A3. Generalităţi privind limbajele de programare

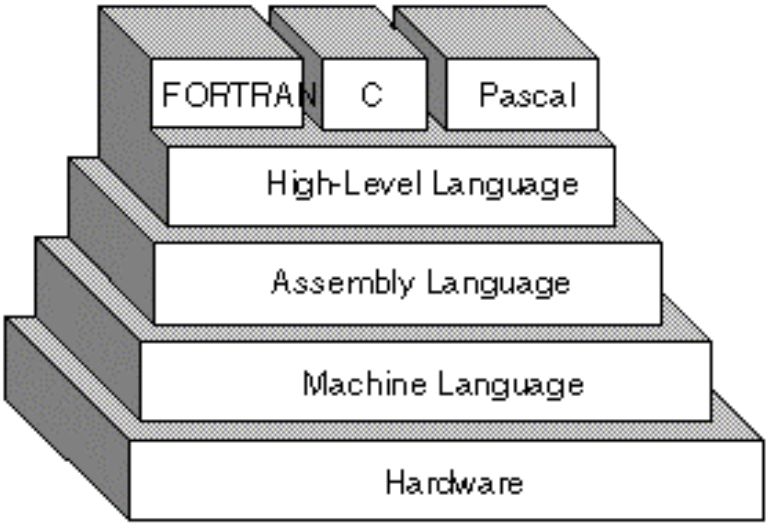
Introducere

Noţiunea de limbaj: este definită ca un sistem pentru comunicare. Limbajele scrise folosesc simboluri (care sunt caractere) pentru a construi cuvinte. Întreg setul de cuvinte formează *vocabularul limbajului*. Modul în care cuvintele pot fi combinate pentru a fi înţelese este definit de *sintaxa şi gramatica limbajului*. Sensul dat de cuvinte sau combinaţii de cuvinte este definit de *semantica limbajului*.
În domeniul computerelor, limbajele umane sunt denumite *limbaje naturale*. Din nefericire computerele nu sunt de ajuns de sofisticate pentru a înţelege limbajele naturale. Prin urmare comunicarea cu computerele se face prin intermediul unor limbaje specifice computerelor denumite *limbaje de programre*.

Noţiunea de limbaj de programare: este definită ca fiind ansamblul format de un vocabular şi un set de reguli gramaticale, necesar instruirii unui computer pentru a realiza anumite activităţi.
După modul cum au evoluat în timp limbajele de programare pot fi:

- limbaje de prima generaţie: limbajul maşină (machine language);
- limbaje de generaţia a doua: limbajul de asamblare (assembly language);
- limbaje de generaţia a treia: limbajele de înalt nivel (high-level programming languages);
- limbaje de generaţia a patra: limbaje mai apropiate de limbajul uman decât limbajele de înalt nivel (ex. de comandă: FIND ALL RECORDS WHERE NAME IS "SMITH")

În figura de mai jos sunt prezentate primele trei generaţii de limbaje de programare şi modul cum interacţionează acestea cu computerul.



Figura

Limbajul maşină

Când un computer urmează instrucţiunile unui program se spune că programul este în execuţie (running). Înainte de a fi executat programul trebuie să fir rezident în memorie. Adică programul trebuie să ocupe un set de bytes consecutivi în memorie. Totodată programul trebuie scris într-un limbaj maşină intern. Fiecare tip de procesor are propriul limbaj maşină. Acesta este conceptul de bază cu privire la modul de execuţie a unui program. Faptul că programul ce se execută este stocat (chiar şi parţial) în memoria principală (RAM) duce la concluzia că numai prin schimbarea programului din memoria RAM computerul poate trece la execuţia altui proces (task)/program.
Aşa cum s-a prezentat mai sus toate computerele au un limbaj maşină intern (specific tipului de procesor). Acest limbaj este codat într-o reprezentare binară şi este foarte greoi (plictisitor) de utilizat pentru scrierea unui program. Majoritatea instrucţiunilor programului vor conţine astfel două părţi:

- o parte care se referă la operaţia de codare – se vor indica ordinea operaţiilor;
- o parte care se referă la adresa din meorie - indică locaţia de memorie ce se va utiliza ca operand al instrucţiunii.

operation code	**address**	**meaning**
00010101	10100001	load c(129) into accumulator
00010111	10100010	add c(130) to accumulator
00010110	10100011	store c(accumulator) in location 131

