

Universitatea Tehnică a Moldovei

Catedra Sisteme Optoelectronice

Programare

Ciclu de prelegeri

Limbajul de programare Pascal

Chişinău 2001

Adnotare

Pentru studenții anului I și II , facultatea de Radioelectronică

specializările IMT și SOE (de zi și fără frecvență)

Alcătuitor: lector asistent Sergiu Istrati

Redactor responsabil: conf. univ. dr. Pavel Nistiriuc

Recenzent: conf. univ. dr. Veaceslav Sidorenco

© U.T.M.,2001

Tema 1: Scurt istoric al apariției calculatoarelor

Activitățile mintale efectuate de către oameni necesită uneori un volum de muncă foarte mare și un procent de inteligență mediu sau scăzut. Multe dintre aceste activități necesită calcule elaborate. Pe parcursul timpului matematicienii au încercat să creeze mașini, care să efectueze aceste calcule automat, lăsând astfel oamenilor timpul necesar pentru a se concentra asupra problemelor esențiale. Dar cu toate eforturile depuse, mașinile automate de calcul sau, cu alte cuvinte, calculatoarele, au apărut doar în momentul în care tehnologia a permis acest lucru, pe la mijlocul anilor 40 ai secolului XX. Însă pînă în acel moment au fost puse bazele tehnicii de calcul moderne.

În 1641 matematicianul și filosoful Blaise Pascal inventează o mașină de calcul care putea efectua primele două ecuații aritmetice elementare cu șase cifre zecimale. Cu cîteva decenii mai tîrziu, în 1671 această mașina este perfecționată de matematicianul Wilhelm Leibnitz, asigurînd-o cu patru operații aritmetice.

În 1823, matematicianul și economistul englez Charles Babbage a conceput construcția mașinii diferențiale, care a fost concepută pentru realizarea automată a tabelelor matematice necesare în navigație. Calculatorul era capabil de a rula un singur algoritm. Din necesitatea de a crea un calculator de uz general, în 1834 el a conceput mașina analitică. În 1834 Babbage elaborează proiectul mașinii analitice, care conține unitățile de bază ale unui calculator modern:

- memoria (numită și depozit)
- unitatea aritmetică (moara sau fabrica)
- unitatea de comandă
- dispozitivele de intrare – ieșire.

Această mașină putea memora 1000 numere a cîte 50 cifre zecimale și realiza adunarea a două cifre într-o secundă, iar înmulțirea - într-un minut.

Primul dispozitiv de calculator cu o comandă-program este realizat de savantul german Konrad Zuse în 1941. Programul era memorat pe o bandă de cinematograf și era citită serial. Calculatorul era construit din 2600 releuri, putea număra 64 cifre a cîte 22 cifre binare.

În 1946 în SUA firma "Bell Telephone" realizează mai multe calculatoare. Primul calculator se numea Bell-v. El era construit din 9000 de rele și ocupa o suprafață de 90 m² avînd masa de 10 tone.

În 1943–1946 se construiește primul calculator electronic universal , ce se numea ENIAC (Electronic Numerical Integrator And Computer). Calculatorul conținea 18 mii tuburi electronice și 1500 rele, suprafața era de 135 m², cu masa de 30 tone. Structura și principiile de funcționare ale unui calculator modern au fost propuse de americanul Neumann: un calculator modern universal trebuie să includă dispozitivul aritmetic, dispozitivul de comandă, memoria și unitățile de intrare–ieșire. Astfel în memorie se vor înmagazina nu numai datele de prelucrare, dar și programul. Instrucțiunile unui program se încarcă în memorie în acelaș mod ca și datele de prelucrat. După începerea procesului de calcul instrucțiunile din memoria calculatorului vor fi extrase și executate automat.

În ceea ce privește dezvoltarea tehnicii de calcul în timp, ea a cunoscut mai multe etape, respectiv generații de calculatoare:

Prima generație de calculatoare o constituie calculatoarele proiectate în baza tuburilor electronice (pînă în 1959). Aceste calculatoare aveau un volum mare, viteză de calcul redusă (circa 1000 operații/sec.), capacitate de memorare redusă, programabile în limbaj-mașină.

Generația a doua cuprinde calculatoarele realizate între anii 1959-1964, locul tuburilor fiind luate de tranzistoare. Drept consecință volumul calculatorului s-a micșorat, iar viteza de funcționare a crescut simțitor, de asemenea a crescut și capacitatea memoriei, s-au extins și posibilitățile de programare: a apărut posibilitatea programării în limbajele Assembler, Cobol, Fortran; volumul sporit de memorie a stimulat lucrările îndreptate spre automatizarea programării calculatoarelor; au apărut primele sisteme operaționale. Exemple de calculatoare de generația a doua sînt Minsk-22, Minsk-32, БЭСМ-4, БЭСМ-6.

Generația a treia a apărut în anii 1964-1972. La baza acestor calculatoare stau circuitele integrate, datorită cărora dimensiunile calculatoarelor au scăzut cu mult, viteza de lucru și capacitatea memoriei au crescut cu cîteva ordine.

Din anul 1972 pînă în 1980 calculatoarele proiectate aparțin generației a patra. Ca element de bază servește microprocesorul și alte circuite integrate mari.

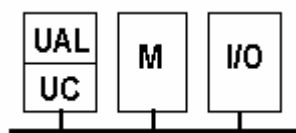
Din 1980 pînă în zilele noastre se dezvoltă generația a cincea de calculatoare, la baza proiectării cărora stau circuite integrate foarte complicate și cu capacități de lucru foarte mari. Aceste circuite conțin sute de mii de tranzistoare în corpul lor, avînd dimensiuni foarte mici.

Parametrii principali ce caracterizează un calculator modern sînt: Memoria operativă RAM(Mb), Memoria permanentă ROM(Gb), Frecvența de lucru(MHz).

Tema 2: Concepții generale despre calculatoare

2.1 Principiul de funcționare al calculatoarelor și structura lor.

Deși un calculator este ansamblul de dispozitive și circuite foarte diverse și complicate, studiul funcționării lui este mult ușurat prin faptul, că circuitele componente sînt grupate în unități avînd funcții mai complexe bine precizate. Structura clasică a calculatorului este următoarea:



unde UAL - unitate aritmetică-logică;

UC - unitate de control sau dirijare;

(UC+UAL=procesor);

M - memoria;

I/O - dispozitive de intrare/ieșire (input/output).

Pentru a înțelege mai bine principiul de funcționare a unui calculator, o vom face pe baza analizei lucrului unui program introdus în memoria calculatorului. Deci, avînd un text al unui program scris într-un limbaj oarecare, prin intermediul dispozitivelor de intrare (tastatura, unitatea de disc flexibil, ș.a) el se introduce în memoria calculatorului, de unde este citit și procesat de către unitatea aritmetică-logică; apoi datele rezultante sînt scoase la răspuns prin intermediul dispozitivelor de ieșire (monitorul, imprimanta ș.a). Toate aceste operații sînt efectuate sub supravegherea unității de control.

Din punct de vedere a dispozitivelor de bază, calculatorul modern este alcătuit din următoarele componente :

- unitatea centrală, menită sa gestioneze funcționarea întregului sistem;
- monitorul, destinat afișării datelor alfanumerice și grafice;
- tastatura , a carei destinație este introducerea datelor sau programelor;
- unitățile de discuri flexibile;
- unitățile de discuri rigide;

În afară de componentele de bază la un calculator personal mai pot fi conectate:

- imprimanta - unitate pentru tipărirea pe hartie a informațiilor alfanumerice și grafice;
- șoricelul sau mouse-ul, a carui destinație este facilitarea proceselor de introducere a informației și de manipulare a programelor;
- videomaneta sau joystick, utilizată în dispozitivele de antrenament și diverse simulatoare, la animarea jocurilor pe calculator;
- scannerul, a carui destinație este copierea imaginii de pe hârtie în memoria calculatorului;
- alte dispozitive ce ușurează lucrul la calculator.

2.2 Destinația dispozitivelor de bază.

Unitatea centrală a calculatorului se compune din microprocesorul central, memoria operativă RAM, controlere și porturile de intrare/ieșire (I/O).

Microprocesorul central determină viteza de lucru a calculatorului. În cazul microprocesoarelor slabe, cum ar fi INTEL8088, INTEL80286, INTEL80386 nu există comenzi speciale pentru efectuarea operațiilor complicate (în virgulă flotantă) și se recomanda folosirea coprocesoarelor aritmetice, care măresc viteza de lucru de 5-15 ori. Cu apariția procesoarelor Pentium aceasta problemă dispare și vitezele se măresc simțitor.

În memoria operativă se încarcă temporar componentele sistemului de operare necesare la un moment dat, precum și programul utilizatorului cu datele acestuia. Datele se păstrează în memoria operativă pînă la reîncărcarea calculatorului, apoi dispar. Capacitatea memoriei operative se masoară în octeți (bytes). Un octet reprezintă o structura de memorie de 8 poziții (celule), numite biți.(1Kb=1024 octeți, 1Mb=1024Kb). Calculatoarele personale actuale au o memorie operativă între 8Mb și 256Mb (pe cînd 30 pagini Word formatate = 250 Kb, 1 pagina text=2.5Kb).

Controlerele reprezintă dispozitive electronice care controlează activitatea imprimantei, monitorului, unităților de disc flexibil și rigid etc.

Porturile de intrare/ieșire (I/O) realizează schimbul de date între unitatea centrală și unitățile periferice. Porturi I/O există de 2 feluri: paralele (LPT1, LPT2 ...) și seriale asincrone (COM1, COM2 ...). Porturile paralele efectuează operațiile de I/O cu o viteză mai mare decît porturile seriale, dar necesită un numar mai mare de circuite pentru schimbul de date. La porturile I/O se pot conecta diferite dispozitive periferice: imprimanta, mouse-ul, scanner-ul, joystick-ul, tastatura, monitorul, etc.

Unitățile de discuri flexibile asigură scrierea și citirea informației pe dischete. Dischetele sînt purtători de informație, ce permit transportarea ei de la un calculator

la altul, păstrarea ei în formă de copii de arhivă etc. Capacitatea dischetelor se masoară în Kb și Mb. În prezent cele mai răspândite sînt dischetele cu capacitatea de 1.44 Mb avînd diametrul de 3.5 inches (89 mm). Mai există și dischete cu diametrul de 5.25 inches (133 mm) cu o capacitate de 360Kb, 720Kb, 1.2Mb. Înainte de a pune în aplicare, dischetele trebuie formatate. Formatarea dischetelor se realizează cu ajutorul comenzii *FORMAT* al SO MS-DOS. Este de menționat faptul că fiecare model de unitate de disc flexibil este calculat și destinat pentru folosirea unui anumit tip de dischete.

Unitățile de discuri rigide servesc la păstrarea permanentă a informației în calculator: programe, chituri de instalare, editori de texte și alte produse soft, documente importante în formă textuală, grafică. Discurile rigide sînt fixate. Din punct de vedere al utilizatorului discurile rigide se deosebesc prin durata de acces și capacitate (volumul de informație care poate fi stocat pe disc). Calculatorul IBM PC/XT posedă disc rigid cu capacitatea de zeci Mbytes, iar discul rigid al calculatorului IBM AT are o capacitate de sute Mbytes. Calculatoarele moderne posedă discuri rigide cu o capacitate de ordinul Gbytes.

Durata de acces este timpul necesar unității de disc pentru a poziționa capul pentru citire-scriere pe pista indicată și pentru a citi sectorul cerut de unitatea centrală. Durata de acces la discul rigid al calculatorului cu procesor intel 80386 este aproximativ egală cu 12-25 ms.

În sistemele de operare ale calculatoarelor unitățile de disc se notează prin litere latine urmate de simbolul ":". Ex: A:,B:,C:,D:, ... Z: .

Imprimanta este unitatea periferică destinată tipăririi informației. Toate imprimantele pot tipări informații numerice și grafice, alb-negru și color. Există mai multe tipuri de imprimante:

- matriciale;
- cu jet de cerneală;
- laser.

Imprimanta matricială are un cap de tipărit care conține un set de ace metalice subțiri, care la momentul dat lovesc prin banda de tipărire în foaia de hîrtie. La *imprimanta cu jet de cerneală* caracterele tipărite se formează din picături microscopice pulverizate pe hîrtie de niște duze speciale. Ele asigură o calitate destul de înaltă a tiparului și sînt comode pentru imprimarea color. Viteza de imprimare este comparabilă cu cea a imprimantelor matriciale. Prețul imprimantelor cu jet de cerneală este mai mare ca la cele matriciale. *Imprimanta laser* asigură cea mai bună calitate a tiparului. La baza acestor imprimante se află principiul de xerografie. Caracterul se imprimă pe un cilindru pe care a fost format, prin intermediul semnalelor electrice cu vopsea specială - toner. Viteza de tipărire este mai mare ca la primele 2 tipuri de imprimante și constituie 3-15 sec. pentru o pagina. Aceste imprimante sînt cele mai scumpe.

Monitorul e destinat afișării pe ecran a informației alfanumerice și grafice. Există monitoare color și monocolor. Ele pot funcționa în două moduri: alfanumeric și grafic. În cazul regimului alfanumeric, ecranul e împărțit în zone convenționale numite zone caracter. De obicei aceste zone alcatuiesc 25 linii cu 80 caractere pe linie. În fiecare zonă poate fi afișat un singur caracter dintr-un set de 256 caractere. Printre

caracterele afișate pe ecran pot fi: litere mari și mici ale alfabetului latin, cifre, semne de punctuație și matematice, precum și alte litere din alte alfabete. Regimul grafic al monitorului este destinat afișării pe ecran desenelor, graficelor, diagramelor și textelor. Ecranul e alcătuit din puncte numite pixeli, fiecare punct poate fi pe monitorul monocrom iluminat sau întunecat, sau de una din culorile de care dispune monitorul color. Numarul de puncte pe orizontală și verticală determină rezoluția monitorului. Ex: 640x480, 800x600, 1024x768.

Din punct de vedere al rezoluției și graficii există mai multe norme internaționale pentru monitoare. Ex: CGA, EGA, VGA, SVGA.

Tastatura este dispozitivul care permite introducerea informației în calculator și gestiunea sistemului. Tastaturile se clasifică în funcție de țara căreia îi sînt destinate. Tastatura are mai multe tipuri de taste:

- alfa-numerice (a,b,c,...,z,0,1,...,9,(.,:;,"*,\$,@, etc.);
- funcționale (F1-F10);
- speciale (Shift, Alt, Ctrl, Enter).

Mouse-ul are destinația de a mări comoditatea lucrului la calculator, de a gestiona sistemul. Mouse-ul poate fi cu 2 sau 3 clape. Cele mai răspîndite sînt cele cu 2 clape. Clapa din stînga are funcția confirmării unei comenzi și este echivalentă cu tasta <ENTER> de pe tastatură. Clapa din dreapta deschide meniuri adăugătoare ajutătoare, avînd specificații diferite pentru fiecare obiect în particular.

2.3 Bazele aritmetice ale calculatorului.

2.3.1 Sistemele de numerație.

În calculatoarele digitale informația de orice categorie este reprezentată, stocată și prelucrată în formă numerică. Numerele se reprezintă prin simboluri elementare numite cifre. Totalitatea regulilor de prezentare a numerelor împreună cu mulțimea cifrelor reprezintă un sistem de numerație. Numarul cifrelor folosite servește drept bază a sistemului de numerație. Cel mai frecvent sînt folosite sistemele:

- Zecimal: sistem de numerație în baza 10, cifrele utilizate: 0..9;
- Binar: baza-2, numarul de cifre utilizate 2: 0,1. Cifrele se numesc biți;
- Ternar: baza-3, cifre: 0,1,2;
- Octal: baza-8, cifre 0..7;
- Hexazecimal baza-16, 16 cifre:0,1,...,9, A(zece), B,C,D,E,F(15)

Regula de reprezentare a numerelor din sistemul zecimal rezultă din urmatorul exemplu: $(3856,43)_{10} = 3 \cdot 10^3 + 8 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0 + 4 \cdot 10^{-1} + 3 \cdot 10^{-2}$. Se observă, că în această reprezentare semnificația (valoarea) fiecărei cifre depinde de poziția pe care o ocupa în numar (sute, mii, sutimi, miimi etc).

Sistemele în care semnificația cifrelor depinde de poziția ocupată în cadrul numerelor se numesc sisteme de numerație poziționale.

Presupunem că numărul N are partea întregă formată din n+1 cifre, iar partea fracțională din m cifre: $N = C_n \cdot C_{n-1} \dots C_1 \cdot C_0, C_{-1} \dots C_m$.

Valoarea acestui numar se evaluează în funcție de baza sistemului de numerație: $(N)_b = C_n b^n + C_{n-1} b^{n-1} + \dots + C_0 b^0 + C_{-1} b^{-1} + \dots + C_{-m} b^{-m}$.

Efectuând calculele respective se va realiza conversia numărului $N(b)$ din baza respectivă în baza zece . Ex:

$$(110.11)_{10} = 1 \cdot 10^2 + 1 \cdot 10^1 + 0 \cdot 10^0 + 1 \cdot 10^{-1} + 1 \cdot 10^{-2} = 110.15;$$

$$(110.11)_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 6.75;$$

$$(110.11)_3 = 1 \cdot 3^2 + 1 \cdot 3^1 + 0 \cdot 3^0 + 1 \cdot 3^{-1} + 1 \cdot 3^{-2} = 12.444 \dots;$$

$$(110.11)_8 = 1 \cdot 8^2 + 1 \cdot 8^1 + 0 \cdot 8^0 + 1 \cdot 8^{-1} + 1 \cdot 8^{-2} = 72.140625;$$

$$(110.11)_{16} = 1 \cdot 16^2 + 1 \cdot 16^1 + 0 \cdot 16^0 + 1 \cdot 16^{-1} + 1 \cdot 16^{-2} = 272.06640625;$$

Formal sistemul zecimal nu prezintă nici un avantaj deosebit față de celelalte sisteme de numerație. Se presupune că acest sistem a fost adoptat din cele mai vechi timpuri datorită faptului că procesul de numărare a folosit ca instrumente inițiale degetele mâinilor.

