|  |
| --- |
| Universitatea „Ștefan cel Mare” Suceava |
| Toy – limbaj de programare |
| Proiect laborator „Proiectarea compilatoarelor I” |

|  |
| --- |
| Alexandru Dascălu  1/21/2016 |

Cuprins

[Arhitectură proiect 2](#_Toc441236694)

[Structură director 2](#_Toc441236695)

[Creare 2](#_Toc441236696)

[Interpretor 3](#_Toc441236697)

[Structură limbaj 4](#_Toc441236698)

[Tipuri de date: 4](#_Toc441236699)

[Afișare ecran: 4](#_Toc441236700)

[Instrucțiuni: 4](#_Toc441236701)

[Operatori matematici 5](#_Toc441236702)

[Operatori relaționali 5](#_Toc441236703)

[Funcții aritmetice 5](#_Toc441236704)

[Funcții trigonometrice 6](#_Toc441236705)

[Funcții proprii (conversii distante): 6](#_Toc441236706)

[Teste 7](#_Toc441236707)

[Utilizarea funcții trigonometrice și calcularea ariei unui cerc. 7](#_Toc441236708)

[Utilizarea structurii for și funcții matematice 8](#_Toc441236709)

[Bibliografie și resurse 9](#_Toc441236710)

# Arhitectură proiect

„Toy” este un interpretor al limbajului de programare cu același nume. După cum sugerează și numele, „toy” este un limbaj de programare de test. Aplicația care interpretează codul a fost generată cu ajutorul aplicațiilor windows **win\_flex** și **win\_bison**.

Codul scris este interpetat de aplicație și este executată o secvență de cod C corespunzătoare. De asemenea, se obține și codul echivalent în python.

## Structură director

* 5\_ex\_corecte: conține 5 exemple de cod corect
* 5\_ex\_gresite: conține 5 exemple de cod greșit
* src: contine codul sursă
* documentatie – documentul curent
* readme.md – indicații scurte
* ruleaza\_teste\_ok.bat – fișier batch – execută testele corecte și crează fișierele aferente de cod python
* ruleaza\_teste\_not\_ok.bat – fișier batch – execută testele gresite

## Creare

toy.exe

GCC

## Interpretor

Modalități de test:

1. In linia de comandă: **toy.exe**. Se deschide interpretorul toy și fiecare linie scrisă va fi executată pe loc. Toate instrucțiunile corecte vor fi translatate în python în fișierul cod\_python.py.
2. In linia de comanda: **toy.exe nume \_sursă**. Se va compila și executa fișierul sursă linie cu linie, de la dreapta la stânga. Codul python va fi scris în fișierul cod\_python.py
3. In linia de comanda: **toy.exe nume\_sursă nume\_output.** Se va compila și executa fișierul sursă linie cu linie, de la dreapta la stânga. Codul python va fi scris în fișierul nume\_output.py.

Procesul este relativ simplu: executabilul toy.exe generat de compilatorul GCC, poate să primească input de text de la tastatură sau dintr-un fișier text. Textul va fi împărțit în lexeme și în funcție de fiecare simbol se va scrie în fișierul de output codul echivalent în python și se va executa o anumită secvență de cod asociată expresiei respective.

# Structură limbaj

Limbajul „toy” (după cum sugerează și numele) este un limbaj de programare formal. Nu este foarte solid și are un număr limitat de funcționalități.

Toy 🡪 Python

|  |  |
| --- | --- |
| Toy | PYTHON |
| stop | exit() |

## Tipuri de date:

|  |  |
| --- | --- |
| Toy | PYTHON |
| var (întreg, real, natural) |  |

## Afișare ecran:

|  |  |
| --- | --- |
| Toy | PYTHON |
| -> „text” | print „text” |

## Instrucțiuni:

|  |  |  |
| --- | --- | --- |
| Instrucțiune | Toy | PYTHON |
| Instrucțiunea de atribuire | nume\_variabila=valoare  EX: x=2 | Nume\_var = valoare;  EX: a=1; |
| Instructiunea IF | **?** expresie instructiune 1 **??** instructiune 2  EX: ? x< 2 ->”Da” | If expresie\_booleana:  Instructiune1  EX: if x < 2:  Print „Da” |
| Instrucțiunea IF - ELSE | **?** expresie instructiune 1 **??** instructiune 2  EX: ? x< 2 ->”Da” ?? -> „Nu” | If expresie\_booleana:  Instructiune1  else:  Instructiune2  EX: if x < 2:  Print „Da”  else:  Print „Nu” |
| Instrucțiunea FOR | **de (**expr **ori):** instructiune  EX: var i  **de (5 ori): -> i**  **executa** | **For** index **in range(expre):**  instructiune  EX**:**  **for i in range (5):**  **print i;** |
| Instrucțiunea WHILE | **cat\_timp** conditie **executa**  instructiuni | **while** conditie:  instructiuni |

