NOȚIUNI INTRODUCTIVE

1

1.1. STRUCTURA GENERALĂ A UNUI SISTEM DE CALCUL

Calculatorul reprezintă un sistem electronic (ansamblu de dispozitive şi circuite diverse) complex care prelucrează datele introduse într-o formă prestabilită, efectuează diverse operații asupra acestora şi furnizează rezultatele obținute (figura 1.1.).

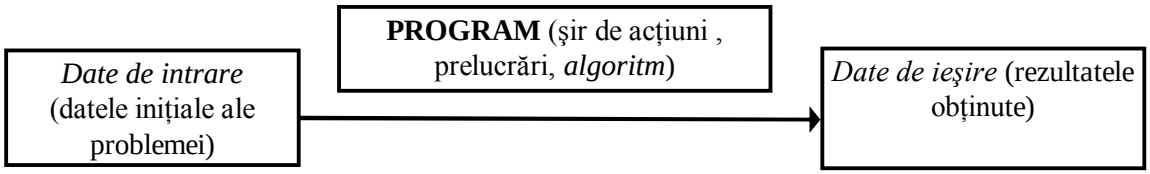


Figura 1.1. Calculatorul - sistem automat de prelucrare a datelor

Principalele avantaje ale folosirii calculatorului constau în:
- viteza mare de efectuare a operațiilor;
- capacitatea extinsă de prelucrare şi memorare a informației.

Deşi construcția unui calculator - determinată de tehnologia existentă la un moment dat, de domeniul de aplicație, de costul echipamentului şi de performanțele cerute - a evoluat rapid în ultimii ani, sistemele de calcul, indiferent de model, serie sau generație, au o serie de **caracteristici comune**. Cunoaşterea acestor caracteristici uşurează procesul de înțelegere şi învățare a modului de funcționare şi de utilizare a calculatorului.

În orice sistem de calcul vom găsi două părți distincte şi la fel de importante: **hardware**-ul şi **software**-ul.
- Hardware-ul este reprezentat de totalitatea echipamentelor şi dispozitivelor fizice;
- Software-ul este reprezentat prin totalitatea programelor care ajută utilizatorul în rezolvarea problemelor sale (figura 1.2.).

  Software-ul are două componente principale:
  - Sistemul de operare (de exploatare) care coordonează întreaga activitate a echipamentului de calcul. Sistemul de operare intră în funcțiune la pornirea calculatorului şi asigură, în principal, trei funcții:
    - ✓ Gestiunea echitabilă şi eficientă a resurselor din cadrul sistemului de calcul;
    - ✓ Realizarea interfeței cu utilizatorul;
    - ✓ Furnizarea suportului pentru dezvoltarea şi execuția aplicațiilor.

  Exemple de sisteme de operare: RSX11, CP/M, MS-DOS, LINUX, WINDOWS NT, UNIX.
  - Sistemul de aplicații (de programare): medii de programare, editoare de texte, compilatoare, programe aplicative din diverse domenii (economic, ştiințific, financiar, divertisment).

Componentele unui sistem de calcul pot fi grupate în unități cu funcții complexe, dar bine precizate, numite **unități funcționale**. Modelul din figura 1.3. face o prezentare simplificată a structurii unui calculator, facilitând înțelegerea unor noțiuni şi concepte de bază privind funcționarea şi utilizarea acestuia. Denumirea fiecărei unități indică funcția ei, iar săgețile - modul de transfer al informației.

