

# Paradigma de Programare



*Titular*  
M. Mihai Zaharia

# Observații generale

- Linux?
- boot "bare metal"
- Aspecte generale cu privire la examinare
  - teorie
  - lab
- Bologna vs Harețian?

# Cum este cu înregistrările????

- No way!!!
- Why?

# Laptop recomandat

- minim 16 GB RAM(daca are linux)
- minim 20 (daca are win in special 10)
- ideal 32 GB
- evitati procesoarele cu extensia M,LV etc
- preferabil cele cu HQ
- minim I5 (dar de ultima gen)
- recomandat tastură luminată
- recomandat I7 sau I9
- 2 hdd-uri din care cel de boot si os obligatoriu SSD
- la limita merge si un SSHD dar prost (pentru cei care nu pot baga al doilea hdd) -
  - min Mon 15,6 inch glossy/mate
  - verificati ca ecranul are buna vizibilitate si in soare direct
  - preferabil cu placa retea intel in extremis qualcomm/atheros
  - firme low cost - acer deoarece au un program de proiectare bios/hw care tine cont de linux
- nu este obligatorie placa video performanta - papa baterie si cam atat daca nu ai ce face dezactivati-o din bios sau in linux din bumble bee (dar preferabil nvidia pt cuda)

# Obiectivele cursului

1. Limbajele care vor fi folosite de-a lungul cursului sunt:

1. *UML*
2. *C/C++*
3. *Java Script*
4. *Python*
5. *Kotlin*
6. *R*
7. *Polyglot*
8. *Prolog*

# Criterii folosite în evaluarea activității

- **Participarea:** la orele de curs și de laborator:
  - Neparticiparea la mai mult de 50% din laboratoare conduce la refacerea disciplinei.
  - neparticiparea la curs conduce la probleme la examenul teoretic
- **Laboratoare:** pentru a putea lua 10 la laborator trebuie ca studentii să fie capabili la intrebarile asistenților cu privire la conținutul cursului curent (și pentru care a fost creat laboratorul) - 30% - atenție asistentii nu stau să repredea!! ci numai notează corectitudinea răspunsului și trec mai departe
- **Examen final 70%** (este o singura notă) defalcată astfel:
  - Proba de **laborator** – **ELIMINA-TORIE 40%** cu biletă și două ore maxim la dispoziție. Un subiect din două trebuie să fie îndeplinit integral pentru a se putea nota (min 5).
  - Proba **teoretică** – “**PICĂ-TORIE**” **40%** - test docimologic - conține și întrebări cu caracter practic specifice laboratorului (min 5)
  - **Teme acasă: semi** - “**PICĂ-TORIE**” la majoritatea laboratoarelor **20%** (atenție se poate să aveți 5 la lab și teorie și să picați din teme nepredate)
- În caz de variantă on line (SARS-CoV-II)
  - testul practic - se aleg automat două probleme din pool și se trimit
  - testul teoretic - oral cu întrebări selectate automat din pool

# Îndatoririle studentului

1. **Citiți materialele** recomandate cu o zi înainte de fiecare curs – elaborați o lista de întrebări eventual.
2. **Rezolvați temele** pentru acasă în săptămâna în care le-ați primit.
3. **Participați la cursuri** (suportul de curs livrat va conține uneori numai desenele/cod din slide-uri).
4. **Rezervați un minim de 2-4 ore** de studiu individual pe săptămâna pentru aceasta materie.
5. **Verificați înțelegerea teoretică și practică** a noțiunilor asimilate prin ajutarea colegilor cu răspunsuri legate de partea teoretică sau ajutor (NU tema copiată) în rezolvarea problemelor practice.



