

Limbaje formale si translatoare - introducere

February 26, 2024

Outline

Administrativ

- ▶ Examen final - 60%: ≥ 4 pentru a se aduna laboratorul
- ▶ Laborator - 40%: ≥ 5 pt intrare in examen
 - ▶ Colocviu Lex si Yacc
 - ▶ Proiect (individual sau echipe de 2 studenti)
 - ▶ teme
- ▶ puncte suplimentare: Kahoot
- ▶ anca.marginean@cs.utcluj.ro
- ▶ Registration key Moodle - *lex&YaccX*, $X = 1..9$
 - ▶ Camelia Pintea - 1,2
 - ▶ Flaviu Cojocaru - 1
 - ▶ Adrian Burzo - 2
 - ▶ Alex Garleanu - 1
 - ▶ Stefan Popescu - 2
 - ▶ Constantin Senila - 1

- ▶ Syntax errors: Java vs python
- ▶ see BNF(BackusNaur form) for JAVA
- ▶ see AST for Javascript: tokens, AST tree

Bibliografie

- ▶ Curs introductiv - Capitolul 1. Michael Scott "Programming language pragmatics" third edition
- ▶ **I.A.Letia, E.S.Chifu "Limbaje formale si translatoare"**
- ▶ J.E. Hopcroft, R. Motwani, J.D. Ullman "Introduction to Automata Theory, Languages, and Computation"
- ▶ A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman : "Compilers: Principles, Techniques, and Tools" (dragon book)
- ▶ M. Sipster: "Introduction to the theory of computation" (third edition)

Outline

Motivatie

Evolutia limbajelor de programare

Compilare - Interpretare. Fazele unui compilator

Limbaje ezoterice

```
primesTo m = sieve [2..m]
    where
        sieve (x:xs) = x : sieve (xs \\ [x,x+x..m])
        sieve [] = []
```

```
import java.util.LinkedList;
import java.util.BitSet;

public class Sieve{
    public static LinkedList<Integer> sieve(int n){
        LinkedList<Integer> primes = new LinkedList<Integer>
            ();
        BitSet nonPrimes = new BitSet(n+1);

        for (int p = 2; p <= n ; p = nonPrimes.nextClearBit(
            p+1)) {
            for (int i = p * p; i <= n; i += p)
                nonPrimes.set(i);
            primes.add(p);
        }
        return primes;
    }
}
```

Limbaje formale si translatoare

- ▶ Un limbaj e un set legal de propozitii

Limbaje formale si translatoare

- ▶ Un limbaj e un set legal de propozitii
- ▶ O propozitie e o secventa de simboluri

Limbaje formale si translatoare

- ▶ Un **limbaj** e un set legal de propozitii
- ▶ O **propozitie** e o secventa de simboluri
- ▶ Un **simbol** poate fi un caracter, un cuvant, un semn de punctuatie, ...

Limbaje formale si translatoare

- ▶ Un **limbaj** e un set legal de propozitii
- ▶ O **propozitie** e o secventa de simboluri
- ▶ Un **simbol** poate fi un caracter, un cuvant, un semn de punctuatie, ...
- ▶ Un **limbaj formal** este un limbaj definit de un **set finit de reguli** neambigue care delimiteaza propozitiile legale de cele ilegale

Limbaje formale si translatoare

- ▶ Un **limbaj** e un set legal de propozitii
- ▶ O **propozitie** e o secventa de simboluri
- ▶ Un **simbol** poate fi un caracter, un cuvant, un semn de punctuatie, ...
- ▶ Un **limbaj formal** este un limbaj definit de un **set finit de reguli** neambigue care delimiteaza propozitiile legale de cele ilegale

Obiective curs

- ▶ Cum se poate descrie un limbaj formal?
- ▶ Cum se poate recunoaste si prelucra un limbaj?
- ▶ Automate finite deterministe si nedeterministe, automate stiva pentru parsare
- ▶ + Elemente introductive de procesare a limbajului natural

Donalt Knuth(1938-): programming - "the art of telling another human being what one wants the computer to do"