O mașina de calcul automată poate fi construită să lucreze în orice sistem de numerație. Pe parcursul dezvoltării tehnicii de calcul s-a stabilit că cel mai avantajos este sistemul binar. Acest sistem a fost preferat din următoarele motive:

- Simplitatea regulilor pentru operațiile aritmetice și logice.
- Materializarea fizică a cifrelor în procesul prelucrării sau stocării numerelor se face mai ușor pentru două simboluri decât pentru zece: perforat-neperforat, contact închis - contact deschis, prezență de curent sau absență de curent etc.
- Circuitele care trebuie să diferențieze numai între 2 stări sînt mai sigure în funcționare decât cele care trebuie să diferențieze între zece stări.

În procesul dezvoltării civilizației umane au fost create și *sisteme de numerație nepoziționale*. Drept exemplu poate servi sistemul roman, care utilizează cifrele I, V, X, L, C, D, M. Întrucît regulile de prezentare a numerelor și de efectuare a operațiilor aritmetice sînt foarte complicate, sistemele nepoziționale au o utilizare foarte restrînsă.

2.3.2 Conversia numerelor dintr-un sistem în altul.

Conversia numărului $(N)_b$ în echivalentul său zecimal se efectuează conform formulei : $(N)_b = C_n b^n + C_{n-1} b^{n-1} + \dots + C_0 b^0 + C_{-1} b^{-1} + \dots + C_{-m} b^{-m}$.

Conversia numărului zecimal $(N)_{10}$ în echivalentul său în baza b se efectuează conform următoarelor reguli:

- 1) Pentru partea întreaga. Se împarte la baza respectivă partea întreagă și cîturile obținute după fiecare împărțire, pîna se obține cîtul zero. Rezultatul conversiei părții întregi este compus din resturile obținute luate în ordinea inversă apariției.
- 2) Pentru partea fracționară. Se înmulțește cu baza partea fracționară, apoi toate părțile fracționare obținute din produsul anterior pîna cînd partea fracționară a unui produs este zero sau pîna la obținerea unui număr fracționar cu numărul de cifre după virgulă dorit. Rezultatul conversiei părții fracționare este constituit din părțile întregi ale produselor considerate în ordinea apariției.

Analizăm cîteva exemple:

- 1) De efectuat conversia numărului zecimal 37.0625 în echivalentul său binar.

Pentru partea întreagă: $37:2=18+1/2$; $18:2=9+0/2$; $9:2=4+1/2$; $4:2=2+0/2$; $2:2=1+0/2$; $1:2=0+1/2$;

Pentru partea fracționară: $0.0625 \cdot 2 = 0.125$; $0.125 \cdot 2 = 0.250$; $0.250 \cdot 2 = 0.5$; $0.5 \cdot 2 = 1.0$;

Prin urmare $(37.0625)_{10} = (100101, 0001)$;

2) De transformat numărul 43.9 din sistemul zecimal în sistemul binar.

Pentru partea întreaga: $43:2=21+1/2$; $21:2=10+1/2$; $10:2=5+0/2$; $5:2=2+1/2$; $2:2=1+0/2$; $1:2=0+1/2$;

Pentru partea fracționară: $0.9 \cdot 2 = 1.8$; $0.8 \cdot 2 = 1.6$; $0.6 \cdot 2 = 1.2$; $0.2 \cdot 2 = 0.4$; $0.4 \cdot 2 = 0.8$; $0.8 \cdot 2 = 1.6$; ...

Observăm că procesul înmulțirii poate continua până la infinit. În acest caz rotundim cifra cu numărul de poziții după virgulă dorit. Deci $(43.9)_{10} = 101011, 111001100\dots)_2$.

3) Să se transforme $(1456.40625)_{10}$ în sistemul octal.

Partea întreaga: $1456:8=182+0/8$; $182:8=22+6/8$; $22:8=2+6/8$; $16:8=2+0/8$; $2:8=0+2/8$;

Partea fracționară: $0.40625 \cdot 8 = 3.25$; $0.25 \cdot 8 = 2.0$;

Prin urmare: $(1456, 40625)_{10} = (20660, 32)_8$;

4) Să se efectueze conversia numărului zecimal 3786,25 în echivalentul său hexazecimal.

Partea întreaga: $3786:16=236+10/16$; $236:16=14+12/16$; $14:16=0+14/16$;

Partea fracționară: $0.25 \cdot 16 = 4$.

Deci: $(3786, 25)_{10} = (ECA, 4)_{16}$.

2.3.3 Conversia numerelor din sistemul binar în octal, hexazecimal și invers.

Deoarece $8=2^3$, conversia din binar în octal și invers se poate face direct. Orice cifră octală se reprezintă prin 3 cifre binare: $0=000$; $1=001$; $2=010$; $3=011$; $4=100$; $5=101$; $6=110$; $7=111$.

Dacă se consideră un număr octal, pentru conversia lui în binar se va scrie fiecare cifră octală prin 3 cifre binare. De exemplu:

$(352, 451)_8 = (011\ 101\ 010, 100\ 101\ 001)_2$;

$(126, 23)_8 = (001\ 010\ 110, 010\ 011)_2$;

$(5, 065)_8 = (101, 000\ 110\ 101)_2$;

Dacă se consideră un număr binar, pentru conversia lui în octal se vor grupa câte 3 cifre binare pornind de la poziția virgulei spre stânga pentru partea întreagă, respectiv spre dreapta pentru partea fracționară, găsind corespondentul în octal. Pentru completarea unui grup de 3 cifre binare se pun zerouri înaintea numărului pentru partea întreagă, respectiv după număr pentru partea fracționară; aceste schimbări a numărului nu vor modifica valoarea lui. Exemple:

$(101, 100101)_2 = (101, 100\ 101)_2 = (5, 45)_8$;

$(10, 10011)_2 = (010, 100\ 110)_2 = (2, 46)_8$;

$(1010, 1010)_2 = (001\ 010, 101\ 000)_2 = (12, 50)_8$;

În mod similar se procedează și în cazul sistemului hexazecimal, baza căruia $16=2^4$. Orice cifră hexazecimală se reprezintă prin 4 cifre binare:

$0=0000$; $1=0001$; $2=0010$; $3=0011$; $4=0100$; $5=0101$; $6=0110$; $7=0111$;
 $8=1000$; $9=1001$; $A=1010$; $B=1011$; $C=1100$; $D=1101$; $E=1110$; $F=1111$;

Exemple de conversii hexazecimal – binar:

$$(4B5F, B7)_{16} = (0100\ 1011\ 0101\ 1111, 1011\ 0111)_2 ;$$

$$(A51, 3DE)_{16} = (1010\ 0101\ 0001, 0011\ 1101\ 1110)_2 ;$$

Exemple de conversii binar – hexazecimal:

$$(1011, 100111)_2 = (1011, 1001\ 1100)_2 = (B, 9C)_{16} ;$$

$$(10, 11011001)_2 = (0010, 1101\ 1001)_2 = (2, D9)_{16} ;$$

2.3.4 Operații aritmetice în binar.

Operațiile aritmetice cu numerele binare, octale, hexazecimale diferă de la un sistem la altul. Cele mai simple sînt operațiile cu numerele binare. Toate operațiile aritmetice cu numerele binare sînt bazate pe regulile de adunare, scădere și înmulțire binară, prezentate în tabel.

Operații aritmetice cu numerele binare.

Adunarea binară	Scaderea binară	Inmulțirea binară
$0 + 0 = 1$	$0 - 0 = 0$	$0 * 0 = 0$
$0 + 1 = 1$	$1 - 0 = 1$	$0 * 1 = 0$
$1 + 0 = 1$	$1 - 1 = 0$	$1 * 0 = 0$
$1 + 1 = 10$	$10 - 1 = 1$	$1 * 1 = 1$

Pentru însușirea mai ușoară a temei, vom prezenta exemple concrete:

Ex.1: Adunarea în binar a numerelor zecimale 17 și 38:

Pentru a efectua adunarea binară a 2 numere zecimale, întâi de toate ele vor fi convertite în sistenul binar:

$$(17)_{10} = (10001)_2 ; 17:2=8+1/2; 8:2=4+0/2; 4:2=2+0/2; 2:2=1+0/2; 1:2=0+1/2;$$

$$(38)_{10} = (100110)_2 ; 38:2=19+0/2; 19:2=9+1/2; 9:2=4+1/2; 4:2=2+0/2;$$

$$2:2=1+0/2; 1:2=0+1/2;$$

Adunarea binară: deci: $(10001)_2 + (100110)_2 = (110111)_2$

 iar $(110111)_2 = 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = (55)_{10} ;$

$$\begin{array}{r} 10001 \\ 100110 \\ \hline \end{array} +$$

$$110111$$

Verificare $(17)_{10} + (38)_{10} = (55)_{10} ;$

Ex.2: Efectuați adunarea binară a numerelor $(26)_{10}$ și $(23)_{10}$;

$$(26)_{10} = (11010)_2 ; (23)_{10} = (10111)_2 ;$$

Adunarea binară: Deci $(11010)_2 + (10111)_2 = (110001)_2 ;$

$$\begin{array}{r} 11010 \\ 10111 \\ \hline \end{array} +$$

$$110001$$

Verificare : $(110001)_2 = (49)_{10} ;$

$$(26)_{10} + (23)_{10} = (49)_{10} ;$$

Ex.3: De efectuat scăderea în binar a numerelor zecimale: $(45)_{10} - (35)_{10} ;$

$$(45)_{10} = (101101)_2 ; (35)_{10} = (100011)_2 ;$$

Efectuăm scăderea:

$$\begin{array}{r} 101101 \\ 100011 \\ \hline \end{array} -$$

$$001010$$

Deci $(101101)_2 - (100011)_2 = (001010)_2 ;$

Verificăm: $(001010)_2 = (10)_{10} ;$

$$(45)_{10} - (35)_{10} = (10)_{10} ;$$

La efectuarea adunării și scăderii binare a numerelor zecimale fracționare părțile întregi și părțile fracționare sînt adunate sau scazute aparte.

Ex.4: Înmulțirea în binar a numerelor zecimale: $(5, 125)_{10} * (2, 25)_{10}$;

După transformare primim: $(5, 125)_{10} = (101, 001)_2$; $(2, 25)_{10} = (10, 01)_2$;

Efectuăm înmulțirea binară:

10,01 Am primit ca rezultat: $(101, 001)_2 * (10, 01)_2 = (1011,10001)_2$;

101,001

1001

0000

0000

1001

0000

1001

1011,10001

Verificăm: $(1011,10001)_2 = (11, 53125)_{10}$;

$(5, 125)_{10} * (2, 25)_{10} = (11, 53125)_{10}$;

Ex.5: De efectuat împărțirea binară a numerelor zecimale: $(120)_{10} : (4)_{10}$;

$(120)_{10} = (1111000)_2$; $(4)_{10} = (100)_2$; Efectuăm împărțirea binară:

1111000 | 100

100 | _____

111 11110

100

110

100

100

100

0

Rezultatul este: $(1111000)_2 : (100)_2 = (11110)_2$;

Verificare: $(11110)_2 = (30)_{10}$;

$(120)_{10} : (4)_{10} = (30)_{10}$;

Notă: În cazul înmulțirii sau împărțirii virgula, care desparte partea întreagă de cea fracționară este poziționată ca și în cazul sistemului de numerație zecimal.

Operații aritmetice se pot efectua de asemenea și asupra numerelor octale și hexazecimale. Pentru ușurarea proceselor de adunare, înmulțire există tabele cu rezultatele adunării și înmulțirii a două cifre dintr-un sistem sau altul. Aici nu vom precăuta efectuarea operațiilor în octal și hexazecimal, presupunînd studierea acestei teme în viitor la un obiect specializat cum ar fi “Tehnica numerică de calcul”.

Tema 3: Metodica rezolvării problemelor cu ajutorul calculatoarelor.

3.1 Etapele de rezolvare a problemelor cu ajutorul tehnicii de calcul

Orice problemă, destinată rezolvării cu ajutorul calculatorului, în procesul realizării sale neapărat trece prin cele 5 etape:

1. Alegerea modelului matematic
2. Elaborarea algoritmului
3. Verificarea algoritmului
4. Realizarea algoritmului
5. Verificarea programului
6. Documentarea programului.

1. **Alegerea modelului matematic.** Fiecare problemă independent de complexitatea sa sau de ramura disciplinei poate fi formulată matematic înaintea rezolvării sale. Această formulare constituie modelul matematic al problemei. El este simplu sau complicat după natura problemei în studiu. La alegerea modelului matematic influențează următorii factori:


- marginea cunoștințelor programatorului în ceea ce privește numărul metodelor.
- comoditatea prezentării datelor.
- simplitatea datelor.
- posibilitățile tehnicii de calcul.

După alegerea modelului matematic, problema în cauză se reformulează în termenii obiectelor matematice corespunzătoare.

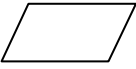
2. **Elaborarea algoritmului.** Prin algoritm se înțelege o mulțime de reguli care indică o succesiune de operații necesare pentru rezolvarea unui tip specific de probleme. Mulțimea de reguli, care indică modul de obținere a rezultatelor, constituie pașii algoritmului. Execuția pașilor algoritmului în ordinea cerută duce la rezolvarea problemei. Orice algoritm are 3 proprietăți de baza:

- Generalitate (constă în faptul că algoritmul trebuie să fie realizat nu în caz particular, ci în caz general, fiind luate în considerație toate situațiile ce pot să apară în tipul respectiv de probleme;
- Realizabilitate (este calitatea care scoate în evidență faptul, că tot ceea ce se întreprinde este exprimat corect, adică posibil de realizat);
- Finitudine (înseamnă că într-un timp finit rezolvarea problemei ia sfârșit, adică în cadrul algoritmului lipsesc procese ce duc mersul rezolvării în impas ciclic).

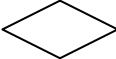
Orice algoritm mai are date de intrare, în urma prelucrării cărora se capătă datele rezultante - de ieșire. În ceea ce privește reprezentarea algoritmului, există o mare varietate de forme. Cele mai răspândite din ele sînt: a) reprezentarea cu ajutorul schemelor logice, b) reprezentarea cu ajutorul unui limbaj de tip pseudocod, etc. Reprezentarea algoritmului sub forma de schemă logică folosește următoarele elemente logice:


1. START/STOP. Bloc-delimitator. Indică începutul/sfârșitul schemei. 

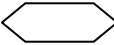
2. Bloc de atribuire. 

3. Bloc de intrare. Pune în evidență o operație de citire a datelor. 

4. Bloc de ieșire. Indică necesitatea scoaterii datelor pentru a le vizualiza. 

5. Blocul de condiție(decizie). 

6. Blocul de procedură. Indică corpul unui subprogram încorporat în cel principal. 

7. Blocul de ciclu FOR. Destinat realizării proceselor ce necesită câteva repetări. 

8. Nod. Este folosit în cazul intersecției și suprapunerii săgeților într-un punct. ○
9. Elementul de trecere pe altă foaie (bloc de legătură). □
10. Săgeata de legatură între blocuri. ↓

Între elementele schemei logice de calcul sînt unele grupări pentru reprezentarea proceselor mai complicate folosindu-se elementele prezentate mai sus.

3. **Verificarea algoritmului.** Este una din cele mai complicate etape. Presupune verificarea corectitudinii mersului rezolvării, încercarea algoritmului cu diferite date de intrare, și echivalența rezultatelor primite prin alte metode de rezolvare. Algoritmul se descrie sub pași numerotați (0...n) și se face analiza, controlul și motivația fiecărui pas, de asemenea este necesară demonstrarea faptului că algoritmul este finit(pentru aceasta trebuie controlate toate datele potrivite și primite rezultatele respective.
4. **Realizarea algoritmului.** Presupune analiza profundă a sarcinilor algoritmului, alegerea limbajului de programare și traducerea algoritmului în limbajul respectiv.
5. **Verificarea programului.** Pentru verificarea unui program cele mai utilizate metode sînt: a) metoda analitică, b) metoda constructivă, c) metoda testării. Ne oprim la metoda 3.
Testarea programului este însoțită de depanarea lui. Depanarea stabilește cauzele, care au determinat apariția erorilor și constă în folosirea unor metode și instrumente pentru înlăturarea lor. În cazul limbajului BP7.0 depanarea se face cu ajutorul compilatorului încorporat în mediul BP.
6. **Documentarea programului.** Prevede descrierea metodei algoritmului, a programului, a tuturor variabilelor folosite, construcțiilor logice, rezultatelor obținute și indicații pentru lucrul cu programul.

Tema 4: Limbajul de programare Pascal

4.1 Noțiuni generale

4.1.1 Alfabetul, vocabularul și sintaxa limbajului.

Numim limbaj de programare un limbaj prin care putem comunica unui calculator metoda de rezolvare a unei probleme. Iar metoda de rezolvare a problemei, după cum știm deja o numim algoritm. Întreaga teorie informatică se ocupă, de fapt, cu elaborarea unor noi calculatoare, limbaje de programare și algoritmi. Datoria oricărui limbaj de nivel înalt este să ne pună la dispoziție o sintaxă cât mai comodă prin care să putem descrie datele cu care lucrează programul nostru și instrucțiunile care trebuiesc executate pentru a rezolva o anumită problemă.

Limbajul de programare Pascal, ca și orice alt limbaj de programare își are alfabetul său și specificul de utilizare a simbolurilor. Alfabet al unui limbaj de programare se numește un set de simboluri permis pentru utilizare și recunoscut de compilator, cu ajutorul căruia pot fi formate mărimi, expresii și operatori ai acestui limbaj de programare. Alfabetul oricărui limbaj de programare conține cele mai

simple elemente cu semnificație lingvistică, iar sintaxa limbajului definește modul în care se combină elementele vocabularului pentru a obține fraze corecte (instrucțiuni, secvențe de instrucțiuni, declarații de tipuri, variabile, constante, etichete, funcții, proceduri etc.).

Elementele vocabularului sînt alcătuite din caractere. Orice caracter este reprezentat în calculator, în mod unic, printr-un număr natural cuprins între 0 și 127, numit cod ASCII.