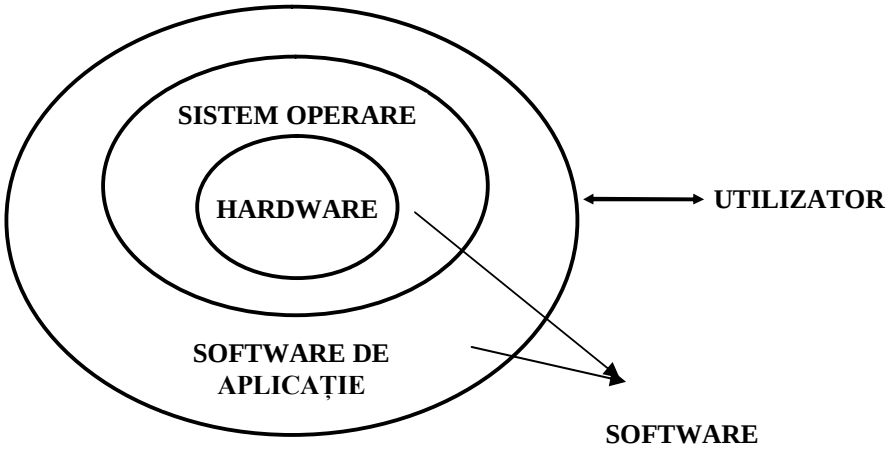


Figura 1.2. Echipamentul de calcul ca un sistem hardware-software

Vom utiliza în continuare termenii de **citire** pentru operația de introducere (de intrare) de la tastatură a datelor inițiale ale unei probleme, şi **scriere** pentru operația de afişare (de ieşire) a rezultatelor obținute. În cazul în care utilizatorul doreşte să rezolve o problemă cu ajutorul calculatorului, informația de intrare (furnizată calculatorului de către utilizator) va consta din datele inițiale ale problemei de rezolvat şi dintr-un program (numit program sursă). În programul sursă utilizatorul implementează (traduce) într-un limbaj de programare un **algoritm** (acțiunile executate asupra datelor de intrare pentru a obține rezultatele). Această informație de intrare este prezentată într-o **forma externă**, accesibilă omului (numere, text, grafică) şi va fi transformată de către calculator într-o **forma internă**, binară.

Unitatea de intrare (cu funcția de citire) realizează această conversie a informației din format extern în cel intern. Din punct de vedere logic, fluxul (informația) de intrare este un şir de caractere, din exterior către memoria calculatorului. Din punct de vedere fizic, unitatea de intrare standard este tastatura calculatorului. Tot ca unități de intrare, pot fi enumerate: mouse-ul, joystick-ul, scanner-ul (pentru introducerea informațiilor grafice).

Unitatea de ieşire (cu funcția de scriere, afişare) realizează conversia inversă, din formatul intern în cel extern, accesibil omului. Din punct de vedere fizic, unitatea de ieşire standard este monitorul calculatorului. Ca unități de ieşire într-un sistem de calcul, mai putem enumera: imprimanta, plotter-ul, etc.

Informația este înregistrată în **memorie**.
Memoria internă (memoria **RAM - R**andom **A**cces **M**emory) se prezintă ca o succesiune de octeți (octet sau **byte** sau locație de memorie). Un octet are 8 **biți**. **Bit**-ul reprezintă unitatea elementară de informație şi poate avea una din valorile: 0 sau 1.

Capacitatea unei memorii este dată de numărul de locații pe care aceasta le conține şi se măsoară în multiplii de 1024 (2^{10}). De exemplu, 1 Mbyte=1024Kbytes; 1Kbyte=1024bytes.
Numărul de ordine al unui octet în memorie se poate specifica printr-un cod, numit **adresă**. Ordinea în care sunt adresate locațiile de memorie nu este impusă, memoria fiind un dispozitiv cu acces aleator la informație.

În memorie se înregistrează două categorii de informații:
- Date - informații de prelucrat;
- Programe - conțin descrierea (implementarea într-un limbaj de programare) a acțiunilor care vor fi executate asupra datelor, în vederea prelucrării acestora.

În memoria internă este păstrată doar informația prelucrată la un moment dat. Memoria internă are capacitate redusă; accesul la informația pastrată în aceasta este extrem de rapid, iar datele nu sunt păstrate după terminarea prelucrării (au un caracter temporar).

Unitatea centrală prelucrează datele din memoria internă şi coordonează activitatea tuturor componentelor fizice ale unui sistem de calcul. Ea înglobează:

- **Microprocesoru**l- circuit integrat complex cu următoarele componente de bază:
  - ✓ Unitatea de execuție (realizează operații logice şi matematice);
  - ✓ Unitatea de interfață a magistralei (transferă datele la/de la microprocesor).
- **Coprocesorul matematic** – circuit integrat destinat realizării cu viteză sporită a operațiilor cu numere reale.