Limbaje formale si translatoare - definire limbaj de programare si implementare

- ▶ specificam limbajul de programare folosind modele formale - gramatici si automate
- ▶ transformam aceste modele formale intr-o implementare

Implementare limbaje de programare: 3 strategii

- ▶ interpretare - *source* $\xrightarrow{\text{interpret}}$ *actions/results*
- ▶ compilare - *source* $\xrightarrow{\text{translates}}$ *machine / assembly language* $\xrightarrow{\text{execute}}$ *actions/results*
- ▶ hibrid
 - ▶ Just-In-Time(JIT) compilers - interpreteaza parti din program, compileaza alte parti in timpul executiei
 - ▶ compilatoare care translateaza programul in *alte limbaje de programare*(precum C) sau limbaj intermediar (Java *bytecode*) pentru care exista un translator sau compilator

Observatie: strategia este specifica implementarii unui limbaj, si nu unui limbaj (exista interpretoare C si compilatoare Lisp)

Outline

Motivatie

Evolutia limbajelor de programare

Compilare - Interpretare. Fazele unui compilator

Limbaje ezoterice

Evolutia limbajelor de programare

Machine language - instructiuni in binar sau hexazecimal care controleaza direct unitatea centrala de procesare (CPU)

```
55 89 e5 53 83 ec 04 83 e4 f0 e8 31 00 00 00 89 c3 e8 2a 00
00 00 39 c3 74 10 8d b6 00 00 00 00 39 c3 7e 13 29 c3 39 c3
75 f6 89 1c 24 e8 6e 00 00 00 8b 5d fc c9 c3 29 d8 eb eb 90
```

Listing 1: gcd in hexazecimal

Limbaj de asamblare

- ▶ one-to-one correspondence: mnemonics - machine language instructions
- ▶ programare dependenta de setul de instructiuni al masinii

```
pushl    %ebp                                jle     D
movl     %esp, %ebp                          subl    %eax, %ebx
pushl    %ebx                                B:    cmpl    %eax, %ebx
subl     $4, %esp                            jne     A
andl     $-16, %esp                         C:    movl    %ebx, (%esp)
call     getint                             call     putint
movl     %eax, %ebx                         movl     -4(%ebp), %ebx
call     getint                             leave
cmpl     %eax, %ebx                         ret
je       C                                D:    subl    %ebx, %eax
A: cmpl   %eax, %ebx                        jmp     B
```


Limбай masina - limбай asamble

```
55 89 e5 53 83 ec 04 83 e4 f0 e8 31 00 00 00 89 c3 e8 2a 00  
00 00 c9 c3 74 10 8d b6 00 00 00 00 39 c3 7e 13 29 c3 39 c3  
75 f6 89 1c 24 e8 6e 00 00 00 8b 5d fc c9 c3 29 d8 eb eb 90
```

Correspondenta one-to-one *cmpl %eax, %ebx* e reprezentat de
secventa 39 c3

1950 Fortran - first high-level programming language

*FOR*mula *TRAN*slator; Rapid urmat de Lisp si Algol
expresii aritmetice, If, Do, Goto

```
10  if (a .EQ. b) goto 20
    if (a .LT. b) then
        a = b - a
    else
        b = a - b
    endif
    goto 10
20 end
```

Evolutie

- ▶ COBOL(1959) - Common Business-Oriented Language; type declarations, record types, file manipulation
- ▶ LISP - McCarthy, MIT, 1958; functional: recursive
- ▶ APL (array manipulation) - IBM, 1960; imperative, matrix-centric; symbols α
- ▶ Algol, Pascal(1970), Clu, Modula, Ada - Imperative, block-structured language, formal syntax definition, structured programming
- ▶ SNOBOL, Icon - string processing languages
- ▶ 1964 - BASIC - Beginner's All-purpose Symbolic Instruction Code; programming for the masses; la Dartmouth College
 - ▶ 1975 - Altair BASIC (Bill Gates, Paul Allen) - Micro-Soft
- ▶ C(1969)- procedural, imperative
- ▶ Simula, Smalltalk, C++, Java (1991), C# - object oriented
- ▶ ML, Miranda, Haskell(1990) - functional languages with types
- ▶ sh, awk(1977), perl, tcl, python(1989), php - scripting langs
- ▶ SQL(1974 IBM) - database queries
- ▶ Prolog(1972) - logic programming language