Alfabetul limbajului de programare Pascal este compus din:

1. Simboluri folosite pentru formarea identificatorilor:
 - literele mari și mici ale alfabetului latin,
 - cifrele arabe de la 0 la 9,
 - simbolul de subliniere `_` (cod ASCII:95).
2. Simboluri de despărțire:
 - simbolul ‘spațiu’(cod ASCII:32).Are funcția de despărțire a cuvintelor cheie și numerelor.
 - simboluri de conducere (ASCII:0...31), se pot folosi la descrierea constantelor. Simboluri tabulare (ASCII:9) și simbolul de trecere în alt rînd

$$\{a := b + c - D \text{ e identic cu } \begin{cases} a := b \\ + c - D \end{cases} .$$

3. Simboluri speciale care îndeplinesc anumite funcții în alcătuirea diferitor construcții ale limbajului de programare Pascal
`+ - * / { } [] () < > . , ‘ : ; ^ @ # $`
4. Simboluri compuse – grupe de simboluri, recunoscute de către compilator ca un tot întreg:
`<= => := (* *) (. .) .. ,`
5. Simboluri “neutilizate”. Simboluri ale tabelii ASCII extinse (cod 128...255). Aici se conțin literele alfabetului rus, simboluri de pseudografică, și cîteva simboluri ale tabelului ASCII (Ex.: `& , ! , % , ~ , “` și altele).Ele nu intră în alfabetul limbajului, dar pot fi folosite în textul comentariilor, în textul mesajelor.

Vocabularul limbajului de programare Pascal.

Cele mai simple elemente, alcătuite din caractere și care au semnificație lingvistică sînt unitățile lexicale. Acestea formează vocabularul limbajului.

Distingem următoarele unități lexicale:

- simboluri speciale, identificatori, numere, șiruri de caractere, etichete, comentarii, directive;
- simboluri speciale;
- cuvintele cheie:

and array begin case const div do downto else end file for function goto if in label mod nil not of or origin otherwise packed program record repeat set then to type until var while with etc.

Cuvintele cheie sînt rezervate, adică sînt utilizate în program doar cu semnificația dată explicit prin definiția limbajului. Celelalte simboluri speciale sînt folosite ca operatori și delimitatori.

Identificatorii sînt nume asociate constantelor, variabilelor, tipurilor de date, procedurilor și funcțiilor. Primul caracter al numelui este o literă sau '\$', iar celelalte sînt litere, cifre, semnul '\$' sau '-'.

Numerele pot fi de tip întreg sau real. Numerele de tipul întreg desemnează numere întregi și sînt scrise în reprezentare zecimală, precedate sau nu de semnele '+' sau '-'. Numerele de tipul real desemnează numere raționale cu număr finit de zecimale. În mod uzual ele sînt reprezentate prin numere fracționare (în baza 10) cu partea fracționară separată de partea întreagă prin punct. La scrierea numerelor de tipul real se poate utiliza și un factor de scală. Acesta este un număr întreg precedat de litera E (Ex. 7,354E+02) și are efectul de înmulțire a numărului real cu 10 la puterea egală cu valoarea factorului de scală.

Șirurile de caractere sînt șiruri de caractere imprimabile, delimitate de apostrof, în șirul de caractere, apostroful apare dublat. Ex. 'șir de caractere'

Etichetele sînt șiruri de cifre zecimale. Ex. 1, 01, 20, 0.

Comentariile sînt șiruri de caractere precedate de '{' și urmate de '}'.

Comentariile conțin informație despre:

- numele fișierului în care se păstrează programul,
- o descriere scurtă a destinației programului,
- drepturile de autor,
- limbajul de programare folosit,
- versiunea programului,
- însemnări despre destinația unor părți a programului etc.
- Directivele sînt cuvintele rezervate forward, external, nonpascal și %include.

În scrierea oricărei unități lexicale nu se face distincție între literele mari și mici. La scrierea consecutivă a identificatorilor, a cuvintelor cheie, a literelor numerice ele trebuie să fie despărțite de ' ' (spațiu).

Prin sintaxa unui limbaj de programare se înțelege, în general, un ansamblu de reguli de agregare a unităților lexicale pentru a forma structuri mai complexe (instrucțiuni, declarații, programe etc.). Forma generală a unui program în limbajul de programare Pascal de asemenea își are regulile ei de sintaxă după cum urmează: un antet, urmat de o parte declarativă și de o instrucțiune compusă. Și pentru descrierea în detaliu a acestor componente sînt necesare, desigur și alte reguli.

4.1.2 Structura generală a unui program

Programele, scrise în limbajul de programare Borland Pascal 7.0 se compun conform unor reguli extinse și puțin mai slabe ale sintaxei standardului limbajului de programare Pascal. Însă și aceste reguli trebuie neapărat să fie respectate. Structura generală a unui program în limbajul de programare Pascal se poate împărți în cîteva părți de bază:

- antetul de program ,
- partea declarativă ,
- partea procedurilor și funcțiilor,
- partea blocului principal.

care sînt amplasate în cadrul programului în următorul mod:

1. Antet
Program nume;
2. Partea declarativă
 - {\$....} directive globale ale compilatorului
 - USES bibliotecile (pentru folosire) incluse
 - LABEL declararea etichetelor globale
 - CONST declararea constantelor globale
 - TYPE declararea tipurilor globale
 - VAR declararea variabilelor globale
3. Partea procedurilor și funcțiilor:
 - Procedure (function) antetul procedurii(funcției)
 - LABEL declararea etichetelor locale
 - CONST declararea constantelor locale
 - TYPE declararea tipurilor locale
 - VAR declararea variabilelor locale
 - BEGIN blocul principal al procedurii (funcției)
 - END;
4. Partea blocului principal
 - BEGIN blocul principal al programului
 - END.

1. În prima parte se află rîndul antetului programului. El constă din cuvîntul rezervat PROGRAM și denumirea programului. În Borland Pascal antetul nu este obligatoriu, dar e preferabil de-l folosit pentru o comoditate în citirea programului și informației suplimentare despre program.

2. În partea a doua se comunică compilatorului cu ce identificatori se notează datele, deasemenea se determină tipurile noi create de programator. În această parte a programului se poate de-i dat compilatorului unele indicații, ce determină regimurile de lucru la translarea programului. Aceste indicații se oformează în textul programului ca comentarii, ce se încep cu simbolurile ‘{\$’ și se termină cu ‘}’. Aceste comentarii pot conține indicații la includerea în textul programului a fragmentelor din alte programe (din fișierele corespunzătoare), informație despre necesitatea folosirii coprocesorului aritmetic.

Operatorul USES joacă un rol important în includerea în textul programului a modulelor de sistem din biblioteci. În acest operator se indică compilatorului, din ce bibliotecă se folosesc module în cadrul programului dat, pentru a fi incluse în program. Bibliotecă – este un set de module, fiecare din care este închis, își are numele său, se compilează aparte și se include în program ca un tot întreg cu o interfață cunoscută. Fiecare modul (bloc,UNIT) reprezintă un program ce conține declarații de tipuri și variabile, proceduri și funcții. Denumirea bibliotecilor, ce se includ în program cu ajutorul operatorului USES se despart prin virgulă:

Ex.: USES CRT,GRAPH,DOS,SYS;

După rîndul operatorului USES urmează declararea etichetelor, constantelor, tipurilor, variabilelor. (Ele pot fi amplasate în ordine aleatoare).

În partea de declarare a etichetelor se conțin, despărțite prin virgulă, numele etichetelor de trecere, care nu trebuie să fie dublate. Numele etichetei de trecere poate fi un număr întreg (0..9999), un șir de caractere sau o construcție din numere și caractere. Ex.: LABEL 1, 2, a, b, 1a, 2b;

Descrierea constantelor ce vor fi folosite în program are loc în secțiunea CONST. Constantele pot avea orice nume compus din caractere sau construcții din caractere și cifre. Însă în orice caz un nume de constantă nu poate începe cu o cifră. După numele constantei urmează semnul “=” după care este specificată valoarea constantei. O constantă poate avea valoare de diferite tipuri:

întreg, real, caracterial etc. Ex.: CONST a=19; AN=1998; MONTH='Iulie'; pi=3.14. Partea de descriere a tipurilor permite programatorului să determine tipuri noi în program.

Ex.: TYPE zi = (luni, marți, miercuri, joi, vineri, sâmbătă, duminică);

Partea de descriere a variabilelor globale conține lista variabilelor folosite în cadrul programului cu indicarea tipurilor lor.

Ex.: VAR A,B,C:INTEGER; NUME:REAL;

Părțile LABEL, CONST, TYPE și VAR se pot plasa în program în ordine aleatoare.

3. Proceduri și funcții.

“Procedură” și “funcție” sînt termenii în Pascal ce se folosesc pentru identificarea într-un mod special a unei consecutivități de instrucțiuni(subprograme). Dacă în program se folosesc proceduri (funcții), atunci e necesar de descris antetul lor indicînd lista parametrilor folosiți de ele (așa ca tipul sau valoarea variabilelor). În așa fel se realizează posibilitatea chemării unei proceduri/funcții cu diferite date și din diferite locuri a programului. Ex.: Procedure suma (VAR sum: INTEGER; VAR x, y: INTEGER); În interiorul procedurii se poate de declarat etichete, constante, tipuri, variabile care vor fi accesibile numai în interiorul procedurii/funcției unde au fost declarate. Corpul (blocul principal) funcției/ procedurii joacă același rol ca și corpul programului. Și are ca hotare cuvintele cheie BEGIN și END, cu o deosebire că după END urmează “;” și nu “.”

4. Corpul (blocul principal) programului.

Corpul programului constă dintr-o succesiune de operatori, lucrul programului începîndu - se de la primul operator. Corpul este mărginit de cuvintele cheie BEGIN și END care pot conține în interior operatori, diferite construcții, apelări la funcții/proceduri și aceleași construcții BEGIN_END incluse una în alta , dar cu “ ; “ la sfîrșit .Variabilile,constantele,etichetele,tipurile noi pot fi folosite în program numai în cazul declarării la început.

4.2 Tipuri de date.

Un program în limbajul Pascal conține o descriere a acțiunilor ce trebuie să fie executate de calculator și o descriere a datelor ce sînt prelucrate de aceste acțiuni. Acțiunile sînt descrise prin instrucțiuni, iar datele prin declarații(sau definiții). Prin tip de date se înțelege o mulțime de valori care pot fi atribuite unei variabile sau constante. Pe tipurile de date ale unui program se definesc o serie de operații, rezultatul fiind o algebră multisortată avînd ca domenii aceste tipuri. Se disting trei categorii de tipuri de date: simple(elementare), compuse(structurale) și de

referință(pointer). În general, tipurile de date sînt definite explicit prin declarațiile TYPE, iar operațiile asociate - prin declarațiile FUNCTION sau PROCEDURE, și sînt specifice programului în care apar. Există însă tipuri de date elementare de interes mai general, numite tipuri predefinite a căror definiție se consideră cunoscută și nu cade în sarcina programatorului. O serie de operații relative la aceste tipuri sînt deasemenea predefinite.

4.2.1 Tipuri de date simple

Există 3 categorii de tipuri simple: predefinite, enumerate și subdomeniu(interval). Tipurile simple se mai numesc scalare.

4.2.1.1 Tipuri predefinite .

Există 5 tipuri de date simple predefinite: INTEGER, REAL, BOOLEAN, CHAR, TEXT,.

Tipul INTEGER este o mulțime de numere întregi între cel mai mic și cel mai mare număr întreg, ce se pot reprezenta pe un calculator. Adică orice număr întreg N trebuie să respecte condiția: $-\text{maxint} \leq N \leq +\text{maxint}$. Prelucrarea unui număr întreg din afara acestui diapazon duce la rezultat greșit sau la stoparea executării programului. Asupra numerelor întregi se pot efectua următoarele operații de bază, rezultatul cărora de asemenea este un număr întreg. Toate aceste operații se îndeplinesc asupra 2 argumenti și sînt următoarele: '+' adunarea; '-' scăderea; '*' înmulțirea; div -împărțirea fără rest; mod - restul de la împărțire;

Ex.: $3+5=8$; $5-3=2$; $5*3=15$; $5\text{div}2=2$; $5\text{mod}2=1$;

După gradul de îndeplinire aceste operații sînt aranjate în felul următor: (*,div, mod, + și -). Mai există și alte operații asupra numerelor întregi cu un grad mai jos: ABS(x); x - întreg, rezultatul-modulul lui x; SQR(x); x-întreg, rezultatul-pătratul lui x; TRUNC(x); x-real, rezultatul-parteă întregă a lui x; ROUND(x); x-real, rezultatul-valoarea rotunjită a lui x: pentru $x>0$ $\text{round}(x)=\text{trunc}(x+0,5)$; pentru $x<0$ $\text{round}(x)=\text{trunc}(x-0,5)$. Deoarece tipul întreg este ordonat, asupra numerelor întregi se pot aplica și funcțiile: SUCC(x); rezultatul: următorul număr întreg după x; PRED(x); rezultatul: numărul precedent lui x. Adică operațiile și funcțiile ce pot fi aplicate asupra numerelor întregi și dau un rezultat întreg sînt: *, DIV; MOD; +; -; ABS(X); SQR(X); TRUNC(X); ROUND(X); SUCC(X); PRED(X).

Tipul REAL este mulțimea numerelor reale și ocupă un loc deosebit printre tipurile scalare. În particular tipul real nu-i ordonat și funcțiile SUCC(X) și PRED(X) nu se aplică. În comparație cu tipul întreg, tipul real conține o submulțime infinită de numere reale. Numerele reale se reprezintă incorect în calculator (cu aproximație). Din această cauză și operațiile asupra lor se execută incorect (adică după regulile aproximației). Și se recomandă ca operația de comparare a valorilor tipului real să fie ignorată din pricina unui rezultat incorect.

Următoarele operații au un rezultat real cu condiția că cel puțin un argument va fi real, iar celălalt întreg: (+) adunarea; (-) scăderea; (*) înmulțirea; (/) împărțirea. Rezultatul va fi real chiar dacă ambii argumenti vor fi reali. Mai există în Pascal funcții ce au un rezultat real independent de tipul argumentului: SIN(X); COS(X);

ARCTAN(X); LN(X); EXP(X); SQRT(X). Funcțiile ABS(X) și SQR(X) de asemenea au rezultat real cu un X real. Formatul numerelor reale este:
Ex.: 7.34501E+03;

Tipul BOOLEAN conține două elemente referite prin constantele predefinite FALSE și TRUE. Operațiile predefinite ale acestui tip sînt AND, OR și NOT și definesc o structură de algebră booleană. Tipul este succesiv în ordinea: FALSE < TRUE; 0 < 1. Operațiile logice AND (conjunție), OR (disjunție), NOT (negație) se pot aplica asupra unor argumenti de asemenea logici și au un rezultat logic.

Conjunție: AND:			Disjunție: OR:			Negație: NOT:	
X	Y	X&Y	X	Y	X v Y	X	X
1	1	1	1	1	1	1	0
1	0	0	1	0	1	0	1
0	1	0	0	1	1		
0	0	0	0	0	0		

În Pascal există funcții ce au un rezultat logic: ODD(X) dacă numărul întreg(x) este impar, rezultatul va fi : true, în caz contrar - false.
EOLN(F) = true dacă cursorul se află la sfîrșitul unui rînd, altfel - false.
EOF(F) = true dacă cursorul e la sfîrșitul fișierului, în caz contrar - false.

Tipul CHAR este o mulțime finită și ordonată de caractere ce conține litere, cifre și caracterul spațiu, adică toate caracterele ASCII. Deoarece tipul char este o mulțime ordonată sînt caractere cu index (număr de ordine) mai mare și mai mic. Conform numerelor de ordine elementele tipului char sînt aranjate astfel: A < a < B < b < ... < Z < z < 0 < 1 < ... < 9 < ?. E posibilă și altfel de ordine 0 < 1 < 2 < ... < 9 < a < b < c < ... < z < ? < A < B < ... < Z.

Ordinea conform numărului de ordine a elementelor tipului CHAR depinde de realizarea limbajului.

O constantă de tip CHAR este un element al mulțimii CHAR delimitat de apostrof.
Exemplu: '7'; 'f'; 'A'.

Pentru reprezentarea directă și indirectă a mulțimii CHAR există 2 funcții:
ORD(X) – întoarce numărul de ordine al caracterului 'x' în mulțimea char.
CHR(I) – întoarce caracterul cu numărul de ordine i.

În Pascal nu există careva operații asupra tipului char care ar avea și rezultatul char. Funcțiile PRED și SUCC aici sînt valabile. Pentru primul (ultimul) element al mulțimii char funcția pred(succ) nu-i determinată. Asupra elementelor mulțimii char se pot aplica operațiile de comparare.

4.2.1.2 Tipul enumerare

Un tip enumerat este o mulțime ordonată de valori specificate prin identificatori. Această specificare are forma: (id1, id2, ..., idn) și induce o relație de ordine astfel că id(i) < id(j) pentru i < j;

Ex.: type lună = (ianuarie, februarie, martie, aprilie, mai, iunie, iulie, august, septembrie, octombrie, noiembrie, decembrie);

var a: lună; B: real;

Operațiile aplicabile elementelor de tip enumerare sînt cele relaționale(= \leq , \gt , \lt , \leq , \gt , \lt), și funcțiile succ, pred, ord; (ord(id1)=0).

4.2.1.3 Tipul subdomeniu (interval)

Fiind dat un tip ordinal, din acest tip se poate genera un nou tip, numit tipul interval. Definiția unui interval indică valoarea constantă cea mai mică și cea mai mare din interval (în sensul numărului de ordine) și cuprinde toate valorile dintre ele. Sintaxa unui tip interval este

Type nume_tip=valoarea_minimă .. valoarea_maximă;

Valoarea minimă trebuie să fie mai mică sau egală cu valoarea maximă.

Subliniem faptul că nu este permisă definirea unui interval al tipului real, deoarece acesta nu este tip ordinal. Exemple de declarare a diferitor intervale:

Type

indice=1..10; {interval de integer}

Litera='A' .. 'Z'; {interval de char}

zile=(l,ma,mi,j,v,s,d); {tip de enumerare}

zile_lucrat=l..v; {interval de tip de enumerare}

var:

i:indice; {valori posibile: 1,2,...,10}

l:litera; {valori posibile: 'A', 'B', ... , 'Z'}

z: zile_lucrat; {valori posibile: l; ma, ...,v}

O variabilă de tip interval moștenește proprietățile variabilelor tipului de bază, dar valorile variabilei trebuie să fie numai din intervalul specificat. În cazul, cînd valoarea variabilei de tip interval nu intră în limitele intervalului definit, programul va fi executat incorect sau va fi primit mesajul de eroare.