Astfel programatorii care utilizează limbajul maşină vor trebuie să fie atenţi în ce zone de memorie se vor stoca date şi în ce zone de memorie se vor executa programele (instrucţiunile). Astfel pot apărea erori de programare datorate suprapunerii scrierii instrucţiunilor peste date. Prin urmare programarea în limbaj maşină presupune o bună capacitate de a interpreta datele şi instrucţiunile la nivel de bit. Totodată aceasta reprezintă si posibilitatea de a se genera alte programe şi de a le executa.
Concluzii:

- este limbajul pe care computerul îl înţelege în mod direct;
- în limbajul maşină programele se scriu în cod binar: succesiuni de 0 şi 1;
- fiecare instrucţiune din limbajul maşină este o combinaţie de 4 bits (LOAD-0000;ADD-0001);
- programatorul trebuie să cunoască detaliat structura hardware a computerului;
- programatorul trebuie să gestioneze fără greşeală alocarea adreselor de memorie pentru un program;
- pot apărea multe erori datorită: concepţiei programului, sintaxei, suprapunerii adreselor de memorie, etc.

Limbajul de asamblare

Atenţia necesară evitării ocupării aceloraşi adrese de memorie este foarte solicitantă (greoaie) în programarea în limbaj maşină. Astfel dacă programatorul modifică un program şi decide să mai introducă ceva atunci toate celelalte adrese de memorie utilizate până atunci se vor

schimba şi trebuie să examineze întregul program din nou şi să decidă iarăşi cu privire la modul cum va aloca memoria datelor şi instrucţiunilor.
Astfel au apărut începând cu 1950 limbajele de asamblare care sunt forme mai prietenoase ale limbajului maşină. Astfel comenzile limbajului maşină au fot înlocuite de comenzi mnemonice (gestionează memoria!!! – nu e foarte ortodoxă). Astfel limbajul de asamblare are grijă să convertească comenzile mnemonice în comenzile corespunzătoare limbajului maşină. Programatorul poate folosi adrese simbolice pentru reprezentarea datelor. Acest limbaj va atribui adresele în limbaj maşină şi va verifica dacă datele distincte nu se suprapun la stocare. Astfel scurtul program de mai sus se poate scrie în limbaj de asamblare astfel:

operation code	**address**
LOAD	A
ADD	B
STORE	C

Evident acest limbaj evită o multitudine de erori de alocare a memoriei. Limbajul de asamblare presupune existenţa unui program numit **assembler** care să traducă programele în limbaj maşină. Asamblorul înlocuieşte codarea mnemonica (cum ar fi ADD) cu coduri binare corespunzătoare limbajului maşină şi alocă adrese de memorie pentru toate variabilele simbolice utilizate (A, B, C) şi se asigură că aceste adrese sunt distincte.
Astfel prin uşurarea procesului de programare s-a introdus un nou nivel de procesare pentru computer. Astăzi limbajele de asamblare sunt încă utilizate pentru unele programe critice deoarece aceste limbaje conferă programatorului un control foarte precis a ceea ce se întâmplă în computer. Limbajele de programare încă necesită ca programatorul să aibă foarte bune cunoştinţe cu privire la structura internă a computerului. Limbajele de asamblare sunt şi ele specifice computerului pe care rulează astfel că programatorul trebuie sa-şi rescrie programul pentru un alt tip de computer.

Concluzii:

- programele se scriu în mod text pentra ca apoi sa fie traduse intr-o formă binară corespunzătoare limbajului maşină;
- limbajul de asamblare este tradus în limbaj maşină de către **assembler**;
- limbajele de asamblare încă solicită programatorului cunoaşterea de multe detalii hardware;
- sunt specifice anumitor tipuri de computere;
- **limbajul de asamblare împreună cu limbajul maşină formează categoria limbajelor de nivel scăzut (low-level languages).**

Limbajele evoluate (High level Languages)

Evident că de la apariţia computerului tot s-a pus problema de a se obţine un proces de programare cât mai uşor. Aceasta ar presupune ca activitatea de programare sa se poată face cu un bagaj de cunoştinţe cu privire la funcţionarea internă a computerului cât mai mic. După cum s-a văzut anterior limbajul maşină şi limbajul de asamblare presupune o bună cunoaşterea funcţionării interne a computerului.
O altă direcţie pentru uşurarea programării ar fi aceea ca programele să fie prezentate într-o limbă cât mai familiară persoanei care doreşte să programeze (să rezolve o anumită problemă).

