DEZVOLTAREA RAȚIONAMENTULUI

După ce v-ați pus de acord cu utilizatorul asupra scopurilor şi rezultatului programului, restul depinde numai de dumneavoastră. Treaba dumneavoastră este să luați acel rezultat şi să stabiliți cum puteți determina calculatorul să-l producă. Ați fragmentat problema globală în instrucțiuni detaliate, pe care calculatorul le poate executa. Aceasta nu înseamnă însă că sunteți gata de a scrie programul, ci din contra. Acum sunteți gata pentru a dezvolta raționamentul ce va produce acel rezultat.

Definirea datelor de ieşire parcurge o etapă bună din descrierea a ***ceea ce*** se presupune că trebuie să efectueze programul. Acum, dumneavoastră trebuie să hotărâți ***cum*** se va îndeplini aceasta. Trebuie să ordonați detaliile astfel încât acestea să opereze într-un mod ordonat în timp. De asemenea, trebuie să hotărâți ce decizii va lua programul dumneavoastră, ca şi acțiunile rezultate din fiecare dintre aceste decizii.

Organigramele

Organigramele şi instrumentele corespunzătoare de concepere a raționamentului constituie elementele de bază ale profesioniştilor din domeniul calculatoarelor. Complexitatea programelor cere ca să găsiți o modalitate oarecare de descriere a raționamentului programului, înainte de a-l scrie. Organigramele pot părea arhaice, dar se constată că programatorii ce nu utilizează astfel de instrumente sunt mai puțin productivi decât ar putea fi.

Se spune că o ilustrație face cât o mie de cuvinte, iar organigrama oferă o reprezentare ilustrată a logicii programului. **Organigrama** nu include toate detaliile programului, ci **reprezintă cursul logic** general al acestuia. Organigrama oferă logica pentru programul final. Dacă organigrama este corect desenată, scrierea efectivă a programului devine o chestiune de rutină. După terminarea întregului program, organigrama poate constitui documentația pentru acesta.

Organigrama descrie caseta centrală a modelului de intrare-prelucrare-ieşire, corespunzător programelor de calculator.

Organigramele sunt compuse din simboluri utilizate în standardele industriale:

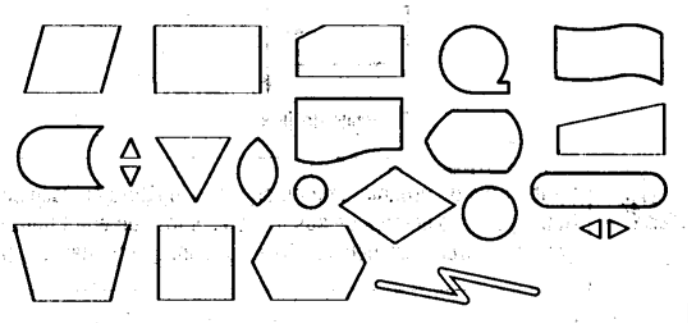

Fig. 1

Numai după ce ați învățat să programați, puteți începe să creați organigrama unui program; cu toate acestea, trebuie să realizați organigrame înainte de a scrie programe. Acest sindrom de tip „oul sau găina" este obişnuit pentru începătorii în programare. Atunci când veți începe să scrieți propriile programe, veți înțelege mult mai bine necesitatea organigramelor.

Aşa cum puteți constata există multe simboluri posibile. Cu toate acestea, numai câteva sunt utilizate în mod curent. Acest curs vă prezintă cel mai des întâlnite simboluri de organigramă. Cu cele pe care le veți cunoaşte aici, veți putea scrie şi realiza toate organigramele care vă vor fi vreodată necesare. Celelalte simboluri constituie doar rafinamente ale celor despre care veți afla aici.

În figura 2 sunt descrise simbolurile de organigramă utilizate în acest curs; ele sunt extrem de des utilizate în literatura referitoare la programare, în ceea ce priveşte proiectarea de sus în jos, puteți utiliza organigramele pentru a descrie orice eveniment, nu numai programele de calculator.

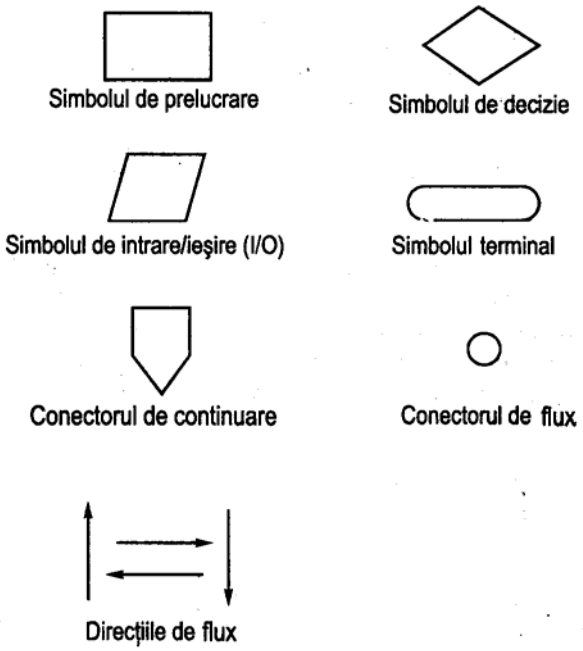

Fig. 2

Simbolurile de organigramă nu au nici o legătură cu casetele din proiectarea de sus în jos. Rețineți că proiectarea de sus în jos este instrumentul utilizat pentru obținerea detaliilor programului. Organigramele furnizează raționamentul necesar pentru obținerea acestor detalii.

În continuare sunt descrise aceste simboluri si modul lor de utilizare.

Prelucrare Conține o descriere a ceea ce se efectuează. Se utilizează un simbol de prelucrare atunci când are loc un proces direct de prelucrare a datelor, cum ar fi un calcul sau inițializare într-un program.

Decizie Utilizată atunci când programul trebuie să ia o decizie bazată pe două alternative cum ar fi tipărirea pe ecran sau la imprimantă, în funcție de locul în care doreşte utilizatorul să se efectueze tipărirea.