Dacă este validată opțiunea de compilare Range Cheking (vezi meniul Options, comanda Compiler), sau dacă este prezentă directiva de compilare $\{\$R+\}$, în execuție se va verifica apartenența valorii unei variabile de tip interval la intervalul desemnat. În caz de neapartenență este semnalată o eroare de execuție și programul se oprește. Implicit nu se efectuează nici o verificare. Exemplu:

Program test;

Type cifra=0..9;

var c1,c2,c3:cifra;

Begin

$\{\$R+\}$ c1:=5; {valid}

c2:=c1+3; {valid,c1+3<=9}

$\{\$R-\}$ c3:=25; {invalid, dar nu se semnaleaza eroare}

$\{\$R+\}$ c3:=30; {invalid, se semnaleaza eroare}

end.

4.2.2 Tipuri structurate în Pascal

Pînă în momentul de față s-au precăutat numai tipurile de date simple(în special cele scalare). Fiecare valoare a oricărui din aceste tipuri de date este cunoscută și compusă dintr-o singură componentă. În cazul tipurilor de date structurate fiecare valoare a oricărui din tipuri reprezintă o structură cu valoare

nedeterminată în sensul că această valoare are mai mult de o componentă. În același timp orice dată concretă la rândul său deasemenea reprezintă o structură de date. Mai pe scurt, tipurile de date structurate reprezintă structuri alcătuite din câteva elemente de același tip, în caz simplu fiecare fiind de tip simplu.

4.2.2.1 Tipul *ARRAY*

Un tablou (masiv) este un set ordonat a unui număr fixat de valori (componente ale masivului). Toate componentele masivului trebuie să fie de unul și același tip care-l numim tipul componentelor sau tip de bază (pentru masiv). În cazul componentelor de tip real avem un vector, adică un masiv real. În cazul componentelor de tip char, masivul poate fi interpretat ca un rând de text. De obicei fiecărui masiv aparte folosit în program trebuie de-i atribuit un nume. Acest nume îl vom numi variabilă completă din cauza că valoarea ei este masivul întreg. Fiecare componentă a masivului trebuie evident notată prin indicarea numelor masivului după care urmează selectorul de componente: indexul luat în paranteze pătrate care redă regula de calculare a numărului componente necesare. Deci, pentru referirea la un element al masivului folosim sintaxa:

< nume masiv>[<index>]; Ex.: x[i];

x[i] se mai numește variabilă parțială din cauză că valoarea ei este o parte componentă a masivului și nu masivul întreg. În caz general, în calitate de index poate fi folosită o expresie, valoarea căreia determină numărul componente masivului. În același timp, în cadrul programului această expresie poate să obțină diferiți parametri. În așa fel, una și aceeași variabilă cu index în procesul îndeplinirii programului poate indica la diferite componente ale masivului.

Notă: Tipul indexului sau a expresiei ce reprezintă indexul trebuie să fie neapărat ordonat. Cele mai mari posibilități pentru prelucrarea masivelor ne oferă indexul de tipul: subdomeniu al tipului întreg.

Sintaxa de declarare a masivului este următoarea :

Var : <nume> : array [C1..C2] of <tip>.

unde <nume> - numele masivului; C1,C2 – limitele dimensiunii masivului

<tip> - tipul componentelor masivului . Ex. : var: x : array [1..4] of real;

În timp ce componentele masivului pot fi de orice tip, indexul masivului neapărat trebuie să fie tip subdomeniu al tipului întreg.

Fie un masiv X din 4 elemente. Inscripția X[1] – indică la primul element al masivului X[2] – la al doilea element etc. X[i] – la elementul cu indexul i. Anume acest index este folosit în cadrul ciclului FOR pentru prelucrarea masivelor: FOR i:=1 to N do;

În Pascal nu există nici o restricție în ceea ce privește tipul elementului masivului. Este necesar numai ca toate elementele să fie de același tip. În particular, elemente ale unui masiv pot fi de asemenea masive. Și dacă aceste din urmă sînt compuse din elemente scalare, atunci avem un masiv bidimensional numit și matrice. Dacă la următoarea etapă elementele masivului sînt de asemenea masive ,atunci avem un masiv tridimensional. Sintaxa de declarare a unui masiv bidimensional este:

ARRAY [<tip index>] OF <tip element>;

Deoarece elementele masivului sînt de asemenea masive, avem

ARRAY [<tip index>] OF ARRAY [<tip index>] OF <tip scalar>;

Această sintaxă poate fi exprimată în următoarea formă:

```
ARRAY [C1..C2 , C3..C4] OF <tip scalar>;
```

unde C1,C2,C3,C4 - mărimile masivului (limitele dimensiunii).

Ex.: X:ARRAY [1..10, 1..20] OF real;

i j

Deoarece avem și linii și coloane există doi indici: i și j.

4.2.2.2 Tipul șir de caractere

Pentru prelucrarea unor seturi de caractere, cuvinte, propoziții în Turbo Pascal este folosit tipul de date șir de caractere numit string. Valoarea unei variabile de acest tip este formată dintr-un număr de caractere oarecare. Din cauza, că variabila de acest tip are ca valoare câteva elemente de tip simplu (și anume de tip char), tipul string este un tip structurat (compus).

Tipul șir de caractere se specifică prin *string[lungime]* sau folosind numai cuvântul *string*. Exemplu:

```
Var a:string[lungime]; b:string;
```

În primul caz "lungime" reprezintă lungimea maximă a șirului de caractere, avînd valori de la 0 la 255. Un tip șir de caractere fără specificarea atributului de lungime reprezintă de fapt un șir de lungime implicit egală cu 255. Variabilele de tip *string[lungime]* pot avea ca valori orice șiruri de caractere a căror lungime nu depășește lungimea declarată. Lungimea unui șir de caractere în unele cazuri este necunoscută. Ea poate fi schimbată pe parcursul realizării programului, sau fiind declarată variabila de tip string fără specificarea lungimii. Pentru a determina valoarea actuală a lungimii unei variabile de tip string în Pascal este folosită funcția standard Length. Exemplu:

```
Var a1:string[10]; a2:string; n1,n2:integer;
```

```
Begin a1:='Programare'; a2:='Radioelectronica';
```

```
n1:=Length(a1); {n1=10}     n2:=Length(a2); {n2=16}
```

```
Writeln ('Lungimea a1 este:',n1); Writeln ('Lungimea a2 este:',n2); end.
```

După cum se vede din exemplu valoarea returnată de funcția Length este de tip întreg, în așa mod determinînd numărul de caractere ce se conțin în variabila de tip string.

Variabilele de tip șir de caractere sînt memorate în locații succesive de memorie, pe lungime+1 octeți, unde octetul de început conține lungimea actuală a șirului de caractere. Anume din acest octet de început este luată valoarea lungimii actuale a șirului de caractere de către funcția Length. Această valoare poate fi modificată de programator, printr-o instrucțiune de atribuire de forma: *sir[0]:=#nr* sau *sir[0]:=chr(ord(nr))*; unde nr este cuprins între 0 și lungime maximă admisă. Dacă nr are valoarea zero, șirul este considerat vid.

O variabilă de tip șir poate fi folosită fie în totalitatea ei, fie parțial, prin referirea unui caracter din șir. În primul caz referirea se face numai prin numele variabilei, în cel de-al doilea caz trebuie specificată între paranteze pătrate poziția caracterului din șir, cu o construcție de forma [expresie], unde rezultatul expresiei trebuie să fie o valoare întregă în intervalul 0 și lungimea declarată a șirului. De exemplu:

```

Var a:string[15];
Begin a:='student'; n:integer;
Writeln ('primul caracter:',a[1]);           {ca rezultat – litera s}
n:=Length(a); Writeln('Ultimul caracter:', a[n]); {ca rezultat – litera t}
writeln('caracterul din mijloc:', a[n-(ndiv2)]); {ca rezultat – litera d} end.

```

Asupra șirurilor de caractere se poate efectua operația de concatenare, notată cu +. Dacă s și t sînt doi operanzi de tip șir de caractere sau char, rezultatul concatenării s+t este compatibil cu orice tip șir de caractere (dar nu și cu tipul char). Dacă lungimea șirului rezultat depășește 255 de caractere, șirul se trunchiază după caracterul cu numărul de ordine 255. Exemplu:

```

Program sir;
var s1:string[10]; s2:string[20]; l:integer;
begin s1:'Turbo'; s2:=s1+'Pascal'; l:=length(s2);
writeln(s2); writeln('Lungime actuala =',l); end.

```

Operatorii relaționali =, <>, <,>, =, >= și <= compară șiruri de caractere, în conformitate cu ordonarea setului extins de caractere ASCII. Deoarece toate șirurile de caractere sînt compatibile, pot fi comparate două valori arbitrare de tip șir. O valoare de tip caracter este compatibilă cu o valoare de tip șir de caractere; cînd aceste valori sînt comparate, valoarea de tip caracter este considerată ca și cum ar fi un șir de lungime 1.

4.2.2.3 Tipul set

În Pascal o variabilă de tip set este echivalată cu o mulțime. Un tip set (mulțime) se definește în raport cu un tip de bază, care trebuie să fie un tip ordinal. Fiind dat un asemenea tip de bază, valorile posibile ale tipului set sînt formate din mulțimea tuturor submulțimilor posibile ale tipului de bază, inclusiv mulțimea vidă. Tipul mulțime se definește astfel: *type numeTip=set of tip_de_baza*

unde tip_de_bază este tip ordinal (char, interval, enumerare, boolean, byte). Cu toate că tipurile întregi sînt ordinale, nu este permis decît tipul *set of byte*.

Dacă tipul de bază are n valori, tipul mulțime va avea 2 la puterea n valori, cu restricția că n<=256.

Exemplu:

```

Type cifre=5..7; {tip interval}
mult=set of cifre; {tip mulțime}
var m:mult; {variabilă de tip mulțime}

```

Variabila m de tip mulțime poate avea 8 valori, și anume mulțimea tuturor submulțimilor posibile ale tipului mult, inclusiv mulțimea vidă. Aceste valori sînt: [5], [6], [7], [5,6], [5,7], [6,7], [5,6,7] și [], ultima valoare reprezentînd mulțimea vidă.

O valoare de tip mulțime poate fi specificată printr-un constructor (generator) de mulțime. Un constructor conține specificarea elementelor separate prin virgule și închise între paranteze pătrate. Un elemnt poate să fie o valoare precizată sau un interval de forma inf. - sup., unde valorile inf și sup precizează valorile limitelor inferioare și superioare. Atît elementul cît și limitele de interval pot fi expresii.

Dacă sup<inf, nu se generează nici un element.

Exemplu:

Program exemplu;

```
type octet=0..255;           {tip interval}
numar=set of octet;         {tip mulțime}
cuvint=set of char;         {tip mulțime}
culoare=(alb,gri,negru);   {tip enumerare}
nuanta=set of culoare;     {tip mulțime}
var: n:numar; c:cuvint; a:nuanta; i:integer;
begin n:=[2..4,8,10..12];    {elementele din constructor:2,3,4,8,10,11,12}
      i:=10; n:=[i-1..i+1, 2*i, 30]; {elemente:9,10,11,20,30}
      c:=['A'..'C','K','S'];    {elementele:'A','B','C','K','S'}
      a:=[alb,gri];           {elementele: alb, gri}      end.
```

Dacă tipul de bază are n valori, o variabilă de tip mulțime corespunzătoare tipului de bază va fi reprezentată în memorie pe n biți, depuși într-o zonă de memorie continuă de: $(n \text{ div } 8) + 1$ octeți, dacă n nu este divizibil cu 8; și $(n \text{ div } 8)$ octeți, dacă n este divizibil cu 8. De exemplu, set of char va fi reprezentat pe $256 \text{ div } 8$, adică pe 32 octeți.

Operațiile care se pot face cu valorile tip mulțime sînt:

Reuniunea: o valoare de tip ordinal c este în a+b, dacă c este în a sau b.

Diferența: o valoare de tip ordinal c este în a-b dacă c este în a și nu în b.

Intersecția: o valoare de tip ordinal c este în a*b, dacă c este în a și în b.

Relații referitoare la mulțimi: Dacă a și b sînt operanzi tip mulțime, relația $a = b$ este adevărată numai dacă a și b conțin exact aceiași termeni; altfel aob.

$a \leq b$ este adevărată dacă fiecare termen al lui a este de asemenea un termen al lui b.

$a \geq b$ este adevărată dacă fiecare termen al lui b este de asemenea un termen al lui a.

Relația de apartenență este aplicabilă în cazul mulțimilor. Fiind X o variabilă de tip ordinal t, iar a o variabilă de tip mulțime (a cărei mulțime de bază este compatibilă cu t), relația $(X \text{ in } a)$ este adevărată, dacă X este element al mulțimei a. În operațiile și relațiile de mai sus a și b trebuie să fie mulțimi compatibile.

Dacă notăm cu elmin cea mai mică valoare ordinală a rezultatului unei operații cu mulțimi, iar cu elmax cea mai mare valoare ordinală a operației, tipul rezultatului este set of a..b. Constructorii pot fi folosiți pentru scrierea mai completă a unor condiții. De exemplu, dacă ch este o variabilă de tip caracter, condiția:

```
if (ch=' T') or (ch='U') or (ch='R') or (ch='B') or (ch='O') then {...}
```

poate fi exprimată prin: `if ch in ['T','U','R','B','O'] then {...}`

iar condiția `if (ch='0') and (ch<='9') then {...}` poate fi exprimată prin:

```
if ch in ['0'..'9'] then {...}
```

Exemplu: Verificarea operațiilor cu mulțimi:

program opmult;

```
type multime=set of 1.. 10;
```

```
var a,b,int,reun,dif: multime; i:integer;
```

```
begin a:=[1..3,7,9,10]; b:=[4..6,8..10];
```

```
int:=a*b; {9,10} reun:=a+b; {1..10} dif:=a-b; {1,2,3,7}
```

```
writeln("intersecție"); for i:=1 to 10 do if i in int then writeln(i);
```

```
writeln("reuniune"); for i:=1 to 10 do if i in reun then writeln(i);
```

```
writeln('diferența');  for i:=1 to 10 do if i in dif then writeln (i);
end.
```

4.2.2.4 Tipul articol

Pînă în momentul de față au fost studiate tipurile de date compuse, elementele cărora aparțineau aceluiși tip de date (simplu sau de asemenea compus). În cazul unui masiv, toate elementele masivului erau de același tip (integer, real, char etc.) fără a fi posibilă atribuirea diferitor elemente ale masivului valori de diferite tipuri. Însă deseori apar situații, cînd este necesară prelucrarea și păstrarea unei informații mai complexe, așa ca: informația despre o persoană, despre reușita unui student, orarul lecțiilor etc. Dacă precăutăm cazul cu reușita unui student, atunci este simplu de presupus că vor fi necesare următoarele date: numele studentului, grupa, notele la examenele unei sesiuni, balul mediu calculat. Aceste date sînt legate între ele prin faptul că aparțin aceleiași persoane. Ca urmare ar fi justificată tratarea lor ca o singură valoare compusă. Însă tipurile datelor diferă între ele: numele și grupa vor fi de tip string, notele la examene – tip integer, iar balul mediu – de tip real. În consecință nu putem grupa aceste componente ca un tablou, care reprezintă o structură omogenă. Gruparea lor o permite structura de înregistrare în Pascal numită tip *articol*.

Tipul *articol* este un tip compus format dintr-un număr de componente, numite cîmpuri. Numărul componentelor poate să fie fix sau variabil. În cazul numărului de componente fix se spune că avem o structură fixă, în cel de-al doilea caz avem o structură cu variante.

Spre deosebire de tablouri, cîmpurile pot fi de tipuri diferite. Fiecare cîmp are un nume (identificator de cîmp) și un tip (numit tip de cîmp), de asemenea cum și toată structura articol își are numele său.

Forma generală a unei structuri cu articole fixe este:

```
nume_articol = record ;
nume_cîmp_1 : tip_cîmp_1;
nume_cîmp_2 : tip_cîmp_2;
... ..
nume_cîmp_n : tip_cîmp_n;
end;
```

De exemplu:

```
type data=record
  an:1900..2000;
  luna:(ian,feb,mar,apr,mai, iun, iul, aug, sep, oct, nov, dec);
  ziua:1..31;
end;
var astăzi:data;
```

Tipul unui nume de cîmp este arbitrar, astfel un cîmp poate să fie la rîndul lui tot de tip articol, deci se pot defini tipuri imbricate. Exemplu:

```
Type întîlnire = record
cînd:data; {tip articol}
ora: real;
```

```
unde:string[20];
end;
var a:întîlnire;
```

Un nume de câmp trebuie să fie unic numai în tipul articol în care a fost definit. Deci, dacă în program avem declarate mai multe articole, atunci numele câmpurilor din eceste articole pot să se repete, adică să fie aceleași, cu condiția că numele articolelor vor fi unice. Exemplu:

```
Type student1=record
  Nume: string; Grupa: string; Nota1: integer; Nota2:integer; Media: real;
End;
  student2=record
  Nume: string; Grupa: string; nota1: integer; nota2:integer; nota3:integer;
media: real;
End;
```

Pentru ușurarea redactării programelor, declararea câmpurilor de același tip ale unui articol se face ca și în cazul declarării câtorva variabile simple de același tip – ele sînt despărțite prin virgulă. Exemplu:

```
Type student=record
  Nume, Grupa: string; nota1, nota2, nota3: integer; media: real;
End;
```

Valorile variabilelor de tip articol pot lua parte la diferite operații, calcule, formule. Însă aceste valori nu sînt valorile articolului întreg, ci valorile concrete ale câmpurilor articolului. Un câmp al unei variabile de tip articol este referit prin numele variabilei și numele de câmp, separate printr-un punct. Exemplu:

```
Astăzi.an:=1991; astăzi.luna:=nov; astăzi.ziua:=13; a.ora:=13,40; a.cînd.ziua:=5;
Operațiile care pot fi efectuate asupra câmpurilor tipului articol sînt operațiile aplicabile tipului de date, cărui îi aparține câmpul corespunzător. Exemplu: Calcularea balului mediu la sesiune al unui student
```

```
Program p1;
Type student=record
  Nume, Grupa: string; nota1, nota2, nota3: integer; media: real;
End;
Var a:student;
Begin a.nume:='vasile'; a.grupa:='ABC-981';
      a.nota1:=8; a.nota2:=9; a.nota3:=7;
      a.media:=(a.nota1+ a.nota2+ a.nota3) / 3; writeln('media=',a.media);
end.
```

Pentru a ușura scrierea anevoioasă în cazul referirii unui câmp al articolului prin nume de variabile și câmpuri separate prin punct se poate folosi instrucțiunea *with*. Ea permite o referire prescurtată a câmpurilor unui articol. În interiorul unei instrucțiuni *with* câmpurile uneia sau a mai multor variabile de tip articol pot fi referite folosind numai numele câmpurilor lor. Sintaxa instrucțiunii este: *with listă_de_variabile_tip_articol do instrucțiune;* unde "listă_de_variabile_tip_articol" este una sau mai multe variabile tip articol, referirea cărora va avea loc prin intermediul instrucțiunii *with*. Exemplu:

```

type calendar = record
    an:1990..2000; luna:1..12; ziua:1..31; end;
var data:calendar;
begin with data do
    if luna = 12 then begin
        luna:=1; an:=an+1;
    end;
    else luna=luna+1;
end.

