1. Programarea şi rezolvarea problemelor

Comportamentul uman şi gândirea sunt caracterizate de secvențe logice. Un individ învață cum să execute anumite comenzi. De asemenea, învață ce tipuri de comportamente poate aştepta de la alți indivizi.

La o scară mai largă, matematica lucrează cu secvențe logice de paşi pentru rezolvarea problemelor şi demonstrarea teoremelor. La fel, producția de masă nu poate exista fără succesiunile de operații executate într-o anumită ordine.

Atunci când ordonăm un proces, îl *programăm*. Acest curs se referă la programarea unui aparat: *calculatorul*.

Calculatorul este un dispoztiv programabil care poate păstra, regăsi şi procesa date.

Aşa cum un program descrie acțiunile care trebuie executate pentru a atinge un scop, un *program de calculator* descrie paşii pe care trebuie să îi execute calculatorul pentru a rezolva o problemă. În contextul acestui curs, atunci când vom vorbi de *programare* ne vom referi la programarea calculatorului.

Un program de calculator este o listă de instrucțiuni care trebuie urmate de calculator.

Un calculator ne permite să realizăm o serie de acțiuni într-un mod mult mai rapid şi mai precis decât o putem face fără ajutorul său. Pentru a îl folosi, însă, trebuie să specificăm ce dorim să facem, dar şi în ce ordine. Aceste lucruri le facem prin *programare*.

1.1 Cum scriem un program?

Pentru a scrie un program trebuie să parcurgem două faze:

- rezolvarea problemei
- implementarea.

Faza de rezolvare a problemei

1. *Analiza* înseamnă înțelegerea, definirea problemei şi a soluției ce trebuie dată;
2. *Algoritmul* presupune dezvoltarea unei secvențe logice de paşi care trebuie urmați pentru rezolvarea problemei;
3. *Verificarea* este parcurgerea paşilor algoritmului pe mai multe exemple pentru a fi siguri că rezolvă problema pentru toate cazurile.

Faza de implementare

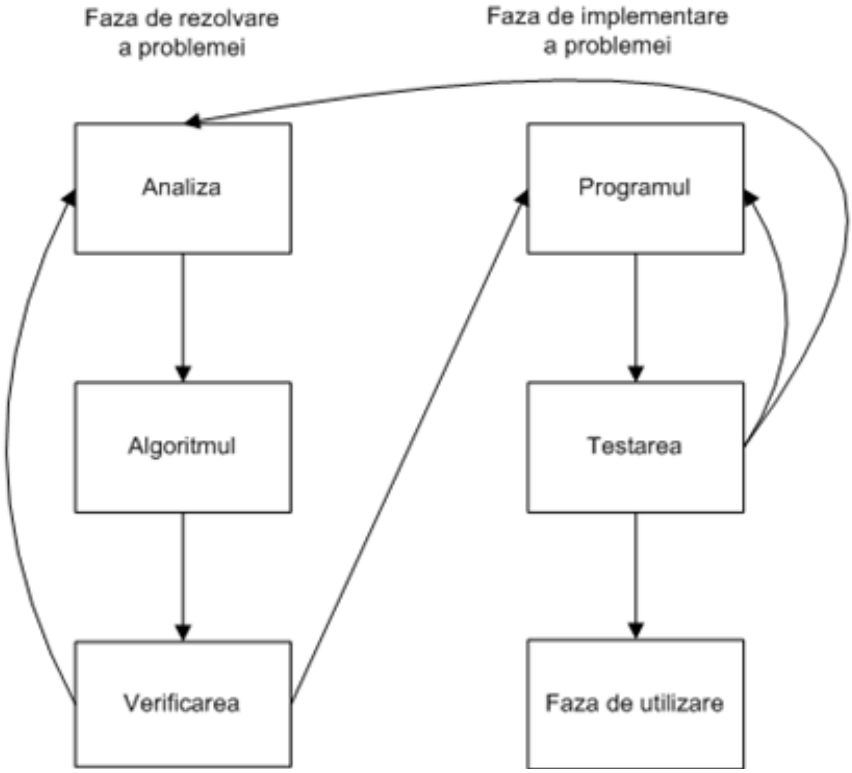
1. *Programul* reprezintă translatarea algoritmului într-un limbaj de programare
2. *Testarea* este etapa în care ne asigurăm că instrucțiunile din program sunt urmate corect de calculator. În situația în care constatăm că sunt erori, trebuie să revedem algoritmul şi programul pentru a determina sursa erorilor şi pentru a le corecta.