Intrare/Iesire Se utilizează pentru orice intrare sau ieşire efectuată de către program, cum ar fi adresarea unei întrebări utilizatorului sau tipărirea unui raport. (Forma înclinată a simbolului I/O indică semnificația sa; semnul"/" din prescurtarea I/O este înclinat în acelaşi mod.)

Terminal Simbolul terminal cu cuvântul START scris în interior este folosit întotdeauna pentru începerea oricărei organigrame. Simbolul terminal cu cuvintele STOP sau RETURN scrise în interior este folosit întotdeauna la sfârşitul oricărei organigrame. Atunci când vă veți referi ulterior la organigramă, nu se va pune întrebarea unde începe şi unde se sfârşeşte.

Continuare Se pune un conector de continuare în partea de jos a oricărei organigrame ce se continuă pe o altă pagină. Numărul paginii următoare se pune în interiorul conectorului de continuare. Se pune un conector de continuare la începutul fiecărei pagini ce continuă organigrama de pe o pagină anterioară. Numărul paginii precedente se pune în interiorul conectorului de continuare ce marchează începutul noii pagini din organigramă.

Conectorul de flux Se utilizează atunci când un flux logic al organigramei trebuie să fuzioneze cu sensul logic existent. De obicei se observă o literă a alfabetului în interiorul conectorului de flux. Un conector de flux corespunzător (cu aceeaşi literă) indică punctul de revenire la raționamentul existent.

Direcția de flux Săgețile conectează fiecare simbol din organigramă şi indică direcția de flux a programului.

Regulile de realizare a organigramelor

Cu toate că fiecare programator desenează organigramele în mod diferit, există câteva reguli clare pe care este bine să le cunoaşteți înainte de a continua. Aceste reguli sunt aproape universal valabile şi trebuie să le cunoaşteți, astfel încât organigramele să poată fi citite de şi alții. Alături de fiecare regulă veți putea observa câte un exemplu care urmează, respectiv încalcă regula, astfel încât vă veți putea face o idee despre modul de utilizare al regulii respective.

Regula 1: *utilizarea simbolurilor de organigramă standard.* Dacă folosiți simbolurile convenționale, ceilalți vor putea înțelege semnificația organigramei dumneavoastră, iar dumneavoastră le veți putea înțelege pe ale lor. În figura 3, sus sunt prezentate modalitățile corecte şi incorecte de a urma această regulă.

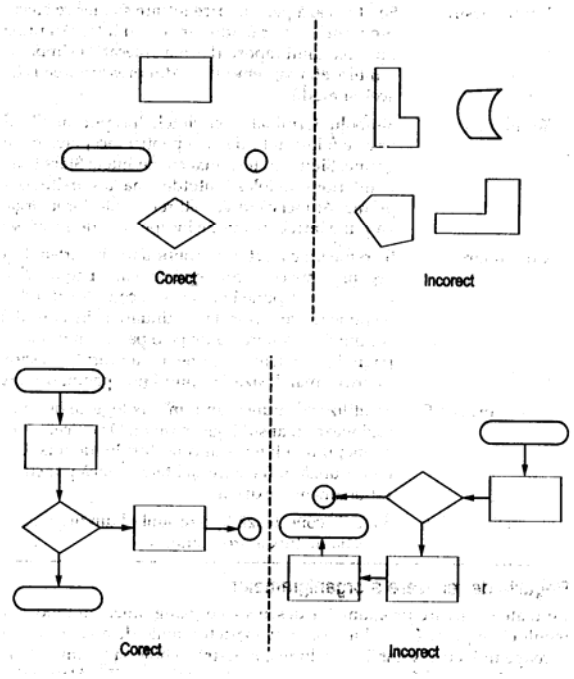

Fig. 3

Regula 2: *de obicei, raționamentul general al organigramei trebuie să se desfăşoare pe pagină de sus în jos şi de la stânga la dreapta.* Dacă organigramele dumneavoastră nu se conformează acestui standard, ele ar putea deveni dezordonate şi dificil de urmărit. În figura 3, jos este prezentată direcția corectă de desfăşurare a organigramelor. Observați că săgețile de direcție a fluxului indică sensul urmat de raționament.

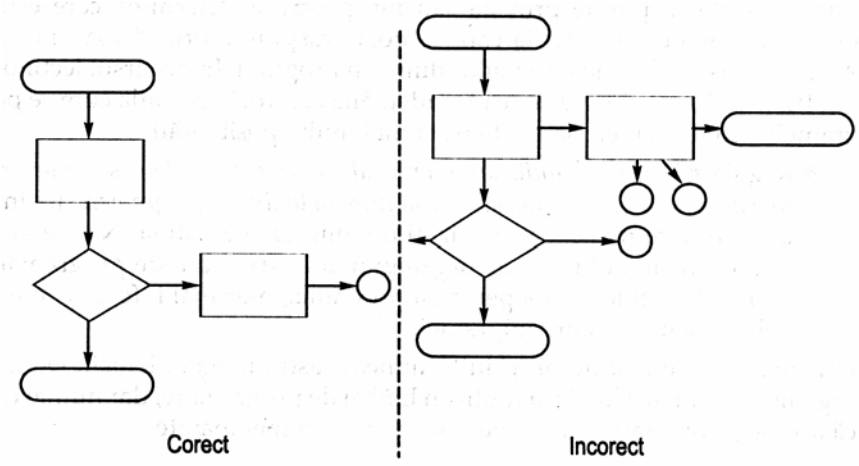

Fig. 4

Unele organigrame nu au nevoie să curgă către dreapta, deoarece descriu secvențial logica programului, însă majoritatea prezintă un fel de curs într-una dintre cele două direcții recomandate. Ar putea exista situații în care o organigramă corectă să pară că încalcă această regulă; în curând veți vedea un astfel de exemplu. Din cauza repetărilor din raționament, este posibil ca organigrama să aibă zone ce curg în sus şi către stânga pentru a repeta secțiuni din organigramă, dar, de fapt, firul logic trebuie să fie continuat în direcțiile preferențiale.

Firul logic *general* trebuie să curgă de sus în jos şi de la stânga la dreapta.

