Recapitulare

Capitolul 1

- ❖ **Structura programelor Pascal**
- ❖ **Vocabularul limbajului**
- ❖ **Constante**
- ❖ **Tipuri standard de date**
- ❖ **Variabile**
- ❖ **Definirea constantelor**
- ❖ **Expresii**
- ❖ **Citirea şi scrierea datelor**
- ❖ **Instrucţiuni**
- ❖ **Probleme propuse**

Limbajul Pascal a fost conceput mai ales în scop didactic, urmărind predarea „artei programării calculatoarelor" conform principiilor programării structurate. Limbajul modelează natural concepte de bază în programare şi permite implementări eficiente.

1.1. Structura programelor Pascal

În descrierea unui program Pascal se diferenţiază acţiunile de efectuat şi descrierea datelor de prelucrat. În structura unui program Pascal intră:

- antetul programului (opţional) – conţine cuvântul rezervat **`Program`** urmat de numele programului;
- declaraţie (opţional) a numelor *unit*-urilor care se vor folosi în program, introdusă de cuvântul rezervat **`uses;`**
- parte obligatorie formată dintr-un bloc, urmat de caracterul punct (`'.'`).

Un *unit* este o colecţie de declaraţii de constante, tipuri, variabile şi subprograme. Un *unit* se compilează şi se utilizează în programele în care se doreşte folosirea acestor constante, tipuri, variabile şi subprograme. Folosirea *unit*-urilor permite scrierea unor programe mari care pot depăşi 64K. Dimensiunea unui program, ca şi cea a unui *unit* nu poate să depăşească 64K. *Unit*-urile standard, oferite de mediul Borland Pascal sunt: `System`, `Dos`, `Graph`, `Crt` etc. *Unit*-ul `System` este încorporat în toate programele Pascal fără să fie necesară includerea declaraţiei **`uses`**. Acest *unit* cuprinde procedurile şi funcţiile cel mai des folosite.

Blocul este cea mai importantă parte dintr-un program şi cuprinde o secţiune de declaraţii (opţional) şi o secţiune de program.

Declaraţiile se pot preciza în orice ordine, pot alterna şi se pot repeta. Acestea sunt structurate astfel:

- declaraţii de constante;
- definiţii de tipuri;
- declaraţii de variabile;
- declaraţii de etichete;
- declaraţii de subprograme.

Instrucţiunile care se vor executa vor fi scrise în secţiunea de program. Aici se pot apela subprograme declarate anterior (în partea de declaraţii sau în *unit*-uri).

Iată un exemplu de program Pascal:

```
{ Intre acolade vom scrie comentarii.                          }
{ Acestea sunt texte explicative care nu se compileaza.        }
{ Enuntul problemei:                                           }
{ Fie a, b si c lungimile laturilor unui triunghi oarecare.    }
{ Sa se calculeze perimetrul triunghiului si aria sa.          }
{ Observatie:                                                  }
{ a, b si c sunt numere naturale strict pozitive.              }

Program triunghi;
type pozitiv=1..MaxLongint;
var a,b,c,peri:pozitiv;
    sp,aria:Real;        { sp ==> semiperimetrul triunghiului }
Begin
  repeat
    Write('Introduceti lungimea laturii a-->'); ReadLn(a);
    Write('Introduceti lungimea laturii b-->'); ReadLn(b);
    Write('Introduceti lungimea laturii c-->'); ReadLn(c);
    if a*b*c=0 then
      WriteLn('Cel putin o latura are lungimea 0! Reluati!)
    else
      if (a<0) or (b<0) or (c<0) then
        WriteLn('Lungimi negative! Reluati!')
  until (a>0) and (b>0) and (c>0);
  if (a+b>c) and (a+c>b) and (b+c>a) then begin
    peri:=a+b+c;
    WriteLn('Perimetrul triunghiului este: ',peri);
    sp:=peri/2;
    aria:=Sqrt(sp*(sp-a)*(sp-b)*(sp-c));
    WriteLn('Aria triunghiului este: ',aria:10:2)
  end else
    WriteLn(a,' ',b,' ',c,' nu pot fi laturile unui triunghi!')
End.
```

În secțiunea **type** am definit tipul de date pozitiv cu valori mari, astfel putem calcula perimetrul şi aria unui triunghi, având lungimile laturilor numere pozitive care pot depăşi valoarea de două miliarde, dar trebuie să fie mai mici sau egale cu valoarea constantei MaxLongint (2147483647). Rezultatele le vom declara de tip Real, deoarece doar tipul acesta permite păstrarea rezultatului cu zecimale exacte.

Variabilele declarate în secțiunea **var** sunt a, b, c, peri şi au tipul pozitiv, iar sp şi aria sunt de tipul Real. Instrucțiunile din program apar între cuvintele rezervate **begin** şi **end** şi sunt separate prin punct şi virgulă (;). Textele cuprinse între acolade ({}) sunt comentarii.

1.2. Vocabularul limbajului

Setul de caractere al limbajului cuprinde: literele mari şi mici ale alfabetului englez, cifrele din baza 10 şi caracterele speciale: +, -, *, /, ^, <, >, (,), [,], {, }, . , ,, :, ;, #, $, @, _ şi spațiul (blancul).

Identificatorii sunt formați din litere, cifre şi caracterul '_' (liniuța de subliniere), dar primul caracter dintr-un identificator nu poate fi o cifră. Este indicat ca numele unui identificator să fie ales după semnificația lui în program. Lungimea numelui identificatorului nu influențează lungimea codului obiect. Compilatoarele Pascal nu fac distincție între literele mari şi literele mici.

Identificatorii min, max2, _Sum13a, prod sunt acceptați de compilatorul Pascal, în schimb a+b, 5a, z y nu sunt identificatori (primul exemplu este o expresie, al doilea începe cu cifră, iar al treilea conține un spațiu).

Identificatorii pot desemna *nume de programe*, *constante*, *tipuri*, *variabile*, *subprograme*, *parametri*, *unit*-uri etc. Identificatorii care au semnificație fixată, şi care nu pot fi folosiți decât în contextul în care sunt definiți în limbaj, alcătuiesc mulțimea *cuvintelor rezervate* ale limbajului.