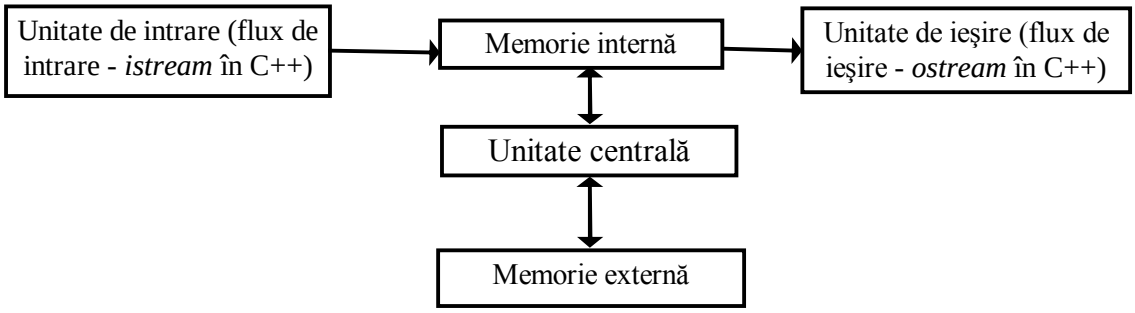


Figura 1.3. Unitățile funcționale ale unui sistem de calcul

În funcție de numărul de biți transferați simultan pe magistrala de date, microprocesoarele pot fi clasificate astfel: microprocesoare pe 8 biți (Z80, 8080); microprocesoare pe 16 biți (8086, 8088, 80286) cu coprocesoarele corespunzătoare (8087, 80287); familii de procesoare pe 32 biți (80386DX, 80486, PENTIUM) cu coprocesoarele corespunzătoare (începând de la 486, coprocesoare sunt încorporate microprocesoarelor).

Memoria externă este reprezentată, fizic, prin unitățile de discuri (discuri dure-**hard disk**, discuri flexibile-**floppy disk**, discuri de pe care informația poate fi doar citită-CDROM, DVDROM, etc). Spre deosebire de memoria internă, memoria externă are capacitate mult mai mare, datele înregistrate au caracter permanent, în dezavantajul timpului de acces la informație.

1.2. ALGORITMI

1.2.1. NOȚIUNI GENERALE

Algoritmul este *conceptul fundamental* al informaticii. Orice echipament de calcul poate fi considerat o maşină algoritmică. Într-o *definiție aproximativă algoritmul este un set de paşi care defineşte modul în care poate fi dusă la îndeplinire o anumită sarcină.* Exemplu de algoritm: algoritmul de interpretare a unei bucăți muzicale (descris în partitură). Pentru ca o maşină de calcul să poată rezolva o anumită problemă, programatorul trebuie mai întâi să stabilească un algoritm care să conducă la efectuarea la sarcinii respective.

Exemplu:

Algoritmul lui Euclid pentru determinarea celui mai mare divizor comun (cmmdc) a 2 numere întregi pozitive.

Date de intrare: cele 2 numere întregi

Date de iesire: cmmdc

1. Se notează cu A şi B- cea mai mare, respectiv cea mai mică, dintre datele de intrare
2. Se împarte A la B şi se notează cu R restul împărțirii

3. a. Dacă R diferit de 0, se atribuie lui A valoarea lui B şi lui B valoarea lui R. Se revine la pasul 2.
   b. Dacă R este 0, atunci cmmdc este B.

Probleme legate de algoritmi

Descoperirea unui algoritm care să rezolve o problemă echivalează în esență cu descoperirea unei soluții a problemei. După descoperirea algoritmului, pasul următor este ca algoritmul respectiv să fie reprezentat într-o formă în care să poată fi comunicat unei maşini de calcul. Algoritmul trebuie transcris din forma conceptuală într-un set clar de instrucțiuni. Aceste instrucțiuni trebuie reprezentate într-un mod lipsit de ambiguitate. În acest domeniu, studiile se bazează pe cunoştințele privitoare la gramatică şi limbaj şi au dus la o mare varietate de scheme de reprezentare a algoritmilor (numite limbaje de programare), bazate pe diverse abordări ale procesului de programare (numite paradigme de programare).

Căutarea unor algoritmi pentru rezolvarea unor probleme din ce în ce mai complexe a avut ca urmare apariția unor întrebări legate de limitele proceselor algoritmice, cum ar fi:

- Ce probleme pot fi rezolvate prin intermediul proceselor algoritmice?
- Cum trebuie procedat pentru descoperirea algoritmilor?
- Cum pot fi îmbunătățite tehnicile de reprezentare şi comunicare a algoritmilor?
- Cum pot fi aplicate cunoştințele dobândite în vederea obținerii unor maşini algoritmice mai performante?
- Cum pot fi analizate şi comparate caracteristicile diverşilor algoritmi?

1.2.2. DEFINIȚII ŞI CARACTERISTICI

Definiții:

Algoritmul unei prelucrări constă într-o secvență de primitive care descrie prelucrarea.
Algoritmul este un set ordonat de paşi executabili, descrişi fără echivoc, care definesc un proces finit.

Proprietățile fundamentale ale algoritmilor:

- *Caracterul finit:* orice algoritm bine proiectat se termină într-un număr finit de paşi;
- *Caracterul unic şi universal*: orice algoritm trebuie să rezolve toate problemele dintr-o clasă de probleme;
- *Realizabilitatea*: orice algoritm trebuie să poată fi codificat într-un limbaj de programare;
- *Caracterul discret*: fiecare acțiune se execută la un moment dat de timp;
- *Caracterul determinist*: ordinea acțiunilor în execuție este determinată în mod unic de rezultatele obținute la fiecare moment de timp.