## Operatori matematici

|  |  |
| --- | --- |
| Toy | PYTHON |
| + | + |
| - | - |
| \* | \* |
| / | / |
| %, mod | % |

## Operatori relaționali

|  |  |
| --- | --- |
| Toy | PYTHON |
| > | > |
| < | < |
| == | == |

## Funcții aritmetice

|  |  |
| --- | --- |
| Toy | PYTHON |
| a putere n, a^n | math.pow (a,n) |
| rad a | math.sqrt (a) |
| Log2(a) | Math.log (a, 2) |
| Log10(a) | Math.log(a, 10) |
| |sus (a) –rotunjire adaugare | Math.ceil(a) |
| |jos (a) – rotunjire scadere | Math.floor(a) |
| !(a) – factorial | Math.factorial(a) |
| modul (a) | Math.abs (a) |

## Funcții trigonometrice

|  |  |
| --- | --- |
| Toy | PYTHON |
| Sin (a) | math.sin(a) |
| Cos (a) | math.cos(a) |
| Tan(a) | Math.tan(a) |
| Sinh(a) | Math.sinh(a) |
| Cosh(a) | Math.cosh(a) |
| Tanh(a) | Math.tanh(a) |

## Funcții proprii (conversii distante):

|  |  |
| --- | --- |
| Toy (fișierul func.h) | PYTHON (fișierul func.py) |
| m\_>>\_km(a) | func.m\_la\_km(a) |
| km\_>>\_m(a) | func.km\_la\_m(a) |

# Teste

Directoarele 5\_ex\_corecte și 5\_ex\_greșite conțin codul sursă pentru test. Se pot executa automat folosind batch-urile corespunzătoare:

* **ruleaza\_teste\_ok.bat**
* **ruleaza\_teste\_not\_ok.bat**

## Utilizarea funcții trigonometrice și calcularea ariei unui cerc.

|  |  |
| --- | --- |
| Toy | PYTHON |
| var unghi = 1  -> sin(unghi)  -> cos (PI)  var const = PI  -> const  var razaCerc = 4  -> "Raza"  -> razaCerc  var aria = PI \* razaCerc ^ 2  -> "Aria"  -> aria  aria = PI \* razaCerc putere 2  -> "Aria2"  -> aria  stop | #!/usr/bin/python  import math  from func import m\_la\_km  from func import km\_la\_m  unghi = 1.00  print math.sin ( unghi )  print math.cos ( math.pi )  const = math.pi  print const  razaCerc = 4.00  print "Raza"  print razaCerc  aria = math.pi \* razaCercmath.pow2.00  print "Aria"  print aria  aria = math.pi \* razaCercmath.pow2.00  print "Aria2"  print aria  exit() |

## Utilizarea structurii for și funcții matematice

|  |  |
| --- | --- |
| Toy | PYTHON |
| var nrNeg = -1  -> "Numar\_negativ"  -> nrNeg  -> "Modul"  var nrPositiv = modul(nrNeg)  -> nrPositiv  de (nrPositiv ori): nrPositiv = nrPositiv + 1  -> "Instructiune\_for"  -> nrPositiv  var factorial = !3  -> "Factorial"  -> factorials  -> "Factorial2"  -> !factorial  stop | #!/usr/bin/python  import math  from func import m\_la\_km  from func import km\_la\_m  nrNeg = - 1.00  print "Numar\_negativ"  print nrNeg  print "Modul"  nrPositiv = abs ( nrNeg )  print nrPositiv  **for idx in range ( nrPositiv ) : nrPositiv = nrPositiv + 1.00**  print "Instructiune\_for"  print nrPositiv  factorial = math.factorial3.00  print "Factorial"  print factorial  print "Factorial2"  print math.factorialfactorial  exit() |

# Bibliografie și resurse

* <https://www.gnu.org/software/bison/>
* <http://gnuu.org/2009/09/18/writing-your-own-toy-compiler/>
* <http://stackoverflow.com/questions/19200808/parsing-a-while-loop-in-bison>
* <https://github.com/Package/Bison-Flex-Calculator>
* <http://stackoverflow.com/questions/25709334/output-ast-from-flexbison-to-main-cpp>