Identificatorii creați şi referiți de utilizator trebuie declarați în program într-o linie sursă scrisă anterior referirii.

1.3. Constante

Tipurile de constante întâlnite în limbajul Borland Pascal sunt:

- constantele *întregi* (1977, 6, 3, -127 sunt numere întregi în baza 10; $afe2 este un număr în baza 16);
- constantele *reale* (-4.23, 0.0, 67E-2 este egal cu $67 \cdot 10^{-2} = 0.67$) ;
- *şirurile de caractere* pot fi:
  - un şir de caractere imprimabile, cuprins între apostrofuri: 'elev';
  - un şir de numere întregi de la 0 la 255, precedate fiecare de caracterul #, scrise fără separatori (concatenate);

  - un şir de caractere de control (care au codurile ASCII între 1 şi 31), exprimate fiecare printr-un caracter precedat de ^ (^A este **Ctrl+A**);
  - şiruri obținute prin concatenarea şirurilor definite anterior;
- constante simbolice.

Dacă dorim caracterul apostrof, atunci acesta trebuie dublat în şir: `'Windows''98'`.

Constantele simbolice sunt desemnate printr-un identificator. Dacă folosim aceste constante oferim programului claritate şi independență față de eventualele modificări ale acestor constante. O constantă simbolică *nu* poate fi modificată în timpul execuției programului.

1.4. Tipuri standard de date

Prin *dată* înțelegem o entitate asupra căreia poate opera calculatorul. Limbajele de nivel înalt folosesc tipurile de date pentru abstractizarea reprezentării unei date. *Tipul de dată precizează mulțimea de date şi operațiile care se pot efectua cu ea*.

Tipurile simple de date sunt nestructurate şi se împart în: *tipuri reale* şi *tipuri ordinale*. Un tip ordinal defineşte o mulțime finită şi ordonată de valori. Orice tip ordinal are o valoare minimă şi una maximă. Orice valoare de tip ordinal are o valoare *predecesor* (excepție făcând minimul) şi o valoare *succesor* (excepție făcând maximul).

Tipuri de date

<table>
<tr><td rowspan="6">Tipuri simple de date
(scalare, nestructurale)</td><td>Reale</td><td colspan="2"></td></tr>
<tr><td rowspan="5">Ordinale</td><td rowspan="2">Predefinite în unit-ul <code>System</code></td><td>Întregi
<code>Boolean</code>
<code>Char</code></td></tr>
<tr><td rowspan="2">Utilizator</td><td>Enumerat</td></tr>
<tr><td>Subdomeniu (interval)</td></tr>
<tr><td rowspan="5">Structurate</td><td>Tablou</td><td colspan="2" rowspan="5"></td></tr>
<tr><td>String</td></tr>
<tr><td>Articol (înregistrare)</td></tr>
<tr><td>Mulțime</td></tr>
<tr><td>Fişier</td></tr>
<tr><td>Tipul reper (pointer, referință)</td><td colspan="3"></td></tr>
</table>

În unit-ul `System` există definite funcțiile aplicabile unei valori de tip ordinal:

`Succ(x)`	Furnizează succesorul lui *x* (dacă *x* nu este maxim în tipul său) sau returnează prima valoare corespunzătoare tipului (dacă *x* coincide cu valoarea maximă din tipul său); rezultatul are acelaşi tip ca şi *x*.
`Pred(x)`	Furnizează predecesorul lui x (dacă *x* nu este minim în tipul său) sau ultima valoare corespunzătoare tipului (dacă *x* coincide cu valoarea minimă din tipul său); rezultatul are acelaşi tip ca şi *x*.
`Ord(x)`	Furnizează numărul de ordine pe care îl are *x* în mulţimea valorilor din tipul lui *x*; rezultatul funcţiei este de tip întreg.

Compararea valorilor ordinale *x* şi *y* se poate face cu ajutorul operatorilor relaţionali: = (egal), <> (diferit), <, >, <=, >=. Rezultatul comparării este de tip `Boolean`, adică poate avea valorile `true` sau `false`.

Prin reguli stabilite în limbaj, tipurile simple de date stau la baza alcătuirii tipurilor structurate.

1.4.1. Tipul `Boolean` (logic)

O variabilă de tip `Boolean` poate avea două valori: *adevărat* sau *fals*. Există mai mulţi operatori logici care pot fi aplicaţi asupra datelor booleene. Expresia va avea rezultat de tip `Boolean`:

Operatori binari:

`and` – şi logic

`or` – sau logic

`xor` – sau exclusiv (`x` **`xor`** `y` este `true` numai dacă `x<>y`)

`not` – negaţie logică (operator unar)

Valoarea `false` are asociat numărul de ordine 0, iar valoarea `true` are asociat numărul de ordine 1. Valorilor booleene li se pot aplica operatorii relaţionali, obţinând orice operaţie logică dorită. Operatorul = exprimă echivalenţa logică, iar operatorul <= exprimă implicaţia logică (`x<=y` are valoarea `false` dacă şi numai dacă *x* este `true` şi *y* este `false`).

1.4.2. Tipul `Char`

Setul extins de caractere ASCII reprezintă domeniul de valori pentru o dată de tip `Char`. O dată de tip `Char` se reprezintă pe un octet (8 biţi).

Exemple de date de tip `Char`: `'k'`, `#10` (caracterul având codul ASCII 10), `''''` (caracterul apostrof), `^A` caracterul care se obţine prin apăsarea tastelor **Ctrl+A**.

Funcţiile care sunt definite în *unit*-ul `System` pentru date de tip ordinal se pot utiliza şi pentru datele de tip `Char`.

- `Succ('b')` furnizează caracterul `'c'`

- `Pred('c')` furnizează caracterul `'b'`
- `Ord('C')` furnizează codul ASCII al caracterului `'C'`, codul returnat este 67
- `Chr(68)` furnizează caracterul `'D'`

Funcția `Chr(n)` returnează caracterul al cărui cod ASCII este egal cu numărul *n*. Numărul *n* trebuie să se încadreze în intervalul [0, 255]. Dacă valoarea lui *n* este mai mică decât 0, atunci funcția `Chr` generează caracterul care are codul ASCII egal cu $k \cdot 256 + n$, unde *k* este cel mai mic număr natural pentru care expresia $k \cdot 256 + n$ este mai mare sau egală cu 0. Dacă numărul *n* este mai mare decât 255, atunci funcția `Chr` returnează caracterul având codul ASCII egal cu restul împărțirii întregi a lui *n* la 256. O valoare de tip `Char` se poate scrie şi se poate citi. Datele de tip `Char` se pot compara utilizând operatorii relaționali.