```

Această instrucțiune este echivalentă cu următoarea:

```

if data.luna = 12 then begin
    data.luna:=1;
    data.an:=data.an+1; end
else data.luna:=data.luna+1;

```

În interiorul unei instrucțiuni with, la întâlnirea unui nume de variabilă prima dată se testează dacă variabila poate fi interpretată ca un nume de câmp al unui articol. Dacă da, variabila va fi interpretată ca atare, chiar dacă în acel moment este accesibilă și o variabilă avînd același nume

```

Type punct = record
    x,y:integer; end;
var x:punct; y:integer;

```

în cazul nostru atît x, cît și y, pot să reprezinte fie o variabilă, fie un câmp al articolului. Dar în instrucțiunea: *with x do begin x:=1; y:=2; end;* identificatorul x situat între with și do se referă la variabila de tip articol x, iar identificatorul x între begin și end se referă la variabila de tip integer, adică la câmpul articolului. În cazul necesității referirii din una și aceeași instrucțiune with a mai multor variabile tip articol se procedează în felul următor:

```

with v1,v2,...,vn do instructiune;

```

Această instrucțiune este echivalentă cu următoarea:

```

with v1 do with v2 do {...} with vn do instructiune;

```

Articol cu variante. Uneori apare necesitatea să se includă în structura unui articol informații care depind de o altă informație deja prezentă în articol. Fie că se prelucrează informația despre 3 persoane în ceea ce privește numele, salariul și pregătirea profesională. Pregătirea profesională poate fi elementară, medie și superioară. Și este impusă următoarea condiție: (a)dacă pregătirea este superioară, atunci apare necesitatea de informația despre anul absolvirii și validitatea licențierii ; (b)dacă pregătirea este medie, atunci se mai adaugă și balul mediu de absolvire; (c)dacă pregătirea este elementară- nu trebuie nici o informație suplimentară. Problema aceasta poate fi rezolvată cu ajutorul articolului cu structură fixă, declarînd toate câmpurile susnumite și folosindu-le numai în caz de necesitate. În acest caz unele câmpuri vor fi declarate, dar nu vor fi folosite în program, ceea ce duce la folosirea nerațională a resurselor calculatorului. În Pascal există tipuri articol, care au o structură flexibilă (cu variante). Și anume în cazul respectării unei condiții anumite apare câmpul necesar adăugător, iar în cazul, cînd condiția nu este respectată, în

componenta articolului acest câmp lipsește. Astfel de tipuri sînt articole cu variante. Forma generală a unei astfel de structuri este:

```
Nume_articol = record
  nume_cîmp_1 : tip_1; {cîmpurile fixe}
  ... ..
  nume_cîmp_n : tip_n;
case
cîmp_sel : tip_sel of {cîmpuri variabile}
const_1 : (cîmp_var1_1 : tip_var1_1; cîmp_var1_2 : tip_var1_2; ... cîmp_var1_n :
tip_var1_n) const_2 : (cîmp_var2_1 : tip_var2_1; cîmp_var2_2 : tip_var2_2; ...
cîmp_var2_m : tip_var2_m)
... end;
```

Diferitele variante sînt selectate de valorile posibile ale lui tip_sel (tip selector) al câmpului cîmp_sel (cîmp selector). Fiecare din variantă este "etichetată". Aceste "etichete" notate mai sus cu const_1, const_2, ... conțin diferite valori ale tipului selector; valorile specificate pot fi scrise și sub forma de interval de valori, astfel: inf. - sup. Fiecare valoare a tipului selector trebuie să apară într-una din aceste "etichete".

Dacă o variantă este vidă, adică nu are nici un câmp, forma ei este *const: ()*. O listă de câmpuri poate să conțină numai o singură clauză *case*, plasată după câmpurile fixe. Tipul selector trebuie să fie ordinal.

Exemplu: *Type studii = (elem, medii, sup);*

```
inform = record
Nume : string[20];
salar : 5000..15000;
Case preg : studii of
{preg-cîmp selector, studii-tip selector. Cîmpul preg de tip studii}
elem : (); {nu avem nici un câmp, variantă vidă}
medii : (bal : real);
{dacă câmpul selector = medii, atunci apare un câmp nou: bal tip real}
sup : (an_abs : 1950..2000; licența : boolean) {cînd preg = sup}
end;
var x, y, z : inform;
begin
x.nume := 'JOHN'; x.salar := 6 0 0 0; x.preg := elem;
y.nume := 'MARY'; y.salar := 7000; y.preg := medii; y.bal := 8.50;
z.nume := 'BILL'; z.salar := 8000; z.preg := sup; z.an_abs := 1990; z.licența := true;
end.
```

Numai o singură variantă poate să fie activă la un moment dat. Astfel, dacă este activă varianta medii, prin *y.preg := medii*, nu are sens o atribuire de forma *y.anabs := 1980*.

Lungimea unei variabile de tip articol este egală cu lungimea părții fixe propriu zise plus lungimea câmpului selector, plus lungimea celei mai mari părți variabile (în caz că există).

4.2.2.5 Tipul reper

În timpul de astăzi problema memoriei în tehnica de calcul este cea mai stringentă. Pe parcursul dezvoltării sale tehnica de calcul se perfecționează și pune la dispoziția utilizatorului posibilități mai vaste de lucru. Odată cu dezvoltarea tehnicii de calcul se dezvoltă cu pași giganti și asigurarea soft a ei. Astfel apar noi aplicații cu cerințe mai mari către resursele calculatorului. În cele mai dese cazuri problema folosirii acestor aplicații constă în insuficiența memoriei operative RAM. Astfel de situații pot apărea și în cazul programării Pascal, când se prelucrează un volum mare de date și programele devin voluminoase și complicate necesitând un volum mare de memorie RAM. Pentru așa cazuri Pascal își are instrumentul său numit variabile dinamice.

În Turbo Pascal variabilele pot fi statice sau dinamice. Variabilele statice sînt alocate în memorie în timpul compilării, iar locul ocupat de ele în memorie nu poate fi modificat în execuție; ele există pe durata întregii execuții a blocului (program, procedură, funcție). Însă în Pascal există variabile care pot fi create și distruse dinamic în timpul execuției programului; o astfel de variabilă este denumită variabilă dinamică.

Crearea și distrugerea variabilelor dinamice se realizează cu procedurile `New` și `GetMem` respectiv `Dispose` și `FreeMem`. Aceste proceduri alocă, respectiv eliberează spațiu de memorie pentru variabilele dinamice. Adresa zonei de memorie alocată unei variabile dinamice va fi depusă într-o variabilă de tip special, numit reper. Lungimea zonei de memorie atribuită unei variabile dinamice depinde de tipul variabilei dinamice: în funcție de tip se alocă un număr variabil de octeți variabilei respective. De exemplu, dacă tipul variabilei dinamice este întreg, se alocă 2 octeți, dacă tipul este real, se alocă 6 octeți. În consecință, variabila de tip reper care va conține adresa zonei alocate variabilei dinamice trebuie să comunice procedurilor de alocare de memorie tipul variabilei dinamice. Menționăm că variabilele dinamice sînt alocate într-o zonă specială de memorie, denumită "heap".

Definirea unui tip reper se poate face în secțiunea `type`, în felul următor:

```
type nume_reper = ^tip_variabilă_dinamică;
```

unde semnul `^` semnifică o adresă. Mulțimea valorilor reper de tip "`nume_reper`" constă dintr-un număr nelimitat de adrese; fiecare adresă identifică o variabilă de tip "`tip_variabilă_dinamică`". La această mulțime de valori se mai adaugă o valoare specială, numită `nil`, care nu identifică nici o variabilă.

Limbajul permite ca în momentul întâlnirii tipului variabilei dinamice acesta să nu fie cunoscut încă (referire înainte); acest tip însă trebuie declarat mai târziu, în aceeași declarație de tip. De exemplu, secvența următoare este corectă:

```
Type rep = ^art; {referire inainte}
```

```
art = record x,y:integer; end;
```

```
  var  r1,r2:rep; {reperarea variabilelor tip art}
```

```
      r3:^integer; {reperarea var. dinamice}
```

```
      r4:^char; {de tip intreg si caracter}
```

O altă facilitate a limbajului constă în posibilitatea utilizării tipurilor care se autoreferă (sînt definite recursiv). De exemplu,

```
type lista = ^articol; articol = record
```

a,b: integer; urmator : lista; end;
var : l:lista;

Aici tipul reper "lista" reperează un tip "articol", în care câmpul "următor" la rindul lui este de asemenea de tip "listă". Această facilitate poate fi folosită de exemplu la alcătuirea listelor înlănțuite.

După crearea unei variabile dinamice a cărei adresă este depusă într-o variabilă de tip reper, ea poate fi accesată prin așa zisa dereperare: numele variabilei de tip reper este urmat de semnul ^. Acest semn poate urma și după un alt calificator (de câmp, de tablou, etc.).

Dereperarea unei variabile de tip reper cu conținut nil declanșează o eroare de execuție. Variabilele de tip reper sînt alocate pe 4 octeți (2 octeți pentru memorarea adresei de segment, 2 octeți pentru memorarea deplasamentului).

Operațiile relaționale care sînt permise cu operanzii de tipul reper compatibile sînt = și < >. Dacă p1 și p2 sînt două valori tip reper compatibile, relația p1 = p2 este adevărată dacă sînt egale părțile de segment și de deplasament. Astfel pot exista două valori de tip reper care indică aceeași locație, dar totuși ele nu sînt egale în sensul de mai sus. De exemplu, \$0040:\$0049 și \$0000:30449 indică aceeași adresă fizică, totuși în sensul limbajului ele nu sînt egale. Adresele returnate de procedurile New și Getmem sînt totdeauna normalizate, adică partea de deplasament este în intervalul \$0000...\$FFFF. Astfel comparările vor fi efectuate corect. Cînd însă valorile de reper sînt create cu funcția Ptr, trebuie de acordat o mare atenție la compararea lor, deoarece deplasamentele nu sînt neapărat normalizate. Exemplul 1:

Operații simple cu variabile dinamice:

```
type pc=^char; pintrg= ^ integer; pcmp=^art;  
art=record x:integer; y:array[1..2] of integer; end;  
var c:pc; intrg:pintrg; cmp:pcmp;  
begin new(c); {crearea unei var. dinamice tip caracter}  
c^ := '*'; {încărcarea variabilei create}  
new(intrg); {crearea unei variabile dinamice}  
intrg^:=123; {de tip întreg și încărcarea ei}  
new(cmp); {crearea unei variabile dinamice}  
cmp^.x:=4; {de tip articol și încărcarea}  
cmp^.y[1]:=5; {cîmpurilor variabilei}  
cmp^.y[2] :=6; {...}  
writeln(c^); writeln(intrg^); write(cmp^.x); write(' ',cmp^.y[1]);  
writeln(' ',cmp^.y[2]); {variabilele dinamice nu mai sînt necesare}  
dispose(cmp); {distrugerea variabilei dinamice}  
dispose(intrg);  
dispose (c); {se poate reutiliza spațiul de memorie ocupat anterior}  
{...} end.
```

4.2.2.6 Tipul pointer

Tipul predefinit pointer este un tip reper fără tip de bază. Astfel, o variabilă de acest tip poate să repereze o variabilă de orice tip, motiv pentru care acestui tip i se mai spune și tip reper liber. Declararea unei variabile de acest tip se face de exemplu

astfel: *Var a:pointer*; Variabilele de tip pointer nu pot fi dereperate: scrierea simbolului ^ după o astfel de variabilă constituie o eroare. Variabilele de tip pointer sînt utilizate pentru memorarea valorii unor variabile de tip reper legat. Unei variabile de acest tip îi poate fi atribuită și valoarea predefinită nil. Observație: *Valori de tip reper pot fi create și cu operatorul @, și cu funcția standard Ptr. Aceste valori sînt tratate ca și cum ele ar fi repere pentru variabile dinamice.*

```
Exemplu:  program test;
           type e=integer; pe=^e; pin=^integer;
           var  alfa:e; beta:integer; p,p1:pointer; vpe:pe; vpi:pin;
begin writeln('Test cu tipul pointer');
  alfa:=5; beta:=6;
  vpe:=@alfa; vpi=@beta;
  writeln('alfa=',vpe^, 'beta=', vpi^);
  p1:=@alfa;
  {writeln('alfa=',p2^); {eroare 64: Cannot Read or Write variables of this type }
                        {(Nu pot fi citite sau scrise variabile de acest tip) }
  {alfa1^:=p1^};      {eroare 26:Type mismatch (Incompatibilitate de tip) }
  p:=@beta; vpi:=p; vpe:=p1; writeln('alfa=',vpe^,'beta=',vpi^);
readln; end.
```

Pe ecran vor fi afișate următoarele rezultate:

Test cu tipul pointer

alfa=5 beta=6

alfa=5 beta=6

4.2.2.7 Fișiere în Pascal

4.2.2.7.1 Generalități despre fișiere.

Prin fișier în general se înțelege o structură de date care constă dintr-o secvență de componente. Fiecare componentă din secvență are același tip. Numărul acestor componente nu este fixat. Aceasta este o caracteristică prin care se distinge clar fișierul de tablou. La un moment dat însă, este accesibilă direct numai o singură componentă a secvenței. Celelalte componente sînt accesibile progresînd secvențial în fișier. Progresarea în componentele unui fișier se realizează prin subprograme de citire și de scriere. Datele fișierului sînt stocate de obicei pe un suport magnetic. În limbajul Pascal, pot fi definite trei structuri de tip fișier: 1)fișier cu tip; 2)fișier text; 3)fișier fără tip.

Să notăm cu *f* o variabilă de tip fișier. Înainte ca variabila să fie utilizată, ea trebuie asociată cu un fișier extern, prin apelul procedurii *Assign*. În general, fișierul extern este un fișier pe disc, dar variabila de fișier poate fi asociată și cu un dispozitiv (de exemplu tastatura, ecran). Fișierul extern marchează informațiile scrise în fișier sau furnizează informațiile depuse. După ce s-a stabilit asocierea cu un fișier extern, fișierul trebuie "deschis". Această deschidere pregătește fișierul pentru citire și/sau scriere. Un fișier existent poate fi deschis cu ajutorul procedurii *Reset*. Un fișier nou poate fi deschis cu procedura *Rewrite*. Fișierele de tip text deschise cu *Reset* permit doar operații de citire; fișierele de tip text deschise cu procedura *Rewrite* sau *Append* permit numai operații de scriere. Fișierele cu tip sau fără tip permit atât citirea cât și

scrierea, indiferent dacă ele au fost deschise cu *Rewrite* sau cu *Reset*. Fișierele text standard Input și Output sînt deschise automat în momentul în care începe execuția programului. Input este un fișier text care permite numai operații de citire și este asociat cu claviatura. Output este un fișier text care permite numai operații de scriere și este asociat cu ecranul. Fiecare fișier este o secvență liniară de componente, fiecare componentă avînd tipul de bază al variabilei fișier. Fiecare componentă a fișierului are asociat un număr, numit numărul componentei. Prima componentă a fișierului este considerată avînd numărul de componentă zero. Cea de a doua componentă are numărul 1 ș.a.m.d. În mod normal accesul la componentele fișierului este secvențial. Aceasta înseamnă, că atunci cînd se citește o componentă cu ajutorul procedurii standard *Read*, sau cînd se scrie o componentă cu ajutorul procedurii standard *Write*, poziția curentă de fișier se deplasează la următoarea componentă, în sensul ordonării numerice. În afară de acest mod de acces, fișierele cu tip și fișierele fără tip permit și accesul direct (aleator) la componentele fișierului. Accesul direct se bazează pe procedura de căutare *Seek*, care mută poziția curentă în fișier pe o componentă specificată. După această poziționare componenta astfel aleasă poate fi citită. Funcțiile standard *FilePos* și *FileSize* permit determinarea poziției curente în fișier, respectiv a dimensiunii actuale a fișierului. Cînd prelucrarea componentelor unui fișier se termină, fișierul trebuie închis cu procedura standard *Close*. După ce fișierul a fost închis, se vor actualiza datele fișierului extern asociat. După fiecare apel de procedură și funcție standard de intrare/ieșire se testează automat reușita operației de intrare/ieșire. În cazul în care apare o eroare, programul se termină și se afișază un mesaj de eroare în execuție. Directiva de compilare *\$I* permite ca această eroare să fie tratată în program. În stare decuplată *{\$I-}*, programul nu se oprește la o eroare de intrare/ieșire. Pentru a analiza cauza erorii, se apelează funcția standard *IOResult*, care în caz de operație reușită returnează valoarea zero, iar în caz de eșec returnează o valoare diferită de zero, care codifică natura erorii.

