LUCRAREA DE LABORATOR nr. 1

*Tema:* **Cartea de vizită a studentului—informatician**

*Scopul lucrării:* Evidenţierea *nivelului* de cunoştinţe a fiecărui student la informatică, în mod deosebit algoritmizarea, pentru elaborarea unui plan individual de studiu pe viitor la lucrările practice şi lucrările de laborator pe parcursul semestrului.

## Obiectivele temei

1. Aprecierea personală a nivelului de cunoaştere a următoarelor compartimente:

* a echipamentelor periferice şi unităţilor-componente ale calculatorului electronic personal (PC);
* a acţiunilor elementare referitoare la sistemele de operare (SO) MsDOS, Windows etc.;a manipulărilor de bază referitor la principalele activităţi în prelucrarea fişierelor de sistem cu ajutorul membranelor FarManager/DOSNavigator/ NortonCommander, Start, MyComputer, bare şi ferestrele Windows-ului instalat, Office 200x;
* a mediului dezvoltat integrat Turbo Pascal (TP) şi TC, a algoritmizării şi a procedeelor de scriere/citire a datelor şi utilizarea construcţiilor elementare a limbajului TP predestinate calculelor valorilor conform formulelor din matematică şi fizică, elaborând algoritmi cu structuri liniare, ramificative, ciclice;
* a tehnicilor de programare şi principilor de elaborare a modelelor de calcul.
* Însuşirii principiului algoritmizării şi programării cu rezolvarea propriilor variante:
  + însuşirea unei gândiri algoritmice
  + dezvoltarea abilităţilor de proiectare de soluţii algoritmice

### Subiectele temei şi ordinea executării

Subiectele temei se împart în două categorii în dependenţă de nivelul de pregătire a fiecărui student.

*I. Pentru cei care sunt mai* ***slab iniţiaţi*** *în programare se recomandă să expună concis în editorul Word 200x următoarele:*

* 1. Compartimentele cunoscute din domeniul informaticii, performanţele şi posibilităţile utilizării PC, sistemele de calcul şi metodele algoritmizării.
  2. Nivelul însuşirii comenzilor de bază a SO şi cunoaşterea destinaţiei panourilor şi tastelor funcţionale. Membranele SO de tipul Norton Commander sau scenariul de lucru în SO Windows şi Word.
  3. Nivelul de cunoaştere al mediului de programare Turbo Pascal şi C. Reprezentarea meniului principal, submeniurilor din mediul integrat dezvoltat Turbo Pascal şi comenzile editorului.
  4. După însuşirea noţiunilor de bază ale algoritmizării din îndrumarul prezent:

-- notarea şi descrierea blocurilor schemelor logice;

--regulile elaborării diferitor structuri ale algoritmilor;

--modul de transpunere ale algoritmilor şi organizare a programului TP pentru prelucrarea datelor citite, calculul şi afişarea lor.

* 1. Principiile depănării programului simplu şi executarea programului pas cu pas, urmărirea evaluării expresiilor etc.
  2. Validarea rezultatelor cu ajutorul PC-ului, verificarea corectitudinei alcătuirii programelor conform condiţiilor problemelor.
  3. În TurboPascal să efectueze calculele conform formulelor propuse fie din matematică/fizică, fie din Anexa lucrării , utilizînd funcţiile predefinite în TP. Să reformuleze problemele şi să elaboreze schemele logice conform cărora se vor efectua calculele.
  4. Să elaboreze la indicaţia profesorului un set de algoritmi în MS Word.

*II. Pentru cei* ***avansaţi*** *în programare se recomandă să expună concis în editorul Word 200x următoarele:*

* 1. Compartimentele din obiectivele temei, expuse mai sus, prin noţiuni şi definiţii concise, totodată, apreciind propriul nivel de cunoaştere în sistemul zecimal.
  2. Să se elaboreze schema logică a algoritmului şi programul în limbajul TP pentru efectuarea calculelor conform condiţiilor problemelor propuse din matematică/fizică, utilizînd diverse structuri de date (tablouri, şiruri, fişiere, liste etc.). Să folosească principiul modular, utilizînd propriile subprograme cu comentariile de vigoare.

***Conţinutul şi forma dării de seamă conform modelului recomandat***

1. Coperta (foia de titlu) cu indicarea numărului lucrării de laborator şi denumirea temei.

1. Varianta, numărul problemei, scopul şi sarcina (descrierea succintă a sarcinii reformulate, adică adaptată la metoda rezolvării).
2. Expunerea părţii teoretice se efectuează în baza materialului teoretic, formulând întrebări cu răspunsuri.
3. Schema logică a algoritmului/schiţa algoritmului şi tabelul variabilelor utilizate în program. Noţiunile generale, formulele de bază, răspunsurile la întrebări, descrierea tehnicilor şi nuanţelor de realizare a programului (perfectate în Word).
4. Conţinutul programului (listing-ul complet).
5. Rezultatele intermediare şi finale
6. Analiza rezultatelor în baza testelor elaborate şi concluzia: simularea numerică.
7. Bibliografia (doar sursele real utilizate cu referinţe concrete la capitole, paragrafe, etc)
8. Anexe. Exemple şi rezultate. Fişiere intrare/ieşire. Tabele/diagrame.
9. Conţinutul raportului/dării de seamă se prezintă tipărit pe hîrtie şi pe suport magnetic.

***Notă:*** Aceste cerinţe către “***Conţinutul şi forma dării de seamă”*** trebuie să se respecte la toate lucrările de laborator.

# **1. Noţiuni generale**

#### ***1.1. Modelul de calcul al problemei***

Programarea veritabilă n-are frontieră între ştiinţele exacte şi artă. Ştiinţele exacte îţi coordonează conduita, iar arta îţi sugerează soluţiile netradiţionale prin posibilitatea neordinară de a găsi nuanţele “colore” al succesului în timpul analizei şi soluţionarea problemei. Practica **rezolvării problemelor** folosind un *limbaj de programare* a determinat de-a lungul timpului diverse abordări în funcţie de performanţa limbajului de programare, performanţa calculatorului şi nu în ultimul rând, în funcţie de metodele şi tehnicile avansate, privind implementarea raţionamentelor pentru demonstraţiile corespunzătoare problemelor

Este cunoscut faptul că în timpul studiului problemei trecem prin *etapele pregătirii problemei* (conştient sau inconştient, în mod implicit sau explicit) pentru rezolvarea ei cu ajutorul PC-ului: formularea problemei generale, determinarea soluţiei generale, în dependenţă de datele iniţiale şi metodele logico-matematice, prin strategii şi tehnici eficiente, redefinirea corectă şi elaborarea algoritmilor (algoritmi de calcul numeric, algoritmi de prelucrare a informaţiei) în baza celor tradiţionali sau euristici, reieşind din propriile cunoştinţe şi experienţe acumulate. Apoi transpuneţi respectivii algoritmi în programe al limbajului de programare ales. După aceasta depanări, derulări, ameţeli, verificări la corectitudine, testări la eficienţă şi la sfârşit, după ce aţi obţinut obiectivele propuse, treceţi într-o stare euforică cu un refren-bîzîit, plăcut: "Ce geniu sunt!".

Caracteristica fundamentala a prelucrării automate a informaţiei este natura algoritmica a proceselor ce se executa pe sistemele de calcul. Rezolvarea prin calcul automat a unei probleme este posibila doar daca se precizează exact datele de intrare, calculele ce trebuie efectuate si succesiunea lor.

Astăzi, perfomanţa unui *informatician/programator* este determinată de experienţa şi competenţa obţinută în desfaşurarea celor două **etape ( ANALIZĂ, PROGRAMARE)**:

* --        **�etapa gândirii obiectuale (ANALIZĂ / PROIECTARE)** -- modul de analiză şi abstactizare a problemelor prin definirea corectă a obiectelor, a tipurilor de obiecte, a relaţiilor între obiecte şi a operatorilor specifici(*elaborarea UAP ; etapa de concepţie şi analiză- proiectare*) ;
* --        **etapa gândirii algoritmice (PROGRAMARE / EXECUŢIE)** -- alegerea şi aplicarea corectă a unor metode de rezolvare prin precizarea exactă a operatorilor de prelucrare a obiectelor, reprezentarea corectă a strategiilor algoritmice, reprezentarea codificată a obiectelor şi a prelucrărilor conform unui limbaj de programare (*elaborarea algoritmului şi programului ; etapa de programare - codificare implementare şi execuţie*).

**Etapele elaborării programului**. În rezolvarea problemelor cu calculatorul se recomanda respectarea următoarelor etape:

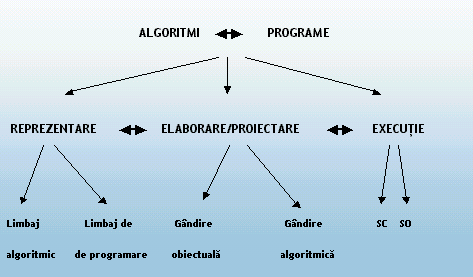
1. Specificarea - precizarea completa a problemei de rezolvat prin stabilirea unui enunţ clar si definirea exacta a datelor de intrare si a celor de ieşire.

2. Proiectarea - căutarea algoritmului de rezolvare si descrierea logicii fiecărei reguli de calcul si a relaţiilor dintre reguli.