1.4.3. Tipuri întregi

În Pascal există cinci tipuri întregi predefinite: `Shortint`, `Integer`, `Longint`, `Byte`, `Word`. Domeniul de valori şi modul de reprezentare se pot vedea în meniul `Help`.

Asupra datelor de tip întreg se pot aplica următorii operatori aritmetici: + (adunarea), – (scăderea), * (înmulțirea), **`div`** (împărțirea întreagă), **`mod`** (restul împărțirii întregi a primului operand la al doilea) şi operatorii pe biți: **`not`** (*negație*, fiecare bit egal cu 1 devine 0 şi fiecare bit egal cu 0 devine 1), **`and`** (*şi* logic face conjuncția bit cu bit între operanzi), **`or`** (*sau* logic face disjuncția bit cu bit între operanzi), **`xor`** (*sau exclusiv* între operanzi), **`shl`** (deplasează la stânga un număr de poziții binare egal cu valoarea celui de-al doilea operand; primii biți se pierd, în locul ultimilor biți se generează valoarea 0), **`shr`** (deplasează la dreapta un număr de poziții binare egal cu valoarea celui de-al doilea operand; ultimii biți ai primului operand se pierd, pe primele poziții se copiază valoarea 0).

Operația de deplasare la stânga cu o poziție este echivalentă cu o înmulțire a primului operand cu 2, iar deplasarea spre dreapta cu o poziție a unui număr pozitiv este echivalentă cu o împărțire întreagă la 2. Aceste operații de deplasare la stânga şi la dreapta a valorii biților se realizează mult mai rapid decât atunci când se utilizează operatorii * sau **`div`**.

Tipurile întregi sunt tipuri ordinale. Fiecare număr întreg are asociat un număr de ordine care coincide cu valoarea respectivă. Operanzilor de tip întreg li se pot aplica funcțiile `Succ`, `Pred`, `Ord` şi operatorii relaționali. Alte funcții care se pot aplica operanzilor de tip întreg sau real şi care returnează un rezultat de tip întreg sunt:

- `Abs(n)` returnează valoarea absolută a argumentului *n*;
- `Sqr(n)` determină pătratul argumentului *n*;
- `Trunc(n)` furnizează cel mai mare întreg mai mic sau egal cu *n* pentru $n > 0$, sau cel mai mic întreg, mai mare sau egal cu *n*, pentru $n < 0$;
- `Round(n)` furnizează cel mai apropiat întreg față de argumentul *n*.

1.4.4. Tipuri reale

Datele de tip real primesc valori reale aparţinând domeniului de valori corespunzător tipului respectiv. Calculele cu valori întregi modelează exact aritmetica întregilor, dar operaţiile cu date de tip real produc rezultate aproximative. Aceste aproximări provin din erorile de rotunjire generate de numărul finit de cifre semnificative folosite în reprezentarea valorilor reale. Tipurile reale care există în limbajul Pascal sunt: `Real`, `Single`, `Double`, `Extended`, `Comp`. Domeniul de valori şi modul de reprezentare se pot vedea în meniul `Help`. Valorile datelor de tip `Comp` sunt numere întregi cuprinse în intervalul [0, 2^{63}-1]. Dacă din doi operanzi cel puţin unul este de tip `Real`, atunci putem aplica operatorii: + (adunare), – (scădere), * (înmulţire), / (împărţire cu rezultat de tip real; acest operator se poate aplica şi operanzilor de tip întreg, însă rezultatul va fi de tip real). Dacă aplicăm operatorii relaţionali asupra unor operanzi de tip real vom obţine un rezultat de tip `Boolean`.

Tipurile reale nu sunt ordinale, prin urmare asupra valorilor reale nu putem aplica funcţiile `Ord`, `Succ`, `Pred`.

Valorile reale pot fi citite şi scrise. Principalele funcţii standard de lucru cu valori reale sau întregi se găsesc prezentate în meniul `Help` (`Abs`, `Sqr`, `Sin`, `Cos`, `Arctan`, `Ln`, `Exp`, `Sqrt`, `Int`, `Frac`, `Pi`).

1.4.5. Tipuri ordinale definite de utilizator

În limbajul Pascal utilizatorul poate să definească tipuri ordinale proprii: tipul *enumerat* şi tipul *subdomeniu*.

Tipul ***enumerat*** modelează cazurile în care exprimarea nu este numerică. De exemplu, atunci când lucrăm într-un program de tip agendă cu zilele săptămânii, programul va fi mult mai lizibil dacă se lucrează chiar cu denumirea zilelor săptămânii şi nu cu numerele de ordine corespunzătoare lor.

Un tip enumerat defineşte o mulţime ordonată de valori prin enumerarea identificatorilor care vor desemna valorile. Primul identificator desemnează cea mai mică valoare, cu numărul 0. Ceilalţi identificatori au numerele următoare, conform succesiunii lor. Prelucrarea datelor de acest tip se face rapid, deoarece valorile corespunzătoare se reprezintă în memorie prin numerele lor de ordine. Aceşti identificatori sunt trataţi în program la fel ca şi constantele.

Fiind un tip special, diferit de orice alt tip, nu putem utiliza identificatorii unui tip enumerat în definiţia altui tip enumerat.

Asupra datelor de tip enumerat nu se pot aplica operatori aritmetici. Datele de tip enumerat se comportă similar constantelor simbolice, de asemenea, nu se pot citi sau scrie. Iniţializarea datelor de tip enumerat se face prin atribuiri. Dacă, de exemplu, am declarat tipul:

```
type Saptamana=(luni,marti,miercuri,joi,vineri,sambata,duminica)
```

apelul `Ord(marti)` returnează `2`, `Pred(marti)` returnează `luni`.