Aceste două faze de dezvoltare a programului sunt urmate de *faza de utilizare* a programului care înseamnă folosirea acestuia pentru rezolvarea problemelor reale, cele pentru care a fost conceput. Ulterior pot interveni *modificări* ale programului fie pentru a răspunde noilor cerințe ale utilizatorilor, fie pentru corectarea erorilor care apar în timpul utilizării şi care nu au putut fi găsite în faza de testare.

Calculatorul nu este inteligent. El nu poate analiza problema şi nu poate să dea o soluție. Programatorul trebuie să analizeze problema, să dea soluția şi apoi să o

comunice calculatorului. Avantajul folosirii calculatorului este că el rezolvă problemele rapid şi fără erori, eliberându-ne totodată de realizarea unor operații repetitive şi plictisitoare.

Programatorul începe operația de programare prin analiza problemei şi dezvoltarea unei soluții generale numită *algoritm*. *Algoritmul* este o procedură care descrie paşii ce trebuie parcurşi pentru rezolvarea unei probleme într-un timp finit. Algoritmul este esențial pentru procesul de programare. Programul este de fapt un algoritm scris pentru calculator.



Un algoritm este o secvență logică de acțiuni. Folosim algoritmi în fiecare zi: rețetele, instrucțiunile de folosire sunt exemple de algoritmi care nu sunt, însă, programe. Un exemplu de algoritm poate fi o succesiune de paşi care trebuie urmați pentru a porni o maşină. Un alt exemplu este calculul sumei care trebuie plătită unui salariat într-o săptămână.

1. Verificarea sumei plătite pe oră
2. Determinarea numărului de ore lucrate în timpul săptămânii
3. Dacă numărul de ore este mai mic sau egal cu 40, se înmulțeşte numărul de ore cu suma plătită pe oră
4. Dacă numărul de ore depăşeşte 40, atunci se scade 40 din numărul de ore lucrate, iar diferența de ore se înmulțeşte cu 1,5 ori suma plătită pe oră
5. Adună sumele de la punctele 3 şi 4 şi stabileşte suma finală.

După dezvoltarea soluției generale, programatorul poate testa algoritmul mental sau în scris. Dacă algoritmul nu este corect, reluăm paşii descrişi mai devreme.

Când programatorul este satisfăcut de algoritm, poate să îl translateze într-un program scris într-un *limbaj de programare*.

Limbajul de programare este un set de reguli, simboluri şi cuvinte speciale folosite pentru a construi un program.

Limbajul C++ este o variantă simplificată a limbii engleze şi care are un set strict de reguli gramaticale. Datorită numărului mic de cuvinte disponibile, sunteți obligați să scrieți instrucțiuni simple şi exacte.

Codarea este translatarea algoritmului într-un limbaj de programare.

Execuția este rularea programului pe calculator (*running*).

Depanarea este faza de determinare şi corectare a erorilor (*debugging*).

Implementarea este combinația dintre codarea şi testarea algoritmului.

O parte importantă a programării este scrierea documentației. *Documentația* este reprezentată de un text scris şi de comentariile necesare înțelegerii de către alte persoane a programului scris de noi.

După scrierea programului, trebuie să transmitem calculatorului informațiile sau datele necesare rezolvării problemei.