# TIOBE Programming Community Index for 2017

Feb 2017	Feb 2016	Programming Language	Ratings	Change
1	1	Java	16.676%	-4.47%
2	2	C	8.445%	-7.15%
3	3	C++	5.429%	-1.48%
4	4	C#	4.902%	+0.50%
5	5	Python	4.043%	-0.14%
6	6	PHP	3.072%	+0.30%
7	9	JavaScript	2.872%	+0.67%
8	7	Visual Basic .NET	2.824%	+0.37%
9	10	Delphi/Object Pascal	2.479%	+0.32%
10	8	Perl	2.171%	-0.08%
11	11	Ruby	2.153%	+0.10%
12	16	Swift	2.125%	+0.75%
13	13	Assembly language	2.107%	+0.28%
14	38	Go	2.105%	+1.81%
15	17	R	1.922%	+0.73%
16	12	Visual Basic	1.875%	+0.02%
17	18	MATLAB	1.723%	+0.63%
18	19	PL/SQL	1.549%	+0.49%
19	14	Objective-C	1.536%	+0.13%
20	23	Scratch		

# TIOBE Programming Community Index for 2018

<https://www.tiobe.com/tiobe-index/>

Feb 2019	Feb 2018	Change	Programming Language	Ratings	Change
1	1		Java	15.876%	+0.89%
2	2		C	12.424%	+0.57%
3	4	▼	Python	7.574%	+2.41%
4	3		C++	7.444%	+1.72%
5	6		Visual Basic .NET	7.095%	+3.02%
6	8		JavaScript	2.848%	-0.32%
7	5		C#	2.846%	-1.61%
8	7		PHP	2.271%	-1.15%
9	11		SQL	1.900%	-0.46%
10	20		Objective-C	1.447%	+0.32%
11	15		Assembly language	1.377%	-0.46%
12	19		MATLAB	1.196%	-0.03%
13	17		Perl	1.102%	-0.66%
14	9		Delphi/Object Pascal	1.066%	-1.52%
15	13		R	1.043%	-1.04%
16	10		Ruby	1.037%	-1.50%
17	12		Visual Basic	0.991%	-1.19%
18	18		Go	0.960%	-0.46%
19	49		Groovy	0.936%	+0.75%
20	16		Swift	0.918%	-0.88%



Nov 2022

Nov 2021

Change

Products  
ProgrammingQuality Models  
LanguageMarkets  
RatingsSchedule a demo  
Change

Rank	Rank Change	Icon	Language	Market Share	Market Share Change
1	1		Python	17.18%	+5.41%
2	2		C	15.08%	+4.35%
3	3		Java	11.98%	+1.26%
4	4		C++	10.75%	+2.46%
5	5		C#	4.25%	-1.81%
6	6		Visual Basic	4.11%	-1.61%
7	7		JavaScript	2.74%	+0.08%
8	8		Assembly language	2.18%	-0.34%
9	9		SQL	1.82%	-0.30%
10	10		PHP	1.69%	-0.12%
11	18		Go	1.15%	-0.06%
12	15		R	1.14%	-0.14%
13	11		Classic Visual Basic	1.10%	-0.46%
14	17		Delphi/Object Pascal	1.08%	-0.14%
15	20		MATLAB	1.02%	-0.15%

# Ergonomie

1. Alegere monitor – mat
2. Dimensiune minima ecran
3. Alegere densitate / rezoluție ecran
4. Reglare luminozitate și culoarea albastră
5. Poziție corectă de lucru – desktop/laptop
6. Sporturi recomandate

# Criterii generale de proiectare a unui limbaj

- Putere (excelează în rezolvarea...)
- Flexibilitate
- Expresivitate
- Ușor de scris (vezi C vs Pascal)
- Implementare eficientă
- Support pentru abstractizări

- Simplitate
- Claritate
- Consistență (ortogonalitate)  
(puține structuri de control cu puține posibile combinații)
- Usurință în urmărirea cod
- Aplicabilitatea în domeniul problemei (>=gen4)
- Portabilitate

# Definirea unui limbaj

- **Sintaxa:** se definește gramatica unui limbaj
  - Ce înseamnă o propoziție corectă? dar un program corect?
  - De obicei se folosesc notații formale ca BNF sau forma sa extinsă EBNF.
- **Semantica:** interesul elementelor limbajului
  - De obicei a fost legat de limbajele naturale (umane)
  - Notații formale există dar nu sunt foarte folosite

