Lecția 2

Etapele rezolvării unei probleme cu ajutorul calculatorului

1. ***Analiza problemei*: presupune:**

· *citirea şi analiza* cu atenţie a enunţului problemei date pentru identificare datelor de intrare (datele iniţiale ale problemei). (Ce se cunoaste ?)

· *Înţelegerea şi identificarea rezultatului* aşteptat (stabilirea datelor de iesire). (Ce se cere?)

· *Alegerea strategiei de rezolvare* a problemei (identifiarea algoritmului optim de rezolvare a problemei) (Cum rezolv?) Pentru rezolvarea unei probleme date pot exista mai multe metode. În acest moment trebuie identificat care este cel mai optim mod de rezolvare a problemi astfel încât în cel mai scurt timp şi cu cele mai puţine resurse să obţinem rezultatul dorit. Analiza timpului de lucru a unui program se numeste complexitate[1], dar pentru acest subiect vom discuta mai pe larg intr-un capitol separat.

1. ***Descrierea în limbaj natural* a modului de rezolvare propus, stabilirea tipurilor de informaţii care trebuie prelucrate precum şi a tipurilor de rezultate aşteptate. Limbajul natural descrie în termeni largi etapele propuse pentru rezolvarea problemei date.**
2. ***Descrierea algoritmului în limbaj pseudocod:* Limbajul pseudocod este un limbaj intermediar între limbajul natural şi limbajul de programare. În aceasta etapă se stabilesc paşii de algoritm, tipurile datelor de intrare, de manevră şi de ieşire, se identifică condiţiile care impun şi se scriu în expresii matematice care sunt uşor de interpretat de calculator (calculatorul nu face presupuneri si nici nu ghiceşte ce doritţi dumneavoastră ci doar execută rapid comenzile pe care le-a primit). Un algoritm scris în limbaj pseudocod nu este un program care poate fi rulat direct pe**

calculator, dar poate si transcris uşor într-un limbaj de programare ţinând cont de vocabularul, semantica si sintaxa limbajului de programare. În această carte vom folosi un limbaj pseudocod scris în limba română (pentru a fi mai uşor de inteles logica algoritmicii). Limbajul pseudocod se va baza implementarea ideii de rezolvare într-un format apropiat de limbajul C++, dar care nu pune accent pe sintaxa riguroasă care trebuie respectată atunci când scrii într-un limbaj de programare.

3. ***Scrierea codului sursă:*** **Aceasta este etapa în care algoritmul scris în limbaj pseudocod este transcris în limbaj de programare, ţinând cont de regulile impuse de acesta şi repectând semnificaţia construcţiilor sintactice corecte impuse de limbajul de programare.**
4. ***Corectarea erorilor sintactice:*** **Corectarea unui cod sursă din punct de vedere sintactic se numeste *compilare[2]*. În aceasta etapă programul este verificat, iar eventualele greşeli sau omisiuni ale respectării regulilor sintactice sau semantice sunt semnalate de compilator print-o listă de erori. Cel care a scris programul are obligaţia să corecteze aceste erori folosindu-se de masajele de eroare afişate precum şi de cunoştinţele sale. Mesajul de eroare afişat, care semnalează o eroare de compilare, nu indică întotdeauna corect modalitatea de corectare (el este mai mult informativ). Erorile de eroare vor fi căutate începând de la poziţia semnalat de compilator sau de la poziţia semnalată spre începutul programului. O serie de greşeli frecvent întâlnite la începători sunt generate de definirea eronată a tipurilor de date care vor contine rezultatele. Dacă rezultatul aşteptat depăşeşte ca valoare domeniul de definiţie stabilit atunci aceasta greşeala nu va fi identificată de către compilator, deci nu va fi corectată in acestă etapă. La execuţia programului vom avea însă surpriza unor rezultate neaşteptate, chiar daca soluţia de rezolvare găsită este corectă si programul scris nu are greşeli de compilare.**
5. ***Testarea programului:*** **În această etapă se verifică corectitudinea rezolvării problemei prin analiza rezultatului obţinut pentru diferite**

valori ale datelor de intrare. Tot aici se identifică si eventualele *erori de gândire*, care apar prin alegerea unui algoritm de rezolvare greşit. În această etapă trebuie indentificate seturi de date de intrare numite şi *cazuri limită* pentru care programul s-ar putea bloca sau ar genera rezultate greşite datorită unei neglijenţe în etapa de analiză şi implementare a programului.

[1] Complexitatea unui algoritm este determinată ţinând cont de spaţiul de memorie utilizat (complexitate spaţiu) si de timpul de obţinere a rezultatului (complexitate timp). Prin *complexitate spaţiu* înţelegem dimensiunea spaţiului de memorie utilizat de program. Un program necesită un spaţiu de memorie constant, independent de datele de intrare, pentru memorarea codului său, a constantelor, a variabilelor simple şi a structurilor de date de dimensiune constantă alocate static şi un spaţiu de memorie variabil, a cărui dimensiune depinde adesea de datele de intrare de cerinţele problemei de rezolvat şi din spaţiul de memorie necesar apelurilor de proceduri şi funcţii. Prin *complexitate timp* înţelegem timpul necesar execuţiei programului. Înainte de a evalua timpul necesar execuţiei programului ar trebui să avem informaţii detaliate despre sistemul de calcul folosit.

[2] Un compilator este un program (sau set de programe) care traduce textul scris într-un limbaj de programare (limbajul sursă – low level) într-un alt limbaj de calculator (numit limbaj ţintă- hight level). Sursa originală se numeşte de obicei cod sursă iar rezultatul cod obiect. În general codul sursă este compilat pentru a crea un program executabil.