Tipul ***subdomeniu*** se obține dintr-un tip ordinal definit anterior sau predefinit, numit tip de bază al subdomeniului. Mulțimea valorilor acestui tip cuprinde un interval de valori ale tipului de bază, ale cărui limite se specifică în definiția tipului subdomeniului. Constantele care reprezintă limitele din stânga şi dreapta ale subdomeniului sunt de acelaşi tip ordinal, iar cea din stânga trebuie să fie strict mai mică decât cea din dreapta.

Operațiile permise asupra datelor de tip subdomeniu sunt cele permise de tipul de bază al acestuia.

1.5. Variabile

Variabilele reţin date, iar conținutul unei variabile poate fi modificat pe parcursul execuției programului. Fiecărei variabile i se asociază un identificator şi un tip de dată. Tipul precizează *mulțimea de date* din care poate lua valori variabila respectivă şi *operațiile* care pot fi efectuate cu acea variabilă. Tipul se asociază variabilei printr-o declarație care poate utiliza *identificatori de tip* sau poate referi un tip *anonim*. Două tipuri anonime se consideră diferite, chiar dacă descriu aceeaşi structură, dar identificatorii aparținând aceleiaşi liste într-o declarații de variabile, folosind tipuri anonime, vor avea tipuri identice.

Pentru fiecare variabilă declarată în program se alocă o zonă de memorie în timpul compilării. Dimensiunea zonei de memorie pentru variabila respectivă depinde de tipul variabilei.

1.6. Definirea constantelor

Constantele simbolice se definesc de către utilizator în secțiunea **`const`**. Constantele simbolice reprezintă valori care nu se modifică pe parcursul execuției programului. Constantele simbolice pot fi de mai multe tipuri: *întreg*, *real*, *boolean*, *şir de caractere*.

În limbajul Pascal se pot defini *constante cu tip* care sunt variabile inițializate în faza de compilare. Valorile lor se pot modifica în timpul execuției programului. Folosirea constantelor cu tip poate fi avantajoasă pentru inițializarea variabilelor având tipuri structurate.

Exemple

```
const a=1977;                              { constanta simbolica }
      Zi='marti';
      Par:Boolean=Odd(a);                     { constanta cu tip }
```

1.7. Expresii

O expresie descrie o regulă de calcul într-un şir de operatori şi operanzi. Fiecare expresie evaluată returnează un rezultat. În evaluarea unei expresii se respectă prioritatea operatorilor. Dacă avem operatori cu aceeaşi prioritate, atunci se execută operaţiile de la stânga la dreapta. Dacă dorim schimbarea ordinii, vom folosi paranteze rotunde). Ordinea descrescătoare a priorităţii operatorilor este:

Prioritate	**Operator**
1. (maximă)	**`not`**, `+`, `-` (opratori unari)
2.	**`and`**, `*`, `/` , **`div`**, **`mod`**, **`shl`**, **`shr`** (operatori multiplicativi)
3.	**`or`**, **`xor`**, `+`, `-` (operatori aditivi)
4. (minimă)	`=`, `<>`, `<`, `<=`, `>`, `>=` (operatori relaţionali)

O expresie este alcătuită fie dintr-o expresie simplă, fie din două expresii legate printr-un operator relaţional. În calculul valorii expresiei se vor evalua mai întâi factorii, apoi termenii, după aceea expresiile simple, iar în final expresia relaţională.

1.8. Citirea şi scrierea datelor

Datele de intrare ale unui program se pot citi de la dispozitivul standard de intrare sau dintr-un fişier (o colecţie de date stocată în memoria externă a sistemului de calcul). Datele de ieşire în urma execuţiei unui program se pot afişa la dispozitivul standard de ieşire sau într-un fişier reţinut în memoria externă a sistemului de calcul.

Citirea datelor se realizează prin executarea procedurilor standard `Read` şi `ReadLn`. Scrierea datelor se realizează prin executarea procedurilor `Write` şi `WriteLn`. Acestea sunt declarate în *unit*-ul `System`.

Sintaxa apelului procedurii `Read` este: `Read(`*listă_de_variabile*`)`, unde *listă_de_variabile* poate conţine variabile de tip întreg, real sau şir de caractere.

În urma apelării fiecărei variabile prezente în lista de variabile i se atribuie o valoare. Valorile se iau de pe suportul extern, începând din punctul în care s-a ajuns după citirea anterioară. Valorile sunt luate astfel încât să corespundă tipului variabilei respective. Dacă acest lucru nu este posibil se va semnala eroare în execuţie. De exemplu, dacă *x* este o variabilă de tip întreg, iar *u* şi *v* sunt variabile de tip caracter, atunci instrucţiunea `Read(x,u,v)` atribuie o valoare întreagă variabilei *x* şi câte o valoare de tip caracter variabilelor *u* şi *v*.

Dacă pe suportul extern se află `'125 A'`, atunci *x* va primi valoarea `125`, variabila *u* va primi valoarea `' '` (caracterul spaţiu), iar *v* va primi valoarea `'A'`. Dacă suportul extern conţine `'12.4 A'`, atunci execuţia se va termina cu un mesaj de eroare întrucât valoarea `12.4` nu are tipul întreg. Tot cu eroare se va termina execuţia dacă pe suportul extern avem `'text 15'`, deoarece prima valoare întâlnită nu este un număr întreg.

Procedura `Write` asigură transferul informației din memoria internă pe suportul extern. Apelul procedurii are forma generală: `Write(`*listă_de_expresii*`)`. Lista de expresii Pascal poate conține expresii de orice tip, cu excepția tipurilor structurate.

Efectul acestui apel este evaluarea expresiilor prezente în listă, apoi tipărirea valorilor obținute.

Pe suportul extern datele de transferat se află scrise sau vor fi tipărite pe o succesiune de înregistrări. Considerăm că o înregistrare este constituită dintr-o secvență de date, urmată de marca de sfârşit de linie. În cazul apelării procedurilor `Read` şi `Write`, atunci când se epuizează o înregistrare se începe o altă înregistrare. Dar, adeseori este necesar să se treacă la o altă înregistrare înainte de a fi epuizată precedenta. Pentru aceasta se pot folosi procedurile `ReadLn` şi `WriteLn` care au sintaxa şi semnificația procedurilor `Read` şi `Write`, dar cer în plus ca la terminarea execuției lor să se treacă la începutul unei noi înregistrări.