Nerespectarea acestor caracteristici generale conduce la obținerea de algoritmi neperformanți, posibil infiniți sau nerealizabili.

1.2.3. REPREZENTAREA ALGORITMILOR

Reprezentarea (descrierea) unui algoritm nu se poate face în absența unui limbaj comun celor care vor să îl înțeleagă. De aceea s-a stabilit o mulțime bine definită de **primitive** (blocuri elementare care stau la baza reprezentării algoritmilor). Fiecare primitivă se caracterizează prin *sintaxă* şi *semantică*. Sintaxa se referă la reprezentarea simbolică a primitivei; semantica se referă la semnificația primitivei. Exemplu de primitivă: aer- din punct de vedere sintactic este un cuvânt format din trei simboluri (litere); din punct de vedere semantic este o substanță gazoasă care înconjoară globul pământesc.

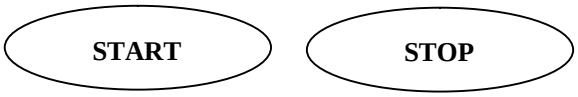
Algoritmii se reprezintă prin:

- scheme logice;
- pseudocod.

1.2.3.1. Reprezentarea algoritmilor prin scheme logice

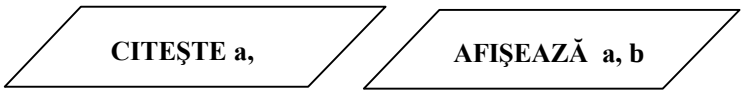
Primitivele utilizate în schemele logice sunt simboluri grafice, cu funcțiuni (reprezentând procese de calcul) bine precizate. Aceste simboluri sunt unite prin arce orientate care indică ordinea de execuție a proceselor de calcul.
Categorii de simboluri:

- Simboluri de *început* şi *sfârşit*



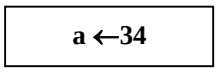
Simbolul START desemnează începutul unui program sau al unui subprogram.
Simbolul STOP desemnează sfârşitul unui program sau al unui subprogram. Prezența lor este obligatorie.

- Simbolul *paralelogram*



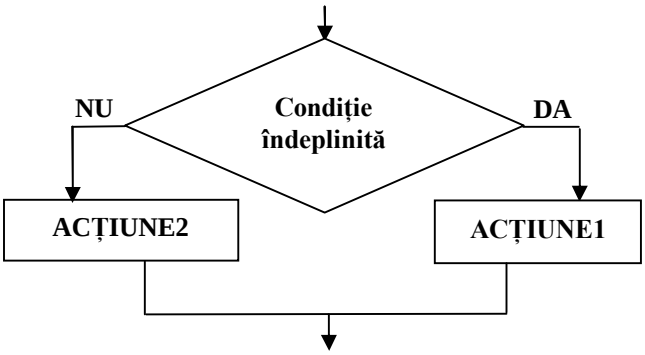
Semnifică procese (operații) de intrare/ieşire (citirea sau scrierea)

- Simbolul *dreptunghi*



Semnifică o atribuire (modificarea valorii unei date).

- Simbolul *romb*

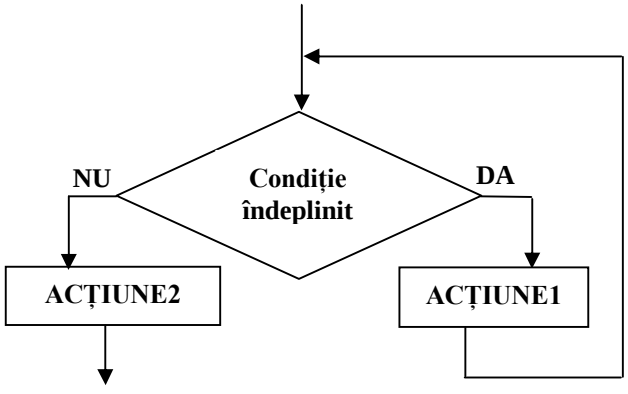


Simbolul romb este utilizat pentru decizii (figura 1.4.). Se testează îndeplinirea condiției din blocul de decizie. ***Dacă*** această condiție este îndeplinită, se execută ACȚIUNE1. Dacă nu, se execută ACȚIUNE2. La un moment dat, se execută ***sau*** ACȚIUNE1, ***sau*** ACȚIUNE2.

Figura 1.4. Structura de decizie

Cu ajutorul acestor simboluri grafice se poate reprezenta orice algoritm.
Repetarea unei secvențe se realizează prin combinarea simbolurilor de decizie şi de atribuire.
Structurile repetitive obținute pot fi: cu test inițial sau cu test final.