3. Implementarea soluţiilor -- *elaborarea algoritmilor* şi *codificarea* acestora într-un limbaj de programare modern.

1. Analiza soluţiilor prin simularea numerică -- *eficienţa soluţiilor* raportată la resursele utilizate: *memorie, timp, utilizarea dispozitivelor I/O*, etc.
2. Testarea şi depanarea -- *verificarea execuţiei programului* cu diverse seturi de date de intrare pentru a putea răspunde rezolvării oricărei probleme pentru care aplicaţia a fost elaborată.
3. Actualizarea şi întreţinerea -- adaptarea soluţiilor implementate pentru *eliminarea erorilor* în rezolvarea unei anumite probleme şi *compatibilitatea* cu sistemul de calcul şi sistemul de operare folosite.
4. Documentarea - scrierea documentaţiei de utilizare a programului sau completarea programului cu comentariile de rigoare.

Conform acestor faze iese în evidenţă **interdependenţa** între urmatoarele *activităţi* importante: REPREZENTARE , ELABORARE / PROIECTARE şi EXECUŢIE, şi anume se poate sintetiza prin următoarea schemă(SC=sistem de calcul, SO=sistem de operare):



Prelucrarea informatiilor intr-un sistem de calcul se realizeaza prin executia unor operatii simple. Operatiile se aplica asupra unor reprezentari ale informatiilor, numite date . Inlantuirea acestor operatii simple, respectand logica de rezolvare, va conduce la obtinerea rezultatelor dorite.

Algoritmul reprezinta ansamblul de reguli (de calcul) împreuna cu ordinea in care se succed, in vederea solutionarii unui tip de probleme.

Spre exemplu, notand cu N si M doua numere intregi pozitive si cu C cel mai mare divizor comun al lor, atunci algoritmul lui Euclid pentru aflarea celui mai mare divizor comun a doua numere intregi pozitive date poate fi descris astfel:

•  CITESTE valorile lui M si N.

•  ATRIBUIE lui C valoarea lui N.

•  ATRIBUIE lui R restul împartiri intregi a lui M la N.

•  DACA R <> 0 (R diferit de 0) ATUNCI TRECI LA pasul 7.

•  SCRIE valoarea lui C.

•  STOP.

•  ATRIBUIE lui M valoarea lui N.

•  ATRIBUIE lui N valoarea lui R.

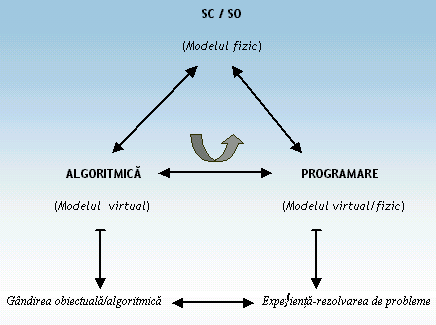
•  TRECI la pasul 2.

Modul in care este prezentat acest algoritm este “imperativ”; propozitiile fiind de fapt niste comenzi pe care trebuie sa le execute cineva (omul sau masina). Fiecare comanda a algoritmului specifica o operatie (actiune) ce se aplica datelor.

**Algoritmul** va trebui să *simuleze* execuţia pe o maşină virtuală asemănătoare maşinii reale, iar **Programul** se va *executa* pe o *maşină fizică/reală* reprezentată de un sistem de calcul (SC) şi un sistem de operare(SO). Acest lucru înseama că *restricţiile* *modelului fizic* trebuie considerate şi în *modelul virtual*, prin urmare execuţia algoritmului trebuie să simuleze execuţia programului pe maşină reală.

Trebuie de menţionat că constantele si variabilele reprezinta datele asupra carora opereaza algoritmii. Variabilele pot fi interpretate ca nume de zone de memorie iar valorile atribuite lor în timpul executiei algoritmului trebuie interpretate ca fiind continutul acestor zone.

De asemenea, *interdependenţa* precizată mai sus se transmite şi între componentele de pe nivelul inferior din schema arborescentă alăturată. **Competenţa** şi **experienţa** în **rezolvarea problemelor**� se pot obţine doar dacă permanent se are în vedere această interdependenţă şi dacă se întreprind eforturi pentru î*nsuşirea de noi cunoştinţe* şi pentru *conoaşterea* corespunzătoare a tuturor aspectelor privind MODELUL FIZIC , respectiv MODELUL VIRTUAL, aspecte determinate de interdependenţa SISTEM DE CALCUL � ALGORITMICĂ � PROGRAMARE:



Pentru a obţine un model de calcul al unei probleme care să fie implementat pe PC trebuie de respectat etapele algoritmizării şi creării programului conform regulilor sintactice şi semantice ale limbajului de programare ales ca să atingem scopul scontat cât mai eficient. De aceea pentru adoptarea unor decizii cât mai corecte şi eficiente, este necesar să se folosească rezultatele obţinute în toate celelalte etape (niveluri de cunoaştere a metodelor de rezolvare, formule, experimente sau măsurătoarele efectuate, legi stabilite etc.). Aceste rezultate pot fi sintetizate şi folosite într-un *model de calcul.* Modelul de calcul poate fi reprezentat printr-un model matematic, adică „dublicatul” fenomenelor analizate, conceput în funcţie de cunoştinţele noastre asupra naturii problemei, închegate în cadru unei *teorii sau experiment*. Pe baza acestor supoziţii, se definesc variabilele modelului (care pot fi mai mult sau mai puţin agregate), legile care intervin în fenomenul analizat, formulele, metodele şi restricţiile, precum şi criteriile de optim. Construirea modelului este dependentă de informaţiile obţinute, mai ales de cele numerice (date de iniţializare, coeficienţi sau parametri care intervin în funcţiile care descriu o anumită legitate sau restricţie etc.).

În unele cazuri se dispune de date analitice, iar alte ori de mărime agregată (de regulă ca o combinaţie liniară a mărimilor analitice). În momentul elaborării modelului apare problema luării în consideraţie şi a resurselor de calcul (echipamente de calcul, viteză de prelucrare, memorie etc.), în funcţie de o anumită precizie. Pentru rezolvarea modelului se folosesc diverşi algoritmi.

Pentru problemele complexe algoritmul, la un mod general, reprezintă o succesiune coerentă de operaţii aritmetice şi/sau logice desfăşurate în vederea realizării unui anumit scop (rezolvarea modelului) într-un număr de paşi finit.*.* Soluţia modelului obţinută cu ajutorul unui algoritm ales de noi va fi analizată, în scopul fundamentării eficiente a deciziei finale. Dacă în urma unei ample analize(ştiinţifice, tehnice, economice, psihologice etc.) soluţia este considerată corespunzătoare, se trece la transpunerea în practică numărându-se, efectele care se obţin. În caz contrar, se revine la analiza fie la datele de intrare, fie la algoritm, fie chiar la model.

Modelarea matematică presupune observarea fenomenelor (obţinerea formulelor şi datelor necesare), elaborarea modelelor în conformitate cu cea mai riguroasă teorie cunoscută, elaborarea unui algoritm de rezolvare a modelului şi, în final, folosirea unui echipament de calcul pentru a aplica algoritmul elaborat în vederea obţinerii soluţiei optime. Acest tip de modelare este foarte dificil de implementat în practică din lipsa unor cunoştinţe profunde.

De multe ori, se evită elaborarea modelului şi în acest caz, se procedează astfel:

* în funcţie de informaţiile disponibile se elaborează algoritmii de calcul în baza căruia se obţine programul în limbajul de programare care se derulează şi, apoi, se analizează soluţia obţinută;
* dacă soluţia obţinută satisface criteriile considerate în analiză se trece la aplicare. Dacă nu, se reia observaţiile sau se elaborează un alt algoritm sau se încearcă elaborarea unui model, corectându-se algoritmul de calcul în mod corespunzător.

Pentru problemele simple prin algoritm vom înţelege un ansamblu de reguli de calcul, care indica operaţiile necesare rezolvării unei probleme si ordinea lor de efectuare. În acest caz se mai poate spune că unalgoritm reprezintă o metoda de rezolvare specifică a problemelor de un anumit tip. Deci a rezolva o problema înseamnă a obţine, pentru anumite date de intrare, rezultatul problemei (date de ieşire ). Astfel algoritmul este constituit dintr-o succesiune de operaţii care descriu, pas cu pas, modul de obţinere a datelor de ieşire într-un număr de paşi finit, plecând de la datele de intrare. De obicei*,* regulile de calcul constituie paşii elementari al algoritmului cu o succesiune anumită şi clară.

Exemple de algoritmi sunt:

- algoritmul lui Euclid pentru calculul celui mai mare divizor comun a doua numere naturale

- regula lui Cramer pentru rezolvarea sistemelor liniare

- ciurul lui Eratostene pentru determinarea numerelor prime, etc.

Deci, pot exista cazuri în care modelul nici nu se mai elaborează. Numai dacă elaborarea algoritmului nu a fost satisfăcătoare se trece la elaborarea modelului care cere reformularea problemei, reieşind din cunoştinţele acumulate şi condiţiile reale. Este posibil că acest model să nu fie complet. După aplicarea algoritmului şi analiza unor rezultate, modelul se completează, se aplică din nou algoritmul, se analizează din nou rezultatele etc. În acest caz , avem un proces iterativ cu o modelare multilaterală, numită şi procedurală. Deci , modelarea procedurală este caracterizată prin acordarea unui rol mai mare algoritmului şi a unuia secundar modelului. De fapt, în modelarea procedurală nu se mai poate face o distincţie prea clară între model şi algoritm. Se poate de scris chiar un „algoritm de elaborare a modelului” . În consecinţă, este necesar ca şi din punct de vedere teoretic să acordăm o atenţie deosebită algoritmilor. În general, un algoritm trebuie să îndeplinească trei proprietăţi: generalitatea (universalitatea), unicitatea (determinismul) şi finititudinea . Prima condiţie se referă la posibilitatea că algoritmul să asigure prelucrarea unui număr mare de date de intrare (chiar “infinit”). A doua condiţie se referă la claritatea operaţiilor din fiecare bloc. Prin intermediul regulilor algoritmului, informaţia iniţială este transformata în rezultate bine definite, care determina în mod unic pasul următor. Adică, la orice execuţie a algoritmului cu date de intrare identice, rezultatele sunt aceleaşi. A treia se referă la numărul de paşi finit în care se furnizează rezultatele (care trebuie să fie din punct de vedere practic de ordinul secundelor /minutelor ). Această caracteristică precizează că orice algoritm se termina într-un număr finit de paşi.