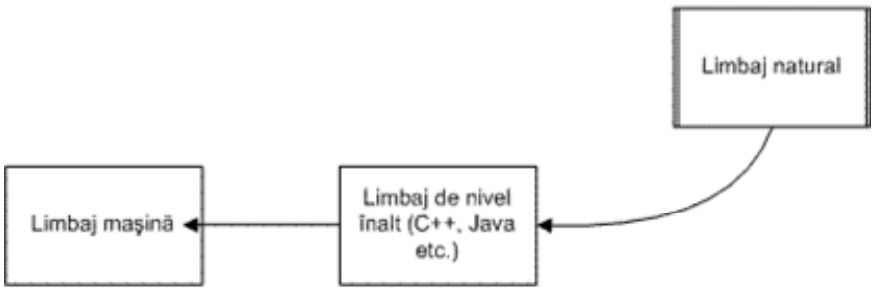
Informația este orice cunoştință care poate fi comunicată, inclusiv idei abstracte sau concepte.

Datele sunt informații transpuse într-o formă care poate fi înțeleasă de calculator.

1.2 Ce este un limbaj de programare?

Programatorii scriu instrucțiuni în diverse limbaje de programare, unele care sunt înțelese în mod direct de calculator, altele care necesită mai mulți paşi de *translatare*. În prezent există sute de limbaje de programare care pot fi împărțite în trei tipuri generale:

1. Limbaje maşină
2. Limbaje de asamblare
3. Limbaje de nivel înalt



Singurul limbaj de programare pe care calculatorul îl poate executa în mod direct este un set primitiv de instrucțiuni numit *limbaj maşină* sau *cod maşină*. Acesta este "limbajul natural" al unui calculator şi este definit de alcătuirea hardware a fiecărui calculator în parte. Un anumit limbaj maşină poate fi folosit doar pentru un anumit tip de calculator. Limbajul maşină este alcătuit din instrucțiuni codate binar şi poate fi folosit direct de calculator. Limbajele maşină sunt greu de folosit de programatori, aşa cum se poate vedea din următoarea secțiune de program scris în limbaj maşină care adună o sumă suplimentară de bani la suma de bază pe care o primeşte un angajat, rezultând suma finală.

Exemplu

```
+1300042774
+1400593419
+1200274027
```

Pe măsură ce calculatoarele au devenit tot mai populare, a devenit evident că limbajul maşină este greu de folosit, dezvoltarea aplicațiilor este foarte lentă şi probabilitatea de apariție a erorilor este foarte mare. În loc să se folosească numere pentru a programa calculatoarele, s-a trecut la folosirea unor abrevieri ale unor cuvinte din limba engleză care reprezintă operații elementare pentru calculator. Aceste abreviații formează baza *limbajelor de asamblare*. În acelaşi timp au fost dezvoltate *programe de translatare* sau *asambloare* pentru a converti programele scrise in limbaj de asamblare către programe în limbaj maşină. Secvența de instrucțiuni de mai jos realizează aceleaşi operații ca cele din exemplul anterior, dar într-o manieră mai clară decât echivalentul în limbaj maşină.

Exemplu

```
LOAD BASEPAY
ADD OVERPAY
STORE GROSSPAY
```

În prezent se folosesc *limbaje de nivel înalt*, mult mai uşor de folosit decât codul maşină şi care accelerează procesul de dezvoltare software. Un program numit *compilator* translatează un program scris într-un limbaj de nivel înalt în limbaj maşină. Iată o variantă scrisă într-un limbaj de nivel înalt a programului de mai sus.