1.9. Instrucțiuni

1.9.1. Instrucțiunea de atribuire

Instrucțiunea de atribuire are scopul de a da valori unor variabile. Ea are sintaxa:
variabila`:=`*expresie*`;`
unde *variabila* şi *expresie* sunt de acelaşi tip. *variabila* şi *expresie* pot fi de tip întreg, real, boolean, enumerare sau orice alt tip structurat cunoscut în programul în care apare instrucțiunea, cu excepția tipului fişier.

Instrucțiunea mai întâi evaluează expresia din partea dreaptă a semnului de atribuire `':='`. Dacă tipul expresiei şi cel al variabilei sunt identice, atunci valoarea expresiei este atribuită variabilei din stânga semnului de atribuire. Dacă tipul variabilei şi cel al expresiei diferă, atunci se semnalează eroare, dar există şi excepții:

1. Dacă tipul expresiei este `Integer`, iar cel al variabilei este `Real`, atunci se converteşte valoarea întreagă a expresiei într-o valoare reală care se atribuie variabilei.
2. Dacă tipul variabilei şi cel al expresiei sunt de tip enumerat sau subdomeniu cu acelaşi tip de bază şi dacă valoarea expresiei aparține tipului variabilei, această valoare se atribuie variabilei.
3. Dacă variabila şi expresia sunt de tip mulțime, atunci este posibil ca unul sau ambele tipuri să fie subdomenii ale aceluiaşi tip ordinal. Atribuirea este permisă dacă valorile din mulțimea rezultată sunt incluse în tipul de bază al variabilei din stânga semnului de atribuire.
4. A patra excepție se referă la tipul **`string`** (şir de caractere); tipul variabilei şi cel al expresiei putând fi tipuri **`string`** diferite (şiruri de caractere de lungimi diferite). Atribuirea este corectă dacă valoarea ei se poate atribui variabilei din stânga atribuirii.

În partea stângă a semnului de atribuire variabila poate fi una simplă sau o componentă a unui tablou sau a altui tip de dată structurată. Atribuirea `A:=B` este posibilă dacă `A` şi `B` sunt variabile de acelaşi tip. În toate cazurile se cere ca partea dreaptă a atribuirii să poată fi evaluată în mod „sigur" (toate variabilele să fie iniţializate, sau citite, eventual să aibă valoare obţinută din calcule precedente), altfel vom spune că variabila `B` este neiniţializată, iar atribuirea va putea genera rezultat eronat.

1.9.2. Instrucţiunea compusă

Sintaxa unor instrucţiuni impune folosirea, în cadrul ei, a unei singure instrucţiuni. Pentru a putea include în aceste locuri grupuri de instrucţiuni, în Pascal le vom grupa într-o entitate simplă. Această entitate se numeşte *instrucţiune compusă* şi are sintaxa:

`begin`
instrucţiune$_1$;
instrucţiune$_2$;
...
instrucţiune$_n$
`end;`

În limbajul Pascal caracterul punct şi virgulă (`;`) separă două instrucţiuni. El nu are rolul de a marca sfârşitul acestora, ca atare, acest caracter nu „trebuie" scris în faţa cuvântului **`end`**, dar acest lucru nu este interzis. În acest caz se consideră că între `;` şi **`end`** se află o *instrucţiune vidă*, având efect nul.

1.9.3. Instrucţiuni condiţionale

Există două instrucţiuni Pascal care permit execuţia unor instrucţiuni în funcţie de îndeplinirea unor condiţii: instrucţiunile **`if`** şi **`case`**. Instrucţiunea **`if`** are sintaxa:

`if` *condiţie* **`then`** *instrucţiune*`;`

sau

`if` *condiţie* **`then`** *instructiune*$_1$ **`else`** *instructiune*$_2$`;`

unde *condiţie* este o expresie logică, iar *instrucţiune*$_1$ şi *instrucţiune*$_2$ sunt instrucţiuni Pascal. În ambele variante instrucţiunea mai întâi evaluează expresia logică *condiţie*. Dacă valoarea obţinută este `true`, atunci se execută instrucţiunea *instrucţiune*$_1$ cu care se încheie execuţia instrucţiunii **`if`**. Dacă valoarea obţinută este `false` atunci, în cazul variantei a doua se execută instrucţiunea *instrucţiune*$_2$, iar în cazul primei variante nu se execută nimic.

Exemplu

```
if X < 0 then V:=-X
         else V:=X;
```

Instrucțiunea **case** permite selectarea unei instrucțiuni dintr-o mulțime de instrucțiuni marcate, în funcție de valoarea unui *selector*. Sintaxa instrucțiunii este:

case *expresie_ordinală* **of**
  lista_c: *instrucțiune*$_1$;
  lista_c: *instrucțiune*$_2$;
  ...
  lista_c: *instrucțiune*$_n$
[**else** *instrucțiune*$_{n+1}$]
end;

unde *expresie_ordinală* este o expresie de tip ordinal (selector), *lista_c* este o listă de *constante case* având tipul ordinal al expresiei *expresie_ordinală*. În *lista_c* în locul unei constante poate să apară şi un subdomeniu $c_1..c_2$ (pentru o scriere mai condensată în cazul în care constantele sunt consecutive).

Efectul instrucțiunii **case** constă în evaluarea expresiei *expresie_ordinală*, obținându-se o valoare *v*, care constituie valoarea selectorului. Apoi se caută în listele *constantelor case* valoarea egală cu *v* şi se execută instrucțiunea din dreptul *constantei case* egale cu *v*.

Dacă nu există nici o instrucțiune cu eticheta *v* atunci, în prezența cuvântului **else**, se execută instrucțiunea care urmează după acest cuvânt, altfel nu se execută nici o instrucțiune.

Exemplu

```
case a of
   'A': Write('A');
   'B': Write('B');
   'C': Write('C')
else
   Write('alta litera!!!')
end;
```

1.9.4. Instrucțiuni repetitive

În Pascal există trei instrucțiuni care execută repetat o instrucțiune; acestea sunt: **for**, **while** şi **repeat**.