Ștergerea fișierului extern asociat unui fișier închis poate fi realizată cu procedura *Erase*. În procesul de citire a unui fișier, faptul că s-a ajuns la sfîrșitul fișierului poate fi urmărit cu funcția *Eof*. Această funcție returnează valoarea logică *true* dacă s-a ajuns la sfîrșitul fișierului, respectiv *false* în caz contrar.

4.2.2.7.2 Fișiere cu tip

Fișierul cu tip este o secvență de componente, fiecare componentă avînd același tip, numit tipul de bază al fișierului. O variabilă de acest tip poate fi declarată printr-o declarație de forma:

var nume_fișier : file of tip_de_bază; unde *tip_de_bază* este un tip arbitrar, exceptînd tipul fișier.

Exemplul 1: Crearea unui fișier cu tip cu nume extern *persoana.txt*. Componentele sînt de tip articol, cu informațiile referitoare la angajații unui institut.

Program fișiercutip;

type angajat=record

marca:integer; nume:string[20]; salar:integer; end;

var f:file of angajat; { variabila f este un fișier tip articol}

a:angajat; contin:char; k:integer; {răspunsul operatorului}

```

begin
assign(f, 'persoana.txt');    {asocierea variabilei f cu fișierul de pe disc persoana.txt}
rewrite(f);                  {deschiderea fișierului pentru prima dată}
repeat writeln('marca='); readln(a.marca);
    writeln('nume='); readln(a.nume);
    writeln('salariu='); readln(a.salar);
write(f,a);                  {scriere de componenta în fișier}
writeln('Continuam? d/n'); readln(contin);
until upcase(cont)='N';
k:= filesize(f)              {funcția FilSize determină mărimea actuală a fișierului}
writeln('Numarul de componente în fișier=',k);
close(f);                    {închiderea fișierului}
end.

```

Exemplul 2. Acces direct la fișierul persoana.txt. Se livrează informațiile referitoare la un angajat, pe baza numărului de componentă asociat unui angajat.

```

Program accesdirect;
type angajat=record
    marca:integer; nume:string[20]; salar:integer; end;
var f :file of angajat; a:angajat; fs:integer; nr:integer;
begin
assign(f, 'persoana.txt') {asocierea variabilei f cu fișierul de pe disc persoana.txt}
reset(f);                 {deschiderea fișierului existent pentru citire/scriere}
fs:=filesize(f);          {funcția FilSize determină mărimea actuală a fișierului}
writeln('nr.componentei='); readln(nr);
if nr>fs then writeln('eroare, fișierul e mai mic') else
begin seek(f,nr);         {poziționare pe componenta cu numărul nr.}
read(f,a);                {citirea compnentei selectate din fișier}
writeln('marca=',a.marca); writeln('nume=', a.nume); writeln('salar=',a.salar);
end; close(f);            {închiderea fișierului}
end.

```

4.2.2.7.3 Fișiere de tip text

Fișierul text conține caractere structurate pe linii, fiecare linie fiind terminată cu un caracter de sfârșit de linie (EOLN caracter). Lungimea liniilor este variabilă. Caracterul de sfârșit de linie este de regulă CR[ENTER] (ASCII #13). Un fișier text este terminat cu un caracter de sfârșit de fișier CTRL-Z. Un fișier text este declarat prin tipul predefinit *text*, de exemplu: *var f:text*. Un fișier text nu este echivalent cu un fișier de tip *file of char*. Lungimea liniilor fiind variabilă, poziția unei linii în cadrul fișierului nu este calculabilă. În consecință, la fișiere text accesul nu poate fi decât secvențial. Asocierea numelui fișierului la suportul extern este realizată cu procedura *Assign*. Deschiderea fișierului poate fi realizată prin trei subprograme standard. Un fișier nou se deschide cu procedura *Rewrite*, un fișier existent poate fi deschis fie la începutul fișierului, fie la sfârșitul lui, în vederea adăugării liniilor noi. Deschiderea la început se realizează cu procedura *Reset*, iar la sfârșit cu procedura *Append*. Pentru fișiere text, formele speciale ale subprogramelor *Read* și *Write* permit

citirea și scrierea nu numai a valorilor de tip caracter, ci și a valorilor de tip întreg, real și de tip șir de caractere. De exemplu, *Read(f,i)*, unde *i* este o variabilă de tip întreg, va citi o secvență de cifre, care vor fi interpretate ca un întreg zecimal, și care va fi depus în variabila *i*. Detectarea caracterelor de sfârșit de linie poate fi realizată cu funcția *Eoln*, care returnează valoarea true dacă s-a ajuns la sfârșitul liniei curente. Funcția *SeekEoln* este similară cu *Eoln*, exceptând faptul că se sare peste blanc-uri și tab-uri, după care se testează starea de sfârșit de linie. Starea de sfârșit de fișier poate fi testată cu funcția *Eof*, precum și cu funcția *SeekEof*, aceasta din urmă sare peste blanc-uri și tab-uri, după care se returnează true dacă s-a ajuns la sfârșitul fișierului. Procedura *SetTextBuf* permite atașarea unui tampon de intrare/ieșire de lungime dată la un fișier text. Golirea tamponului unui fișier text deschis pentru scriere poate fi realizată cu procedura *Flush*.

Așa cum s-a arătat la generalități despre fișiere, există două fișiere text standard: *Input* și *Output*. Fișierul *Input* este destinat numai pentru operații de citire și este asociat cu fișierul standard de intrare al sistemului de operare (de regulă claviatura). Fișierul *Output* este destinat numai pentru operații de scriere și este asociat cu fișierul standard de ieșire al sistemului de operare (de regulă ecranul). Aceste fișiere sînt deschise automat înainte de începutul execuției, ca și cum ar fi prezente instrucțiunile

```
Assign(Input, ' '); Reset(Input); Assign(Output, ''); Rewrite(Output);
```

Aceste fișiere sînt închise automat după ce s-a terminat execuția programului.

Unele proceduri și funcții de intrare/ieșire permit ca numele fișierului să nu fie specificat în lista de argumente; în acest caz se presupune că fișierul text implicit este sau *Input*, sau *Output*, în funcție de natura subprogramului. De exemplu, *Read(x)* este echivalent cu *Read(Input,x)*, iar *Write(x)* este echivalent cu *Write(Output,x)*. Un fișier text deschis cu *Reset* suportă numai proceduri și funcții orientate spre citire. Analog, un fișier deschis cu *Rewrite* sau *Append* permite utilizarea acelor proceduri și funcții, care sînt orientate spre scriere.

Exemplul 1: Crearea unui fișier text 'carte.txt'; liniile fișierului sînt introduse de la tastatură (fișier INPUT)

```
Program crtext;
```

```
  var c:char; f:text;
```

```
begin
```

```
  assign(f, 'carte.txt'); {asocierea variabilei f cu fișierul extern carte.txt}
```

```
  rewrite(f);           {deschiderea fișierului nou pentru scriere}
```

```
  while not eof do     {control sfârșit de fișer: eof(INPUT)}
```

```
  begin while not eoln do {control sfârșit de linie: eoln(INPUT)}
```

```
  begin read(c);      {citire de la tastatura: read(INPUT,c)}
```

```
  write(f,c);         {variabila c va fi scrisă în fișier} end;
```

```
  readln;             {readln(INPUT)}
```

```
  writeln(f);         {scrie EOLN în f} end;
```

```
  close(f);           {scrie EOF în f și-l inchide}
```

```
end
```

Exemplul 2. Afișarea fișierului creat anterior pe ecran, cu numerotarea liniilor.

```
Program extext;
```

```

var c:char; f:text; numlin:integer;
begin assign(f,'carte.txt'); reset(f); numlin:=0;
while not eof(f) do begin
numlin:=numlin+1; write(numlin,' ');
while not eoln(f) do begin
read(f,c); write(c); end;
readln(f); {citeste EOLN din f}
writeln; end;
close(f); end.

```

4.2.2.7.4 Fișiere fără tip

Fișierele fără tip sînt canale de intrare/ieșire de nivel inferior, utilizate în primul rînd pentru accesul direct la orice fișier-disc, indiferent de tipul și de structurarea internă a fișierului. În operațiile de intrare/ ieșire, la fișierele fără tip informațiile sînt transferate direct între fișierul de pe disc și variabile, economisîndu-se astfel spațiul necesar zonei tampon. Un fișier fără tip este compatibil cu orice fișier (cu tip sau text). O variabilă de acest tip se declară cu tipul predefinit "file", și nimic altceva; de exemplu: *var f:file;*

Pentru fișiere fără tip, procedurile de deschidere *Reset* și *Rewrite* permit folosirea unui parametru auxiliar. Acest parametru specifică lungimea unei componente a fișierului. Valoarea acestui parametru, din motive istorice, este 128. Se recomandă alegerea valorii 1, deoarece această valoare permite reflectarea corectă a dimensiunii exacte a fișierului (cînd această valoare este 1, nu sînt posibile componente fracționate).

Exceptînd procedurile *Read* și *Write*, toate procedurile și funcțiile standard utilizabile pentru fișierele cu tip sînt permise și pentru fișierele fără tip. În locul procedurilor *Read* și *Write*, pentru realizarea transferurilor rapide de date sînt folosite procedurile *BlockRead* și *BlockWrite*. Procedura *BlockRead* citește din fișierul fără tip un număr precizat de componente, care se depun în memorie de la o adresă specificată. La revenire din procedură, o variabilă - care se poate specifica opțional - va conține numărul componentelor citite efectiv. Dacă această variabilă nu este specificată și dacă nu s-a reușit citirea tuturor componentelor, va apare o eroare de intrare/ieșire.

Procedura *BlockWrite* scrie în fișierul fără tip un număr precizat de componente, componentele fiind luate de la o adresă specificată. La revenire din procedură o variabilă opțională va conține numărul componentelor scrise efectiv. Dacă această variabilă nu este specificată și dacă nu s-a reușit scrierea tuturor componentelor, atunci va apare o eroare de intrare/ieșire.

Este permis atît accesul secvențial, cît și cel direct, datorită faptului că toate componentele au aceeași lungime. Poziționarea pe o componentă dorită se realizează cu procedura *Seek*. Numerotarea componentelor începe de la zero. Numărul componentelor se determină cu funcția *FileSize*, iar numărul componenteii actuale se determină cu funcția *FilePos*. Procedura *Truncate* permite trunchierea fișierului, pornind de la componenta actuală din fișier. Exemplu. Programul următor prezintă o utilizare a fișierelor fără tip:copierea unui fișier de tip arbi trar.

```

program copiere;
var sursa,dest:file;
citit,scris:word; tampon:array[1..2048] of char;
numsurs,numdest:string[14];
begin      writeln('Fisierul sursa:');      readln(numsurs);
assign(sursa,numsurs);
reset(sursa,1);      {lungimea componenta = 1}
writeln('Fisierul destinatie:'); readln(numdest);
assign(dest,numdest); rewrite(dest,1);
writeln (' Copiere de', FileSize (sursa), ' octeti...');
repeat
BlockRead(sursa,tampon,2048,citit);
BlockWrite(dest,tampon,citit,scris);
until (citit = 0) or (scris <> citit); close(sursa); close(dest);
end.

```

4.3 Instrucțiuni

O instrucțiune este alcătuită dintr-o etichetă opțională prin intermediul căreia poate fi referită din alte instrucțiuni, urmată de instrucțiunea propriu-zisă, prin care este descrisă acțiunea realizată în momentul execuției sale. Se face distincție între instrucțiuni simple (instrucțiunea de atribuire, apelul de procedură, instrucțiunea de efect nul și instrucțiunea de transfer necondiționat) și instrucțiunile structurate (instrucțiunea compusă, instrucțiunile iterative, instrucțiunile condiționale, instrucțiunea with ș.a.).

4.3.1 Instrucțiuni simple.

O instrucțiune se numește simplă dacă nu conține alte instrucțiuni.

4.3.1.1 Instrucțiunea de atribuire.

Această instrucțiune are forma $V:=E$; unde V – o variabilă, iar E – o expresie. Prin execuția instrucțiunii de atribuire, expresia E este evaluată și rezultatul se atribuie variabilei V . Variabila și rezultatul evaluării trebuie să fie de tipuri identice sau tipul uneia să fie un subdomeniu al celeilalte, sau ambele să fie subdomenii ale aceluiași tip, iar rezultatul în subdomeniul variabilei. Se admite ca excepție cazul când variabila este de tipul real, iar rezultatul de tipul integer sau un subdomeniu al acestuia. În dependență de tipul variabilei și rezultatului există: atribuire aritmetică, logică, caracterială. Atribuirea aritmetică servește pentru atribuirea unei valori variabilei de tipul real sau integer. În calitate de expresie pot fi diferite funcții și operații ce pot fi aplicate asupra tipurilor real și integer. În cazul atribuirii logice de partea stîngă se află o variabilă de tipul boolean, iar în partea dreaptă o expresie logică pentru calcularea unei valori logice(true sau false).De obicei la atribuirea caracterială în partea stîngă se află o variabilă tip char, iar în dreapta - o expresie caracterială ce redă regula de determinare a valorii de tipul char, adică un caracter aparte. Ex.: a = 'l'; b :=succ (l);

4.3.1.2 Instrucțiunea apel de procedură

Această instrucțiune se inserează în program în locul în care se dorește executarea instrucțiunilor specificate de o declarație de procedură asupra unităților particulare transmise ei din locul de apel. Sintactic apelul de procedură poate fi reprezentat astfel: P(L), unde P – numele procedurii, L – lista parametrilor actuali.

Parametrii actuali din L trebuie să corespundă ca număr, ordine și tip cu parametrii formali specificați în antetul declarației de procedură. Ei pot fi expresii, funcții, proceduri. Ex.: o procedură cu antetul procedure P(x, y: integer; var z, t: real) se poate apela prin P(a, b, u, v) sau P(3, 5, u, v) sau P (3, 2*(a + b - c), u, v) etc. unde a, b, c desemnează variabile întregi. Instrucțiunea de apel la procedură va fi analizată mai concret în punctul 4.4.

4.3.1.3 Instrucțiunea de transfer necondiționat.

Instrucțiunile unui program sînt executate secvențial, așa cum apar scrise în textul programului. Instrucțiunea de transfer necondiționat oferă posibilitatea de a întrerupe această secvență și a relua execuția dintr-un alt loc al textului. Această instrucțiune are forma: *GOTO e* unde *e* – etichetă declarată prin *label*. Declararea etichetei *e* prin *label* este obligatorie. Execuția instrucțiunii *GOTO e* are ca efect transferul controlului la instrucțiunea precedată de *e*.

Instrucțiunea *GOTO e* și instrucțiunea marcată cu *e* trebuie să îndeplinească condiția: instrucțiunea precedată de *e* conține instrucțiunea *GOTO e* sau face parte dintr-o secvență de instrucțiuni *s*, iar instrucțiunea *GOTO e* este una, sau e conținută în una dintre instrucțiunile secvenței *s*.

Ex.: *label 5,8*
Const a=1,4; b=7,03;
Var x, y, z: real
Begin
Writeln ('y'); readln (y); X:=sqr (y);
5: if x>5 then goto 8
Z:= x+a+b;
X:= sqrt(z);
Goto 5;
8: end;

4.3.1.4 Instrucțiunea de efect nul.

Executarea acestei instrucțiuni nu are efect asupra variabilelor programului (starea acestora rămîne neschimbată). În textul programului instrucțiunea de efect nul nu este reprezentată prin nimic dar, deoarece instrucțiunile sînt despărțite între ele prin “;”, prezența sa este marcată de apariția acestui delimitator. Ex.: *a:=b+3;;;*

4.3.2 Instrucțiuni structurate.

O instrucțiune se numește structurată cînd are în componența sa cîteva instrucțiuni simple (2 și mai mult).

4.3.2.1 Instrucțiunea compusă BEGIN-END.

Uneori, pentru compilarea corectă a unui program în limbajul Pascal este nevoie ca într-un loc oarecare al construcției sintactice să fie prezent numai un singur operator (instrucțiune), în timp ce după mersul algoritmului în acest loc trebuie să fie un set de instrucțiuni. Pentru rezolvarea acestui conflict dintre sintaxa limbajului și mersul algoritmului este folosită instrucțiunea compusă begin - end, care unește un set de operatori simpli într-un tot întreg și care este înțeles de compilator ca o singură instrucțiune aparte. Cuvintele cheie la instrucțiunea compusă sînt <begin> și <end>. Sintaxa este următoarea : begin <instr.1; instr.2;...; instr. n> end.

Instrucțiunea compusă begin – end poate conține una sau mai multe instrucțiuni în corpul său (despărțite prin ‘;’). Executarea acestei instrucțiuni se reduce la executarea consecutivă a tuturor operatorilor incluși în corpul său.

Ex.: begin a: =b+3 end; begin y: =s + cos s; z: = y - 4ac; end;

Begin I:=1; begin c: = a + b; y: = sin c end; end;

4.3.2.2 Instrucțiuni condiționale.

O instrucțiune condițională selectează o singură instrucțiune dintre alternativele sale, pe care apoi o execută. În structurile ramificate de calcul, unele etape nu întotdeauna se îndeplinesc în una și aceeași ordine, dar în dependență de careva condiții, care sînt controlate(verificate) pe parcursul calculelor, se aleg pentru executare diferite consecutivități de instrucțiuni. Pentru descrierea astfel de procese în limbajul Pascal se folosesc instrucțiunile ramificate (sau condiționale).

Instrucțiunea condițională IF

Instrucțiunea condițională IF are în Pascal 2 forme: completă și prescurtată. Forma completă este următoarea:

If <condiție> THEN <instrucțiune> ELSE <instrucțiune>;

Forma prescurtată este: IF <condiție> THEN <instrucțiune>;

unde IF(dacă), THEN(atunci) și ELSE(altfel) sînt cuvinte rezervate. Forma completă a instrucțiunii condiționale se mai poate prezenta în felul următor:

If C THEN I1 ELSE I2; (C-expresie logică, I1, I2-instrucțiuni). Executarea acestei instrucțiuni structurate se reduce la executarea unei din instrucțiuni I1 sau I2. Dacă condiția C din cadrul instrucțiunii se respectă (C primește valoarea TRUE), atunci următoarea instrucțiune executabilă este I1, în caz contrar se îndeplinește I2. Exemple de instrucțiuni IF formă completă:

IF x<0 THEN i:=i+1 ELSE k:=k+1;

IF (x>0) and (y>0) THEN a: =SQRT (x * y) ELSE a: =SQR(x * y);

Notă 1. Dacă în instrucțiunea IF C THEN I introducem înainte de I instrucțiunea de efect nul (IF C THEN;I) atunci I nu intră în componența instrucțiunii condiționale, deci este executată indiferent de valoarea lui C.