# Sintaxa

- Definește simbolurile și gramatica unui limbaj
  - BNF sau
  - EBNF

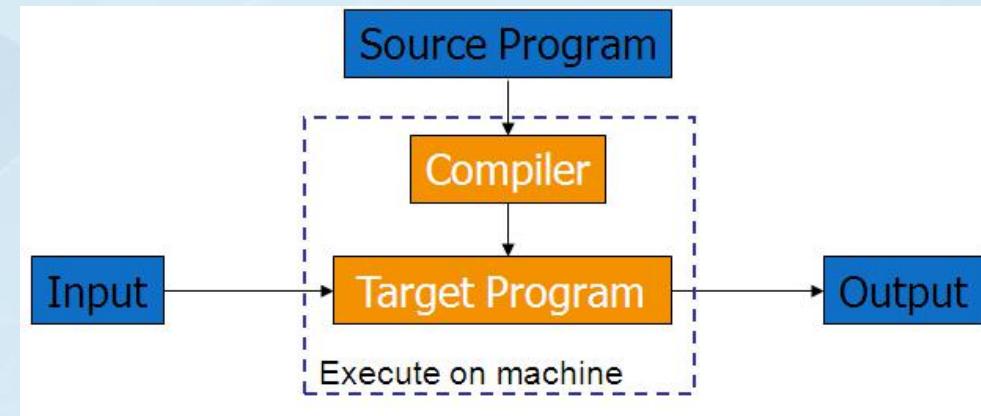
*if-statement* ::= if ( *expression* ) *statement-block*  
[ else *statement-block* ]

*statement-block* ::= *statement* ';'  
| '{' *statement* ';' [...] '}'

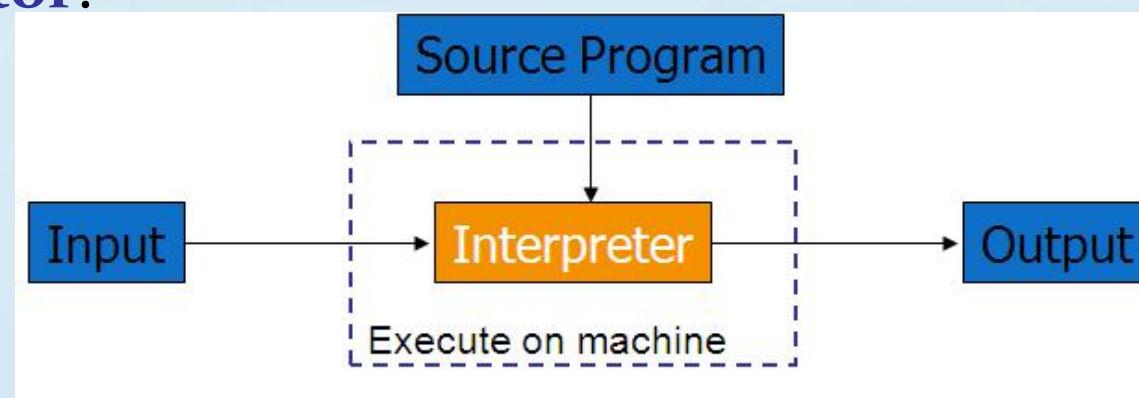
*statement* ::= *if-statement* | *assignment-statement* |  
*while-statement* | ...etc...