În continuare , prezentăm o clasificare a algoritmilor din punct de vedere a complexităţii problemei şi exactităţii soluţiei. Se poate defini trei clase mari de algoritmi: algoritmi *exacţi*  (riguroşi), *algoritmi aproximativi şi algoritmi euristici.*

Pentru a caracteriza aceşti algoritmi, introducem următoarele notaţii: Xk=vectorul soluţiei furnizat de algoritm; X\*= vectorul adevăratei soluţii, =vector al unor abateri admisibile, dinainte stabilite.

*Algoritm exact* (riguros) are următoare proprietate : X\*- Xk=0 (1.1)

*Algoritmul aproximativ* are proprietatea că pentru orice  dat se găseşte soluţia Xk astfel ca:  X\*- Xk<= (1.2)

*Algoritmul euristic* este de căutare prin încercări bazate prin intuiţie şi experiment care se îmbunătăţeşte în general succesiv, cu proprietatea că , pentru un  dat, nu avem siguranţa că găsim o soluţie Xk, astfel încât să se respecte proprietatea (1.2).

Algoritmul euristic se compune în general , dintr-o mulţime de reguli inteligibile, care sunt încercate în timpul calculelor, şi se îmbunătăţesc pe baza analizei rezultatelor obţinute. În urma acestor încercări, programatorul constată care sunt regulile cele mai adecvate pentru anumite situaţii (tipuri de probleme, metodele etc.). Rezultă că mulţimea regulilor elementare care pot fi folosite într-un algoritm euristic este practic infinita, deoarece acestea sunt create pe măsură ce algoritmul se aplică de mai multe ori. Dar, o procedură la care nu se cunosc regulile nu se poate algoritmiza (deoarece la elaborarea schemei unui algoritm ar trebui să ştim ce reguli putem să aplicam). Sar părea, deci, că este un nonsens să vorbim despre un algoritmi euristici. Contradicţia se rezolvă făcând convenţia că pentru o anumită perioadă sau un anumit număr de aplicaţii, algoritmul se compune dintr-un anumit număr de aplicaţii, algoritmul se compune dintr-un număr finit de reguli elementare, cunoscute iniţial. În afara regulilor elementare (independente), nu pot exista reguli compuse (“trunchiate” sau “amendate” etc.). Dacă regula de compunere este cunoscută, atunci numărul de reguli compuse poate fi finit, procedura fiind algoritmizabilă. În consecinţă, rezultă că o procedură euristică poate fi, cel puţin temporar, algoritmizabilă.

Mulţimea algoritmilor euristici aplicaţi pentru adoptarea deciziilor se numeşte *euristică*. Întrucât un algoritm euristic se compune din reguli elementare (şi eventual reguli compuse ), putem afirma că orice euristică reprezintă o mulţime deschisă de reguli (elementare şi/sau compuse), adică depinde de experienţa programatorului şi atunci e obligator de elaborat şi scenariul de testare al programului în baza unui set de date. Însă astfel de cazuri vor fi analizate la alte teme de lucrări de laborator.

Prin operaţia de descriere a algoritmilor se urmăreşte prezentarea acestora într-o forma precisa si clara, accesibila omului sau calculatorului. Descrierea algoritmilor se poate face prin: scheme logice, limbaj pseudocod, limbaj de programare.

Atunci când vorbim despre instrucţiunile specifice programării structurate, trebuie să amintim de teorema de structură a lui Bohm şi Jacopini: ***Orice algoritm poate fi compus din numai trei structuri de calcul:***

* ***Structuri secvenţiale*** = secvenţe - în care instrucţiunile se execută în ordine. Presupune executarea unei prelucrari in ordinea precizata . ( ele nu se executa conditional sau repetitiv ). Exista cateva tipuri , printre care:

1.) Declararea variabilelor se face cu sintaxa ,, tip data variabila ’’. Definirea variabilei pune in legatura numele variabilei cu tipul sau .

2.)Citirea variabilei reprezinta operatia prin care continutul unei variabile e incarcat de la tastatura.

3.)Atribuirea este operatia prin care o valoare e asociata unei variabile.

4.)Afisarea se realizeaza cu sintaxa scrie exp 1 , ... , exp n

5.)Instructiunea compusa reprezinta un set de prelucrari cuprinse intre acolade .

* ***Structuri alternative*** = decizii - decid dacă să se continue execuţia pe o ramură sau alta. SCOP : permite efectuarea unei prelucrari in functie de valoarea unei conditii şi permite executarea unei prelucrari daca o conditie este indeplinita.
* ***Structuri repetitive*** = cicluri - execuţia repetată a unui corp de instrucţiuni

Fiecare dintre aceste structuri are o singură intrare şi o singură ieşire.

Deci un algoritm de rezolvare presupune stabilirea pasilor necesari. Pasii necesari pot fi descrisi intr-un limbaj de programare , insa nu toata lumea cunoaste acel program si niciun program nu s-a impus ca fiind singurul utilizat de programatori . Din acest motiv s-au impus niste metode generale de descriere a algoritmilor independente de limbajul de programare . Programele se scriu pe baza acestor descrieri . Metodele cele mai importante sunt :

∙ pseudocodul

∙ schema logica

***1.2.* Proiectarea algoritmului,** folosind ca instrument **pseudocodul.**

PSEUDOCODUL reprezinta un set restrans de cuvinte in romana sau engleza asociate unor operanti.Astfel , cuvintele utilizate sunt : intreg , real , citeste , scrie , daca , atunci , altfel , sfarsit daca , pentru , cat timp , sfarsit cat timp , executa cat timp .

Pseudocodul folosit cuprinde:

* operaţii de intrare / ieşire:

*citeşte var1, var2,…*

*scrie expresie1, expresie2,…*

* structuri de control:
* decizia:

*dacă expresie atunci*

*instrucţiune1;*

*altfel*

*instrucţiune2;*

* ciclul:

*cât timp expresie repetă*

*instrucţiune;*

* secvenţa:

{ *instrucţiune1;*

*...*

*instrucţiunen;*

}

## Algoritmul dezvoltat pe baza acestui pseudocod este:

reali a,b,c,delta,x1,x2,xr,xi;

citeşte a,b,c;

dacă a=0 atunci

dacă b0 atunci

scrie –c/b;

altfel

dacă c0 atunci

scrie “nu avem nici o solutie”;

altfel

scrie “o infinitate de solutii”;

altfel

{ delta=b\*b-4\*a\*c;

dacă delta >= 0 atunci {

x1=(-b-sqrt(delta))/(2\*a);

x2=(-b+sqrt(delta))/(2\*a);

scrie x1,x2;

}

altfel {

xr=-b/(2\*a);

xi=sqrt(-delta)/(2\*a);

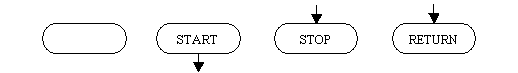
scrie xr,xi;

} }

***1.3.* Descrierea algoritmilor prin scheme logice** Schema logica este descrierea grafica a unui algoritm cu ajutorul unor simboluri care indica paşii algoritmului si ordinea lor de execuţie. Fiecărei operaţii sau pas i se ataşează un simbol geometric în interiorul căruia se înscrie operaţia de executat. Succesiunea operaţiilor este simbolizata prin săgeţi. Schema logica reflecta de fapt modul de execuţie al programului corespunzător algoritmului. Vom analiza în continuare simbolurile grafice utilizate.

a) terminal - marchează începutul, sfârşitul sau un punct de revenire în schema logica. Într-o schema logica exista un singur terminal de START si un singur terminal de STOP.

START STOP RETURN

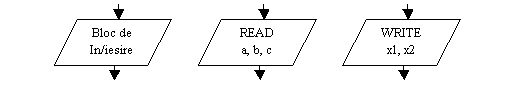
 Figura 5.1. Simbolul terminal al schemei logice

Observaţii: Un algoritm poate să aibă doar un singur bloc **start;** Un algoritm trebuie să aibă cel putin un bloc **stop**

b) bloc de intrare/ieşire (citire/scriere) - permite precizarea datelor de intrare în problema prin initializarea cu valori corespunzătoare a variabilelor care codifica aceste date sau permite afisarea unor rezultate (obtinute ca date de ieşire din algoritm).