Notă 2. Dacă însă în instrucțiunea IF c THEN I1 ELSE I2, introducem după I1 “;”, obținem un program incorect sintactic din pricina că această instrucțiune se consideră un tot întreg și nu poate suporta prezența simbolului “;” în interiorul său.

La formularea algoritmilor deseori este tipică situația cînd analizînd o condiție oarecare și în caz de respectare a ei este necesar de îndeplinit careva acțiuni, iar în cazul cînd condiția nu se respectă – nu trebuie de executat nici o instrucțiune.

Ex.: if $x < 0$ then $x := \text{ABS}(x)$ (Aflarea modulului unui număr). În astfel de situații este destul de comodă forma prescurtată a instrucțiunii condiționale:

if c then i. În acest caz dacă valoarea lui c este TRUE, atunci este executată instrucțiunea I , în caz contrar nici o acțiune din partea compilatorului nu-i îndeplinită, adică se controlează numai condiția C . Ex.: if $a > b$ then $c := a - b$;

Instrucțiunea condițională CASE.

În cazul, când în algoritm este necesar de luat în considerație mai mult de 3 variante posibile, construcția *if – then – else* poate fi foarte complicată. În așa cazuri în calitate de alternativă se folosește comutatorul de tipul CASE. Această instrucțiune condițională prezintă o structură alcătuită după principiul MENIU și conține toate variantele posibile ale condițiilor și instrucțiunilor care trebuie de îndeplinit în fiecare din cazurile concrete.

Instrucțiunea CASE constă dintr-o expresie numită selector și o listă de instrucțiuni, fiecare precedată de o constantă de același tip cu selectorul sau de cuvântul cheie OTHERWISE. Instrucțiunea CASE se execută astfel: se evaluează expresia ce definește selectorul și apoi se execută fie instrucțiunea precedată de constanta egală cu valoarea selectorului, fie instrucțiunea precedată de OTHERWISE, dacă valoarea selectorului nu coincide cu nici una din constantele ce etichetează instrucțiunile componente. În acest caz lipsa specificației otherwise conduce la erori. Sintaxa instrucțiunii case este următoarea:

CASE I OF		CASE I OF
C1:<instrucțiunea 1>;		C1:<instrucțiunea 1>;
C2:<instrucțiunea 2>;	sau	C2:<instrucțiunea 2>;
.....	
Cn: <instrucțiunea n>;		Cn: <instrucțiunea n>;
OTHERWISE <instrucțiune>		ELSE<instrucțiune>;
end;		end;

unde i este selector; c_1, c_n -constante. În standardul inițial al limbajului Pascal construcția case este folosită cu cuvântul cheie *otherwise*. În Turbo Pascal în loc de otherwise se folosește *else*. În calitate de selector poate fi o variabilă de tipul INTEGER sau CHAR, însă nici într-un caz de tipul real. În cazul, când selectorul I coincide cu constanta c_1 se îndeplinește <instrucțiunea 1>, în caz contrar compilatorul trece la controlul condiției din ramura 2. Dacă selectorul coincide măcar cu una din constantele $c_1 \dots c_n$ atunci se îndeplinește instrucțiunea respectivă, în caz contrar se îndeplinește instrucțiunea ce urmează după OTHERWISE sau ELSE.

Ex.: *var I, s: integer;*

Begin writeln ('culege I'); case I of 1: s:=3+I; 2: s:=3+sqrt(I);

3: s:=3+sqrt(I); else s:=3+exp(I) end;

4.3.2.3 Instrucțiuni iterative(ciclice)

Instrucțiunile precăutate mai sus redau operații care trebuie efectuate conform algoritmului și fiecare din ele se îndeplinesc numai o dată. În cazul, când una și aceeași instrucțiune trebuie să fie executată de n ori cu diferite valori ale parametrilor se folosesc instrucțiunile ciclice. Distingem 3 instrucțiuni ciclice în Pascal:

1) Instrucțiunea ciclică cu parametru (FOR)

- 2) Instrucțiunea ciclică cu postcondiție (REPEAT)
- 3) Instrucțiunea ciclică precedată de condiție (WHILE)

Instrucțiunea ciclică FOR.

Ciclul FOR posedă următoarele caracteristici:

- numărul de repetări ale ciclului este cunoscut de la începutul executării lui;
- conducerea ciclului este efectuată cu ajutorul unei variabile de tip scalar, care, în acest proces ciclic primește valori consecutive de la valoarea inițială dată pînă la valoarea finală dată.

Pentru o redare mai compactă a proceselor de calcul de așa tip se folosește următoarea construcție a ciclului: `FOR I:=E1 to E2 do S;`

unde `FOR,TO,DO` – sînt cuvinte cheie, `I` – variabilă tip scalar (în afară de real) numită parametrul ciclului, `E1, E2` – expresii de tip identic cu parametrul ciclului, `S` – operator numit corpul ciclului. În calitate de `S` poate fi prezent o singură instrucțiune sau un set de instrucțiuni unite în operatorul `begin - end`. Acest operator ciclic(`FOR`) presupune atribuirea parametrului ciclic `I` valori consecutive de la valoarea inițială `E1` pînă la valoarea finală `E2` și executarea instrucțiunii `S` pentru fiecare valoare a lui `i`. Valorile expresiilor `E1` și `E2` sînt calculate o singură dată, iar valoarea parametrului `I` nu trebuie să fie schimbată în rezultatul executării instrucțiunii `S`. Dacă `E2` este mai mică decît `E1` (ceia ce-i posibil) atunci instrucțiunea `S` nu se execută nici o dată. Exemplu de ciclu `for`: `y:=0; for I:=1 to n do y:=y+1/I;`

În unele cazuri este comod ca parametrul ciclului să primească valori consecutive însă nu în ordine crescătoare ci în ordine descrescătoare. Pentru așa cazuri în Pascal este prevăzut operatorul ciclului cu parametru de felul următor: `FOR I:=E1 downto E2 do S`

unde `downto` (micșorîndu-se pînă la) – cuvînt cheie. În acest caz, dacă valoarea `E1` e mai mică ca `E2` atunci ciclul nu se va repeta nici odată, adică instrucțiunea `S` nu va fi executată. Parametrul ciclului `I` poate să nu fie folosit în corpul ciclului (adică în interiorul lui `S`) din cauză că destinația lui de bază este conducerea cu numărul de repetări al ciclului. Pasul schimbării lui fiind egal cu 1.

Ex.: `y:=1; for I:=1 to n do y:=y * x;`

Instrucțiunea ciclică cu postcondiție (repeat).

Ciclul `FOR` cu parametru de obicei este folosit în cazurile, cînd numărul de iterații este cunoscut de la începutul executării ciclului. Însă deseori numărul de iterații nu este cunoscut de la început, dar se determină în timpul realizării acestui proces ciclic. În acest caz este folosit ciclul `REPEAT` cu ajutorul cărui este formulată condiția, în cazul respectării căreia, acest proces ciclic trebuie să fie terminat. Ciclul cu postcondiție are următoarea sintaxă: `REPEAT S;S;...;S UNTIL B`

unde: `Repeat`(de repetat) și `until`(pînă la) – cuvinte cheie, `S` – un operator oarecare, `b` – expresie logică.

În tipul executării acestui proces ciclic setul de instrucțiuni `S` ce se află între cuvintele `repeat` și `until` este executat de una sau de mai multe ori. Acest proces se termină atunci, cînd după executarea setului de instrucțiuni `S` expresia logică `B` va fi echivalată(pentru prima dată) cu `TRUE`. În așa fel, cu ajutorul expresiei logice `B` este determinată condiția pentru terminarea executării operatorului ciclului. Deoarece în acest caz controlul condiției este efectuat după îndeplinirea setului de instrucțiuni `S`,

acest ciclu e numit ciclu cu postcondiție. De asemenea ca și ciclul for, ciclul repeat conține un parametru, care conduce cu numărul de repetări al ciclului. Acest parametru trebuie să fie prezent în expresia logică B. Spre deosebire de ciclul for unde parametrul nu poate fi schimbat în corpul ciclului, în cazul ciclului repeat valoarea parametrului neapărat trebuie să fie schimbată spre majorare sau micșorare. Adică parametrul ciclului se va schimba în ordine crescătoare sau descrescătoare. Valoarea cu care se mărește(micșorează) parametrul ciclului se numește pasul ciclului. Și în comparație cu ciclul for, în cazul dat pasul ciclului poate fi și de tip real și mai mare ca 1. Ex. : $I:=I$; *repeat* $y:=y+x$; $I:=I+1$ *until* $I>n$;

Deoarece în instrucțiunea repeat - until condiția se controlează numai la sfârșitul ciclului, setul de instrucțiuni S din care este compus corpul ciclului este executat măcar o singură dată. Și în cazurile, când conform algoritmului este necesar ca un set de operatori să fie îndepliniți cel puțin o dată, este binevenit ciclul repeat - until.

Instrucțiunea ciclică precedată de condiție(while).

În cazul ciclului repeat setul de instrucțiuni S va fi executat cel puțin o dată. Însă destul de frecvente sînt cazurile când numărul de iterații nu este cunoscut, dar pentru niște valori concrete ale datelor inițiale acțiunile prevăzute pentru executarea în cadrul ciclului nu trebuie să fie executate nici o dată și chiar îndeplinirea de o singură dată a ciclului poate duce la rezultat incorect sau nedeterminat. Pentru a reda astfel de procese de calcul în Pascal este folosit ciclul WHILE, sintaxa cărui este următoarea: WHILE B DO S ;

unde while și do sînt cuvinte cheie, b—expresie logică, s-operator. Aici operatorul S se execută de 0 sau mai multe ori, însă înainte de fiecare următoare executare se evaluează valoarea expresiei B și operatorul S este executat numai în cazul, când expresia B are valoarea TRUE. Executarea operatorului ciclului ia sfârșit atunci, când expresia B pentru prima dată se egalează cu FALSE. Dacă expresia B se egalează cu FALSE chiar la prima evaluare a sa, atunci operatorul S nu va fi executat nici o dată. Din cauză că condiția de terminare a procesului ciclic este controlată pînă la executarea operatorului S, această instrucțiune poartă denumirea de proces ciclic precedat de condiție. Este remarcabil faptul, că operatorul ciclic precedat de condiție este cel mai universal dintre toți operatorii ciclici. Cu ajutorul lui este posibil de redat procese ciclice determinate de instrucțiuni ciclice cu parametru și cu postcondiție. De exemplu ciclu cu parametru for $i:=E1$ to $E2$ do S; este echivalent cu următoarea succesiune de instrucțiuni: $i:=E1$; *while* $i<=E1$ do *begin* S; $i:=succ(i)$; *end*; și în primul caz, și în al doilea rezultatul executării acestor fragmente de program va fi același, diferența fiind numai în sintaxă și succesiunea de instrucțiuni. Operatorul ciclului cu postcondiție de obicei de asemenea se poate de redus la operatorul ciclic precedat de condiție, corespunzător schimbînd condiția, adică expresia logică. Avînd la dispoziție ciclul cu postcondiție în următoarea componentă:

$i:=E1$ *repeat* S; $i:=i+1$ *until* $i>E2$;

îl putem reda cu ajutorul operatorului precedat de condiție while:

$i:=E1$ *while* $i<=E2$ do *begin* S; $i:=i+1$ *end*;

Deci, fiind cunoscute cele 3 instrucțiuni ciclice FOR, WHILE și REPEAT le putem folosi în diferite scopuri aparte pentru cazuri concrete. În cazul când

cunoaștem numărul de repetări al ciclului, pasul indexului fiind =1 folosim instrucțiunea ciclică FOR. În cazul când numărul de repetări al ciclului e necunoscut, dar corpul ciclului trebuie să fie executat măcar o singură dată folosim instrucțiunea ciclică REPEAT. Și în cazul când nu este cunoscut numărul de repetări al ciclului, iar corpul ciclului e necesar să fie executat de 0 sau mai multe ori, în dependență de o condiție folosim instrucțiunea ciclică WHILE.

4.4 Organizarea modulară a programelor. Proceduri și funcții în Pascal

Deseori, în timpul alcătuirii unui program apar situații, când e necesar ca una și aceeași secvență de instrucțiuni să fie executată de mai multe ori, dar prelucrând date diferite. În așa cazuri e binevenit de folosit subprograme. Un subprogram corect alcătuit și înscris în program o singură dată, care conține tipul și numărul necesar de parametri poate fi apelat de mai multe ori, din diferite puncte ale programului, cu diferite valori ale parametrilor, însă realizând unul și același algoritm.

În Pascal există 2 feluri de subprograme: proceduri și funcții. Deosebirea dintre ele constă în numărul valorilor calculate și returnate în punctele din care a fost făcut apelul. Procedura calculează oricâte asemenea valori, pe când funcția- numai una, permițând ca apelul ei să se facă chiar din expresia care are nevoie de valoarea calculată. Procedurile și funcțiile se definesc în partea de declarare a subprogramelor și sînt ultimele declarații din secțiunea datelor. O astfel de declarație asociază un identificator cu o parte a programului, astfel încît aceasta poate fi activată cu ajutorul instrucțiunii de apel la procedură în orice moment.

4.4.1 Proceduri

Declararea procedurilor este amplasată în partea procedurilor și funcțiilor și are următoarea sintaxă:

Procedure nume(lista parametrilor formali);

Var [lista variabilelor locale];

Begin [corpul procedurii] *end*;

Pentru redarea mai clară a temei vom folosi un exemplu concret. Fie că trebuie de calculat aria unui patrulater pentru care sînt cunoscute cele 4 laturi și o diagonală. Pentru a calcula aria patrulaterului, calculăm aria fiecărui triunghi component în parte folosind formula lui Heron:

$S = \sqrt{p(p-a)(p-b)(p-c)}$, unde $p = (a+b+c)/2$; a,b,c-lungimea laturilor triunghiului.

Fără folosirea procedurilor rezolvarea acestei probleme va fi următoarea:

Program patrulater1;

Var AB, BC, CD, DA, BD, a,b,c,p,s,s1,s2:real;

Begin readln(AB,BC,CD,DA,BD);

a:=AB; b:=DA; c:=BD; p:=(a+b+c)/2;

S1:=sqrt(p*(p-a)*(p-b)*(p-c));

A:=BC; b:=CD; c:=BD; p:=(a+b+c)/2;

S2:=sqrt(p*(p-a)*(p-b)*(p-c));

S:=S1+S2; writeln('S=',S); end.

Se observă repetarea unui set de instrucțiuni, și anume calculul pentru p și S . Anume această situație cere folosirea unei proceduri. Versiunea programului cu folosirea procedurii este următoarea:

```
Program patrulater2;  
Var AB, BC, CD, DA, BD, a,b,c,p,s1,s2:real;  
Procedure aria;  
Begin p:=(a+b+c)/2; S:=sqrt(p*(p-a)*(p-b)*(p-c)); end.  
Begin readln(AB,BC,CD,DA,BD);  
a:=AB; b:=DA; c:=BD; aria;  
S1:=S; A:=BC; b:=CD; c:=BD; aria;  
S2:=S; S:=S1+S2; writeln('S=',S);  
End.
```

Se observă 2 adresări la procedura *aria*. Însă înainte de fiecare adresare se îndeplinesc câteva instrucțiuni de atribuire, care determină valorile variabilelor a, b, c . Fiecare adresare activează procedura și ca rezultat este primită valoarea suprafeței triunghiului pentru laturile cărui au fost făcute atribuirile. Deci legătura dintre procedură și programul principal este efectuată prin intermediul variabilelor a, b, c și S . Însă procedura mai conține și variabila p , care este locală procedurii și nu apare în corpul programului principal. În așa situație declararea și descrierea variabilei p poate fi făcută înăuntru procedurii. Așa variabile sînt numite variabile locale (referitor la procedură). Ele nu pot acționa nici într-un mod asupra variabilelor globale (din programul principal) cu același nume. Diferența dintre ele constă numai în domeniul de vizibilitate: variabila globală este văzută (de către compilator) pe parcursul întregului program, iar variabila locală procedurii este văzută numai în interiorul procedurii.

Noțiunea de bloc și domeniu de vizibilitate

Corpul unui subprogram (procedură sau funcție) este un bloc sau o directivă. Un bloc este constituit din declarații de etichete (label), definiții de constante (const), definiții de tip (type), declarații de variabile (var), declarații de procedură (procedure), declarații de funcție (function) și instrucțiuni (partea executabilă a blocului). Deoarece programele sînt incluse în blocul programului principal, iar blocurile subprogramelor pot conține la rîndul lor alte subprograme, rezultă că blocurile pot fi imbricate (suprapuse). Această imbricare de blocuri este denumită structura de bloc a programelor Pascal, concept introdus de limbajul ALGOL60.

Întrucît blocurile pot fi imbricate în alte blocuri prin declarații de proceduri sau funcții, fiecărui bloc îi putem atașa cîte un nivel de imbricare. Blocul programului principal sau unit-ului principal este considerat de nivel 0, un bloc definit în acesta este de nivelul 1. În general, un bloc definit în nivelul n este de nivelul $n+1$.

O definiție sau declarație introduce un nume, care poate fi un identificator -care servește la denumirea tipurilor, variabilelor, subprogramelor etc. Fiecare nume dintr-un program Pascal are un punct de definire/declarare, unde se leagă diferitele atribute de numele respectiv. Prin *domeniu de vizibilitate* al unui nume se înțelege acea parte a programului în care numele respectiv păstrează aceleași atribute.

Un identificator care a fost declarat/definit în blocul programului (unit-ului) principal se numește global. Un identificator este local blocului în care este

declarat/definit. Un identificator este nelocal într-un bloc, dacă declarația/definiția sa a fost făcută într-un bloc exterior primului.