Structuri repetitive cu test initial



Se evaluează condiția de test (figura 1.5.). Dacă aceasta este îndeplinită, se execută ACȚIUNE1. Se revine apoi şi se testează iar condiția. Dacă este îndeplinită, se execută (se repetă) ACȚIUNE1, ş.a.m.d. Abia în momentul în care condiția nu mai este îndeplinită, se trece la execuția ACȚIUNE2.
Astfel, **cât timp** condiția este îndeplinită, se repetă ACȚIUNE1. În cazul în care, la prima testare a condiției, aceasta nu este îndeplinită, se execută ACȚIUNE2. Astfel, este posibil ca ACȚIUNE1 ***să nu fie executată*** niciodată.

Figura 1.5. Structură repetitivă cu test inițial

Exisă şi situaţii în care se ştie de la început de câte ori se va repeta o anumită acțiune. În aceste cazuri se foloseşte tot o structură de control repetitivă cu test inițial. Se utilizează un contor (numeric) pentru a ține o evidență a numărului de execuții ale acțiunii. De câte ori se execută acțiunea, contorul este incrementat.

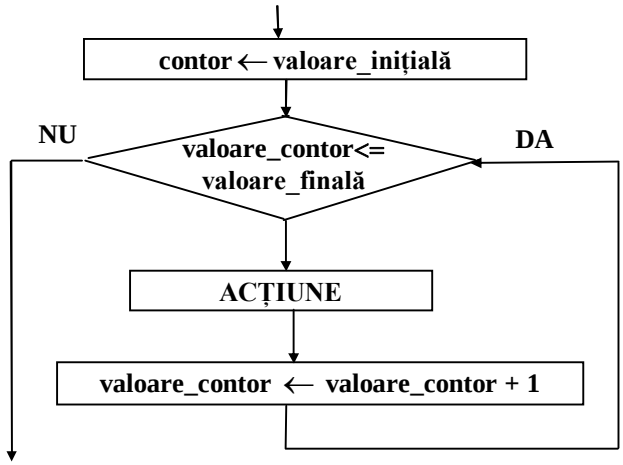


Figura 1.6. Structură repetitivă cu test inițial, cu număr cunoscut de paşi

Se atribuie contorului valoarea inițială (figura 1.6.). **Cât timp condiția** (valoarea contorului este mai mică sau egală cu valoarea finală) este îndeplinită, **se repetă**:

- ACȚIUNE
- incrementare contor (se adună 1 la valoarea anterioară a contorului).

Structură repetitivă cu test final:

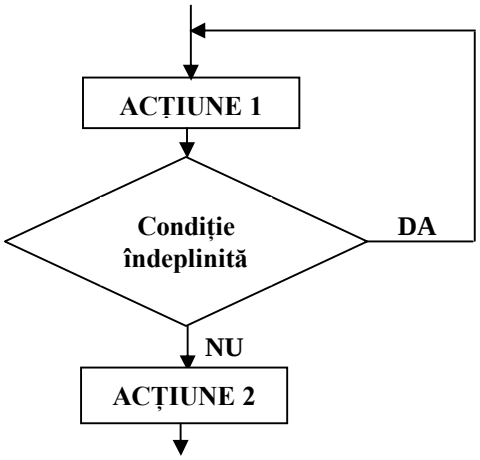


Figura 1.7. Structură repetitivă cu test final

Se execută mai întâi ACȚIUNE1. Se testează apoi condiția (figura 1.7.). Se **repetă** ACȚIUNE1 **cât timp** condiția este îndeplinită.
În acest caz, corpul ciclului (ACȚIUNE1) este executat cel puțin o dată.

1.2.3.2. Reprezentarea algoritmilor prin pseudocod

Pseudocodul este inspirat din limbajele de programare, nefiind însă atât de formalizat ca acestea. Pseudocodul reprezintă o punte de legătură între limbajul natural şi limbajele de programare. Nu există un standard pentru regulile lexicale. Limbajul pseudocod permite comunicarea între oameni, şi nu comunicarea om-maşina (precum limbajele de programare). Pseudocodul utilizează cuvinte cheie (scrise cu majuscule subliniate) cu următoarele semnificații:

```
Sfârşit algoritm:                  SFÂRŞIT
Început algoritm:                      ÎNCEPUT
Citire (introducere) date:         CITEŞTE    lista
Scriere (afişare) date:                SCRIE      lista
Atribuire:                             <-
Structura de decizie (alternativă):    DACĂ condiţie
                                       ATUNCI     acţiune1
                                       ALTFEL     acţiune2
```