Clasificarea limbajelor

declarative

functional	Lisp/Scheme, ML, Haskell
dataflow	Id, Val
logic, constraint-based	Prolog, spreadsheets
template-based	XSLT

imperative

von Neumann	C, Ada, Fortran, ...
scripting	Perl, Python, PHP, ...
object-oriented	Smalltalk, Eiffel, Java, ...

Declarativ - *ce* trebuie sa faca

Imperativ - *cum* trebuie sa faca

Preluat din Michael Scott "Programming language pragmatics" third edition

Popularitatea Limbajelor

- ▶ TIOBE index - “The Importance of Being Earnest”
<https://www.tiobe.com/tiobe-index/>
- ▶ PYPL Index : The PYPL PopularitY of Programming Language Index <http://pypl.github.io/PYPL.html>

Outline

Motivatie

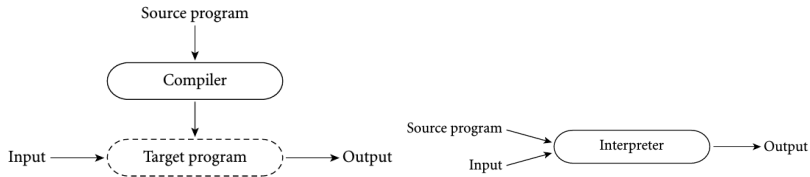
Evolutia limbajelor de programare

Compilare - Interpretare. Fazele unui compilator

Limbaje ezoterice

Compiler vs Interpretor

- ▶ Compiler: **translatarea** dintr-un **limbaj de nivel inalt** in limbaj asamblare sau masina
- ▶ Object code - rezultatul compilarii

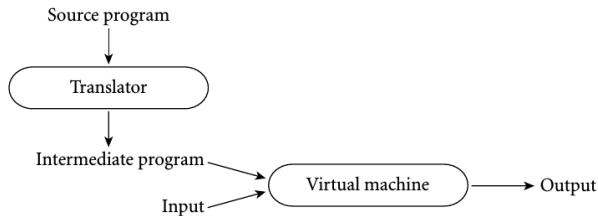


Compilerul nu face parte din executie; interpretorul citește instrucțiuni mai mult sau mai puțin una câte una și le execută

Comparatie

- ▶ Interpretarea - mai mare flexibilitate si mesaje de eroare mai bune
- ▶ Compilarea - performanta mai buna (decizii la momentul compilarii)
exemplu: GHC optimizare tail-recursive calls - apelul recursiv e ultimul statements din functie

Combinare compilare - interpretare



- ▶ Java bytecode - rezultatul compilarii codului sursa; executat in Java virtual machine (JVM)

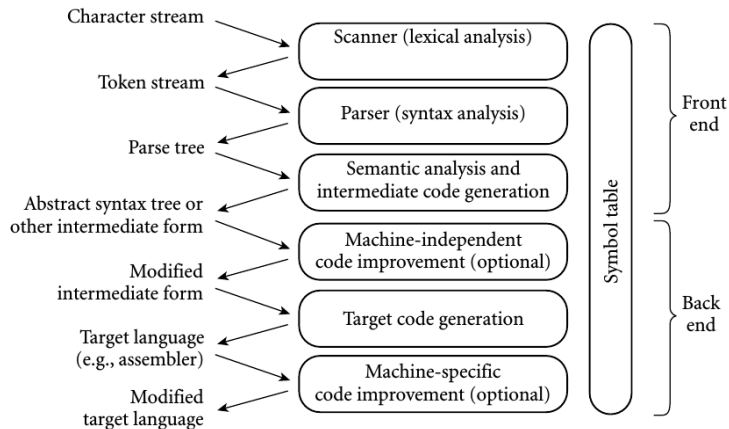
Preprocesare - eliminarea comentariilor, spatiile