A. Instrucțiunea for

Această instrucțiune execută repetat (în număr cunoscut de paşi) o instrucțiune în funcție de valoarea unui contor. Ea are sintaxa:

for *v*:=*lim*$_1$ **to** *lim*$_2$ **do** *instrucțiune*;

sau

for *v*:=*lim*$_1$ **downto** *lim*$_2$ **do** *instrucțiune*;

unde *v* este un identificator de variabilă numită *contor*, iar lim_1 şi lim_2 sunt expresii. Toate au acelaşi tip ordinal. Valorile lui lim_1 şi lim_2 se calculează o singură dată, la începutul execuţiei instrucţiunii **for**. Variabila contor *v* va lua valori între limitele lim_1 şi lim_2 crescător, dacă figurează cuvântul **to** şi descrescător, dacă figurează cuvântul **downto**. Semnificaţia acestei instrucţiuni este dată de următorul algoritm:

1. se calculează valorile expresiilor lim_1 şi lim_2;
2. **dacă** valoarea lui $lim_1 \leq$ valoarea lui lim_2 **atunci**

```
    v ← valoarea lui lim1
    repetă
      se execută instrucţiune
      dacă v < valoarea lui lim2 atunci
        v ← Succ(v)
      sfârşit dacă
    până când v = lim2
  sfârşit dacă
```

În cazul formei **for** *v*:=lim_1 **downto** lim_2 **do** *instrucţiune* vom schimba operatorii $\leq$ cu $\geq$, $<$ cu $>$ şi funcţia *Succ* cu *Pred*. Contorul *v* poate fi folosit în instrucţiune, dar modificarea valorii sale de către programator nu este recomandată.

B. Instrucţiunea `while`

Aceasta este o instrucţiune repetitivă cu număr necunoscut de paşi, *anterior condiţionată*. Sintaxa ei este:

while *condiţie* **do** *instrucţiune*;

unde *condiţie* este o expresie logică. Efectul executării instrucţiunii **while** este:

1. Evaluarea expresiei logice *condiţie*;
2. Dacă valoarea expresiei este `true`, atunci se execută *instrucţiune* şi se revine la pasul 1, altfel execuţia se termină.

Astfel se execută repetat o instrucţiune în funcţie de valoarea de adevăr a unei expresii logice. Dacă iniţial expresia logică este falsă, execuţia instrucţiunii respective nu va avea loc niciodată.

C. Instrucţiunea `repeat`

Instrucţiunea **repeat** permite execuţia repetată în număr necunoscut de paşi a unui grup de instrucţiuni. Instrucţiunea este posterior condiţionată şi are sintaxa:

repeat *instrucţiuni* **until** *condiţie*;

unde *condiţie* este o expresie logică. Semnificaţia instrucţiunii este:

1. Se execută instrucţiunile scrise între cuvintele **repeat** şi **until**;
2. Se evaluează expresia logică *condiţie*;
3. Dacă valoarea expresiei este `true`, atunci execuţia se termină, altfel se revine la pasul 1.

Observații

- expresia *condiție* nu se recomandă a fi foarte complicată, având în vedere că se evaluează la fiecare pas;
- cel puțin o variabilă din expresia logică *condiție* trebuie să-şi schimbe valoarea în mod obligatoriu, altfel există pericolul ca instrucțiunea să se transforme în ciclu infinit.

1.10. Probleme propuse

Problemele propuse în această secțiune pot fi rezolvate în cadrul unei evaluări/testări efectuate de profesor, după recapitulare, cu scopul stabilirii nivelului de cunoştințe a elevilor. Problemele sunt grupate pe 3 niveluri:

1. începători
2. avansați
3. excelenți

1.10.1. Începători

A. Învățarea alfabetului

Copiii din clasele mici învață din greu alfabetul, pe când cei mai mari trag chiulul. În clasa I un copil a „păcătuit" pentru că a încercat să vadă cum este dacă nu îşi învață lecția pentru ora viitoare. Învățătoarea l-a pedepsit, punându-l să scrie *n* litere mari, printre care acesta trebuie să scrie cel puțin o dată toate literele mari pe care le-a învățat până atunci.

Citiți numărul de litere *n* pe care trebuie să le scrie copilul, precum şi literele pe care acesta le-a scris şi scrieți un program care verifică dacă în şirul de litere apar sau nu anumite litere de mai multe ori. Afişați literele care apar de mai multe ori (au numărul de apariții mai mare decât 1) şi numărul aparițiilor acestora.

Exemplu

Intrare	**Ieşire**
5	A 2
A B C A C	C 2

B. Ghiocei şi viorele

Copiii din clasa a VI-a au mers la pădure după ghiocei. Ei au cules ghiocei şi viorele, în total un număr de *n* flori de primăvară. Au observat că ghioceii şi viorelele au crescut după o anumită regulă: între două viorele era un ghiocel, apoi câte doi şi tot aşa mai departe.

Citiți un număr natural *n*, reprezentând numărul florilor şi inițialele lor (`G` pentru ghiocei, `V` pentru viorele) stabiliți dacă florile se găsesc sau nu într-o configurație care

corespunde regulii descrise. Afişaţi un mesaj corespunzător (`'DA'` sau `'NU'`), precum şi poziţiile viorelelor.

Exemple

Intrare	**Ieşire**
`n = 6` `V G V G G V`	`DA` `1 3 6`
Intrare	**Ieşire**
`n = 5` `G V G G V`	`NU`

C. Serviciul pe clasă

La începutul anului şcolar dirigintele a întocmit planificarea serviciului în clasă, fiind planificaţi câte doi elevi pentru fiecare zi. Deoarece, elevii au declarat că doresc să fie de serviciu câte o fată şi un băiat, dirigintele a stabilit următoarea regulă: prima fată din catalog va sta cu primul băiat, a doua cu al doilea şi aşa mai departe. Dirigintele ştie că are *n* elevi în clasă, dar nu a numărat câte fete şi câţi băieţi sunt, aşa că se poate întâmpla să constate la sfârşit că unii copii au rămas fără pereche.

Citiţi numărul de elevi, numele şi genul lor (`'f'` pentru fete, `'b'` pentru băieţi), apoi stabiliţi dacă au rămas elevi fără pereche şi care sunt aceştia.