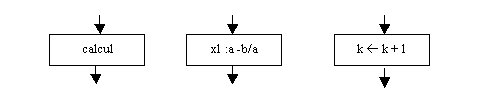
Bloc de Intrare/ieşire READ a, b, c; WRITE x1, x2; -- în TP

scanf printf; în C

Figura 5.2. Blocul de citire/scriere date, reprezentat printr-un paralelogram

c) atribuire sau bloc de calcul - reprezintă o operaţie de calcul cu atribuirea rezultatului variabilei din membrul stâng. Simbolul de atribuire poate fi: ß sau :@

calcul x1 :a -b/a; k ß k + 1

 Figura 5.3. Blocul de calcul

Variabilele din membrul drept al blocului de calcul trebuie sa aiba valori bine definite (rezultate în urma citirii sau calculului); vechea valoare a variabilei din membrul stang se pierde, peste ea scriindu-se noua valoare rezultata în urma operaţiei de calcul.

d) linia de flux - indica directia fluxului operaţiilor

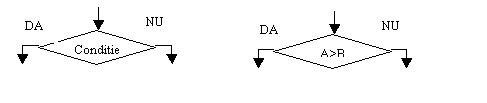
 Figura 5.4. Reprezentarea unei linii de flux

e) conectorul - indica punctul în care o linie de flux se conecteaza cu o alta linie de flux sau o întrerupere a liniei de flux.

 Figura 5.4. Schema unui conector de flux

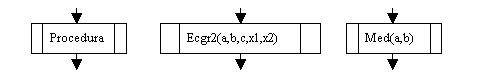
f) decizia logica - indica un test logic al carui rezultat poate fi adevarat (true) sau fals (false), continuarea fluxului de prelucrare facandu-se pe ramura în care condiţia este îndeplinita.

Condiţie DA **A>B** NU DA NU

 Figura 5.5. Reprezentarea grafica a deciziei logice

g) procedura - reprezintă un algoritm sau un subalgoritm descris în alt loc, pentru care se cunosc datele de intrare; procedura returneaza rezultatele sau face doar anumite prelucrări asupra datelor de intrare.

Procedura Ecgr2(a,b,c,x1,x2) Med(a,b)

 Figura 5.6. Bloc pentru apelul procedurilor

Algoritmii pot fi descrisi pe baza unor structuri sau succesiuni de structuri rezultate din combinarea simbolurilor definite mai sus.

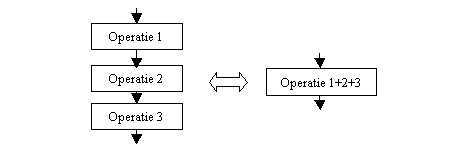
Tipuri de structuri:

1. Structura liniara - este data de un bloc sau de o succesiune de blocuri de calcul.

Operaţie 1

Operaţie 2

Operaţie 3 Operaţie 1+2+3

 Figura 5.7. Schema unei structuri liniare, înlocuita cu un singur bloc de calcul

SCHEMA LOGICA este reprezentarea grafica a unor prelucrari. Aceasta reprezinta un set de simboluri grafice :

START : punctul de pornire al algoritmului

STOP : punctul de oprire al algoritmilor

Citeste variabila : operatia de citire a variabilei

Scrie variabila : operatia de scriere a variabilei.

START

Citeste variabila

Scrie variabila

STOP

2. STRUCTURA ALTERNATIVA

SCOP : permite efectuarea unei prelucrari in functie de valoarea unei conditii.

DECIZIA  SCOP : permite executarea unei prelucrari daca o conditie este indeplinita.

MECANISM DE FUNCTIONARE : se evalueaza conditia sau expresia la o valoare logica. Daca e adevarata se realizeaza prelucrarea P , daca nu , nu se executa nimic.

SELECTIA SCOP : permite executarea unei prelucrari din doua posibile la un moment dat in functie de valoarea unei conditii. Adică permite executarea unei prelucrari din doua posibile la un moment dat in functie de valoarea unei conditii. Se reprezinta in schema logica si pseuocod.

daca\_cond atunci

P1

altfel

P2

sfarsit\_daca

Schema logica:

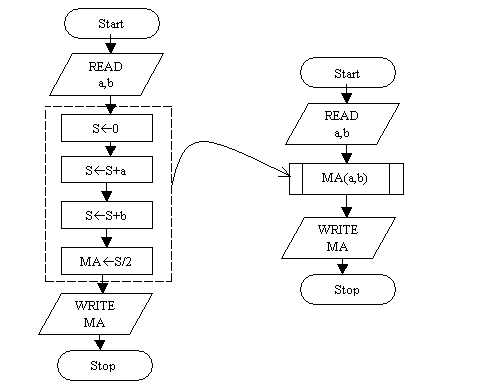
cond=t

P2

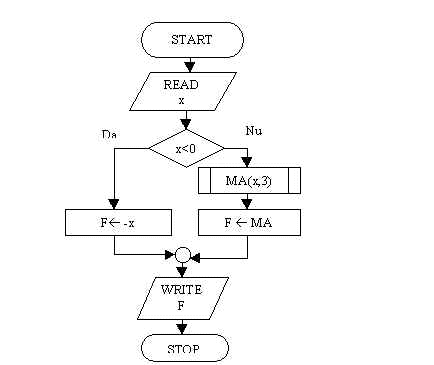
P1

Mecanism de functionare : se evalueaza conditia la o valoare logica.Daca este adevarat se executa P1 , daca nu , se executa P2.

Obs : Atat pentru decizie , cat si pentru selectie se executa in mod conditionat o singura prelucrare. Daca trebuie executate conditionat , pe oricare din ramuri mai multe prelucrari , acestea trebuie cuprinse intre acolade intr-o instructiune bloc.

 Figura 5.8.Exemplu simplu pentru o structura liniara f[

Structura alternativa (IF - THEN - ELSE) - este o structura care are la baza blocul de decizie logica. Ca exemplu se considera schema logica pentru calculul funcţiei:

 Figura 5.9. Exemplu de structura IF-THEN-ELSE

SELECTIA MULTIPLA Este o structura derivata. Ea poate fi inlocuita prin struturi decizionale , deoarece este implementata de instructiuni in limbaje de programare. Ea poate fi prezentata in schema logica:

V1

P1

V2

P2

Vn

Pn

Mecnism de functionare: se evalueaza expresia . Daca valoarea obtinuta = V1 , atunci se executa P1 si se iese din instructiune . Daca nu , se compara valoarea cu V2.Daca sunt egale se executa P2 . Daca nu , valoarea se compara cu Vn , daca e egala se executa Pn si se iese din instructiune.

3. **Structuri repetitive (cicluri)** - sunt structuri utile în situaţii în care o anume operaţie sau grup de operaţii se repeta de un număr finit de ori. Structurile repetitive permit executarea repetata fie a unei instructiuni, fie a mai multor instructiuni, cuprinse în corpul ciclului. Permite executarea unei instructiuni cat timp o conditie este indeplinita. Ca exemplu se va considera calculul sumei elementelor dintr-un şir: a1, a2, …, an : S @ a1 # a2 # … # an.

Operaţiile ce trebuie efectuate pas cu pas sunt:

S = 0 (initializarea sumei)

S = S + a1

S = S + a2 operaţii care se repeta,

… … difera doar indicele

S = S + an

Ciclul cu contor "for" permite executarea unei prelucrari de un numar cunoscut de ori.

Este o instructiune cu test initial si numar cunoscut de iteratii.

Implementeaza structura logica "pentru/sfarsit\_pentru".

Blocul “initializare” realizeaza initializarea contorului cu o valoare initiala. Sunt posibile si alte initializari necesare in program.

Blocul “conditie” testeaza conditia. De regula, conditia este “contor < valoare finala”.

Observatie: Repetitiva poate lucra si descrescator, caz in care conditia se schimba.

Blocul “incrementare” realizeaza cresterea contorului cu o unitate in mod automat.

Ciclul cu contor

initializare

indice = indice\_initial

corp ciclu

indice = indice + pas

indice > indicefinal

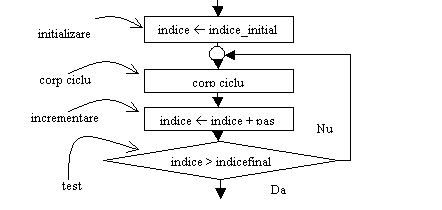
Nu Da

corp ciclu

incrementare

test

initializare

 Figura 5.10 Structura unui ciclu cu contor

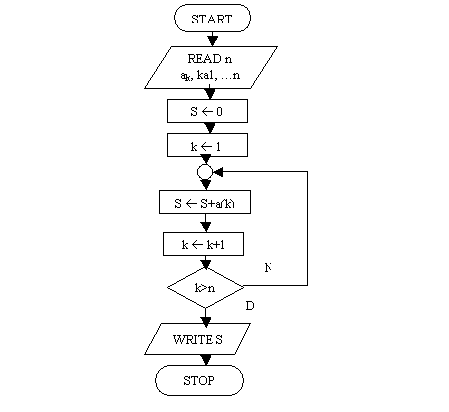
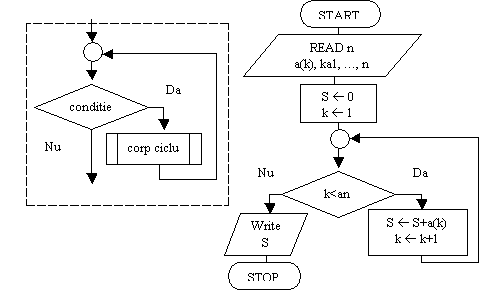
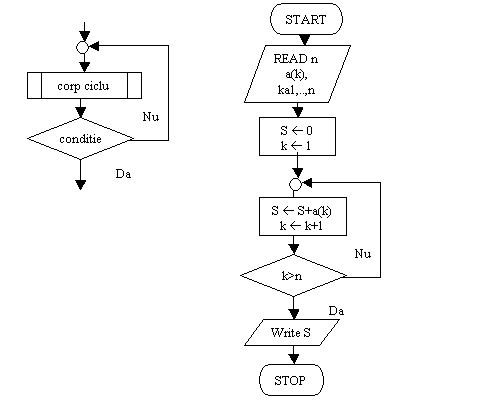


Figura 5.11. Ciclu cu contor pentru calculul sumei unui şir de n elemente

**Ciclul cu test initial (WHILE - DO)** - executa corpul ciclului atat timp cat condiţia este adevarata (Figura 5.12).

 Figura 5.12.Ciclul WHILE - DO

 Figura 5.13. Ciclul cu test final aplicat la calculul sumei elementelor unui şir

La primele teme vom reduce totul la probleme mult mai simple care vor include algoritmi liniari, ramificativi, ciclici şi mixt care trebuie să asigure aceleaşi proprietăţi, însă sunt bazaţi numai pe deducţii logice, evidenţiind următoarele etape pentru rezolvarea problemei concrete:

1) Reformularea strictă a problemei: datele iniţiale, metoda/formulele, forma şi conţinutul rezultatelor.

