşi nume, cu extensia *.exe*.

Execuția. Lansarea în execuție constă în încărcarea programului executabil în memorie şi rularea acestuia.

De obicei, operațiile indicate mai sus se efectuează cu ajutorul unor programe speciale, denumite *medii integrate de dezvoltare* (IDE – *Integrated Development Environment*).

În mod obişnuit, în laboratoarele de informatică din țara noastră se utilizează mediul integrat de programare **Code::Blocks**, care poate fi descărcat de pe Internet folosind adresa http://www.codeblocks.org/. Instalarea acestui mediu de dezvoltare a programelor se face relativ simplu, urmând paşii indicați în manualul de instalare.

În general, mediile de dezvoltare a programelor oferă utilizatorilor următoarele instrumente informatice:

1) Editoare de text, necesare pentru scrierea programelor (codurilor sursă), adică de creare a fişierelor cu extensia `.c` sau `cpp` (respectiv, în limbajele C sau C++).

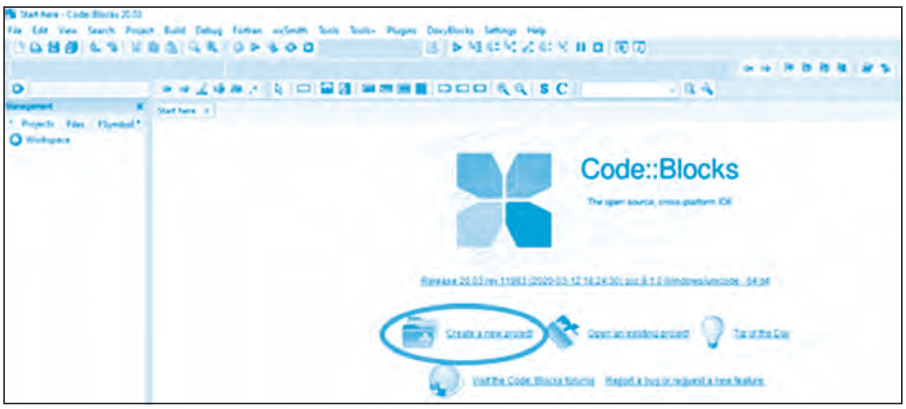
2) Compilatoare, necesare pentru traducerea codurilor sursă în fişiere compilate. Pentru limbajele C/C++ există mai multe compilatoare, cel mai utilizat fiind `gcc` (*GNU C Compiler*).

3) Biblioteci – fişiere ce conţin descrierea tipurilor de date şi a funcţiilor frecvent utilizate, de exemplu a celor destinate citirii şi afişării datelor (fişierul *iostream*). De asemenea, utilizatorii pot crea şi partaja între ei propriile biblioteci.

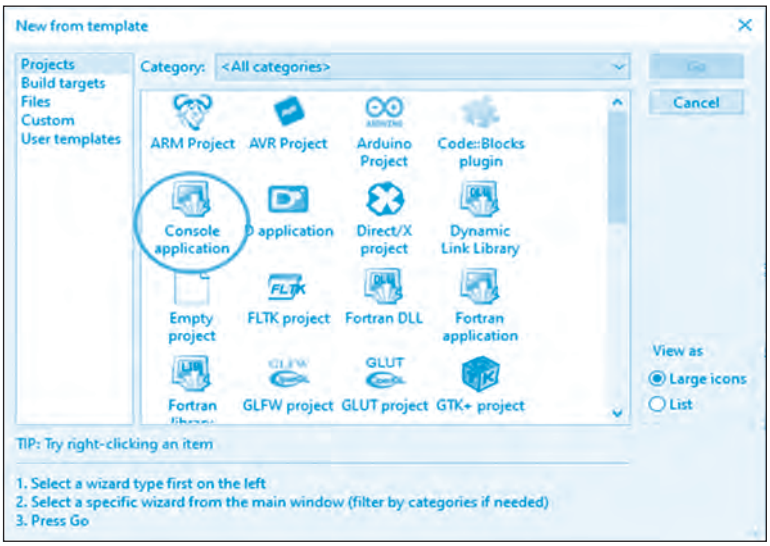
Elaborarea unui program C++ cu ajutorul mediului de dezvoltare a programelor **Code::Blocks** presupune parcurgerea următorilor paşi:

Pasul 1. Lansarea mediului de programare. Se execută lansând comanda respectivă din meniul **Start** al sistemului de operare sau efectuând clic pe pictograma aplicaţiei **Code::Blocks**.