Astfel au apărut **programarea de înalt nivel** care permite formularea soluţiilor problemei de rezolvat în termeni mai apropiaţi de cei folosiţi de oameni. Aceste limbaje au fost concepute pentru a permite programării să fie mult mai uşoară şi cu mai puţine erori iar programatorul nu trebuie să cunoască detalii cu privire la structura internă a unui anumit tip de computer. Aceste limbaje sunt mult mai apropiate de limbajul uman.
Primul limbaj evoluat a fost **FORTRAN II** apărut în 1958. Programul prezentat mai sus se poate scrie în acest limbaj astfel:

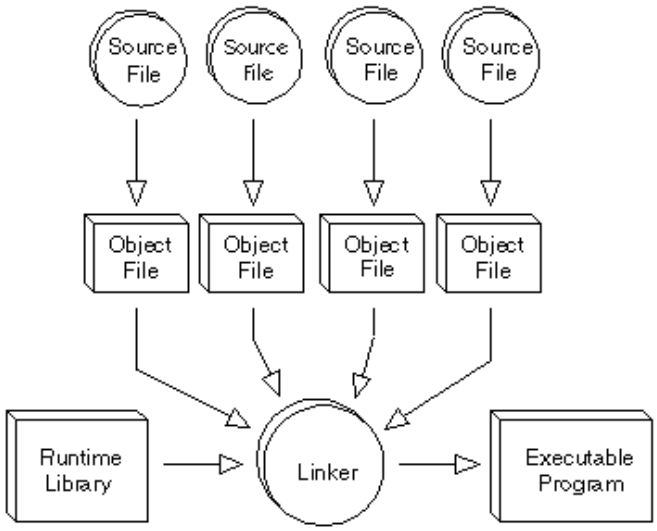
C=A+B

Se observă că spre deosebire de limbajul maşină şi limbajul de asamblare expresia de mai sus este mul mai uşor de înţeles, mai rapid de scris şi evită multe erori de scriere.

Pentru a putea fi executate, ca şi în cazul limbajului de asamblare, computerul nu înţelege în mod direct aceste limbaje evolute şi de aceea ele trebuie procesate printr-un program specializat care traduce limbajele evolute în limbajul maşină intern al computerului respectiv. După modul cum se face această transpunere a programelor evoluate există două tipuri de astfel de programe specializate:

- **interpretor:** traduce succesiv instrucţiunile de înalt nivel într-o formă intermediară care este apoi executată.
- **compiler:** traduce instrucţiunile de înalt nivel direct în limbaj maşină.

Avantajul interpretorului este că poate execută un program imediat. Compilatorul necesită mai mult timp. Programele produse de compilator rulează mul mai rapid decât cele produse de interpretor.
Majoritatea programelor evolute au la dispoziţie atât compiler cât şi interpretor. De obicei interpretorul se foloseşte în timpul realizării unui program pentru testarea unor mici secţiuni ale programului. Unele limbaje evoluate sunt concepute să lucreze numai cu interpretor (BASIC, LISP).



Un alt mare avantaj al limbajelor evoluate este acela că dacă limbajele sunt standardizate atunci fiecare producător de computere (procesoare) va putea să realizeze **compilerul** care să respecte standardele şi să traducă programele în limbajul maşină specific producătorului. Astfel devine posibil ca un program, respectându-se aceste standarde, să poată fi compilat pe diverse computere şi apoi executat.

Prin scrierea unui program într-un limbaj evoluat se face o economie imensă de timp. Astfel programatorul pierde mai puţin timp pentru scrierea într-un limbaj mult mai apropiat de cel uman decât dacă acelaşi program ar fi scris în limbaj maşină (vezi exemplul de mai sus). Timpul de compilare al programului este de ordinul secundelor !!!
De la apariţia limbajului FORTRAN II, au apărut şi dispărut multe limbaje evoluate. Cele mai utilizate la ora actuală sunt:

COBOL	Afacerilor (baze de date, etc)
FORTRAN	Inginerie + Matematică + Fizică + etc
PASCAL	Uz general
C, C++	Uz general – cel mai popular
PROLOG	Inteligenţa artificială
JAVA	Uz general – creşte în popularitate

COBOL (COmmon Business-Oriented Language) - un limbaj de programare dezvoltat între anii 1959-1961. Este utilizat în special pentru aplicatii cu caracter economic si administrativ. A fost initial sustinut de Departamentul de Apărare American si ulterior dezvoltat pentru aplicatii comerciale. Programele scrise în COBOL, limbaj compilat, contin patru diviziuni: identifi-carea, mediul, date si proceduri.