Exemplu

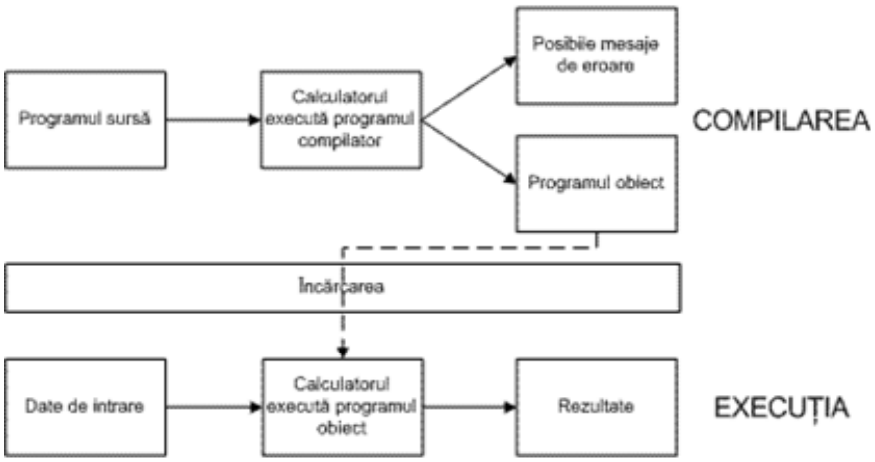
```
grossPay = basePay + overTimePay
```

Pentru a putea rula un program de nivel înalt pe un calculator, pe acesta trebuie să existe un compilator adaptat limbajului dar şi calculatorului.

Programul sursă este un program scris într-un limbaj de nivel înalt.

Programul obiect este versiunea în limbaj maşină a programului sursă şi se obține în urma compilării.

Compilarea şi *execuția* sunt două procese distincte.



Instrucțiunile dintr-un limbaj de programare reflectă operațiile pe care le poate realiza un calculator:

- transferarea datelor dintr-un loc în altul
- citirea datelor de la un dispozitiv de intrare (ex. tastatura) şi transmiterea lor către un dispozitiv de ieşire (ex. ecran)
- stocarea şi aducerea datelor din memorie sau alte dispozitive de memorare
- compararea a două date pentru stabilirea egalității sau a inegalității
- operații aritmetice

Scurt istoric al limbajelor C şi C++

Limbajul C++ a evoluat din limbajul C care, la rândul său, a avut la bază alte două limbaje de programare, BCPL şi B. Limbajul BCPL a fost dezvoltat în 1967 de Martin Richards ca limbaj pentru scrierea sistemelor de operare şi a compilatoarelor. Ken Thompson a modelat multe elemente ale BCPL în limbajul B pe care l-a folosit pentru scrierea uneia dintre primele versiuni ale limbajului UNIX la Bell Laboratories în 1970. Aceste două limbaje de programare nu foloseau tipuri de dată, fiecare dată avea aceeaşi dimensiune în memorie, iar tratarea unei date ca întreg sau real era în responsabilitatea programatorului.

Limbajul C a fost dezvoltat din limbajul B la Bell Laboratories în 1972 de Dennis Ritchie. Inițial a fost folosit pentru dezvoltarea sistemului de operare UNIX, iar astăzi majoritatea sistemelor de operare sunt scrise în C şi C++.

Limbajul C++ este o extensie a lui C şi a fost creat la începutul anilor 1980 de Bjarne Stroustroup tot la Bell Laboratories. Are toate elementele limbajului C, dar oferă posibilitatea *programării orientate pe obiecte*. *Obiectele* sunt, în principiu, componente software *reutilizabile* care modelează elemente din lumea reală. S-a dovedit că folosirea unei abordări modulare, a design-ului şi a implementării orientate pe obiecte poate face ca grupurile de dezvoltare să fie mult mai productive decât atunci când se folosesc alte tehnici de programare, cum ar fi programarea procedurală.

Biblioteca standard C++

Programele C++ constau din elemente numite *clase* şi *funcții*. Puteți programa fiecare piesă de care aveți nevoie pentru a alcătui un program. Dar cei mai mulți programatori folosesc avantajul oferit de bogata colecție de clase şi funcții oferite de biblioteca standard C++. De aceea, învățarea limbajului C++ înseamnă, pe de o parte învățarea limbajului în sine şi, pe de altă parte, deprinderea modului în care se pot folosi clasele şi funcțiile din biblioteca standard C++. Aceste clase şi funcții sunt, de regulă, oferite odată cu compilatorul, dar există multe biblioteci suplimentare care sunt realizate de companii software independente. Unele dintre acestea pot fi descărcate în mod liber de pe Internet.