# Strategii de implementare

- **Compilator:**

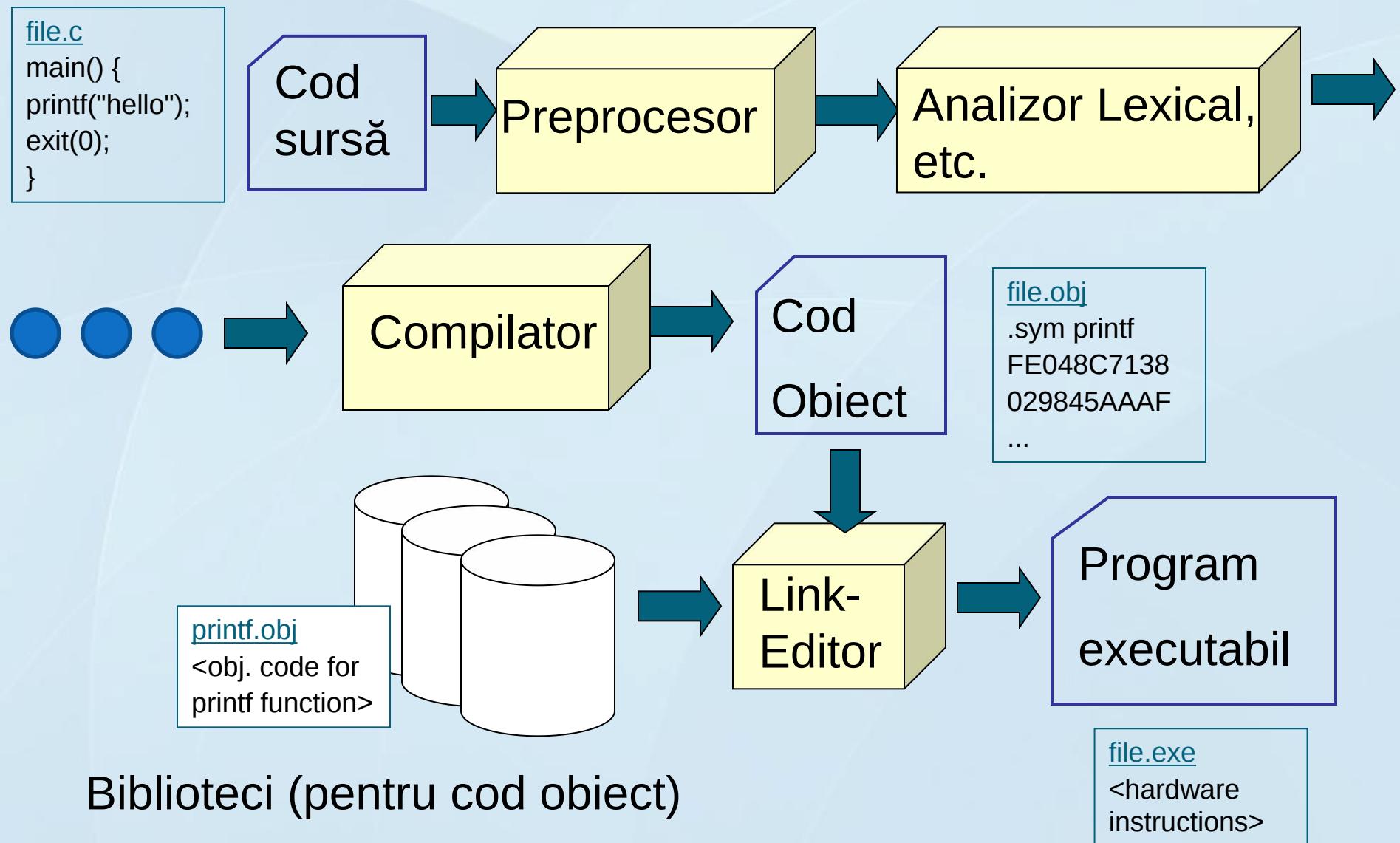


- **Interpreter:**



- **Hibrid:**

# Compilarea unui program



# Etape ale compilării

Program  
Sursă

Analizor Lexical

Codul este convertit în expresii cu simboluri speciale și analizat

Analizor Sintactic/Parser

Se verifică analiza sintactică și atât

Analizor Semantic

Se adaugă informație semantică peste arborele de parsare și se crează tabelul de simboluri

Generator de cod intermediar  
(via assembly sau nu)