FORTRAN (**FOR**mula **TRAN**lation) - primul limbaj de nivel înalt (1954-1958 Jim Bachus) si cel care a introdus si definit concepte ca variabile, expresii, enunturi, iterative, subrutine compilate separat si input/output format. FORTRAN este un limbaj compilat, structurat. Chiar numele aratå originile stiintifice si ingineresti; FORTRAN este încå foarte folosit în aceste domenii.

PASCAL - limbaj procedural concis, proiectat de Niklaus Wirth în perioada 1967-1971. Pascal, limbaj structurat, compilat, construit pe baza limbajului ALGOL, simplificå sintaxa adåugând tipuri de date si structuri ca subzone, tipuri enumerate de date, fisiere, înregistråri si seturi. Acceptarea si utilizarea lui Pascal s-a mărit considerabil o datå cu introducerea în 1984, de către Borland International, a mediului Turbo Pascal, un compilator ieftin, de mare vitezå, utilizat în MS-DOS, care s-a vândut în peste un milion de copii în diverse versiuni. Chiar asa, Pascal pare så piardå teren în favoarea lui C ca limbaj standard de dezvoltare la microcomputere.

C - 1. Limbaj de programare dezvoltat de Dennis Ritchie la Bell Laboratories, New Jersey, în anul 1972, numit asa pentru cå limbajul imediat premergåtor a fost limbajul B. Desi C este considerat de multi a fi mai mult un limbaj de asamblare decât un limbaj de nivel înalt, asocierea strânså cu sistemul de operare UNIX, popularitatea sa enormå cât si standardizarea de către ANSI au fåcut ca acesta så fie aproape un limbaj de programare standard. C este un limbaj compilat ce contine un mic set de functii built-in care sunt machine-dependent. Celelalte functii sunt independente si sunt continute în niste fisiere numite library care pot fi accesate dintr-un program C. Programele C sunt compuse dintr-una sau mai multe functii definite de programator; astfel cå C se considerå a fi un limbaj structurat. Compilatorul nu dispune, în general, de functii de intrare/iesire, din pretentia, de altfel îndreptåtitå, de a se asigura portabilitatea lui; aceste functii fiind într-o bibliotecå foarte vastå. Existå douå implementåri de compilatoare foarte råspândite pe calculatoarele personale: Turbo C, produs de firma Borland în douå versiuni, normalå si profesionalå si Microsoft C, produs de firma Microsoft.

C++ - versiune a limbajului C, orientatå spre obiect, dezvoltatå de Bjarne Stroustrup la începutul anilor 80 la Bell Laboratories. C++ are tråsåturi specifice limbajelor de programare orientate obiect. Conceptul fundamental în C++ este clasa. El contine de asemenea îmbunåtåtiri care nu sunt direct legate de clase, cum ar fi: constante simbolice, substitutia “in-line” a functiilor, argumente cu valori implicite pentru functii, nume de functii supraîncårcate, operatori pentru gestionarea memoriei libere si un tip de referintå

PROLOG (PROgramming LOGic) - limbaj de programare creat în 1973 de Alain Colmeraner la Universitatea din Aix-Marseille. Este un limbaj descriptiv destinat inteligentei artificiale, fiind utilizat în special la realizarea sistemelor expert. Programa-torul formează o bază de cunostinte, adică un set de

reguli si fapte legate de mediul tratat, după care descrie problema de rezolvat. PROLOG-ul este un limbaj care sparge regulile traditionale de programare.

Java este un limbaj de programare de nivel înalt, orientat obiect, proiectat inițial pentru realizarea de aplicații pentru Internet și mai cu seamă pentru Web. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet urilor. În acest sens, multe firme recurg la limbajul Java în procesul de informatizare întrucât oferă un foarte bun suport pentru tehnologiile de vârf și, nu în ultimul rând, pentru faptul că este gratuit și în mod continuu îmbogățit și îmbunătățit.

✧**!NOTA : Donald Ervin Knuth** (născut pe 10 ianuarie 1938) este profesor emerit la Universitatea Stanford.

Este autorul cărții *The Art of Computer Programming* ("Arta prográmarii calculatoarelor"), una dintre cele mai apreciate în acest domeniu și creator al TeX și Metafont.

A primit numeroase premii, printre care: premiul Turing, Medalia națională în științe, medalia John von Neumann și premiul Kyoto. **The Art of Computer Programming** (*Arta programării calculatoarelor*) este una dintre cele mai faimoase cărți din domeniul informatic, scrisă de Donald E. Knuth, carte ce acopera toate genurile de algoritmi cu demonstrații matematice riguroase.

Din această carte monumentală prin dimensiuni și conținut, au apărut trei volume, dar Knuth a anunțat că vor mai urma alte patru.