2) Elaborarea algoritmului rezolvării problemei şi verificarea lui.

3) Alcătuirea programului în conformitate cu algoritmul elaborat la etapa precedentă.

4) Derularea, testarea şi documentarea programului.

5) Analiza rezultatelor programului.

Trebuie de menţionat că cea mai răspândită formă de reprezentare a algoritmului este descrierea prin schemele logice. Mai detaliat se expune şi se examinează în lucrările de laborator ce urmează.

**Aflati media aritmetica a primelor n numere naturale.**

**Simulare numerica :**

P0 n=3 , i=1 , s=0 , med=0;

P1 i=1 , 1<=3 (T) , s=0+1=1 , med=1/1=1; i=1+1=2 ;

P2 i=2 , 2<=3 (T), s=2+1=3 , med=3/2 ; i=2+1=3 ;

P3 i=3, 3<=3 (T), s=3+3=6 , med=6/3 ; i=2+1=4 ;

P4 i=4, 4<=3 (F), med=6/3=2.

START

Citeste n

s=0

Med=0

i<=n

S=s+i

i=i+1

Scrie med

STOP

Med=s/i

DA

NU

# E X E M P L E: Modele de program TP în baza algoritmilor liniari

{Program efectuat de studentul grupei TI 051 NumeFam }

{ Llab. nr.1 V A R I A N T A 7 }

*Program Llb\_nr1;*

{Condiţia problemei: Calculaţi valorile lui d,k conform formulelor următoare: *d:=( X2+sin2 y)/(lnY+√x), k:=(cos x+cos y )/cos(x+y)+ex***.**

Datele iniţiale se introduc de la tastatură. }

{Schiţa algoritmului: algoritmi liniari, utilizînd funcţii predefinite în TP }

Uses Crt;

Var X,Y,D,K : Real; {variabilele X, Y -- iniţiale; var-le D, K -- rezultate}

**Begin**  Clrscr;

Writeln('Introduceti valoarea lui x=> '); Readln(x);

Writeln('Introduceti val. Y=> '); Readln(y);

**d:=(sqr(x)+sqr(sin(y)))/(ln(y)+sqrt(x));**

**k:=(cos(x)+cos(y))/cos(x+y)+exp(x);**

Writeln('Rezult.1: d= ',d:2); Writeln('Rezult.2: k= ',k:2);

**End**.

Program Llab1\_2\_sfera;

{Program efectuat de studentul gr. C-14\_ NumeFam Llab. nr.1, Var-ta 17 }

{ Condiţia problemei: Calculaţi valorile Ariei secţiunii axiale a sferei (M2) şi Volumului sferei (M3) conform formulelor cunoscute. Datele iniţiale se introduc de la tastatură. }

{Schiţa algoritmului: algoritmi liniari în baza formulelor geometrice. }

Uses crt;

Const pi=3.14159; {Acest rînd se putea evita. Pi (π) în TP e determinat}

Var r,l,s,v : Real;

Begin { Introducerea datelor: }

Writeln('Introduceti raza sferei=> '); Readln(r);

{ Calculul variabilelor: }

s:=pi\*(r\*r); v:=4/3\*pi\*sqr(r)\*r;

writeln('Aria sec-i axiale a sferei ', s:8:3,' (M p.)');

writeln('--------------------------------------');

writeln ('Volumul sferei = ', v:8:3,' (M cubi)');

writeln('--------------------------------------'); Readkey;

End.

E X E M P L E: Modele de algoritmi în soluţionarea problemelor:

Problema 2.4.26. Se cere să se realizeze algoritmul care citind o valoare de tip întreg N şi o cifră întreagă C, verifică dacă în reprezentarea zecimală a numărului N apare C. Variantă de soluţie: în acelaşi context, se cere să se precizeze numărul de apariţii ale cifrei C în reprezentarea zecimală

**Algoritmul:**

Start

1

Introducere

N şi C

2

3

a=0

Nmod10=C

nu

4

da

5

a=a+1

6

N=Ndiv10

N>0

da

7

nu

8

Tipărire

Stop

9

Problema 2.4.27. Se cere să se redacteze algoritmul care afişează toate permutările care se pot face cu numerele 1, 2, 3, 4, 5. Indicaţie: se vor utiliza cinci cicluri cuprinse unul în altul, cîte unul pentru fiecare poziţie din cadrul fiecărui ciclu se parcurg toate valorile posibile poziţiei corespunzătoare

Start

1

8

num=0

k=1

2

i=1

k=i

nu

k=j

nu

3

9

10

f=true

da

4

da

11

f=true

j=1

5

12

j=i

l=1

6

nu

l=i

nu

l=j

nu

l=k

nu

13

14

15

da

f=false

da

da

da

7

f=false

16

17

m=1

m=i

m=j

m=k

m=l

nu

nu

nu

nu

18

19

20

21

da

da

da

da

22

f=false

f=true

23

da

24

num=num+1

Tipărire

I,j,k,l,m

25

26

m=m+1

m<6

da

27

nu

28

l=l+1

l<6

da

29

nu

j=j+1

30

j<6

da

31

nu

i=i+1

32

j<6

da

33

nu

Stop

Simulare numerica:

Po n=3 , x=2,4,6 , P=1

P1 i=1 , 1<\_3 (a) , x=2 , P=1\*2=2 , i=1+1=2

P2 i=2 , 2<\_3 (a) , x=4 , P=2\*4=8 , i=1+2=3

P3 i=3 , 3<\_3 (a) , x=6 , P=6\*8=48 , i=1+3=4

P4 i=4 , 4<\_3 (f) , scrie 48.

RETINETI! ∙ Pentru a calcula inversul unui numar trebuie sa folosim formula :

|  |
| --- |
| NINV=NINV\*10+c |

unde c=n%10 , iar n=n/10

∙ Pentru a demonstra daca un numar e palindrom trebuie ca acesta sa fie egal cu inversul sau.

Ex : 123 nu e palindrom

121 e palindrom.

3.) Calculati suma cifrelor unui numar .

Ex : n=123

Suma cifrelor=1+2+3=6

Simulare numerica:

P0:n=123,s=0

P1:123>0 A, c=123?%10=3,s=0+3,n=123/10=12

P2:n=12>0 A,c=12%10=2,s=3+2=5,n=1/10=1

P3:n=1>0 A,c=1%10=1,s=5+1=6,n=1/10=0

P4:n=0>0 F,scrie s=6

**Schema logica**

START

Citeste n

s=0

n>0

C=n%10

n=n/10

Scrie s

STOP

S=s+c

DA

NU

# **Calculati produsul cifrelor unui numar. Schema logica:**

START

Citeste n

p=1

n>0

C=n%10

n=n/10

Scrie p

STOP

p=p\*c

DA

NU

Simulare numerica:

P0:n=123,p=1

P1:n=123>0 A,c=123%10=3, p=1\*3=3,n=123/10=12

P2:n=12>0A, c=12%10=2,p=3\*2=6,n=12/10=1

P3: n=1>0 A,c=1%10=1, p=6\*1=6,n=1/10=0

P4:n=0>0 F,scrie p=6

**Calculati suma primelor n numere naturale .**

Citeste n

i<=n

S=s+i

i=i+1

DA

NU

START

S=0

i=1

Scrie s

STOP

**Calculati produsul primelor n numere naturale . Schema logica**

START

Citeste n

p=1

i=1

i<=n

i=i+1

Scrie p

STOP

P=p\*i

DA

NU

Simulare numerica :

P0 n=3 , p=1

P1 i=1 ; 1<=3 (T) ; p=1\*1=1 ; i=1+1=2 ;

P2 i=2 ; 2<=3 (T) ; p=1\*2=2 ; i=2+1=3;

P3 i=3 ; 3<=3 (T) ; p=2\*3=6 ; i=3+1=4;

P4 i=4 ; 4<=3 (F) SCRIE 6.