Regula 3: *simbolul de decizie este singurul care poate avea mai multe puncte de ieşire şi, de obicei, are două.* Majoritatea simbolurilor de organigramă au un punct de intrare şi unul de ieşire. Săgețile de direcție a fluxului indică punctele de intrare şi de ieşire. Simbolul de decizie are întotdeauna două puncte de ieşire deoarece în acel loc din raționament poate avea loc unul din două lucruri, iar fluxul ulterior al raționamentului este determinat de către rezultatul acelei decizii. În figura 4 este ilustrată această regulă.

Regula 4: *un simbol de decizie trebuie să conțină întotdeauna o întrebare la care se poate răspunde prin da sau nu.* Decizia dintr-o organigramă trebuie să aibă întotdeauna două şi numai două variante (de aici rezultă şi motivația celor două ieşiri de la Regula 3). Decizia se poate vedea în însăşi simbolul respectiv. Majoritatea simbolurilor de organigramă conțin cuvinte ce descriu ceea ce se întâmplă în acel loc din organigramă. Figura 5 ilustrează această regulă.

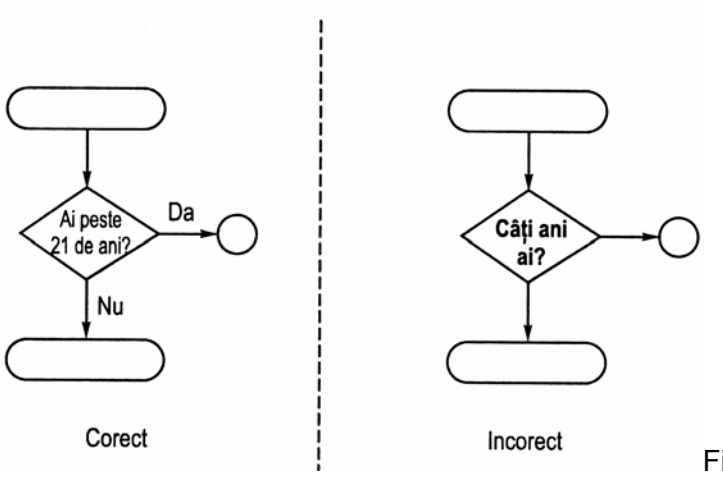

Fig. 5

Ieşirile din fiecare decizie trebuie, de asemenea, etichetate în mod clar. Deoarece ieşirile sunt rezultatele unei întrebări la care se poate răspunde prin da sau nu, acestea se etichetează prin *Da* sau Y, respectiv Nu sau N, astfel încât să se ştie ce indică ieşirea respectivă.

S-ar putea să vă întrebați dacă este rezonabil să vă aşteptați ca toate deciziile să aibă doar două rezultate posibile. Există momente în care programul dumneavoastră trebuie să aleagă o valoare dintre multe altele, bazându-se pe datele primite. Oricum ar fi, mai multe simboluri de decizie vor putea rezolva orice număr de posibilități.

Regula 5: *instrucțiunile din interiorul simbolurilor trebuie să fie descrieri clare în limba română, nu în „calculatorească" sau într-un limbaj de programare,* înainte de a scrie programul, este necesar să realizați organigrama acestuia. Nu trebuie să includeți în interiorul simbolurilor de organigramă instrucțiuni de programare. Dacă în acest moment ați fi fost gata pentru a programa, n-ar mai fi fost nevoie să cheltuiți timp pentru realizarea organigramei.

Organigrama constituie propriul dumneavoastră mers al logicii. De fapt, veți converti organigrama în instrucțiuni dintr-un limbaj de programare, dar numai după ce este sigur că aceasta reprezintă raționamentul corect , nu mai înainte.

Exemplu de organigramă

În figura 6 este prezentată organigrama de apelare telefonică a unui prieten. Pentru a păstra exemplul acesta în limite rezonabile, organigrama include aspectele importante ale problemei, cu toate că mai există încă multe alte modalități de a realiza organigrama.

Vedeți dacă puteți urmări organigrama. Săgețile de direcție vă arată cum să procedați. Urmăriți organigrama de mai multe ori, de la început până la sfârşit, având în vedere fiecare dintre aceste situații diferite:

1. Presupuneți că telefonul nu funcționează atunci când începeți apelul.
2. Presupuneți că prietenul dumneavoastră este acasă şi răspunde la telefon.
3. Presupuneți că telefonul prietenului dumneavoastră este ocupat.
4. Presupuneți că acasă la prietenul dumneavoastră nu este nimeni.
5. Presupuneți că prietenul dumneavoastră nu este acasă, dar răspunde colegul de cameră.

Observați că organigrama nu omite detaliile unui apel real. Pe de altă parte, nici nu caută să presupună prea mult. Pentru dumneavoastră, apelarea telefonică a unui prieten poate să însemne

simpla ridicare a receptorului şi apelarea, dar - atunci când organizați detaliile în organigramă - începeți să observați cât de multe lucruri ați considerat a fi subînțelese.

Observați cum vă ajută utilizarea unei organigrame pentru detalierea raționamentului programului la organizarea şi selectarea detaliilor necesare din program? În cursul anterior ați aflat cât de importantă este fragmentarea scopurilor acestuia în mai multe etape detaliate. Organigrama vă ajută să faceți tocmai aceasta.

Observați şi cum acționează simbolurile conectoarelor pentru a menține organigramă aerisită şi ordonată. Există mai multe locuri în care se repetă porțiuni din organigramă. De exemplu, dacă telefonul prietenului este ocupat, cercul conector notat cu *A* dirijează fluxul înapoi în partea de sus a organigramei. Raționamentele repetitive, cum ar fi acest; pare să meargă în sens opus celui de parcurgere al organigramei, contrar celei de-a doua reguli; în realitate nu se întâmplă aşa, deoarece raționamentul repetat continuă de fapt în jos şi spre dreapta când se va răspunde în sfârşit la telefonul prietenului.