Optimizare cod mem/speed

Generare cod

Program  
țintă

# Program Interpretat vs Compilat

## Interpretat

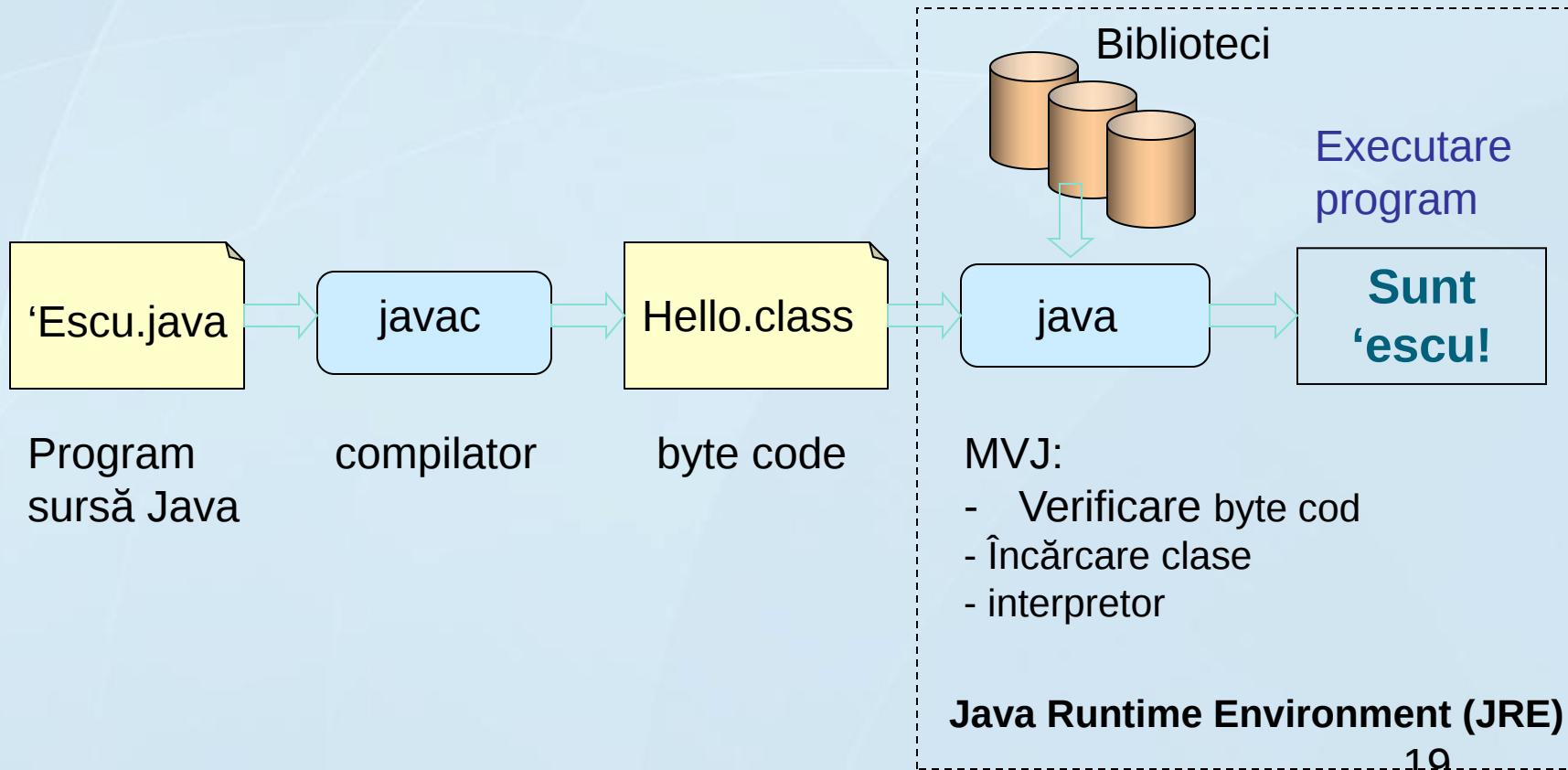
- Flexibil
- Interactiv
- Comportament dinamic mai complex
- Dezvoltare rapidă
- Programul poate fi executat imediat ce este scris/modificat
- Portabil pe orice mașină cu interpretor

## Compilat

- Execuție mult mai eficientă (Java vs .Net vs. C++)
- Analiza datelor este mai amănunțită
- Mai structurat
- De obicei mai scalabil (aplicații mari dimensiuni)
- Trebuie recompilat programul după fiecare modificare
- Trebuie recompilat pentru orice diferență în OS sau HW a mașinii țintă

# Java: o strategie hibridă

- **Compilator Java:** crează un byte code independent de mașină
- **Mașina Virtuală Java (Interpreter):** execută acel cod.



# Classificare Erori

- **Lexicale**: erori la nivel de token, cum ar fi caractere ilegale (greu de distins din erorile de sintaxă).
- **De sintaxă**: erori gramaticale (e.g. “;” lipsa sau cuvânt cheie).
- **Statice de semantică**: care se pot detecta înainte de execuție (variabile nedefinite, erori de tip)
- **Dinamice și de logică**

# Observații cu privire la generarea erorilor

- **Un compilator** va raporta erori lexicale, de sintaxă și pe cele statice. Nu va raporta pe cele dinamice de semantică.
- **Un interpretor** de obicei va raporta erori de sintaxă și lexicale.
- Nici un **translator** nu va raporta o eroare logică.