**Se citesc n valori . Calculati produsul valorilor pozitive si suma valorilor negative .**

Citeste n

i<=n

Citeste x

i=i+1

DA

NU

START

S=0, p=1

i=1

Scrie s, p

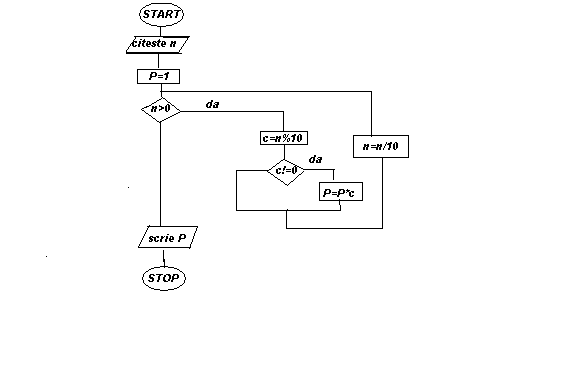
STOP

i<=n

P=p\*x

S=s+x

# **Aflati produsul cifrelor nenule ale unui numar.** **Schema logica :**



Simulare numerica:

P0:n=1230 ,p=1

P1:n=1230>0 A,c=1230%10=1, 0>0(F) ,n=1230/10=123

P2:n=123>0A, c=123%10=3,p=1\*3=3,n=123/10=12

P3: n=12>0 A,c=12%10=2, p=2\*3=6,n=12/10=1

P4:n=1>0 (A) c=1%10=1 , p=6\*1=6 n=1/10=0,

P5:n=0>o(F) scrie 6

# **Problema 2.4.29** Se cere să se redacteze algoritmul care listează tripletele (i, j, k) şi n, natural n=>3 care satisfac relaţia in+jn=kn. Se vor considera tripletele pentru care 1<=i<j<k<M, unde M data de intrare.

Indicaţie: se lucrează în întreg de aceea M şi n trebuiesc astfel aleşi încît Mn<23i-1 adică n\*ln(M)<ln(23i-1). Rezultă că pentru un M precizat 

Algoritmul

1

Start

2

Citire M

3

n=3



4

nu

da

5

Stop

i=1

6

j=i+1

7

k=j+1

in+jn=kn

8

nu

da

9

Tipărire

(i, j, k) şi n

10

k=k+1

11

k<M

da

nu

j=j+1

12

j<M-1

da

nu

13

i=i+1

14

i<M-2

da

15

nu

16

n=n+1

**Notă: Rafinare în paşi succesivi** Adeseori algoritmul de rezolvare a unei probleme este rezultatul unui proces complex, în care se iau mai multe decizii şi se precizează tot ceea ce iniţial era neclar. Observaţia este adevărată mai ales în cazul problemelor complicate, dar şi pentru probleme mai simple în procesul de învăţământ. Este vorba de un proces de detaliere pas cu pas a specificaţiei problemei, proces denumit şi proiectare descendentă, sau rafinare în paşi succesivi. Algoritmul apare în mai multe versiuni succesive, fiecare versiune fiind o detaliere a versiunii precedente. În versiunile iniţiale apar propoziţii nestandard, clare pentru cititor, dar neprecizate prin propoziţii standard. Urmează ca în versiunile următoare să se revină asupra lor. Algoritmul apare astfel în versiuni succesive, tot mai complet de la o versiune la alta.

# **Întrebările de autocontrol**

1. Enumăraţi principalele caracteristici ale lui Windows 9x-200x şi principiul gestiunii ferestrelor.
2. Descrieţi lansarea programului Word for Windows, Crearea, Editarea şi Salvarea unui document nou.
3. Caracterizaţi Mediul de programare TURBO PASCAL: Sistemul de meniuri.şi meniul principal. Rîndul de jos şi Fereastra Edit.
4. Definiţi structura algoritmului şi notaţia în schemele-bloc.
5. Definiţi structura programului TP şi sintaxa expresiilor în limbajul TP.
6. Cum se definesc tipurile de date şi variabile în TP?
7. Ce reprezintă funcţiile predefinite în TP.
8. Ce tipuri de date predefinite cunoaşteţi în TP.
9. Verificaţi ce rezultate se vor obţine pentru următoarele secvenţe de program:
10. Begin {pentru x=1, a=-1 şi x=1, a=1}

f=2\*a\*x+abs(a-1); fx=exp(sqr(x))/sqrt(sqr(a));write(‘f=’,f,’fx=’,fx)End;

1. Begin z:= 3\*7 div 2 mod 7/3 – trunc(sin(1)); writeln(‘z=’, z) end;
2. Begin v:= succ(round(5/2) – pred(3)); writeln(‘v=’, v:4) end;

9\*. Cum trebuie scrisă în TP instrucţiunea de atribuire, fără a utiliza cicluri, pentru calcularea valorii E prin următoarea formulă:

x2 x3 x4

E=1+ x+ — + ── + ── ?

2! 3! 4!

10\*.Câte operaţii în TP se vor efectua în timpul calcului următoarei expresii: (x + 1/2)\*(y + 7/10)–3/4?

11\*. Sunt date punctele pe plan ca valori reale a,b,c,d,s,t,u. Se cunoaşte că punctele (a,b) şi (c,d) nu se află pe dreapta exprimată prin egalitatea s\*x+t\*y+u=0. Să se determine dacă punctele (a,b) şi (c,d) aparţin diferitor semiplane.

12. Descrieţi modalităţile de soluţionarea ale problemelor asistate de PC.

Anexa  **Varintele problemelor Ll.nr.1**

|  |  |  |
| --- | --- | --- |
| **Nr. Var.** | **Formulele pentru calcul** | **Valorile daatelor iniţiale** |
| 1 | (c sin**4**(x-/3)) c+z**2**  Z=─────────; v= ─────  (1/7+ln**2**y); (3+z**3**/5) | x=0,345…7,45; cu pasul 0,1  y= -0,570  c=1,8 |
| 2 | y=|c**vv/a a**-**3a**/c |; y+ sin**2** vv-a  w= ─────────.  5,0001+(y-vv)**3** | c=1,037  a=0,15…1,25; cu pasul 0,1  vv=5,978 |
| 3 | \_\_\_\_  a=e**-h j**cos(h k-j)+**** |j-k| ; b=a ln(a j**4**-sin2h)-k. | j= 0,1…1,0; cu pasul 0,1  k=2,7  h=0,94 |
| 4 | \_\_\_\_ \_\_\_\_  q=│**d2**+f – f**2**ln**3**(d+x)/d│; y=cos**3**q**4**– q/**f2**+d**2** | d=2,359  f=15,5  x=-0,1…2,9; cu pasul 0,1 |
| 5 | \_\_\_  a=g**5**lg**2**(n+m)**2**+m/**g**+n ; b=(ng**3**-a)/(e**ax**-7) | g=1,0…1,64; cu pasul 0,1  m=2,14  n=1,34 |
| 6 | \_\_\_  u=sin**3**(s-1)/p-tg**3**(s-st) ; o=**u**/s+sin**4**(s-p) | P=0,7  S=1,605  t=0,5…1,0; cu pasul 0,1 |
| 7 | \_\_  a=3tg**5**(e-w**2**)-**q**/w ; b=a**3**/q-sin**3**(a-qq) | q=0,5…3,1; cu pasul 0,1  w=1,23  e=5,11 |
| 8 | \_\_\_\_\_  at=arcsin d-w**2**+****| e-w**4**| ; as=at ln(t)+log***2*** at | d=1,711  t,e=1,0…1,  w=2,1101 |
| 9 | af= e4z-dg**3**-d z/g; aa=af arctg(z-d/g)4 | d,z=4,101  g=1,…2,0; cu pasul 0,1 |
| 10 | \_\_\_  w=(s-sin r**2**)+ln**3**(r/q); ff=e**-w**|r-w| /ss; | q=1,01  r=0,2…1,0; cu pasul 0,1  s=1,3; |
| 11 | dd=ln(d-v**2r**+r)/(r-1); dr=dd**3**-d arcsin**4**(r+v)/r | v=2,001  d=0,004…1,07; cu pasul 0,1  r=1,43 |
|  |  |  |
| 12 | \_\_\_\_\_\_\_\_\_ \_\_\_\_\_  ee= e**-2r**+h ****│d-w**2**v ln2r│; k=r**3**-tg****v**2**+ee**2** | v=0,3…5,1; cu pasul 0,1  h=2,04; r=0,101; d=1,711  w=0,1201 |
| 13 | gg=v e**-r** log***2*** rm /(mv-1); g=sin**r**(gg-m**2r**-r) | v=1,3…7,0; cu pasul 0,1  m,r=0,87 |
| 14 | c****│x-y**2** ln2c│ y**4**(x-zz)**2**  ZZ=─────────; v= ─────  (1/7+ln**2**y); (3+zz**3**/5) | x=0,3…4,5; cu pasul 0,1  y= -0,570  c=1,8 |
| 15 | \_\_\_  l= log***2*** v r+m**5**arcsin**2**(m); ll= **l**+1 -e**-m r** | m=0,4…0,5; cu pasul 0,1  r,v=1,8 |

16\*. De alcătuit algoritmul pentru: Se dă un număr natural N şi numerele A1,...,AN. De precăutat subşirurile numerelor formate din:

a) pătrate întregi; b) puterile cinciului; c) numere simple.

Pentru fiecare caz de obţinut cea mai mare lungime a subşirului precăutat. Este de dorit, p*entru cei avansaţi în programare, să încerce să utilizeze* subprograme care ar găsi patratele întregi, puterile numerelor prime, numere multiple şi alte situaţii de prelucrare a şirurilor de numere prin diverşi algoritmi.