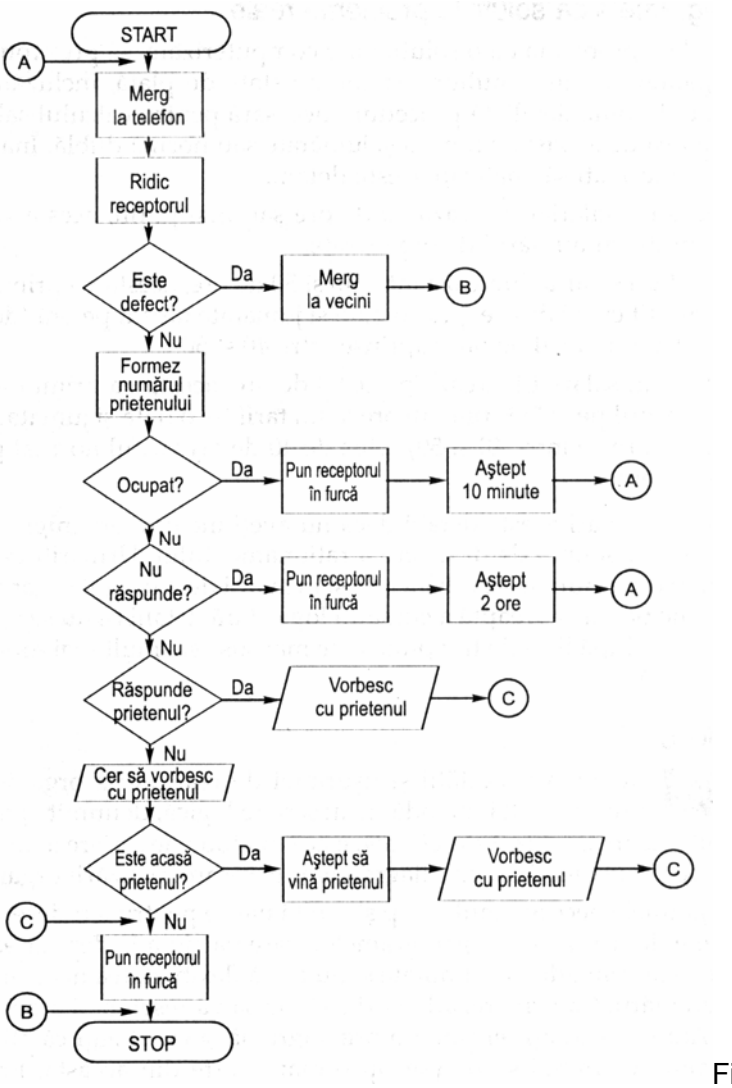

Fig. 6

Organigramele ca soluții la probleme reale

Trecând la o problemă cu o soluționare computerizată, să presupunem că doriți să realizați organigrama mai multor sisteme de state de plată, incluzând orele suplimentare efectuate. Trebuie

detaliată procedura necesară pentru calculul salariului net, dată fiind posibilitatea de a exista o normă şi jumătate sau normă dublă, înainte de a realiza organigrama, încercați să analizați aceste detalii:

• Dacă un salariat lucrează 40 de ore sau mai puțin, acesta va primi tariful pe oră înmulțit cu numărul de ore lucrate.

• Dacă un salariat lucrează între 40 şi 50 de ore, acesta va primi tariful normal pe oră înmulțit cu 40 de ore, plus o dată si jumătate tariful pe oră (de 1,5 ori tariful pe oră) pentru numărul de ore cuprinse între 40 şi 50.

• Dacă un salariat lucrează peste 50 de ore, acesta va primi un tarif dublu (de două ori tariful pe oră), plus 10 ore la un tarif de o dată şi jumătate (pentru numărul de ore cuprinse între 40 şi 50), plus de 40 de ori tariful normal pe oră pentru primele 40 de ore.

Deşi puteți urmări aceste detalii dacă nu aveți încotro, organigrama vă oferă o modalitate mult mai simplă de descriere a raționamentului. Urmăriți organigrama din figura 7. Luați orice număr de ore lucrate şi urmați-l de-a lungul organigramei. Organigrama vă menține pe calea dreaptă a cursului logic, fără detalii ce nu se aplică. A încerca să scrii un program după lista de trei puncte de mai sus este mult mai dificil decât a-l scrie după organigramă.

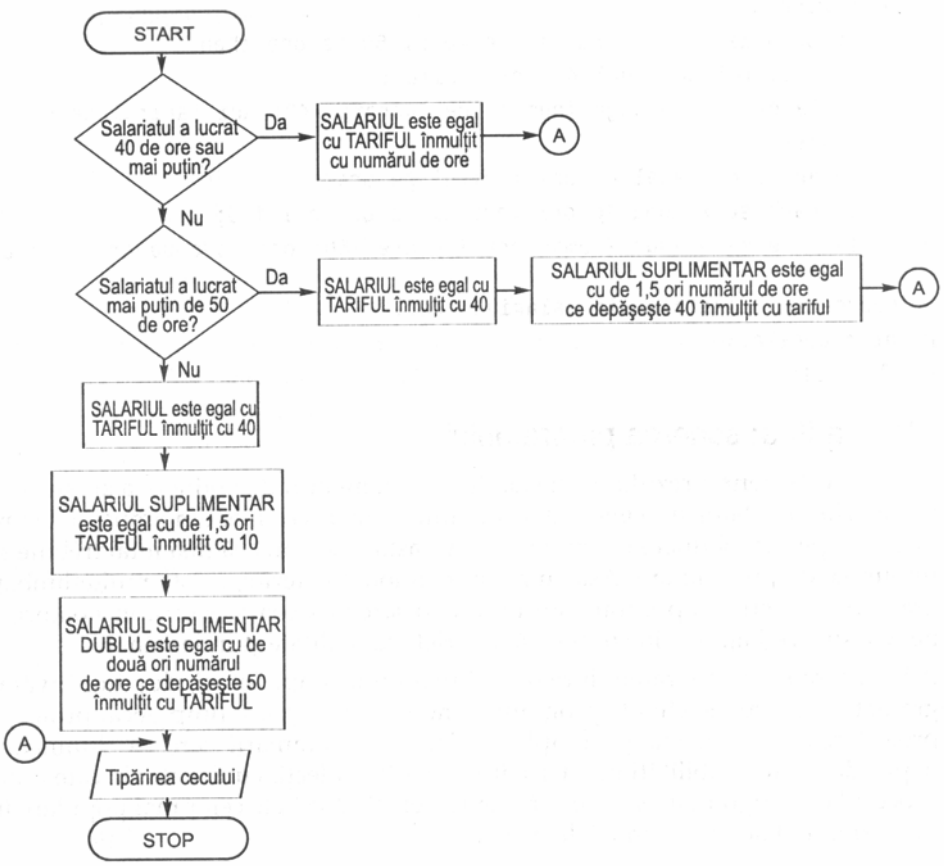

Fig. 7

Pseudocodul

În ciuda solidității si uşurinței de realizare a organigramelor, unele firme preferă o altă metodă de descriere logică, denumită pseudocod.