Pasul 2. Crearea unui proiect nou. Selectaţi în fereastra apărută imediat după lansarea programului opţiunea **Create a new project**:

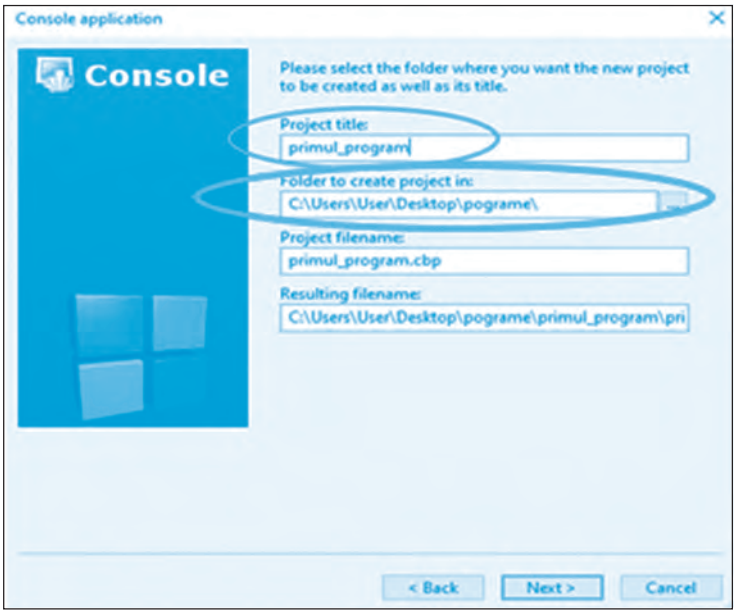


Pasul 3. Selectarea tipului aplicaţiei. În general, mediul de dezvoltare a programelor **Code::Blocks** oferă utilizatorilor posibilitatea de a crea mai multe tipuri de aplicaţii, de exemplu Consolă (în engleză *Console Application*), Bibliotecă (*Dynamic Link Library*), *Arduino* (programe pentru roboţi) etc. Evident, în cazul nostru, se va selecta opţiunea *Console Application*:

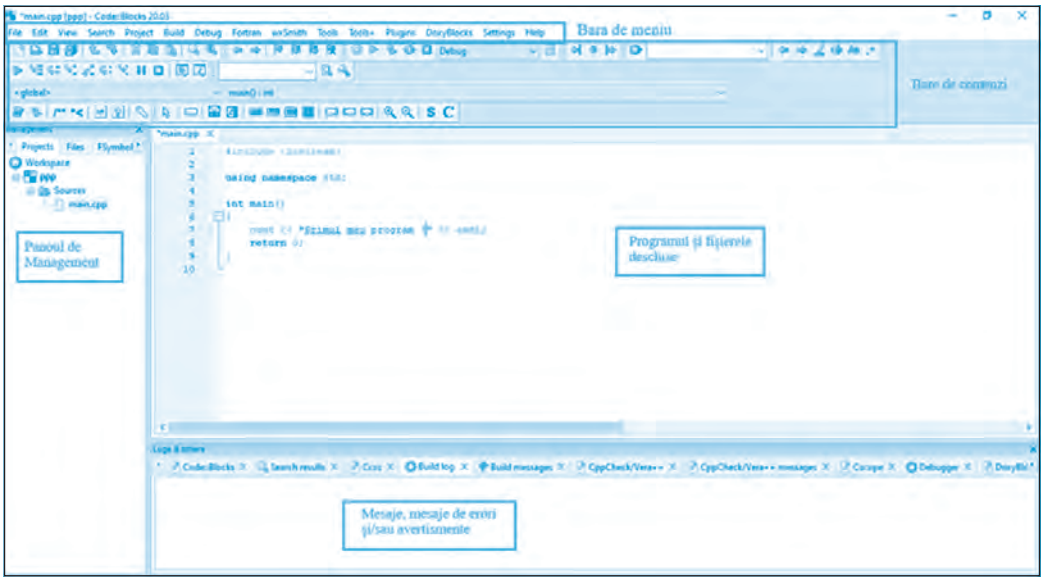


Pasul 4. Selectarea limbajului în care va fi scris programul de elaborat. Desigur, în cazul nostru trebuie selectat limbajul C++.

Pasul 5. Stabilirea denumirii proiectului şi a locaţiei în care el va fi salvat. Pe captura de ecran de mai jos proiectul în curs de elaborare va avea denumirea `primul_program` şi va fi salvat în dosarul `C:\Users\User\Desktop\programe\`:



Pasul 6. Editarea programului. Editarea se efectuează în cadrul ferestrei principale a mediului de dezvoltare a programelor:

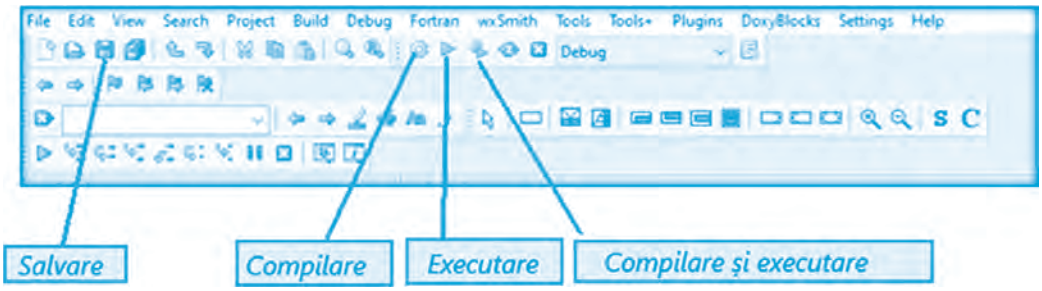


Această fereastră are o structură-tip şi cuprinde:

- Bara de meniuri, care conține mai multe meniuri, cele utilizate mai des în procesul de studiere a limbajului C++ fiind **File**, **Edit**, **Build** (Construiți), **Debug** (Depanați).
- Bare ce conțin comenzile frecvent utilizate, simbolizate prin pictograme;
- Panoul de management, în care este afişată structura arborescentă a proiectului;
- Fereastra de editare, în care sunt redactate programele în curs de elaborare;
- Fereastra ce conține file în care sunt afişate mesaje de eroare, valorile curente ale variabilelor, instrucțiunile executate în regimul pas-cu-pas. Fila ce va fi afişată se selectează cu ajutorul *taburilor* (semnelor de carte) din partea de sus a acestei ferestre.

Accentuăm faptul că utilizatorul poate personaliza fereastra principală a programului, folosind în acest scop comenzile din meniul **View**.

Pasul 7. Compilarea şi lansarea programului în execuție. Comenzile destinate compilării şi lansării programelor în execuție sunt grupate în meniul **Build**, iar cele frecvent utilizate sunt simbolizate prin pictogramele sugestive din bara respectivă:

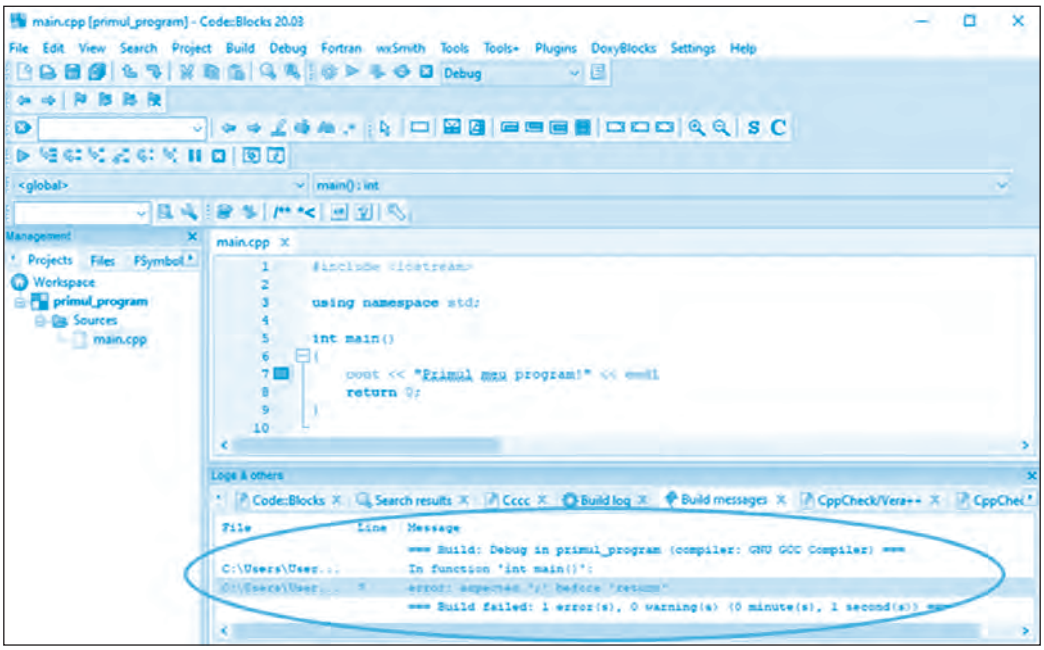


Rezultatul execuției programului va fi afişat pe ecran:

```
C:\Users\User\Desktop\pograme\primul_program\bin\Debug\primul_program.exe
Primul meu program!

Process returned 0 (0x0)   execution time : 0.089 s
Press any key to continue.
```

Dacă programul supus compilării conține erori sintactice, ele vor fi marcate în sursă şi explicate prin mesaje de eroare:



Evident, după corectarea erorilor, programul trebuie compilat din nou şi iar lansat în execuție.

Pasul 8. Depanarea programului. Cu excepția unor programe foarte simple, programele în curs de elaborare conțin nu doar erori sintactice, dar şi erori de logică. Mediile de dezvoltare a programelor nu pot depista în mod automat astfel de erori, însă ele oferă utilizatorului instrumente ce facilitează acest proces.

Pentru a depista eventualele erori de logică, programatorul trebuie să pregătească un set de teste. Fiecare dintre aceste teste conține datele de intrare şi răspunsurile-etalon. Rulând programul elaborat pe fiecare dintre teste, utilizatorul compară răspunsurile furnizate de program cu răspunsurile-etalon. În caz de necoincidențe, programatorul poate executa instrucțiunile programului în regim pas-cu-pas şi afişa valorile curente ale variabilelor din program.

Operațiile de depanare se realizează cu ajutorul următoarelor comenzi:

1) **Debug → Debugging windows → Watches** – afişarea valorilor curente ale variabilelor din programul în curs de execuție;

2) **Debug → Step into** – execuția programului în regim pas-cu-pas.

În general, depistarea erorilor de logică necesită o atenție deosebită din partea programatorului. Chiar şi în cazul unor programe elaborate în scopuri de învățare, scrierea acestora ocupă circa 30% din timpul de muncă al elevului, pe când depanarea programelor respective – celelalte 70%. Prin urmare, în procesul studierii informaticii, elevii vor distribui timpul destinat învățării în aşa mod, încât să aibă posibilitatea nu doar să scrie, dar şi să depaneze programele elaborate de ei.

Mai multe detalii despre mediul de dezvoltare a programelor pot fi găsite în manualul utilizatorului, ce poate fi accesat pe adresa http://www.codeblocks.org/user-manual