Exemplu

Intrare	**Ieşire**

`n=5`
`Ana f`
`Ioana f`
`Andrei b`
`Anca f`
`Paul b`
	`Fara pereche: Anca`

D. Tăierea copacilor

Pe drumul naţional E81 sunt copaci numerotaţi în zig-zag (pe partea dreaptă a drumului copacii sunt numerotaţi cu numere pare şi pe stânga cu numere impare). La un moment dat au fost aleşi copaci pentru tăiere de pe fiecare parte a drumului după următoarea regulă: se va tăia primul acel copac care este numerotat cu un număr ale cărui cifre însumate formează un pătrat perfect, apoi următorul copac având aceeaşi proprietate şi tot aşa mai departe până la capătul drumului.

Citiţi numărul copacilor, apoi determinaţi câţi copaci au fost aleşi pentru tăiere, pe care parte a drumului sunt aceştia, precum şi numărul cu care au fost numerotaţi. Numerotarea copacilor începe de la 1.

Exemplu

Intrare
```
n = 20
```

Ieşire
```
6 copaci
de pe dreapta: 4 10 18
de pe stanga: 1 9 13
```

E. La scăldat

La ştrandul din oraş s-a organizat un concurs de înot dotat cu premii. Din dorința de a câştiga marele premiu s-au înscris şi copii care nu ştiau să înoate. Pentru siguranța lor ei nu au fost acceptați în concurs. Deoarece concurenții aveau vârste diferite, s-a hotărât ca la timpul fiecărui concurent (măsurat în secunde) să se adune vârsta acestuia. Deoarece a început ploaia, s-a anunțat că se oferă un abonament celui care scrie un program care va stabili ordinea finală, pentru ca premierea să aibă loc cât mai repede.

Fişierul de intrare **STRAND.IN** are următorul format:

- pe prima linie este scris numărul n al copiilor înscrişi; copiii sunt identificați cu numere cuprinse între 1 şi n.
- pe linia a doua este scris numărul k al copiilor eliminați;
- pe linia a treia avem k numere, indicând copiii eliminați;
- pe linia a patra sunt scrise $n - k$ numere care reprezintă timpii celor rămaşi în concurs, în ordinea numerelor primite la înscriere;
- pe linia a cincia avem $n - k$ numere indicând vârstele celor rămaşi în concurs, în ordinea numerelor primite la înscriere.

În fişierul de ieşire **STRAND.OUT** se va afişa ordinea finală a copiilor participanți la concurs. Pe fiecare linie se va scrie unul sau mai multe numere, astfel încât numărul de ordine a liniei să fie egal cu poziția pe care copiii scrişi pe linia respectivă au terminat concursul.

Exemplu

STRAND.IN
```
7
2
1 3
60 55 58 55 70
8 14 10 10 9
```

STRAND.OUT
```
6
2 5
4
7
```

F. Drumul spre şcoală

Gigel este un elev căruia nu-i prea place şcoala şi ar face orice să nu mai ajungă acolo. În a 43-a zi a anului şcolar mama sa l-a trimis la şcoală la ora P şi P' minute. El trebuia să ajungă la şcoală la ora S şi S' minute.

În drum spre şcoală el se întâlneşte cu câțiva prieteni care îl țin de vorbă de la ora *D* şi *D′* minute până la ora *DD* şi *DD′* minute. Dacă discută mai mult de 43 de minute, atunci ei vor mai discuta încă 43 de minute.

Scrieți un program care verifică dacă Gigel ajunge la timp la şcoală şi în cazul în care ajunge, să se determine ora şi minutul la care a ajuns.

Durata deplasării de acasă până la şcoală este de 43 de minute.

Date de intrare

Datele de intrare se citesc din fişierul **`GIGEL.IN`** care are următorul format:

- pe prima linie se află ora *P* şi minutul *P′* la care Gigel pleacă de acasă;
- pe a doua linie se află numărul *N* al prietenilor cu care se întâlneşte Gigel pe drum;
- fiecare dintre următoarele *N* linii conțin ora *D* şi minutul *D′* de întâlnire şi ora *DD* şi minutul *DD′* la care se termină conversația;
- pe ultima linie este scrisă ora *S* şi minutul *S′* la care trebuie să ajungă Gigel la şcoală.

Date de ieşire

Rezultatul se va tipări la ieşirea standard. Dacă Gigel ajunge la şcoală în timp util, se va afişa cuvântul `'DA'`, iar dacă nu, se va tipări cuvântul `'NU'`. În cazul în care Gigel reuşeşte să ajungă la şcoală, se va mai tipări ora şi minutul la care ajunge.

Exemplu

`GIGEL.IN`	**`Ieşire`**

`7 30`
`2`
`7 32 7 34`
`7 36 7 37`
`8 43`
	`DA`
	`8 16`

1.10.2. Avansați

A. Numere prime... cu divizori

Elevii clasei a V-a au învățat la matematică despre numerele prime că au exact doi divizori: 1 şi numărul însuşi. La informatică au învățat că numerele se pot reprezenta şi în alte baze, nu doar în baza 10. Cei doi profesori (cel de matematică şi cel de informatică) s-au gândit să le dea o temă copiilor de clasa a V-a, care să combine cele două informații învățate. Ei vor trebui să găsească toate numerele mai mici decât un număr dat *n*, care au proprietatea că scrise în baza 10 nu sunt prime, dar scrise în altă bază şi citite în baza 10, sunt prime.

Scrieți un program care găseşte aceste numere, precum şi bazele în care au această proprietate.

Exemplu

Intrare	**Ieşire**
n = 4	numar in baza 10: 4 nu este prim numar in baza 3: 31 (citit in baza 10 este prim)

B. Matematica... pe calculator

La lecția de matematică profesorul le cere elevilor să efectueze o înmulțire a unor numere al căror produs depăşeşte „limitele matematicii”. Primul care va rezolva această problemă va avea media 10 în acel semestru. Unul dintre ei, fiind mai isteț şi având lângă el un calculator de buzunar, a început să efectueze înmulțirile pe calculator, dar a observat că în acest caz nici calculatorul de buzunar nu îi este de folos. Dar dacă tot este el mai isteț a apelat la informaticieni şi i-a rugat: „Primul care îmi va rezolva problema va primi temele corecte la matematică în tot acest semestru”.