Pseudocodul, denumit uneori şi *engleză structurată,* este o metodă de scriere a raționamentului prin **utilizarea de propoziții**, în locul diagramelor necesare realizării organigramelor.

Organigramele necesită mult timp şi multă hârtie pentru a fi desenate. Cu toate că există programe de realizare a organigramelor, care vă ajută la desenarea si plasarea simbolurilor, acestea sunt adeseori limitate şi nu oferă flexibilitatea necesară de multe ori în logica programării. Ca urmare, adeseori trebuie să vă resemnați a desena organigramele cu mâna. Atunci când ați terminat o organigramă şi constatați că ați omis două simboluri importante, va trebui să redesenați o mare parte din aceasta. Prin natura lor, organigramele necesită mult timp pentru a fi desenate, iar unele firme nu acceptă ca programatorii lor să cheltuiască timp pentru realizarea acestora, atunci când pseudocodul poate să le înlocuiască si este mai eficient ca timp.

Singurul instrument necesar pentru pseudocod este un procesor de text. Procesoarele de text oferă capacitatea de a insera, muta şi a şterge porțiuni de text. Deoarece pseudocodul nu necesită desene, este mai rapid decât realizarea de mână a organigramelor şi mai uşor de întreținut. În plus, mulți programatori sunt de părere că logica pseudocodului este mai uşor de convertit în programul final.

Ca şi în cazul organigramelor, nu există o modalitate de a învăța pe cineva tehnicile pseudocodului, dacă nu a petrecut un timp oarecare programând. Prefixul *pseudo* înseamnă fals; prin urmare, pseudocodul înseamnă literal *cod fals.* Cu cât ştiți mai multe despre limbajele de programare, cu atât vă veți adapta mai bine la pseudocod, aşa că citiți următorul exemplu şi încercați să vă formați o părere despre acesta.

Pseudocodul nu conține instrucțiuni ale unui limbaj de programare, dar nici nu este o engleza cursivă. Este un set de cuvinte din limba engleză, ce permite descrierea logică, observate atât de des în organigrame şi limbaje de programare. Aşa cum se întâmplă şi cu organigramele, se poate scrie un pseudocod pentru orice, nu numai pentru programe de calculator. O mulțime de manuale de instruire utilizează o formă de pseudocod pentru a descrie etapele necesare asamblării componentelor. Pseudocodul oferă o descriere strictă a raționamentului, ce încearcă să lase puțin loc pentru ambiguități.

Mai jos este prezentat raționamentul în problema statelor de plată, sub formă de pseudocod. Observați că puteți citi textul, deşi nu este un limbaj de programare. Indentarea ajută la ținerea evidenței propozițiilor ce merg împreună. Pseudocodul poate fi citit de oricine, chiar şi de către persoanele nefamiliarizate cu simbolurile de organigramă:

FOR fiecare salariat:
IF salariatul a lucrat intre 0 si 40 ore **THEN**
    Salariul net egal numărul de ore lucrate ori tariful pe ora.
OTHERWISE
    IF salariatul a lucrat între 40 si 50 de ore **THEN**
        salariul net egal 40 ori tariful;
        la care se adaugă (număr ore lucrate -40) ori tariful pe ora ori 1,5.
    OTHERWISE
        salariul net egal 40 ori tariful pe ora;
        la care se adaugă 10 ori tariful pe ora ori 1,5;l
        la care se adaugă (număr ore lucrate -50) ori de doua ori tariful pe ora.
    Deduceți impozitele din salariul brut.
PRINT cecurile
END problema

Etapa a III-a: Scrierea programului

După ce este definit rezultatul şi stabilit raționamentul de obținere a acestuia, trebuie să mergeți la calculator şi să generați programul - instrucțiunile din limbajul de programare - necesar pentru obținerea rezultatului. Aceasta înseamnă că mai întâi trebuie să învățați un limbaj de programare. Aşa cum s-a menționat anterior, există multe limbaje de programare, iar scrierea programelor nu este o sarcină uşoară pentru începători, dar dumneavoastră veți ajunge în curând să le scrieți cu abilitate si îndemânare.

A învăța să scrii programe durează cel mai mult timp. Totuşi, după ce învățați să programați, procesul efectiv de programare necesită mai puțin timp decât proiectarea, dacă proiectarea este corectă si completă. Natura programării cere ca dumneavoastră să deprindeți câteva abilități noi.

Procesul de programare şi tehnicile structurate

Acum cunoaşteți etapele ce trebuie parcurse înainte de programare, în decursul următoarelor cursuri vă veți lansa în procesul de programare, începând cu învățarea limbajului C şi scrierea propriilor dvs. programe.

Acum ştiți că două etape trebuie să preceadă întotdeauna scrierea programului - stabilirea rezultatelor de ieşire şi dezvoltarea raționamentului. După ce ați dezvoltat logica, puteți scrie programul utilizând unul dintre multele limbaje de programare disponibile.

Pentru a încheia procesul de programare, trebuie să scrieți, să testați şi să distribuiți programul. Desigur că toate aceste trei etape pornesc de la presupunerea că ştiți un limbaj de programare. Restul cursului va fi dedicat completării procesului de programare, început cu proiectarea rezultatului programului şi dezvoltarea raționamentului.