Avantajul creării propriilor noastre funcții şi clase este că vom şti exact cum lucrează, dar dezavantajul este timpul consumat şi efortul depus pentru proiectarea, dezvoltarea şi întreținerea noilor clase şi funcții pentru a opera corect şi eficient.

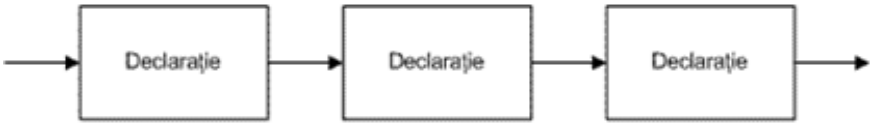
Programarea structurată

După anii 1960, când aplicațiile au început să devină din ce în ce mai complexe şi când costurile de dezvoltare au început să devină foarte mari, lumea a realizat că acest proces este mai complex decât s-a estimat inițial. Activitățile de cercetare în domeniu au rezultat în evoluția către *programarea structurată*, o abordare disciplinată în scrierea programelor care au devenit mai clare, mai uşor de testat, de corectat şi de modificat.

Structuri de program

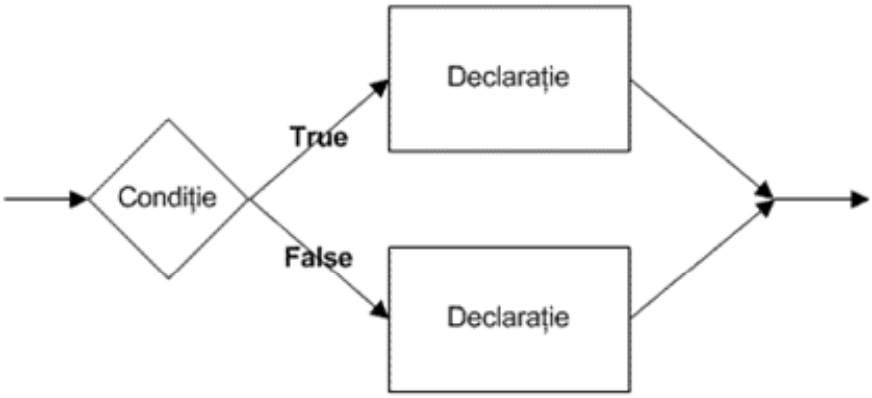
Limbajele de programare folosesc anumite structuri pentru a transpune algoritmii în programe.

Secvența



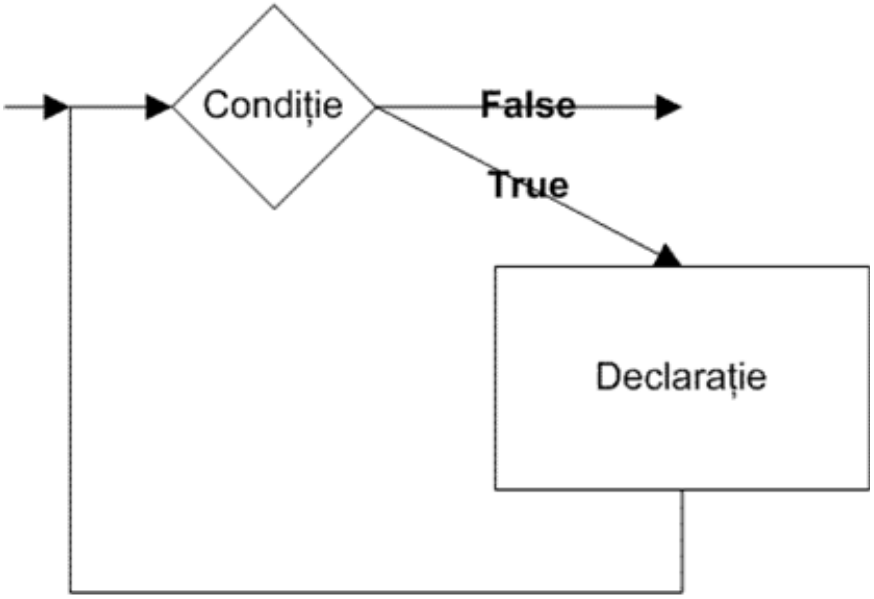
Secvența este o serie de declarații executate una după alta.