Structuri repetitive cu test inițial:

```
CÂT TIMP   condiție
REPETĂ     acțiune
PENTRU contor=val_init LA val_fin [PAS]
REPETĂ acțiune;
```

Structuri repetitive cu test final:

```
REPETĂ acțiune CÂT TIMP condiție
```

sau:

```
REPETĂ acțiune PÂNĂ CÂND condiție
```

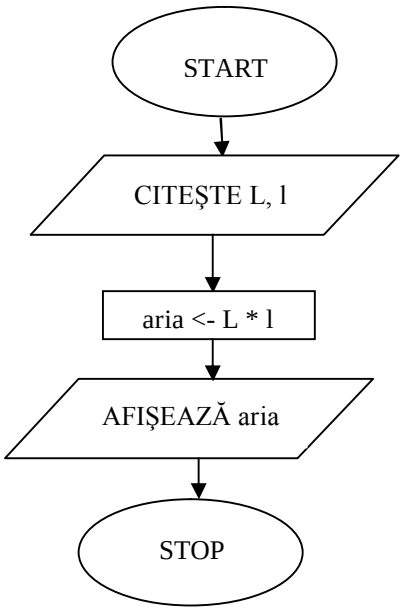
Pe lângă cuvintele cheie, în reprezentarea algoritmilor în pseudocod pot apare şi propoziții nestandard a caror detaliere va fi realizată ulterior.
În cazul în care se realizează un algoritm modularizat, pot apare cuvintele cheie:

```
SUBALGORITM     nume (lista_intrări)
CHEAMĂ          nume (lista_valori_efective_de_intrare)
```

Exemple:

Se vor reprezinta în continuare algoritmii de rezolvare pentru câteva probleme simple (pentru primele 2 probleme se va exemplifica şi modul de implementare a acestor algoritmi în limbajul C++).

1. Se citesc 2 valori numerice reale, care reprezintă dimensiunile (lungimea şi lățimea unui dreptunghi). Să se calculeze şi să se afişeze aria dreptunghiului.




```
ALGORITM aflare_arie_drept
INCEPUT
      CITEŞTE L,l
      aria <- L*l
      AFIŞEAZA aria
SFARŞIT
```

Implementare:

```
#include <iostream.h>
void main( )
{ double L, l;
cout<<"Lungime="; cin>>L;
cout<<"Lațime="; cin>>l;
double aria = L * l;
cout << "Aria="<< aria;
}
```

2. Se citesc 2 valori reale. Să se afiseze valoarea maximului dintre cele 2 numere.

```
ALGORITM max_2_nr
INCEPUT
      CITESTE a, b
      DACA  a >= b
            ATUNCI max<-a
            ALTFEL max<-b
      AFISEAZA max
SFARŞIT
```

Sau:

```
ALGORITM max_2_nr
ÎNCEPUT
      CITEŞTE a, b
      DACA  a >= b
            ATUNCI AFISEAZA a
            ALTFEL AFISEAZA b
SFARŞIT
```

Implementare în limbajul C++:

```
#include <iostream.h>
int main( )
{ float a, b, max;
cout<<"a="; cin>>a;
cout<<"b="; cin>>b;
if (a >= b)
      max = a;
else  max = b;
cout<<"Maximul este:"<<max;
return 0;}
```

```
#include <iostream.h>
void main( )
{ float a, b;
cout<<"a=";cin>>a;
cout<<"b="; cin>>b;
if (a >= b)
    cout<<"Maximul este:"<<a;
else
  cout<<"Maximul este:"<<b; }
```

3. Să se citească câte 2 numere întregi, până la întâlnirea perechii de numere 0, 0. Pentru fiecare pereche de numere citite, să se afişeze maximul.

Algoritm care utilizează structură repetitivă cu test inițial:

```
ALGORITM max_perechi1
INCEPUT
    CITESTE a,b
    CAT TIMP(a#0sau b#0)REPETA
      INCEPUT
      DACA (a>=b)
            ATUNCI AFISEAZA   a
            ALTFEL AFISEAZA   b
      CITESTE a,b
      SFARSIT
SFARSIT
```

```
ALGORITM max_perechi2
INCEPUT
    a ← 3
    CAT TIMP (a#0 sau b#0) REPETA
      INCEPUT
      CITESTE     a, b
      DACA (a>=b)
            ATUNCI AFISEAZA   a
            ALTFEL AFISEAZA   b
      SFARSIT
SFARSIT
```

Algoritm care utilizează structură repetitivă cu test final:

```
ALGORITM max_perechi3
INCEPUT
    REPETA
      INCEPUT
            CITEŞTE a,b
            DACA (a>=b)
                  ATUNCI AFIŞEAZA   a
                  ALTFEL AFIŞEAZA   b
      SFARŞIT
   CAT TIMP (a#0 sau b#0)
SFARŞIT
```

1.3. TEORIA REZOLVĂRII PROBLEMELOR

Creşterea complexității problemelor supuse rezolvării automate (cu ajutorul calculatorului) a determinat ca activitatea de programare să devină, de fapt, un complex de activități.