În continuare veți afla răspunsurile la următoarele chestiuni:

- Ce este necesar pentru scrierea programelor,
- Ce este un editor;
- De ce este atât de importantă programarea structurată;
- Care sunt cele trei construcții din programarea structurata;
- Ce etape sunt necesare pentru testarea unui program;
- Care este diferența dintre verificarea de birou şi testarea beta;
- De ce este atât de importantă testarea paralelă.

Utilizarea unui editor

Editorul este instrumentul utilizat pentru scrierea programelor de calculator. Un editor este asemănător cu un procesor de text, prin aceea că oferă posibilitatea de a scrie linii de program, de a le edita, de a le muta, copia şi salva pe disc. Din acest motiv, editoarele sunt adeseori denumite şi *editoare de text.* Un editor diferă de un procesor de text prin aceea că nu efectuează *aranjarea automată a cuvintelor,* într-un procesor de text, atunci când ați ajuns la capătul rândului, procesorul mută cursorul şi porțiunea de cuvânt de la sfârşitul rândului pe rândul următor. Aranjarea automată a cuvintelor ar constitui un impediment pentru programele de calculator.

Rețineți că limbajele de programare sunt concise. Instrucțiunile de program nu pot fi executate împreună ca un fel de vorbire tipărită, în anumite limbaje de programare se pot pune mai multe instrucțiuni pe acelaşi rând, dar acest obicei nu este recomandat deoarece face citirea programului mai dificilă pentru dumneavoastră si pentru alții. Cu cât un program este mai greu de citit, cu atât va fi ulterior mai dificil de întreținut şi de actualizat.

Programarea structurată

La sfârşitul anilor '60, departamentele de programare au început să se poticnească în sarcini nerezolvate, ce creşteau în ritmuri incredibile. Mai mulți oameni scriau mai multe programe ca oricând, în timp ce mulți programatori trebuiau angajați pentru întreținerea programelor scrise anterior.

Atunci când un program este terminat, el este considerat finalizat doar pentru momentul respectiv. Ipotezele dintr-un program referitoare la sarcina îndeplinită de program se vor schimba în timp. Întotdeauna întreprinderile evoluează în această economie globală. Responsabilii cu prelucrarea datelor începuseră să recunoască faptul că povara întreținerii programelor începea să facă ravagii în detrimentul dezvoltării. Programatorii erau retraşi de la noile proiecte pentru a le actualiza pe cele vechi, întreținerea dura prea mult.

Învățați să scrieți programe ce pot fi citite şi întreținute. Firmele economisesc bani atunci când un programator scrie programe uşor de întreținut.

În timpul crizei întreținerii programelor din anii '60, persoanele ce se ocupau de prelucrarea datelor au început să caute noi modalități de programare. Nu erau neapărat interesați de noi limbaje, ci mai degrabă de noi modalități de a scrie programe, care să le facă să funcționeze mai bine şi mai rapid şi - ceea ce este cel mai important - să le facă uşor de citit, astfel încât ceilalți să le poată întreține fără prea mare bătaie de cap. Tehnicile de programare structurată au fost concepute în această perioadă.

Programarea structurată este o filosofie ce declară că programele ar trebui scrise într-un mod ordonat, fără multe salturi înainte şi înapoi. Dacă un program este conceput astfel încât să poată fi citit cu uşurință, atunci el poate fi mai uşor modificat. Oamenii ştiau de mulți ani că un stil de scriere clar este important, dar acest lucru a devenit evident pentru cei din domeniul calculatoarelor numai după aproape 20 de ani de utilizare a tehnicilor nestructurate.

Există unele controverse referitoare la momentul exact în care programatorii începători ar trebui să înceapă să folosească programarea structurată. Unii sunt de părere că programatorii trebuie instruiți să folosească programarea structurată încă de la început. Alții cred că începătorii trebuie să învețe să programeze în orice mod în care pot rezolva problema respectivă, ca apoi să se adapteze la programarea structurată.

Acum, când ați înțeles organigramele şi limbajul pseudocod, puteți vedea în ce constă programarea structurată cu ajutorul acestor instrumente. Când veți învăța un limbaj de programare veți gândi în stilul programării structurate şi vă veți încadra încă de la început, în mod natural, în tiparul acesteia.

Un program bine scris şi uşor de citit nu înseamnă că este neapărat structurat. Programarea structurată este o abordare specifică a programării, care produce în general programe bine scrise si uşor de citit. Nimic nu poate convinge la o schimbare un programator ce se grăbeşte să termine un program prin ceea ce crede el că este cea mai rapidă modalitate. Adeseori se aud replici ca: „Mai târziu îl voi face structurat, dar deocamdată îl voi lăsa aşa cum este". Acest *„mai târziu"* nu vine

niciodată. Oamenii utilizează un program până într-o zi când trebuie efectuate modificări, iar acestea durează tot atât de mult cât - sau chiar mai mult decât - ar fi nevoie pentru a-l arunca la gunoi şi a-l rescrie de la zero.

Programarea structurată include următoarele trei construcții:

- Secvența;
- Decizia (denumită şi *selecție);*
- Bucla (denumită şi *repetiție* sau *iterație).*

O *construcție* este un bloc de construcție al unui limbaj şi una dintre operațiile fundamentale ale acestuia. Atâta vreme cât un limbaj de programare acceptă aceste trei construcții (majoritatea acceptă), puteți scrie programe structurate. Opusul unui program structurat este cunoscut sub denumirea de *program spaghetti.* Ca şi spaghetele care se revarsă şi se încolăcesc peste tot în farfurie, un program nestructurat - unul plin de instrucțiuni spaghetti - se revarsă peste tot si nu are nici un fel de structură. Un program nestructurat conține o mulțime de *ramificații.* O ramificație apare atunci când un program merge pe o cale sau alta, fără nici o ordine.

Majoritatea limbajelor de programare permit ramificarea printr-o instrucțiune GOTO. Instrucțiunea GOTO funcționează după cum sună; ea îi comunică calculatorului să treacă în alt loc din program şi să continue execuția de acolo. Căutarea următoarei instrucțiuni de executat din program te face să-ți pierzi şirul gândirii.