Selecția (deciza)



`IF` condiție `THEN` declarație1 `ELSE` declarație2

Bucla (repetiția sau iterația)



`WHILE` condiție `DO` declarație

Subprogramul (procedura, funcția, subrutina)

Subprogramul poate fi o combinație a structurilor anterioare. Ne permite scrierea separată a unor părți din program şi apoi asamblarea lor într-o formă finală.

Programarea orientată pe obiecte

Evoluțiile pozitive în dezvoltarea software au început să apară odată cu folosirea programării structurate. În anii 1980, tehnologia programării orientate pe obiecte a început să fie folosită pe scară largă în proiectarea şi dezvoltarea software.

Tehnologia obiectelor este, în principiu, o schemă de „împachetare" care permite crearea unităților software cu o semnificație proprie. Acestea sunt focalizate pe părți specifice ale unei aplicații. Se pot crea obiecte pentru date, pentru plăți, pentru facturi, obiecte video, obiecte fişier etc. De fapt, orice substantiv poate fi transpus într-un obiect.

Trăim într-o lume de obiecte. Există în jurul nostru maşini, avioane, oameni, animale, clădiri etc. Înaintea apariției limbajelor orientate pe obiecte, limbajele de programare (de ex. FORTRAN, Pascal, Basic, C) erau focalizate pe acțiuni (verbe) şi nu pe obiecte (substantive). O lume a obiectelor trebuie transpunsă, puțin forțat, într-o lume a acțiunilor. Paradigma orientării pe obiecte prin limbaje cum ar fi Java sau C++ a făcut posibil ca programarea să devină o prelungire a realității. Acesta este un proces mai natural decât programarea procedurală şi conduce la creşteri semnificative de productivitate.

Una dintre problemele majore ale programării procedurale este ca unitățile de program nu modelează foarte firesc entități ale lumii reale şi, de aceea, nu pot fi reutilizate în mod facil. Fiecare nou proiect în programarea procedurală presupunea scrierea codului „de la zero", lucru care înseamnă o risipă de timp, bani şi resurse umane. Tehnologia obiectelor face ca entitățile create într-un proiect (obiectele), dacă sunt corect concepute, să poată fi folosite şi în proiecte viitoare.

Pe de altă parte, este remarcabil faptul că uneori nu reutilizarea codului este marele avantaj al programării orientate pe obiecte, ci faptul că programele sunt mult mai uşor de înțeles, organizat, de întreținut, de modificat sau de corectat.

Elementele de programare structurată sunt noțiuni cheie în programarea orientată pe obiecte. Fiecare clasă este o unitate care încorporează structuri de program.

1.3 Ce este un calculator?

Un calculator (*computer*) este un dispozitiv capabil să realizeze calcule şi să ia decizii logice cu viteze de milioane sau miliarde de ori mai mari decât oamenii. Aceasta înseamnă că unei persoane îi trebuie câteva milioane de secunde pentru a face calculele pe care le poate face un calculator într-o secundă.

Calculatorul procesează *date* sub controlul unor înşiruiri de instrucțiuni numite *programe de calculator*. Aceste programe dirijează calculatorul să realizeze secvențe de acțiuni care au fost specificate de persoane numite *programatori*.