Pentru ***rezolvarea unei probleme*** trebuie parcurse următoarele ***etape***:

- Analiza problemei (înțelegerea problemei şi specificarea cerințelor acesteia). Se stabileste *ce* trebuie să facă aplicația, şi *nu cum*. Se stabilesc datele de intrare (identificarea mediului inițial) şi se stabilesc obiectivele (identificarea mediului final, a rezultatelor);
- Proiectarea (conceperea unei metode de rezolvare a problemei printr-o metodă algoritmică);
- Implementarea (codificarea algoritmului ales într-un limbaj de programare);
- Testarea aplicației obținute (verificarea corectitudinii programului);
- Exploatarea şi întreținerea (mentenanța, activitatea de modificare a aplicației la cererea beneficiarului sau în urma unor deficiențe constatate pe parcursul utilizării aplicației).

În acest context, activitatea de programare a devenit o activitate organizată, definindu-se metode formale de dezvoltare a fiecărei etape. Etapele descrise anterior alcătuiesc *ciclul de viață al unui produs software* şi constituie obiectul de studiu al disciplinei numite *ingineria sistemulor de programe (software engineering)*.
Teoreticienii ingineriei programării consideră că rezolvarea unei probleme se poate face pe 3 direcții:

- Rezolvarea *orientată pe algoritm* (pe acțiune), în care organizarea datelor este neesențială;
- Rezolvarea *orientată pe date*, acțiunile fiind determinate doar de organizarea datelor;
- Rezolvarea *orientată obiect*, care combină tendințele primelor două abordări.

Abordarea aleasă determină modelarea problemei de rezolvat.

Dintre ***metodele de proiectare orientate pe algoritm*** amintim: ***metoda programării structurate*** şi ***metoda rafinării succesive***. Ambele au ca punct de plecare ***metoda de proiectare top-down***, considerată ca fiind o metodă clasică de formalizare a procesului de dezvoltare a unui produs software.
La baza metodei top-down stă ***descompunerea funcțională*** a problemei P, adica găsirea unui număr de subprobleme P_1, P_2, ... P_n, cu următoarele proprietăți:

- Fiecare subproblemă P_i (1<=i<=n) poate fi rezolvată independent. Dacă nu constituie o problemă elementară, poate fi, la randul ei, descompusă;
- Fiecare subproblemă P_i este mai simplă decât problema P;
- Soluția problemei P se obține prin reuniunea soluțiilor subproblemelor P_i;
- Procesul de descompunere se opreşte în momentul în care toate subproblemele P_i obținute sunt elementare, deci pot fi implementate;

Comunicarea între aceste subprobleme se realizează prin intermediul parametrilor. Implementarea metodei top-down într-un limbaj de programare se face cu ajutorul modulelor de program (funcții sau proceduri în limbajul Pascal, funcții în limbajul C).

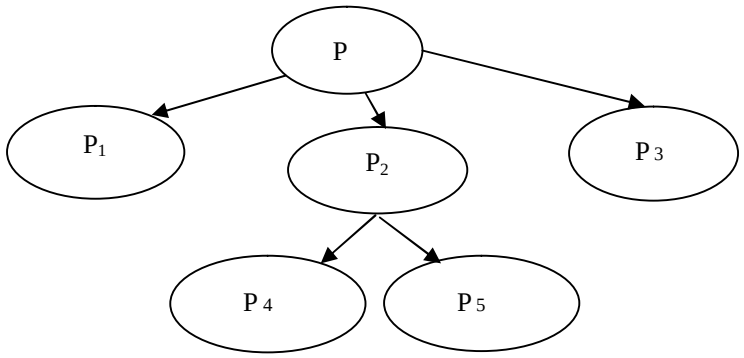


Figura 1.8. Descompunerea funcțională

Descompunerea funcțională a unui program `P` constă în identificarea funcțiilor (task-urilor, sarcinilor) principale ale programului (P_1, P_2, P_3), fiecare dintre aceste funcții reprezentând un subprogram (figura 1.8.). Problemele de pe acelaşi nivel `i` sunt independente unele față de altele.

1.3.1. Etapele rezolvării unei probleme cu ajutorul calculatorului

Să detaliem în continuare ***etapa de implementare***. După analiza problemei şi stabilirea algoritmului, acesta trebuie tradus (**implementat**) într-un limbaj de programare.