Unii programatori şi unele manuale de programare vă avertizează să vă țineți departe de instrucțiunea GOTO. Instrucțiunea GOTO în sine nu este rea, atunci când este utilizată moderat, dar poate face ravagii în posibilitatea de a citi un program, dacă este utilizată prea des.

Cele trei construcții din programarea structurată nu se aplică doar programelor. Veți descoperi că le puteți utiliza în organigrame, pseudocod şi în orice alt set de instrucțiuni pe care le scrieți pentru alții. Construcțiile din programarea structurată garantează că un program nu se ramifică peste tot şi că orice execuție este controlată şi uşor de urmărit.

În continuare se explică fiecare din cele trei construcții din programarea structurată. Citiți-le cu atenție şi veți constata că programarea structurată este un concept uşor de înțeles, învățând despre structură înainte de a învăța un limbaj, veți ajunge să vă gândiți automat la aceasta atunci când vă veți dezvolta abilitățile de programare.

Secvența

Secvența nu este nimic altceva decât un set de două sau mai multe instrucțiuni, una după alta. Instrucțiunile secvențiale reprezintă cea mai simplă din cele trei construcții din programarea structurată, deoarece permit urmărirea programului de la prima până la ultima instrucțiune din secvență, în figura 8 este prezentată o organigramă ce ilustrează o secvență.

Pseudocodul corespunzător secvenței din organigramă este:

GET numărul de ore lucrate.
MULTIPLY numărul de ore cu tariful pe ora.
SUBTRACT impozitele pentru a calcula salariul net
PRINT cecul.

Deoarece calculatoarele trebuie să aibă capacitatea de a lua decizii şi de a executa secvențe repetitive, nu toate programele dumneavoastră pot consta doar dintr-un raționament repetitiv liniar.

Totuşi, secvența merită să fie utilizată pentru raționamentul liniar de programare, atunci când este posibil.

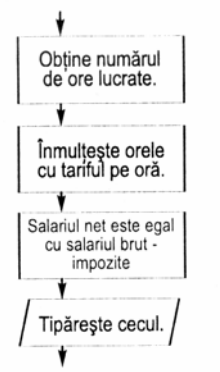

Fig 8

Decizia (Selecția)

Ați întâlnit deja construcția de *decizie,* într-o organigramă, simbolul de decizie este punctul în care este luată decizia. Ori de câte ori un program ia o decizie, trebuie să o ia într-o direcție din două. Evident că o decizie reprezintă o rupere a cursului secvențial al programului, dar este o rupere controlată.

Prin natura sa, o ramificare trebuie efectuată pe baza rezultatului unei decizii (de fapt, programul trebuie să omită instrucțiunile ce nu trebuie executate). Totuşi, spre deosebire de o ramificație dreaptă, o decizie garantează că nu trebuie să vă bateți capul cu secvența ce nu este executată. Nu va trebui să vă întoarceți înapoi şi să citiți partea din program omisă de către decizie, (în funcție de noile date, este posibil ca programul să repete o decizie şi să ia o cale diferită a doua oară, dar, pe de altă parte, puteți întotdeauna presupune că secvența de decizie ce nu este executată în momentul respectiv este irelevantă pentru bucla curentă.)

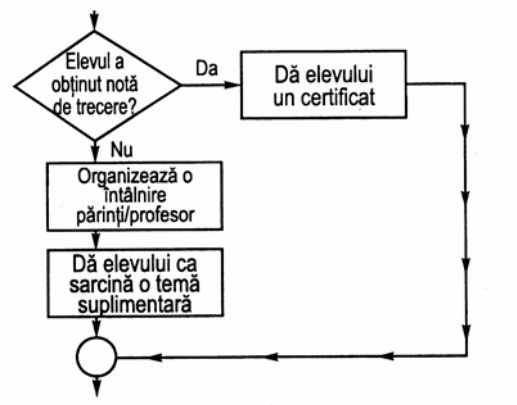

Fig. 9

Pseudocodul corespunzător deciziei prezentate în organigramă este:

IF elevul a obținut A sau B,
    dă elevului un certificat de absolvire.
OTHERWISE:
    Organizează o întâlnire părinți/profesor;
dă elevului o tema suplimentara.

Executarea buclelor sau ciclarea (repetiția şi iterația)

Poate că cea mai importantă sarcină a calculatoarelor o constituie executarea de *bucle* sau *ciclarea* (termenul utilizat pentru repetarea sau iterarea liniilor din program). Calculatoarele repetă porțiuni de program de milioane de ori şi nu se plictisesc niciodată. Calculatoarele sunt tovarăşi perfecți pentru cei care au o mulțime de date de prelucrat, deoarece ele pot prelucra datele, repetând pentru toate datele calculele obişnuite, în timp ce omul poate analiza rezultatele.

Executarea de bucle prevalează în aproape orice program scris. Rareori se întâmplă să scrieți un program format dintr-o secvență liniară de instrucțiuni. Timpul necesar pentru proiectarea si scrierea unui program nu merită întotdeauna efortul, atunci când este vorba doar de o serie liniară de instrucțiuni. Programul funcționează la capacitate maximă atunci când poate repeta o serie de instrucțiuni secvențiale sau decizii.

În figura 10 este prezentată o organigramă ce repetă o secțiune dintr-o buclă. Buclele încalcă doar temporar regula ce spune că organigramele trebuie să curgă în jos şi spre dreapta. Buclele din interiorul organigramei sunt corecte, deoarece, în final, raționamentul va înceta să execute bucle.

Nu uitați de cumplita *buclă infinită.* O *buclă infinită este* o buclă ce nu se termină niciodată, în cazul în care calculatorul dumneavoastră intră într-o buclă infinită, el va continua să o execute, fără a termina vreodată, iar uneori este dificil de recăpătat controlul asupra programului fără a reporni calculatorul. Buclele ar trebui întotdeauna prefațate de către o instrucțiune de decizie, astfel încât, în cele din urmă, decizia să declanşeze sfârşitul buclei, pentru ca restul programului să poată fi executat.

Pseudocodul corespunzător organigramei este:

IF sunt mai mulți cumpărători,
    DO următoarele:
        CALCULATE balanța financiara pentru următorul cumpărător;
        PRINT o factura.
OTHERWISE,
    PRINT raportul total al balanței financiare.