Un calculator este alcătuit din diverse dispozitive, cum ar fi tastatura, mouse-ul, discurile, memoria, CD-ROM-ul sau microprocesorul, toate acestea fiind numite generic *hardware*. Programele de calculator care rulează pe calculator sunt numite *software*. Costurile hardware-ului au scazut foarte mult în ultimii ani până la punctul în care un calculator personal a devenit foarte accesibil ca preț. Din păcate, costurile pe care le implică dezvoltarea software au crescut în tot acest timp pe măsură ce aplicațiile au devenit din ce în ce mai complexe. În acest curs şi în cel din semestrul următor vom studia metode de dezvoltare software cunoscute prin a căror utilizare se

pot reduce costurile: programarea structurată, dezvoltarea top-down, funcționalizarea, programarea orientată pe obiecte, programarea generică.

Organizarea unui calculator

Se poate programa şi fără a şti prea multe despre alcătuirea internă a calculatorului. Cunoaşterea părților componente ajută, însă, la înțelegerea efectului fiecărei instrucțiuni din program.

Calculatorul are 4 componente de bază:

1. *Unitatea de memorare* este o colecție de celule care stochează datele. Fiecare astfel de celulă are o adresă. Aceste celulue se numesc *celule de memorie* sau *locații de memorie*.
2. *Unitatea centrală de procesare* (CPU) este cea care urmăreşte instrucțiunile din program. Are două componente:
    a. *Unitatea aritmetico-logică* (ALU) care realizează operațiile aritmetice şi logice
    b. *Unitatea de control* care controlează acțiunile celorlalte componente astfel încât instrucțiunile să se execute în ordinea corectă
3. *Dispozitivele de intrare/ieşire* (I/O) acceptă date care vor fi procesate şi le prezintă pe cele care au fost procesate
4. *Dispozitivele auxiliare de stocare* păstrează datele şi după oprirea calculatorului

Dispozitivele periferice sunt cele de intrare/ieşire şi cele auxiliare.

Toate aceste componente sunt cunoscute sub numele de *hardware*. Programele care permit hardware-ului să funcționeze se numesc *software*.

Pe lângă programele scrise de noi, existăun set de programe numite *software de sistem* care simplifică *interfața* dintre calculator şi utilizator. În această categorie intră compilatoarele, sistemele de operare sau editoarele de text.

Sistemul de operare coordonează toate resursele calculatorului. El poate rula compilatorul, poate rula programe obiect, poate executa comenzi de sistem

Editorul este un program interactiv folosit pentru crearea şi modificarea programelor sursă sau a datelor.

1.4 Tehnici de rezolvare a problemelor

Adesea în viața de zi cu zi suntem puşi în situația de a *urma* algoritmi. În faza de rezolvare a unei probleme de programare va trebui să *proiectăm* algoritmi. Este important să ne punem cât mai multe întrebări până când înțelegem exact ce avem de făcut.

Folosirea soluțiilor existente – Întotdeauna trebuie să evităm să reinventăm roata. Dacă există o soluție, atunci să o folosim. Dacă am rezolvat o problemă similară înainte, trebuie doar să repetăm soluția pentru că în programare există probleme care apar foarte des (ex. calculul unui minim sau al unui maxim). Dacă avem la dispoziție o secvență de cod care rezolvă o parte a problemei noastre, putem să o folosim. Această metodă se numeşte *software reuse* şi este elementul central în programarea orientată pe obiecte.

Divide et impera – Adeseori este mult mai uşor să rezolvăm o problemă dacă o împărțim în subprobleme mai mici. De altfel, metoda descompunerii unui program în funcții sau tehnica programării orientate pe obiecte se bazează pe acest principiu.

Dificultatea de a începe – Programatorii se confruntă adesea cu o mare dificultate: se găsesc în fața unei foi albe şi nu ştiu cum să înceapă. Privesc

problema şi li se pare foarte complicată. Pentru a depăşi acest moment, rescrieți problema cu propriile voastre cuvinte. Încercați să o descompuneți în subprobleme individuale în loc să o analizați global. Acest lucru vă va ajuta să extrageți componente mai uşor de rezolvat. De asemenea, acest lucru vă va ajuta să sintetizați mai uşor algoritmul de rezolvare a problemei.