17. Problema OJ71 Se ştie că un număr natural este deosebit dacă există un număr natural m astfel încît n=m+S(m) unde S(m) este suma cifrelor lui m. Se cere un program care verifică dacă un număr natural x este deosebit. Metoda folosita: daca x are a cifre atunci şi m va avea m sau a-1 cifre şi ele pot să aibă sumă minimă dacă toate cifrele lui m sînt „1” şi S(m)=a-1 şi suma cifrelor n este maximă dacă toate sînt „9” şi atunci S(m)=a\*9. Deci m poate să varieze de la n-a\*9 pînă la n-a+1

18. De alcătuit algoritmul pentru programul TP(pr1032;) de mai jos.

{ De obţinut toate combinaţiile din 10 elemente (1,...,10), câte 4 elemente in fiecare }

program 1032;

const n=10; num=4;

var i:integer;

a:array[1..n] of 1..n;

procedure tipareste;

var i:integer;

begin

for i:=1 to num do write(a[i],' ');

writeln;

end;

procedure permuta(k:integer);

var i,x:integer;

begin

if k=1 then tipareste else begin

for i:=1 to k do begin

x:=a[i];a[i]:=a[k];a[k]:=x;

permuta(k-1);

x:=a[i];a[i]:=a[k];a[k]:=x;

end;

end;

end;

begin

for i:=1 to n do a[i]:=i;

writeln;

permuta(n);

end.

========================================================================= }

19. De alcătuit algoritmul pentru programul TP de mai jos şi de determinat ce efectuează.

uses crt;

const n=12;

a:array[1..n] of integer=(0,0,-2,0,-2,20,-2,-4,-4,4,-9,5);

var i,j,k,q:byte;

z,w:integer;

begin

clrscr;

for i:=1 to n do begin

write(a[i],' ');

end;

writeln;writeln;

for i:=1 to n do begin

if a[i]>0 then inc(k);end;

i:=n;

repeat

if a[i]>0 then begin

z:=a[i];

for j:=i downto 2 do begin a[j]:=a[j-1];end;

a[1]:=z;

end

else dec(i);

if a[i] < 0 then begin

inc(q); w:=a[i];

for j:=i to n-q do begin a[j]:=a[j+1];end;

a[n-q+1]:=w;

end;

until i=k;

for i:=1 to n do begin

write(a[i],' ');end; readkey;

end.

20. Problema. De alcătuit algoritmul şi programul pentru următoare condiţie: Avionul poate efectua o cursa din Chişinău in una din localitatile L1, L2, ...Ln. Costul unui bilet la cursa "Chişinău - localitatea Li" este de Ai lei. Grupul de pasageri include M1 persoane care pleaca in localitatea L1, M2 persoane care pleaca in localitatea L2 s. a. m. d. Din localitatea Li (i=1,2,...,n) in localitatea Lj (j=1,2,...,n) se circula cu trenul, costul unui bilet fiind de Tij lei. Se considera, ca costul unui bilet direct "Li - Lj" este mai mic decât costul unui bilet cu transportari "Li - Lk - Lj", adica Tij <Tik +Tkj. Elaboraţi un program care determina localitatea unde va ateriza avionul, garantând minimizarea costului total al biletelor procurate de pasageri.}

21. Problema OJ78. De alcătuit algoritmul şi programul pentru următoare condiţie: Fie V o multime ce contine elementele 1 si 0.X=X1,X2...Xn cu n mai mare ca 1 (n este dat)si Xi apartine lui V.Se mai dau:K si s apartin multimii numerelor naturale mai mari ca 1.Sa se genereze toate cuvintele Y=Y1Y2...Yk, unde Y1+Y2+...+Yk=S. Sa se determine pentru fiecare dun cuvintele generate de cite ori apar ca subcuvinte ale lui X.}

22. Problema. De alcătuit algoritmul şi programul pentru următoare condiţie: Fie P o permutare a elementelor [1,2....k] Spunem ca P este un ciclu de lungime l daca o submultime [i1,i2....il] a multimii [1,2...k] este astfel incât urmatoarele relatii sa fie indeplinite:

* 1.P(i1)=i2,....,P(l-1)=P(l), P(il)=i1
* 2.P(x)=x p-u oricare x din [1,2..k]\[i1,i2,...il]

23. Problema OJ72 De alcătuit algoritmul şi programul pentru următoare condiţie: Fie un vector V cu p componente, ocupat partial cu elemente din multimea A=(a,b,c,d,e,f,g). Componentele lui V se considera ocupate incepind in ordine cu I componenta, fara sa existe locuri libere intre componentele ocupate. Fiind date un element y, se cere:

a) Sa se verifice daca y in A si daca se gaseste printre elementele ocupate in vector

b) In cazul in care y nu se gaseste in vector, sa se ordoneze crescator componentele aflate in vector si apoi sa se introduca y pe locul care i se cuvine in ordine crescatoare(lexicografic)

c) Daca y se gaseste in vector, sa se scoata si sa se comprime locurile ocupate astfel incit locurile libere sa ramina ultimele.}

24. Problema De alcătuit algoritmul şi programul pentru următoare condiţie: Se dă o matrice A patratica formata din elementele 0 şi1. Se cere:De gasit sirul B cu proprietatile:

1.Elementele sirului sint distincte doua cite doua.

2.Pentru orice doua elemente l si m consecutive ale sale ,A[l,m]=1.

3.Considerind elementul Bi, succesorul Bi+1 este cel mai mic indice cu proprietatile cerute.

EXEMPLU:

0 1 1 1

A= 1 0 1 0

1 0 1 1

1 0 0 0 ,unde B=(1,2,3,4)

Precautam de la inceput primul element al matricei =1,apoi se precauta pe rindul i=j alt element al matricei =1 .Daca nu s-a gasit trecem la succesorul primului element al matricei =1,etc.

25. Problema OJ129 De alcătuit algoritmul şi de derulat programul pentru următoare condiţie: Se dau mulţimile V=[a,b,c,d] si S=[A,B]. Numim cuvânt format cu elementele lui VUS o succesiune formata conform regulilor:

1)orice x din V-[a,d] este un cuvânt.

2)daca x este cuvânt atunci axd este cuvânt.

3)daca x,y sunt cuvinte atunci xAy, aBy sunt cuvinte.

dându-se succesiune x=x1,x2,...,xn cu elemente VUS, sa se determine daca x este cuvânt.

PROGRAM RADICAL; {Programul 25. Calculează radical din x}

VAR eps, {eps= precizia cu care se calculează}

x, {radical din x, eps>0 si x>=0}

r, {valoarea radicalului x}

a,b, {capetele intervalului ce conţine pe r}

m : REAL; {mijlocul intervalului [a,b]}

BEGIN

WRITELN('Se calculează radical din x cu precizia eps:');

WRITE('eps='); READLN(eps);

WRITE(' x ='); READLN(x);

IF x<1 THEN BEGIN a:=x; b:=1 END {Iniţializează pe a si b}

ELSE BEGIN a:=1; b:=x END;

REPEAT

m:=(a+b)/2;

IF (a\*a‑x)\*(m\*m‑x)<0

THEN b:=m {rădăcina în stânga}

ELSE a:=m; {rădăcina in dreapta}

UNTIL b‑a<eps;

r:=(a+b)/2;

WRITELN; WRITELN;

WRITELN('Radical(',x:6:1,') = ',r:6:3); {r‑rad(x)<eps}

READLN

END.

*26.* De alcătuit algoritmul şi de derulat programul pentru următoare condiţie: *NUMERE PITAGORICE*. Numerele *a,b,c*, se numesc pitagorice dacă 1. Să se tipărească toate tripletele (*a,b,c*) de numere pitagorice, cu 0<*a<b<c* şi *a+b+c**n* ordonate după suma *a+b+c*.

Specificarea problemei este:

*DATE n; {nN; pentru n<12 nu există triplete}*

*REZULTATE toate tripletele de numere pitagorice (a,b,c) cu proprietatea*

*0<a<b<c şi a+b+cn.*

27 De alcătuit algoritmul şi de derulat programul pentru următoare condiţie: Vom nota prin *S* suma *a+b+c*. Se ştie că (3,4,5) este primul triplet de numere pitagorice. În acest caz *S* ia valori de la 12 la *n*. Întrucât *3a<S* variabila *a* ia valori de la 3 la *S*/3. Apoi 2*b*<*S-a* deci *b* va lua valori de la *a+1* la (*S-a*)/2. Algoritmul pentru rezolvarea problemei este dat în continuare :

*Algoritmul NRPITAGORICE este :*

*Date n; {nN; pentru n<12 nu există triplete}*

*Dacă n<12*

*atunci Tipăreşte "Nu există numerele cerute"*

*altfel Pentru S=12,n execută*

*Pentru a=3,S/3 execută*

*Pentru b=a+1,(S-a)/2 execută*

*Fie c:=S-a-b;*

*Dacă c=a+b atunci Tipăreşte(a,b,c) Sf-dacă*

*Sf-pentru*

*Sf-pentru*

*Sf-pentru*

*Sf-dacă*

*Sf-algoritm.*

Programul Pascal corespunzător este dat în continuare.

PROGRAM NRPITAGORICE; {Programul 1.1.2. Numere pitagorice}

VAR n, { nN; a+b+cn }

S, { S = a+b+c }

a,b,c, {(a,b,c) triplet de numere pitagorice}

{ 0 < a < b < c }

k : integer; { contor }

BEGIN

WRITELN('Se tipăresc tripletele(a,b,c) de numere pitagorice');

WRITELN('cu proprietatea: a+b+c<=n, pentru n dat');

WRITE('Daţi valoarea lui n:'); READLN(n);

For k:=1 to 4 do writeln;

k:=0;

IF n<12

THEN WRITELN('Nu exista numerele cerute')

ELSE FOR S:=12 TO n DO

FOR a:=3 TO S DIV 3 DO

FOR b:=a+1 TO (S‑a) DIV 2 DO

BEGIN

c:=S‑a‑b;

IF c\*c=a\*a+b\*b THEN BEGIN

k:=k+1;

WRITELN('Tripletul (a,b,c)',k:3,'= ',a:3, b:3,c:3);

END {IF}

END;

READLN;

END.