După cum vedeți, în cele din urmă nu vor mai fi cumpărători, iar bucla (ce începe cu „do") va înceta a fi efectuată, astfel încât restul logicii va putea prelua comanda.

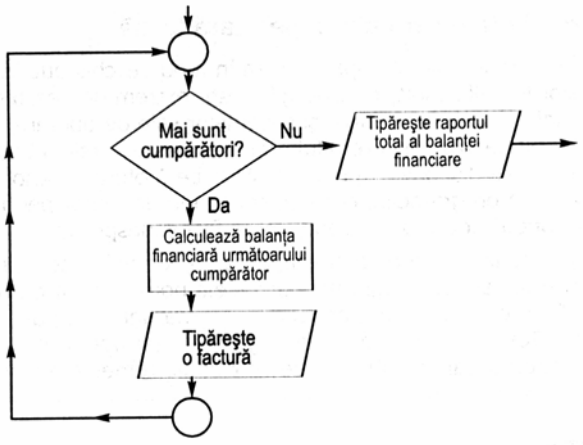

Fig. 10

Testarea programului

Când ați încheiat scrierea efectivă a programului, nu înseamnă că ați terminat complet programul. Trebuie să vă îndreptați atenția către depanarea programului. Desigur că doriți să eliminați cât mai multe hibe posibil din program. Din motive evidente, nu veți dori ca utilizatorul să facă aceasta. Nu veți dori ca utilizatorul programului să găsească tot felul de greşeli făcute de dumneavoastră. Deci, va trebui să testați în amănunt programul.

Mai jos sunt enumerate etapele de testare caracteristice, pe care trebuie să le urmeze programatorii înainte de a distribui utilizatorilor o versiune finală a programului:

1. Efectuarea verificării de birou.
2. Efectuarea unui test beta.
3. Compararea rezultatelor testului beta cu rezultatele testului făcut în paralel pentru vechiul sistem.

Majoritatea programatorilor îşi trec programele printr-o serie de verificări de birou. *Verificarea de birou* înseamnă a sta în fața calculatorului şi a verifica programele utilizând cât de multe cazuri posibile cu date diferite, pentru a depista punctele slabe şi erorile din program. În timpul verificării de birou, programatorii trebuie să testeze valori limită, să tasteze datele de intrare greşite şi în general, să facă tot ce pot pentru a face programul să dea greş. Programatorii trebuie să încerce fiecare opțiune din program, utilizând diverse combinații pentru a vedea ce se întâmplă în toate situațiile posibile.

Atunci când s-a terminat verificarea de birou şi programatorii pot fi cât de siguri se poate că programul este corect, trebuie ca aceştia să desemneze un grup de utilizatori care să încerce să folosească programul, în domeniu, această etapă este cunoscută sub denumirea de faza de *testare beta.* Cu cât veți găsi mai multe persoane care să-l testeze (utilizatori de testare ai programului), cu atât sunt şanse mai bune ca erorile să fie găsite. Adeseori utilizatorii încearcă să facă lucruri care nici nu i-au trecut prin cap programatorului atunci când a scris programul.

Testarea beta există acum pe scară largă

Din ce în ce mai multe companii invită în mod deschis publicul să ajute la testarea beta a produselor lor. Microsoft, de exemplu, este extrem de deschisă în ceea ce priveşte distribuirea de versiuni beta ale aplicațiilor şi sistemelor sale de operare, cu mult înainte ca produsul final să fie disponibil pentru vânzare. Multe dintre aceste versiuni beta pot fi descărcate din Internet. Aceste produse disponibile pentru testare beta oferă criticilor şi celor ce le testează o primă privire asupra programelor, ceea ce constituie un ajutor pentru cei de la Microsoft, deoarece aceşti voluntari pot informa conducerea firmei despre hibele găsite.

Pe măsură ce audiența beta creşte, creşte şi timpul necesar unei firme pentru testarea produsului. Problema este că programele din zilele noastre sunt de o înaltă complexitate, necesitând chiar 100 de programatori pentru producerea unui program cum ar fi o nouă versiune de Windows. Testarea beta pe scară largă este aproape singura modalitate prin care aceste firme pot descoperi o parte dintre hibele ce trebuie remediate înainte de lansarea produsului.

Utilizatorul nu trebuie să renunțe niciodată la un sistem vechi şi să treacă deodată la noul program. Trebuie efectuată mai întâi *testarea în paralel.* De exemplu, dacă scrieți un program pentru state de plată ce va înlocui sistemul manual utilizat într-o curățătorie chimică, nu este indicat ca proprietarul să primească o copie a programului şi s-o utilizeze imediat numai pe aceasta. În schimb, este bine ca acesta să continue sistemul manual de efectuare a statului de plată şi să utilizeze

simultan programul dumneavoastră. Cu toate că aceasta presupune un timp ceva mai lung de fiecare dată când se calculează salariile, *se* pot compara rezultatele programului cu cele ale sistemului manual, pentru a vedea dacă corespund.

Numai după câteva perioade în care plata se face cu testare în paralel, utilizatorul poate căpăta destulă încredere pentru a utiliza programul fără copii de siguranță manuale.

În timpul acestei perioade de testare, este posibil să fie necesar ca programatorii să efectueze mai multe schimbări în program. Aşteptați-vă să fie necesare modificări şi nu vă veți simți dezamăgit, pierzându-vă încrederea în programare. Arareori se întâmplă ca programatorii să scrie un program corect de prima dată. De obicei, sunt necesare multe încercări pentru a realiza programe corecte. Testarea în amănunt descrisă în acest paragraf nu constituie o asigurare perfectă că programul este corect. Unele erori pot apărea numai după ce programul este utilizat o vreme. Cu cât efectuați mai multe testări, cu atât este mai puțin probabil să apară ulterior erori.

Învățarea limbajului de programare

Deocamdată, în acest curs a fost lăsată deoparte o etapă importantă. Aceasta este etapa de învățare a limbajului de programare. Restul cursului este dedicat descrierii limbajului C.