Observatii

Asamblor - traducere din asamblare in cod masina (initial folosind o mapare 1 la 1)

Compiler - traducere din limbaj de nivel inalt in asamblare sau cod masina; substantial mai complicat decat asamblorul (nu exista mapare 1 la 1 intre sursa si target)

Fazele unui compilator



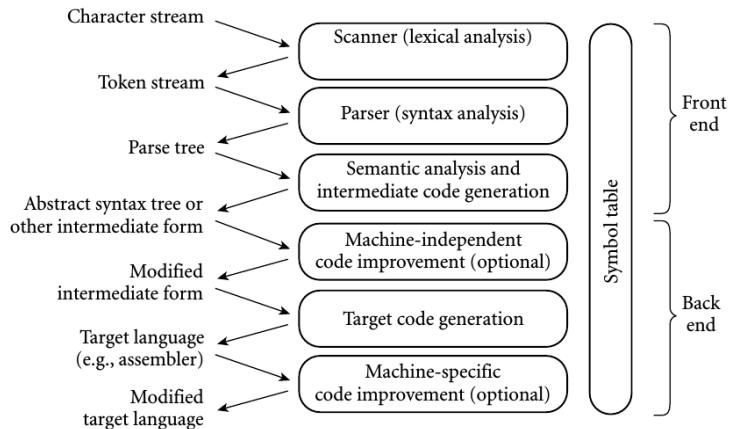
```
int main() {  
    int i = getint(), j = getint();  
    while (i != j) {  
        if (i > j) i = i - j;  
        else j = j - i;  
    }  
    putint(i);  
}
```

Scanare

- ▶ Scanarea (analiza lexicala) - citeste caracterele si le grupeaza in token-i (cea mai mica unitate cu sens a unui program)
- ▶ elimina de obicei si comentariile si spatiile albe
- ▶ reduce dimensiunea inputului pentru parser (numarul de caractere e mult mai mare decat cel de tokeni)

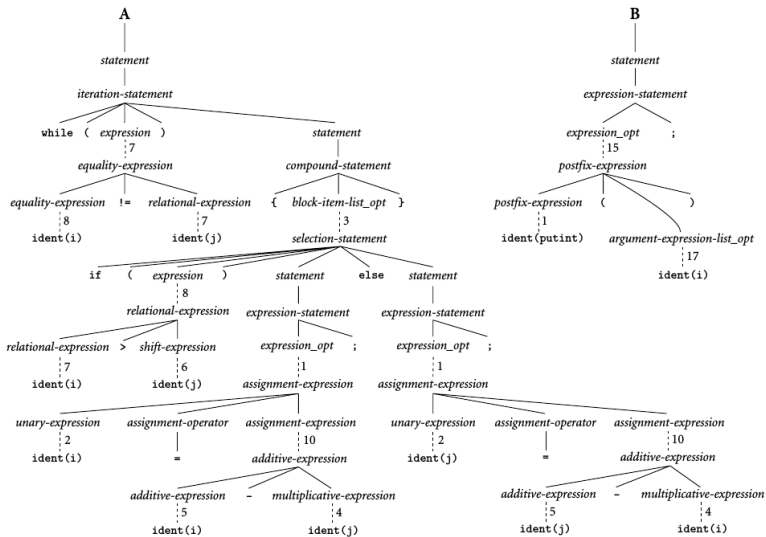
```
int    main    (    )    {    int    i    =  
getint (    )    ,    j    =    getint (      
    )    ;    while (    i    !=    j    )  
{    if    (    i    >    j    )    i  
=    i    -    j    ;    else    j    =  
j    -    i    ;    }    putint (    i  
)    ;    }
```

Fazele unui compilator - parsare



Parsare

- ▶ Parsarea - organizeaza token-ii in arbori de parsare (parse tree)
- ▶ cum formeaza tokenii un program
- ▶ analizor sintactic - derivator sau parser
- ▶ Arbore de parsare(derivare): radacina e programul, frunzele sunt token-ii de la analizorul lexical