Exemplul *patrulater2* ilustrează bine cazul folosirii procedurii fără parametri. Această metodă deseori se dovedește a fi neefectivă din cauza multor instrucțiuni de atribuire pentru a,b,c. Pentru înlăturarea acestor neajunsuri Pascal oferă posibilitatea de a nu fixa valorile inițiale ale variabilelor a,b,c pentru lucrul procedurii, dar face a fi posibilă transformarea acestor variabile în parametri ai procedurii, valoarea cărora va fi concretizată la momentul apelului procedurii. Astfel de parametri se numesc parametri formali. *Parametrii formali* nu prezintă niște valori concrete, dar valori «virtuale». Le fiecare adresare către procedură, parametrii ei formali trebuie concretizați, din această cauză, pentru simplificarea următoarelor apeluri la procedură parametri formali sînt indicați în antetul procedurii și în același timp ordonați după cum au apărut în antet. În același timp, pentru fiecare parametru formal se indică tipul valorii prezentate de acest parametru formal. Ca și în cazul declarării variabilelor globale, acest tip este indicat o singură dată după lista parametrilor formali corespunzători. De exemplu: *procedure aria(a,b,c:real)*; Fiecare parametru declarat în lista parametrilor formali este local procedurii și poate fi referit în blocul asociat procedurii prin identificatorul său. La ardesarea către așa o procedură, în instrucțiunea de apel la procedură după numele procedurii se indică în paranteze lista *parametrilor actuali*. Acești parametri concretizează valorile asupra cărora într-adevăr trebuie să fie aplicată procedura și care în corpul ei au fost notați cu ajutorul parametrilor formali. În exemplul nostru parametri formali sînt de tip real, deci ca parametri actuali pot fi luate orice 3 numere reale. În același timp corespunderea dintre parametri formali și cei actuali este respectată atunci cînd locul, numărul și tipul parametrilor formali este identic cu locul, numărul și tipul celor actuali. Folosind aceste reguli, programul precedent poate avea următoarea redacție:

Program patrulater3;

Var AB, BC, CD, DA, BD, s,s1,s2:real;

Procedure aria(a,b,c:real);

Var p:real;

Begin p:=(a+b+c)/2; S:=sqrt(p(p-a)*(p-b)*(p-c)); end.*

Begin readln(AB,BC,CD,DA,BD);

aria(AB,DA,BD); S1:=S;

aria(BC,CD,BD); S2:=S;

writeln('S=',S1+S2);

End.

După cum se observă variabila globală p folosită numai în cadrul procedurii aria a fost transformată în variabilă locală procedurii. Este ușor de văzut că corespunderea dintre parametri formali și cei actuali este respectată. La apelul *aria(AB,DA,BD)* parametrului actual AB îi corespunde parametrul formal a, lui DA-b, lui BD-c. Parametrii a,b,c există numai atît timp, cît este îndeplinită procedura; după terminarea execuției procedurii acești parametri sînt nimiciți din memorie. La următorul apel al procedurii aria, parametrilor formali noulăscuți le corespund alte valori, respectiv BC,CD,BD.

Mecanismul de substituire a parametrilor formali prin parametrii actuali se numește transmiterea parametrilor.

Transmiterea parametrilor.

În Pascal există două mecanisme de bază pentru transmiterea parametrilor: prin valoare și prin adresă.

Transmiterea prin valoare se realizează ca și cum la intrarea în procedura apelată s-ar găsi o declarație a parametrului formal cu tipul corespunzător și o instrucțiune de atribuire de forma:

parametru formal: =parametru actual;

Valoarea parametrului actual este memorată în locația corespunzătoare parametrului formal. Un parametru formal al unei proceduri sau al unei funcții se comportă ca o variabilă locală a subprogramului, cu excepția faptului că acesta este inițializat la activarea subprogramului cu valoarea parametrului actual corespunzător. Această inițializare este echivalentă cu copierea valorii parametrului actual în spațiul datelor locale subprogramului.

Deseori la transmiterea prin valoare se modifică valoarea unui parametru formal. Printr-o astfel de modificare informația din exteriorul procedurii rămîne nealterată. Din această cauză mecanismul transmiterii parametrilor prin valoare prezintă avantajul siguranței: informația din exteriorul subprogramului nu este schimbată dacă se modifică valoarea parametrilor actuali respectivi.

Sintaxa de declarare a procedurii cu transmiterea parametrilor prin valoare este ca și în exemplu precedent: *Procedure nume(nume_parametru:tip_parametru);*

Parametrul actual corespunzător parametrului formal într-o instrucțiune de apel de procedură este o expresie. Valoarea acestei expresii nu poate să fie de tip fișier sau un tip structurat care conține un tip fișier. Dacă tipul parametrului actual este string, atributul de lungime al parametrului este 255.

Transmiterea prin adresă se folosește atunci cînd o variabilă trebuie redată procedurii sau funcției apelante (adică un rezultat obținut în subprogramul apelat trebuie returnat subprogramului apelant). Această metodă de transmitere este semnalată prin prezența cuvîntului *var* în fața numelui parametrului formal, dar care are și atributul de tip. În acest caz sintaxa va fi următoarea:

Procedure nume(var nume_parametru:tip_parametru);

Parametrul actual trebuie să fie o variabilă. Orice operație ce se referă la parametrul formal se va efectua direct și asupra parametrului actual corespunzător.. Parametrul actual nu poate să fie de tip fișier. Dacă parametrul actual este o referință la componentele unor variabile structurate, dereperările necesare sînt executate înaintea activării procedurii.

Subliniem faptul că parametrul actual nu poate să fie expresie sau constantă. Nu există nici o conversie implicită legată de acest mod de transmitere a parametrilor: întreg pentru întreg, real pentru real, interval pentru interval (și nu un interval oarecare al aceluiași tip de bază).

Exemplul cu programul *patrulater3* prezintă mecanismul de transmitere a parametrilor prin valoare.

Folosind mecanismul de transmitere prin adresă programul precedent va avea forma următoare:

```

Program patrulater4;
Var AB, BC, CD, DA, BD,s1,s2:real;
    Procedure aria(a,b,c:real; var s:real);
        Var p:real;
        Begin p:=(a+b+c)/2; S:= sqrt(p*(p-a)*(p-b)*(p-c)); end;
Begin readln(AB,BC,CD,DA,BD);
    aria(AB,DA,BD,S1);
    aria(BC,CD,BD,S2);
writeln('S=',S1+S2);
End.

```

Se vede că transmiterea parametrilor între AB și a; DA și b; BD și c etc. a fost făcută prin valoare, iar dintre S1 și S (S2 și S) – prin adresă.

Declarații anticipate

Una dintre regulile de bază ale limbajului Pascal este că locul de definiție al unui nume trebuie să fie înaintea utilizării numelui respectiv. Astfel, compilatorul poate să efectueze, printr-o singură trecere peste programul sursă, toate verificările necesare. Respectarea acestei reguli în unele cazuri întâmpină greutăți. Să considerăm următorul exemplu: un program declară două proceduri P și Q astfel încât nici una nu este declarată în cadrul celeilalte. În afară de aceasta, să presupunem că procedura P activează procedura Q, iar procedura Q activează la rândul ei pe P. În acest caz, oricare ar fi forma textuală a programului, utilizarea numelui uneia dintre proceduri apare înaintea locului de definiție a ei.

Pentru rezolvarea acestei probleme, în limbajul Pascal a fost introdusă directiva *forward* prin care se pot separa fizic cele două componente de bază ale declarației unei proceduri: antetul procedurii de blocul procedurii. Directiva *forward*, așezată imediat după antet, înlocuiește blocul programului și permite apariția textuală a blocului undeva mai jos în program, unde va fi precedat numai de identificatorul subprogramului. Astfel, lista parametrilor formali se specifică numai în declarația *forward* (și nu se mai repetă și în declarația de procedură sau funcție).

```

procedure Q(x,y,z:integer); forward;
procedure P (u, v, w: integer);
    Begin ... Q(2,3,4); ... end;
procedure Q;{nu se repeta tipul parametrilor lui Q}
    begin ... P(5,6,7); ... end;

```

4.4.2 Funcții

În Pascal funcțiile sînt aceleași subprograme ca și procedurile. Adică ca și în cazul procedurilor, redactînd corpul unui subprogram-funcție o singură dată în partea declarației subprogramelor, mai apoi putem apela la această funcție în orice loc al programului principal, folosind numai numele funcției respective. Însă există și diferențe dintre proceduri și funcții. S-a spus că procedura calculează cîteva valori, pe cînd funcția- numai una, permițînd ca apelul ei să se facă chiar din expresia care are nevoie de valoarea calculată. Adică în Pascal funcția este un subprogram, care calculează și întoarce programului apelant o singură valoare. În același timp sînt permise numai așa funcții, valorile cărora sînt de tipuri simple. În așa fel valoarea

unei funcții nu poate fi nici într-un caz un masiv, o mulțime sau o orecare altă valoare de tip compus.

În particular, orice expresie aritmetică sau logică în Pascal, care poate nici nu folosește noțiunea de funcție, determină o dependență funcțională orecare. Însă nu orice dependență funcțională poate fi redată în limbajul algoritmic folosind regulile sintactice matematice. În unele cazuri valoarea funcției este determinată de un proces de calcul destul de complicat. De exemplu binecunoscuta dependența funcțională din matematică $n!$ (factorial) este imposibil de redat în Pascal cu ajutorul unei expresii aritmetice de forma $1*2*3*...*n$ din cauza restricțiilor impuse de către sintaxa acestui limbaj. Această problemă este ușor de rezolvat cu ajutorul unei consecutivități de operatori. De exemplu: $y:=1; \text{for } i:=1 \text{ to } n \text{ do } y:=y*i;$

Și dacă necesitatea folosirii acestei dependențe funcționale într-un program Pascal este întâlnită de mai multe ori, ar fi rațională determinarea unei funcții pentru calcularea ei și apelarea funcției în locul dorit. Sintaxa de declarare a unei funcții în Pascal este următoarea: *Function nume(parametri formali): tip_funcție;* unde prin parametri formali se subînțelege lista numelor și tipurilor acestor parametri.

În Pascal este posibilă declararea unei funcții și fără parametri ca și în cazul procedurii. Parametrii funcției (în caz că sînt necesari) pot fi și parametri-valori și parametri-adrese. Lista parametrilor are sintaxa identică ca și la procedură. Mai mult ca atît, tot ceea ce a fost spus la procedură despre parametri formali și actuali sau domeniul de vizibilitate este aplicabil și în cazul funcției. Antetul funcției se termină cu indicarea tipului valorii funcției descrise. Acest tip trebuie să fie un tip deja predefinit în Pascal sau în prealabil declarat în programul principal. Rezultatul calculat de o funcție este asociat numelui ei, care așa cum se vede din antet este caracterizat de un tip concret. Numele funcției apare ca o variabilă în cadrul blocului unde a fost declarată funcția. De aceea numele funcției trebuie să fie folosit în cadrul corpului funcției în partea stîngă măcar a unei atribuirii. Rezultatul funcției va fi ultima valoare astfel atribuită. Apelul de funcție este un operand într-o expresie. El se inserează în locul în care este cerută valoarea calculată de funcție. Cînd expresia este evaluată funcția este activată, iar valoarea operandului devine valoarea întoarsă de funcție.

În așa mod au fost desfășurate cele 3 deosebiri sintactice la declarare dintre procedură și funcție:

- 1) Descrierea funcției se începe cu cuvîntul cheie *function*
- 2) În antetul funcției este indicat tipul valorii descrise de funcție.
- 3) Corpul funcției trebuie să conțină măcar o instrucțiune de atribuire, în partea stîngă a căreie figurează numele funcției și această instrucțiune trebuie să fie executată. Adică valoarea funcției trebuie să fie schimbată în interiorul său. Ca exemplu să descriem funcția $n!$ (factorial) cu ajutorul unei funcții Pascal.

```
Program factorial1;
```

```
Var z:integer; s:real;
```

```
Function fact(n:integer):integer;
```

```
Var i,k:integer;
```

```
Begin k:=1; for i:=1 to n do k:=k*i; fact:=k; end;
```

```
Begin writeln('culege o cifră întreagă'); readln(z);
```

```

    Writeln(z, '! = ' fact(z)); {folosirea nemijlocită a funcției}
S:=135+sin(62)*fact(z)/10;    {folosirea funcției ca operand}
End.

```

Efect colateral

În general scopul unei funcții este să furnizeze ca rezultat o singură valoare. Această valoare urmează să fie utilizată în evaluarea unei expresii în programul apelant. Astfel, singura interfață naturală dintre o funcție și programul apelant este acest rezultat.

Prin efect colateral se înțelege o atribuire (într-o declarație de funcție) a unei valori la o variabilă nelocală funcției sau la un parametru variabil. Această atribuire complică în mare măsură verificarea și depanarea programelor. Din acest motiv se recomandă evitarea utilizării unor funcții ce pot provoca efecte colaterale.

Exemplu:

```

program efectcolateral;
  var a,b:integer;
  function par(x:integer):integer;
  begin b:=b-x;{efect colateral asupra lui b} par:=x*x; end;
  begin b:=10;
  a:=par(b);
  writeln(a,b);
  end.

```

Programul produce următoarele rezultate:(100 0).

4.4.3 Tipul procedură și funcție

O definiție de tip procedură sau tip funcție permite manipularea dinamică a subprogramelor (adică a codului subprogramelor). Aceasta înseamnă că un nume de subprogram poate fi atribuit la o variabilă de tip procedură sau funcție. De asemenea, un nume de subprogram poate fi plasat ca și parametru actual la apeluri. O definiție de tip procedură specifică numele tipului, numele și tipul parametrilor; la o definiție de tip funcție apare și tipul valorii returnate de funcție.

```

De exemplu: type tipf = function (x, y: integer) : integer;
              tipp = procedure(var n:real; v:integer);
              var fl,f2:tipf;{variabile de tip funcție}
                  p;tipp; {variabila de tip procedura}

```

De menționat că la definiția tipului nu apare nici un fel de nume de procedură/funcție.

Având o definiție de funcție de forma:

```

Function f (x, y: integer): integer;
  Begin f:=x+y+1; end.

```

este posibilă o atribuire de forma: $f1 := f$;

Exemplu. În acest exemplu este definit un tip funcție numit tipf. Se realizează un apel direct și indirect al procedurii apelparf cu parametri actuali de tip procedură:

```

{$F+}
program tip;
type tipf : function(x,y:integer):integer; {funcție fara nume}
  var varf:tipf;
  procedure apelparf(fpar:tipf; x,y:integer);
  begin writeln(fpar(x,y));{apel parametru functie} end;
function aduna(x,y:integer):integer;
begin aduna:=x+y; end;
  function scade(x,y:integer):integer;
  begin scade:=x-y; end;
  begin {apel direct}
apelparf(aduna,2,3); apelparf(scade,2,3);
{apel indirect}
varf:=aduna;{o atribuire, nu un apel} apelparf(varf,4,5);
varf:=scade; apelparf(varf,4,5); end.

```

Observații:

- Folosirea variabilelor de tip procedură sau funcție este permisă numai atunci când compilarea se face sub controlul directivei {\$F+}.
- O declarație de procedură/funcție generează implicit o variabilă de tip procedură/funcție. Numele acestei variabile este chiar numele folosit în declarație. Variabila astfel generată conține adresa codului obiect generat de declarația de procedură/funcție,
- Dacă parametrul actual este o variabilă de tip procedură/funcție, parametrul formal corespunzător trebuie să fie de tip procedură/funcție. Listele de parametri trebuie să fie compatibile (același număr, nume și tip de parametri; la funcții trebuie să coincidă și tipul valorilor returnate).

4.4.4 Apel recursiv de subprograme

Folosirea numelui procedurii (funcției) în cadrul textului procedurii (funcției) se numește apel recursiv de procedură (funcție), lucru permis în limbajul Pascal. Menționăm totuși că folosirea tehnicilor recursive nu constituie întotdeauna soluțiile optime. În exemplul următor se apelează recursiv funcția putere, destinată calculării bazei la puterea exponent:

```

program testputere;
function putere(baza,exponent:integer):integer;
begin if exponent <= 0 then putere:=1 else putere:=baza*
putere(baza,exponent-1); end;
begin writeln('2^4=',putere(2,4)); end.

```

Observații: La orice apel de procedură în stivă vor fi depuse următoarele informații:

- adresa de retur (adică adresa instrucțiunii ce urmează după apel);
- valorile parametrilor transmiși prin valoare;
- adresele parametrilor variabili

Astfel, la apeluri recursive, spațiul ocupat din stivă va crește rapid, riscând depășirea spațiului rezervat stivei. La programe recursive se recomandă activarea controlului de depășire de stivă prin directiva de compilare {\$S +}.

BIBLIOGRAFIA

Nr. crt.	Denumirea lucrării (manual, material didactic,îndrumar)	Autor	Anul editării
1.	Pascal și Turbo Pascal	Bălănescu T. ș.a	1992
2.	Inițiere în Turbo Pascal	Kalisz E.	1997
3.	Pascal pe interesul tuturor	Anghel,Florin Stînga	1992
4.	Turbo Pascal 6.0 Ghid de utilizare	Sandur Covax	1993
5.	Turbo Pascal 6.0 Programe	Lucian VasIU ș.a.	1994
6.	Prelucrarea fișierelor în Pascal	Roșca I.	1994
7.	Îndrumare metodică de laborator N510	FCIM	1996
8.	Calculatoare personale	Gremalschi A.	1997
9.	Structura calculatoarelor numerice	Gremalschi A.	1996
10.	Limbajele C și C++ pentru începători	Negrescu L.	1996
11.	Turbo C++	Cojocaru C, O.	1994
12.	Введение в язык Паскаль	Абрамов В.Г.	1988
13.	Введение в программирование на языке Паскаль	Эрбс, Хайнц-Эрих, Штольц, Отто.	1989
14.	Вычислительная техника и программирование	Алексеев В.Е.	1991
15.	Англо-русский словарь по программированию и информатике	Борковский А.Б.	1992
16.	Русско-англо-немецко-французский словарь по вычислительной технике	Масловский Е.К.	1990
17.	Язык программирования Си	Керниган Б. Ритчи Д.	1992
18.	Программирование на языке Си для персональных компьютеров	Григорьев А.	1990
19.	Программирование на языке Си, справочное пособие	Котлинская Г.П.	1991