Se citesc *n* numere întregi. Se cere să se determine produsul acestor numere.

Date de intrare

Pe prima linie a fişierul de intrare **PRODUS.IN** se află un număr natural *n*, iar pe următoarele *n* linii există câte un număr întreg.

Date de ieşire

Produsul celor n numere se va scrie în fişierul de ieşire **PRODUS.OUT**.

Restricții şi precizări

- $2 \le n \le 20$;
- $-25000 \le x_i \le 25000$.

Exemplu

PRODUS.IN	**PRODUS.OUT**
3 1234 -12345 2457	-37429274610

C. La pescuit

Elevii informaticieni, având prea multe concursuri de informatică, s-au gândit că ar fi bine să aibă o perioadă de recreere şi au organizat un concurs de pescuit la care au participat foarte mulți concurenți. Fiecare elev a prins peşti de diferite specii, speciile fiind identificate prin numere. Numărul peştilor prinşi este acelaşi pentru fiecare elev. Se spune că între doi elevi există *relație de legătură* dacă ei au prins cel puțin un peşte de aceeaşi specie, astfel formându-se grupuri dintre copiii care au în „pradă” cel puțin un peşte comun.

Determinați numărul de grupuri care se vor forma şi afişați membrii fiecărui grup.

Date de intrare
Datele de intrare se citesc din fişierul **PESTI.IN** care are următorul format:
- pe prima linie este scris numărul *n* al elevilor;
- pe a doua linie avem numărul *k* al peştilor prinşi de fiecare elev;
- pe fiecare dintre următoarele *n* linii avem câte *k* numere care indică specia fiecărui peşte prins de un anumit elev.

Datele de ieşire
Fişierul **PESTI.OUT** are următorul format:
- pe prima linie se va scrie numărul *g* al grupurilor formate;
- pe fiecare dintre următoarele *g* linii se vor scrie numerele de ordine ale copiilor care fac parte din câte-un grup.

Exemplu

PESTI.IN
```
10
3
1 2 1
1 2 2
3 3 3
3 4 5
7 8 7
7 8 7
8 8 7
9 9 9
1 1 2
7 7 8
```

PESTI.OUT
```
4
1 2 9
3 4
5 6 7 10
8
```

1.10.3. Excelenți

A. Serviciul de gardă
În vederea stabilirii serviciului de gardă, la o unitate militară, comandantul nominalizează *n* soldați (S) şi *n* gradați (G). Comandantul, ştiind că la un moment dat în post intră două persoane şi că ofițerul responsabil îi pune în pereche pe cei prevăzuți în listă, primul cu ultimul, al doilea cu penultimul etc., stabileşte ordinea în listă, astfel încât să nu se formeze perechi mixte (un soldat şi un gradat). Comandantul mai ştie că *n* este număr par şi va folosi această proprietate în momentul în care va întocmi lista.

Scrieți un program care afişează pe ecran lista din care ofițerul va forma perechi, conform cerinței. De asemenea, programul trebuie să afişeze şi perechile care intră în serviciu.

Exemplu

Intrare
```
n=4
Chirila
Macarie
Coza
Domsa
Lador
Orlea
Hila
Pacurar
S, S, G, G, S, G, S, G
```

Ieşire (*o soluție posibilă*)
```
Chirila
Coza
Lador
Pacurar,
Domsa
Macarie
Orlea
Hila

Chirila - Hila
Coza - Orlea
Lador - Macarie
Pacurar – Domsa
```

B. Plasarea oglinzilor

Se consideră un şir de n elemente care au valorile 0 şi 1. Definim operația *plasare* a două oglinzi, una în stânga elementului de pe poziția i, iar alta în dreapta elementului de pe poziția j. Această operație are ca efect interschimbarea elementelor egal depărtate de extreme a elementelor cuprinse între pozițiile i şi j, inclusiv.

Scrieți un program care va citi două şiruri de lungime n şi va stabili pozițiile în care vor trebui plasate cele două oglinzi în cadrul primului şir, astfel încât în urma plasării oglinzilor, primul şir să devină identic cu al doilea şir.

Exemplu

Intrare
```
n = 8
0 0 1 1 0 1 1 0
0 1 0 1 1 0 1 0
```

Ieşire
```
2 6
```

C. Palindrom

Unii matematicieni au fost curioşi şi au început să se „joace” cu numerele; ei au obținut tot felul de rezultate interesante. Pentru un număr oarecare x dat în baza b, definim următorul proces iterativ: la primul pas se adună numărului x oglinditul său, repetând succesiv procedeul cu rezultatul obținut.

Exemplu

Fie $n = 87$ şi $b = 10$.

- pasul 1: 87 + 78 = 165;
- pasul 2: 165 + 561 = 726;
- pasul 3: 726 + 627 = 1353;
- pasul 4: 1353 + 3531 = 4884.

Ultimul rezultat este un palindrom care s-a obținut după patru pași. Întrebarea pe care ne-o punem este următoarea: pornind cu un număr oarecare, se ajunge întotdeauna la un palindrom după un număr finit de pași?

Verificați pentru un număr dat *n* dacă se ajunge sau nu la un palindrom, procedând conform descrierii de mai sus. Veți aplica procedeul de cel mult *k* ori. În cazul în care numărul dat s-a transformat în număr palindrom în cel mult *k* pași afișați numărul de pași necesari, precum numărul palindrom obținut, în caz contrar scrieți în fișier `'NU'`.

Exemplu

Intrare	**Ieşire**
`n = 87`	`4`
`n = 10`	`4884`

D. Numere apropiate

Se dă un număr natural *n* (care conține cel mult 2457 de cifre). Determinați cel mai mic număr natural $m \geq n$ care acceptă ca divizor un număr dat *d* ($1 \leq d \leq 2457$).

Numerele *n* şi *d* se citesc de pe primele două linii ale fişierului **`NUMERE.IN`**, iar numărul *m* va fi scris pe prima linie a fişierului **`NUMERE.OUT`**.

Exemplu

`NUMERE.IN`	**`NUMERE.OUT`**
`14` `5`	`15`