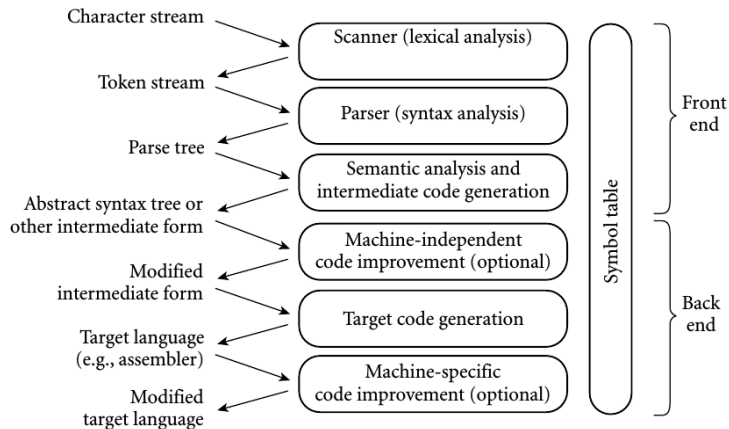
Observatii: Analizor lexical + sintactic

- ▶ Scanarea (analiza lexicala) si parsarea (analiza sintactica) - verifica daca toti token-ii sunt corecti (well formed) si secventa de token-i corespunde sintaxei limbajului

Intrebare - **cum definim limbajul?**

- ▶ Set de productii (gramatici), diagrame sintactice, BNF (Backus Naur Form)

Fazele unui compilator - analiza semantica



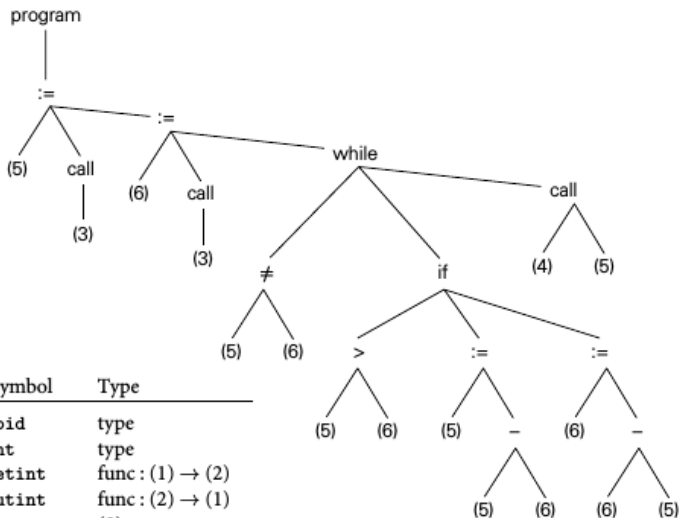
Analiza semantica si Generarea de cod intermediar

Descoperirea sensului programului

- ▶ folosirea aceluiasi identificator - aceeasi entitate; **tabela de simboluri (symbol table)**
- ▶ tipurile identificatorilor si ale expresiilor
- ▶ verificarea unor reguli semantice: ex - nu se aduna un string cu un intreg, procedurile sunt chemate cu nr corect de argumente (static rules)
- ▶ Observatie: exista si reguli semantice care nu pot fi verificate la momentul compilarii, ci doar la momentul rularii; ex: nu se acceseaza un element dintr-un array din afara limitelor acestuia (dynamic rules)

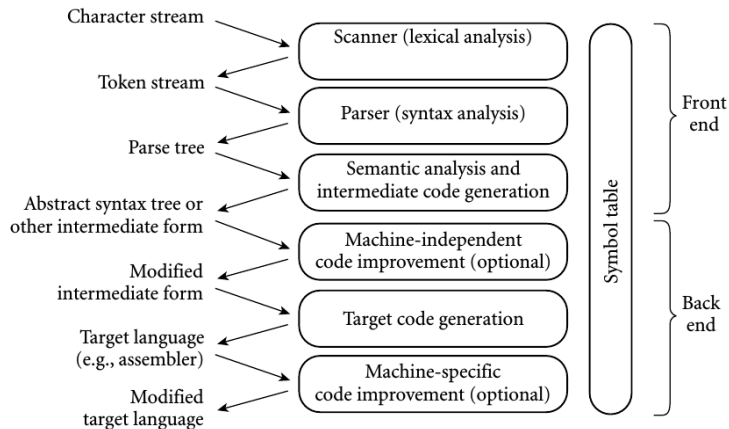
In multe compilatoare: actiuni la momentul identificarii unui pas particular intr-o regula gramaticala (semantic action routines)