# Depanare pas cu pas???

- oare este necesara?
- DA
- nu confundați cu sari la ... și execută

# Java code/bytocode pentru JVM

- outer:
- for (int i = 2; i < 1000; i++)
- {
- for (int j = 2; j < i; j++)
- {
- if (i % j == 0) continue outer;
- }
- System.out.println (i);
- }

```
0:  iconst_2 //începe for
1:  istore_1
2:  iload_1
3:  sipush 1000
6:  if_icmpge 44 //comparatia cu 1000
9:  iconst_2 //începe for
10: istore_2
11: iload_2
12: iload_1
13: if_icmpge 31 // comparația cu i
16: iload_1 //începe if
17: iload_2
18: irem
19: ifne 25 // comparatia cu 0
22: goto 38
25: iinc 2, 1 //j++
28: goto 11
31: getstatic #84; //apel PrintStream
34: iload_1
35: invokevirtual #85;
//PrintStream.println:(I)V
38: iinc 1, 1 // i++
41: goto 2
44: return
```

# Probleme specifice limbajului de asamblare

Programarea este dificilă chiar folosind limbajul macro de asamblare (cu .small etc)

Limbajul nu se potriveste cu maniera în care gandesc oamenii

Programele sunt lungi și dificil de înțeles (dacă nu ai experienta și le scrii cu picioarele)

Erorile de logică în program

Limbajul este dependent de mașină

# Primele încercări de abstractizare

- Fortran, primul limbaj de nivel înalt,

```
PROGRAM EXEMPLU      ! VERSION 0.0.  
CALL HELLO !apelarea subrutinei HELLO.  
CONTAINS  !CONTAINS încheie program principal  
si incepe definirea subrutinelor  
SUBROUTINE HELLO  !definirea corp  
SUBROUTINA  
WRITE(*,*)  "HELLO WORLD!" ! afisez "HELLO  
WORLD!" la CONSOLA.  
END SUBROUTINE HELLO ! Sfarsit SUBROUTINA  
END PROGRAM EXEMPLU !sfarsit PROGRAM
```

```
@echo off cls  
echo Press any key to start  
AProgram.exe!  
pause > nul  
AProgram.exe %1  
if errorlevel 1  
    goto error  
    echo AProgram has  
finished  
whatever it was doing.  
    goto end  
error: echo Something went  
        wrong with Aprogram  
end
```

# Limbaje (foste) de nivel înalt

A treia generație de limbaje, cunoscute și sub denumirea de limbaje de nivel înalt oferă următoarele avantaje:

Sintaxa în limba engleză

Nume descriptive pentru a reprezenta datele

Reprezentare concisă a logicii

Folosirea simbolurilor matematice standard

`total = quantity * price * (1 + taxrate);`

Operatori distincți pentru execuția condiționată a buclelor și expresiilor

`if ( total > 0 ) then writeln("The total is ", total)  
else writeln("No sale.");`

Apariția de unități funcționale ca funcții sau clase cu izolarea variabilelor:

`radius = sqrt( x*x + y*y );`

# Beneficii oferite de a treia generație

- Programarea este mai ușoară și mai rapidă ( yesss!)
- Programatorul gândește la un nivel mai abstract rezolvarea problemei (deci un matematician/ informatician poate scrie cod) fără a ști asamblare
- *Apare independența de masină*
- Se poate aplica la orice nivel o abordare de tip top down, bottom up etc
- Testare
- Reutilizare
- Biblioteci

# Ce este o paradigmă de programare?

- D<sub>1</sub> (generală) Caz exemplar, **model**, **prototip**, situație ideală, structură tip, **arhetip** standard și.a.
- D<sub>2</sub>. (în filozofie, la L.Wittgenstein) Modelele filosofice, acele "tipare" care orientează gândirea noastră în direcții predeterminate.,
- D<sub>3</sub> (în filozofia limbajului) Listă de cazuri tipice de jocuri lingvistice prin care putem înțelege conceptul general..."

# The principal programming paradigms

*"More is not better (or worse) than less, just different."*