- Srierea (editarea) programului sursă.
  Programele sursă sunt fişiere text care conțin instrucțiuni (cu sintactica şi semantica proprii limbajului utilizat). Programul (fişierul) sursă este creat cu ajutorul unui *editor de texte* şi va fi salvat pe disc (programele sursă C primesc, de obicei, extensia ***.c***, iar cele C++, extensia ***.cpp*)**.
  Pentru a putea fi executat, programul sursă trebuie *compilat* şi *linkeditat*.

- Compilarea
  Procesul de compilare este realizat cu ajutorul compilatorului, care translatează codul sursă în cod obiect (cod maşină), pentru ca programul să poată fi înțeles de calculator. În cazul limbajului C, în prima fază a compilării este invocat *preprocesorul*. Acesta recunoaşte şi analizează mai întâi o serie de instrucțiuni speciale, numite *directive procesor*. Verifică apoi codul sursă pentru a constata dacă acesta respectă sintaxa şi semantica limbajului. Dacă există erori, acestea sunt semnalate utilizatorului. Utilizatorul trebuie să corecteze erorile (modificând programul sursă). Abia apoi codul sursă este translatat în cod de asamblare, iar în final, în cod maşină, binar, propriu calculatorului. Acest cod binar este numit cod obiect şi de obicei este memorat într-un alt fişier, numit *fişier obiect*. Fişierul obiect va avea, de obicei, acelaşi nume cu fişierul sursă şi extensia ***.obj***.

- Linkeditarea
  Dupa ce programul sursă a fost translatat în program obiect, el este va fi supus operaţiei de linkeditare. Scopul fazei de linkeditare este acela de a obţine o formă finală a programului, în vederea execuţiei acestuia. Linkeditorul "leagă" modulele obiect, rezolvă referinţele către funcţiile externe şi rutinele din biblioteci şi produce cod executabil, memorat într-un alt fisier, numit *fişier executabil* (acelaşi nume, extensia *.exe*)

- Execuţia
  Lansarea în execuţie constă în încărcarea programului executabil în memorie şi startarea execuţiei sale.

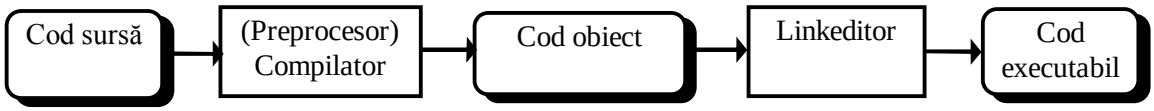


Figura 1.9. Etapele necesare obţinerii fişierului executabil

Observaţii:
1. Mediile de programare integrate (BORLANDC, TURBOC) înglobează editorul, compilatorul, linkeditorul şi depanatorul (utilizat în situaţiile în care apar erori la execuţie);
2. Dacă nu se utilizează un mediu integrat, programatorul va apela în mod explicit (în linie de comandă) un editor de texte, compilatorul, linkeditorul. Lansarea în execuţie se va face tot din linie de comandă.
3. Extensiile specificate pentru fişierele sursă, obiect şi executabile sunt

ÎNTREBĂRI ŞI EXERCIŢII

Chestiuni teoretice

1. Enumeraţi unităţile funcţionale componente ale unui sistem de calcul.
2. Care sunt diferenţele între soft-ul de aplicaţie şi sistemul de operare?
3. Care este deosebirea între algoritm şi program?
4. Care sunt proprietăţile fundamentale ale algoritmilor?
5. Care sunt modalităţile de reprezentare a algoritmilor?

Chestiuni practice

1. Reprezentaţi algoritmul lui Euclid (pentru calculul celui mai mare divizor comun a 2 numere întregi) prin schema logică.
2. Proiectaţi un algoritm care să rezolve o ecuaţie de gradul I (de forma $ax + b = 0$), unde a,b sunt numere reale. Discuţie după coeficienţi.
3. Proiectaţi un algoritm care să rezolve o ecuaţie de gradul II (de forma $ax^2 + bx + c = 0$), unde a,b,c sunt numere reale. Discuţie după coeficienţi.
4. Proiectaţi un algoritm care să testeze dacă un număr întreg dat este număr prim.
5. Proiectaţi un algoritm care să afişeze toţi divizorii unui număr întreg introdus de la tastatură.
6. Proiectaţi un algoritm care să afişeze toţi divizorii primi ai unui număr întreg introdus de la tastatură.
7. Proiectaţi un algoritm care calculează factorialul unui număr natural dat. (Prin definiţie 0!=1)