Arbore sintactic (abstract syntax tree, sau syntax tree) - mai concis decat **arborele de derivare (parse tree)** (uneori numit **concrete syntax tree**)



Index	Symbol	Type
1	void	type
2	int	type
3	getint	func: (1) → (2)
4	putint	func: (2) → (1)
5	i	(2)
6	j	(2)

Fazele unui compilator - Optimizare si generare cod



Generarea de cod target. Optimizare

Pornind de la arborele sintactic si tabela de simboluri - cod limbaj de asamblare sau masina

Optimizare cod - independent sau dependent de masina

Front-end. Back-end

Front-end - verifica daca un program este corect scris in termenii sintaxei si semanticii limbajului de programare

Back end - responsabil cu translatarea sursei in code target (asamblare/masina)

Alt exemplu

$$position = initial + rate * 60$$

1. Analiza lexicala

Alt exemplu

$$position = initial + rate * 60$$

1. Analiza lexicala

Lexeme	Tokeni
position	ID
=	=
initial	ID
+	+
rate	ID
*	*
60	NUM

2. Parsare/Analiza sintactica

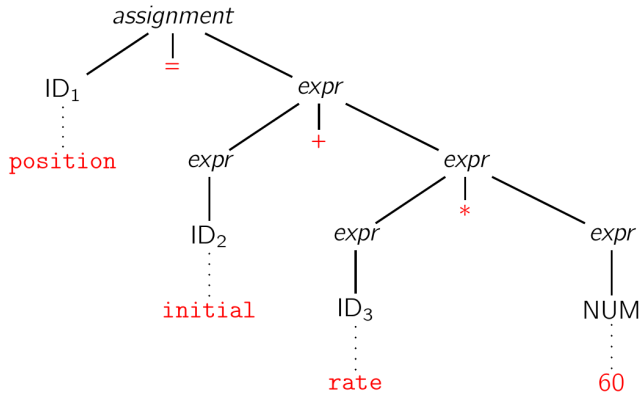
```
<assignment> -> ID "=" <expr>  
<expr>        -> ID | NUM | <expr><op><expr> | ( < expr > )  
<op>          -> + | - | * | /
```

2. Parsare/Analiza sintactica

<assignment> -> ID "=" <expr>

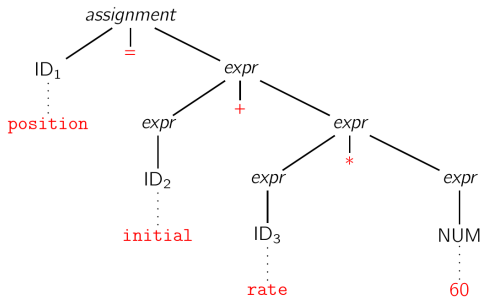
<expr> -> ID | NUM | <expr><op><expr> | (< expr >)

<op> -> + | - | * | /



3. Analiza semantica: verificare de tipuri si alte informatii

Generare cod intermediar din arborele de parsare



```
temp1 = inttoreal(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```

4. Exemplu de optimizare

```
temp1 = inttoreal(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```

```
temp1 = id3*60.0
id1 = id2+ temp1
```