*28. PROBLEME PROPUSE*

*28.1.* Fie *i,j,k.*  Să se scrie un algoritm pentru ca să se determine restul împărţirii numărului natural *ij* la *k*.

*28.2.* Să se scrie un algoritm pentru ca să se tipărească toate tripletele *(i,j,k)* de numere naturale care verifică condiţiile

*i2 + j2 = k2*

*1 < i < j < k  n*

*28.3.* Să se scrie un algoritm pentru ca să se verifice dacă numărul *n* este perfect. (Un număr *n* este perfect dacă este egal cu suma divizorilor lui diferiţi de *n*; exemplu: 6=1+2+3).

*28.4.* Să se scrie un algoritm pentru determinarea numerelor perfecte din intervalul [*a,b*], pentru *a,b* date.

*28.5.* Două numere întregi *x* şi *y* sunt "prietene" dacă suma divizorilor numărului *x* este egală cu suma divizorilor numărului *y*. Să se scrie un algoritm pentru ca să se găsească numerele "prietene" din intervalul [*a,b*].

*28.6.* Să se scrie un algoritm pentru calcul şi să se tipărească primii *n* termeni din şirul Fibonacci, şir definit de relaţia de recurenţă

2, *i*=1,2,...

având 3.

*28.7.* Fie *n,k  Z+* , *nk*. Să se scrie un algoritm pentru calculul numărului combinărilor de *n* elemente luate câte *k*.

*28.8.* Fie *a  N*. Să se scrie un algoritm pentru calculul mediei aritmetice, geometrice şi armonice a tuturor divizorilor lui *a*.

*28.9.* Să se scrie un algoritm care să determine toate numerele întregi de trei cifre 4 cu proprietatea

5.

*28.10.* Să se scrie un algoritm pentru formula de generare a unui şir de numere (*yi*) este

6.

*28.11.* Să se scrie un algoritm care să exprime orice sumă de lei *S*, în minimum de monede de 1 leu, 3 lei, 5 lei, 10 lei , 20 lei, 50 lei şi 100 lei.

*28.12.* Să se scrie un algoritm pentru ca să se găsească soluţiile întregi şi pozitive ale ecuaţiei *ax + by = c*, cu proprietatea *x+y<n*, pentru *a,b,c aparţine lui Z* şi *n*>0.

*28.13.* Se dă *n  .* Să se scrie un algoritm pentru calcularea

7

*28.14.* Fie *a,b*,*c  +.* Să se scrie un algoritm pentru rezolvarea ecuaţiei



unde *d =* (*a,b*) este cel mai mare divizor comun al numerelor *a* şi *b*, iar *p* este probabilitatea ca un număr *n  * ce verifică condiţia *nc*, luat la întâmplare, să fie prim.

*28.15.* Se cere un algoritm pentru determinarea numerelor impare succesive a căror sumă este egală cu *n*3, pentru *n*=1,...,20. (Ex. 8, etc).

*28.16. Analizaţi algoritmul şi scriţi codul sursă pentru soluţionarea ecuaţiei de grad doi aX2+bX+c=0 (a,b,cR \_i a0).*

Metoda de rezolvare a ecuaţiei de gradul doi este cunoscută. Ecuaţia poate avea rădăcini reale, respectiv complexe, situaţie recunoscută după semnul discriminantului d = b2 - 4ac.

*28.17. Analizaţi algoritmul şi scriţi codul sursă pentru soluţionarea următoarelor probleme.* Ca model matematic vom folosi formula:

Formula nu este întotdeauna aplicabilă. Vom distinge următoarele situaţii:

1. a=0, caz în care nu avem de a face cu o ecuaţie de gradul 2, ci

1.1. este posibil să avem ecuaţia de gradul 1: bx+c=0, cu soluţia

x= - c/b , dacă b  0.

1.2. dacă şi b=0, atunci

* + 1. dacă c0, atunci nu avem nici o soluţie, în timp ce
    2. dacă şi c=0, atunci avem o infinitate de soluţii.

2. a0 corespunde ecuaţiei de gradul 2. In acest caz avem alte două situaţii:

* 1. Formula este aplicabilă pentru rădăcini reale (discriminant pozitiv)
  2. Pentru discriminant negativ, întrucât nu dispunem de aritmetică complexă, va trebui să efectuăm separat calculele pentru partea reală şi cea imaginară.

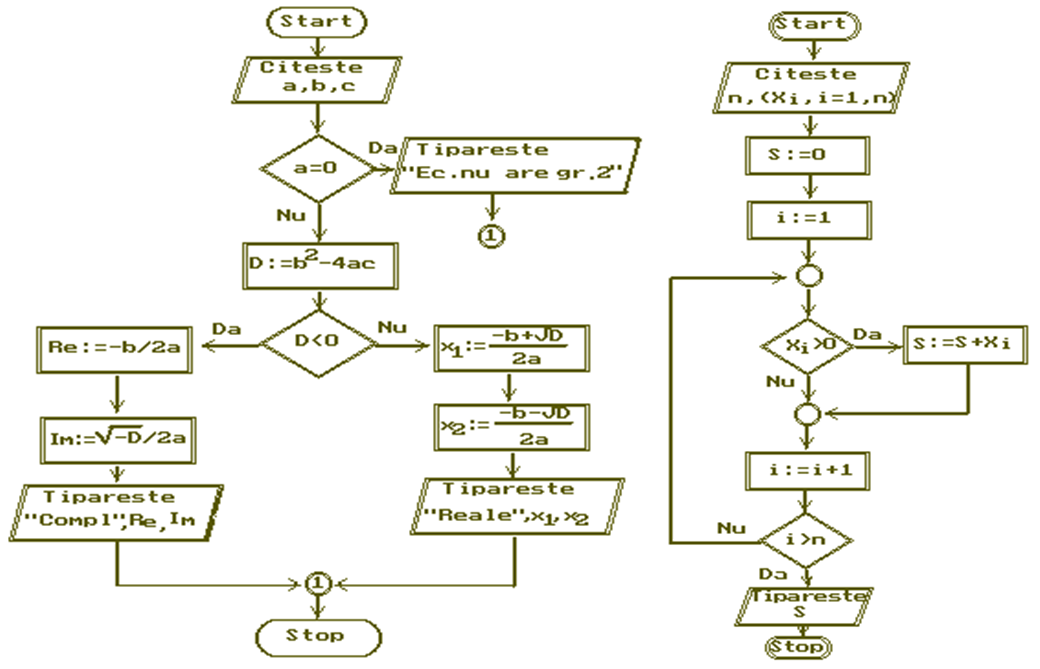


Fig.28.a. Algoritm pentru rezolvarea ecuaţiei de gradul doi Fig.28.b. Algoritm pentru calculul sumei vaorilor

pozitive a unui tablou unidimensional

Algoritmul de rezolvare a problemei va citi mai întâi datele problemei, marcate prin variabilele *a, b* şi *c*. Va calcula apoi discriminantul *d* şi va continua în funcţie de valoarea lui *d*, aşa cum se poate vedea în fig.28.a.

*1.27b. Analizaţi şi scriţi codul sursă pentru calcularea sumei elementelor pozitive ale unui şir de numere reale dat.*

Schema logică (dată în Fig.28.b) va conţine imediat după blocul START un bloc de citire, care precizează datele cunoscute în problemă, apoi o parte care calculează suma cerută şi un bloc de tipărire a sumei găsite, înaintea blocului STOP. Partea care calculează suma *S* cerută are un bloc pentru iniţializarea cu 0 a acestei sume, apoi blocuri pentru parcurgerea numerelor: x1, x2…xn şi adunarea celor pozitive la suma *S*. Pentru această parcurgere se foloseşte o variabilă contor *i*, care este iniţializată cu 1 şi creşte mereu cu 1 pentru a atinge valoarea *n*, indicele ultimului număr dat.

**Bibliografie:**

1. Cristea Valentin, Tehnici de programare. Ed.: Bucur., Teora, 1993. /681.3; T29/
2. Marşanu Radu. MS\_DOS şi UNIX. Sisteme de operare. Utilizare Windows, Lotus, WordPerfect. Bucur., Ed.Tehnicщ, 1995. /691.3; M32/
3. T. Bălănescu ш.a. PASCAL şi TurboPascal. v.1,2. Bucur., Ed.: Tehnica,1992.
4. 13. Vasiu L., Grama R., Aldica A. Turbo Pascal 6.0. Programe-Cl.-Napoca: Micro Informatica ,1994.-399 p. -(Ser.PC). ISBN 973-96274-2-0 CZU 681.3 V-35
5. Odagescu Ioan, Copos Cristina s.a. Metode si Tehnici de programare. /enunturi, solutii, probleme propuse/ Ed.:Bucur.: INTACT, 1994/681.3; O23/
6. Dr. Kris Jamsa si Lars Klander ’’Totul despre C si C++’’, traducere de Eugen Dumitrescu
7. Аммерал Л. Машинная графика на языке Си.(4 кн.) М.: СолСистем. 1993
8. Джехани Н. Программирование на языке Си. М.: Радио и св.. 1988. /681.3; Д409/
9. Уэйтм Н. И др. Язык Си. М.: Радио и св.. 1988./681.3; У97/
10. Берри Р., Микинз Б. Язык Си. Вв. для программистов. М.: Финансы и ст. 1988./681.3; Б518/
11. Programare calculatoarelor. Indrumar de laborator./A.Popescu, S. Marin s.a./- Chişinău.: UTM, 1996.