5. Generare de cod

MOVF id3, R2 The F stands **for** floating-point instruction
MULF #60.0, R2 The # means that 60.0 is **a** constant
MOVF id2, R1 The first and second operand of **each**
 instruction
ADDF R2, R1 specify **a** source and **a** destination
MOVF R1, id1

position := initial + rate * 60

lexical analyzer

$id_1 := id_2 + id_3 * 60$

syntax analyzer

$$\begin{array}{c}
 id_1 \\
 \swarrow \quad \searrow \\
 := \quad + \\
 \swarrow \quad \searrow \quad \swarrow \quad \searrow \\
 id_2 \quad id_3 \quad 60
 \end{array}$$

semantic analyzer

$$\begin{array}{c}
 id_1 \\
 \swarrow \quad \searrow \\
 := \quad + \\
 \swarrow \quad \searrow \quad \swarrow \quad \searrow \\
 id_2 \quad id_1 \quad \text{inttoreal} \\
 \quad \quad \quad 60
 \end{array}$$

intermediate code generator

temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3

code optimizer

temp1 := id3 * 60.0
id1 := id2 + temp1

code generator

MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1

SYMBOL TABLE

1	position	...
2	initial	...
3	rate	...
4		

Outline

Motivatie

Evolutia limbajelor de programare

Compilare - Interpretare. Fazele unui compilator

Limbaje ezoterice

Esolang - esoteric programming language

Brainfuck - 8 caractere

```
+++++++ [ >+++++>+++++++>+++<<<- ] >+ . >+ . ++++++  
..+++ .>+ . <<+++++++ .> .+++ .----- .----- .>+ .
```

Listing 2: "Hello world"

LOLCODE

```
HAI 1.2
CAN HAS STDIO?
PLZ OPEN FILE "LOLCATS.TXT"?
    AWSUM THX
        VISIBLE FILE
    O NOES
        INVISIBLE "ERROR!"
KTHXBYE
```

LOLCODE

```
HAI 1.2
BTW Greetz a friend
I HAS A animal
GIMMEH animal
BOTH SAEM animal AN "cat"
O RLY?
    YA RLY
        VISIBLE "Hello cat"
        VISIBLE "Nice to meet you"
    MEBBE BOTH SAEM animal AN "mouse"
        VISIBLE "Hello mouse"
        VISIBLE "Nice to eat you"
    NO WAI
        VISIBLE "Hello stranger"
OIC
KTHXBYE
```

Exemple de limbaje

- ▶ Geometric figure drawing
- ▶ Music manipulation
- ▶ Table manipulation
- ▶ Finance language
- ▶ A graph language
- ▶ text-based adventure game

<http://www.cs.columbia.edu/~sedwards/classes.html>

`http://www.99-bottles-of-beer.net/toplist.html`

- ▶ See the toplist
- ▶ brainfuck code, dna#,
- ▶ lisp, haskell, python, c++

Semantic - observatii

- ▶ Ceva poate fi sintactic corect, dar fara sens

The rock jumped thorough the hairy planet

- ▶ Sau ambiguu

The chickens are ready to eat.

- ▶ Instrumente pentru reprezentare
 - ▶ Siruri de rescriere
 - ▶ Gramatici - ierarhia lui Chomsky
 - ▶ Derivari si arbori de derivare
- ▶ Gramatici regulate si automate finite
 - ▶ Automate finite
 - ▶ Diagrame de stare si expresii regulate
- ▶ Gramatici independente de context si automate stiva:
Automate stiva
- ▶ Analiza sintactica descendenta:
 - ▶ $LL(k)$
 - ▶ eliminare recursivitate stanga
 - ▶ Factorizare stanga
 - ▶ gramatici $LL(k)$ tari
 - ▶ **Derivator $LL(1)$** - segmente de program
- ▶ Analiza sintactica ascendenta
 - ▶ $LR(k)$
 - ▶ **Derivator $LR(0)$** - functia de tranzitie
 - ▶ $SLR(1)$
- ▶ Elemente de Procesare a limbajului natural - NLP

Rezumat

Motivatie

Evolutia limbajelor de programare

Compilare - Interpretare. Fazele unui compilator

Limbaje ezoterice

Exemplu de proiect ani anteriori

Limbaj propriu pentru desenare (autor Timotei Molcut)

Link actualizat pt 99 bottles

Fun facts