**P175B124 Programavimo kalbų teorija**

Savarankiškas darbas

„Dogescript“

Darbą atliko:

Audrius Frankonis IFF-5/3

Darbą priėmė:

Lekt. Evaldas Guogis

Turinys

[1. Darbo užduotis 3](#_Toc514772181)

[2. Esminės kalbos savybės 3](#_Toc514772182)

[3. Kalbos kūrimo priemonės 3](#_Toc514772183)

[4. Galutinė kalbos gramatika 3](#_Toc514772184)

[5. Apribojimai 6](#_Toc514772185)

[6. Naudojimosi instrukcijos 6](#_Toc514772186)

[7. Pavyzdinis kodas ir jo vykdymo rezultatai 6](#_Toc514772187)

[8. Išvados 9](#_Toc514772188)

[9. Literatūra 9](#_Toc514772189)

# Darbo užduotis

Sukurti pilnai veikiančią programavimo kalbą, atitinkančią šiuos reikalavimus:

1. Sąlygų, ciklo, kintamųjų, funkcijų palaikymas
2. Sisteminių funkcijų kvietimas
3. Bent du duomenų tipai: skaičius, eilutė
4. Bent viena unikali savybė.

# Esminės kalbos savybės

Palaikomos valdymo struktūros: if ir if/else, for ciklas, while/do while ciklai.

Naudojami duomenų tipai: sveikasis skaičius, dešimtainis skaičius, eilutė. Loginiai kintamieji realizuojami sveikaisiais skaičiais – 0 atitinka neigiamą reikšmę, kiti skaičiai – teigiamą.

Kalbos sintaksė paremta C, su papildomais operatoriais: \*\* (realizuotas naudojant atitinkamą Python operatorių) ir <=> (reikšmių sukeitimo operatorius.)

Kalba yra interpretuojama – tai leidžia lankstumą dirbant su tipais ir funkcijų apibrėžimais, bet apsunkina išankstinį problemų aptikimą.

# Kalbos kūrimo priemonės

Kalbos interpretatorius yra realizuojamas Python kalboje, naudojant IDLE programavimo aplinką. Gramatika sukurta ir testuota naudojant GOLD Parser, pagal kūrėjų svetainėje pateiktą C 1976 gramatikos pavyzdį.

# Galutinė kalbos gramatika

"Name" = DOGESCRIPT

"Case Sensitive" = True

"Start Symbol" = <declarations>

!Type declarations

{idHead} = {letter} + [\_]

{idTail} = {idHead} + {digit}

id = {idHead}+{idTail}\*

intLiteral = [ +-]{digit}+

decLiteral = [ +-]{digit}\*(([.]{digit}+[d]?)|[d])

{String Ch} = {Printable} - ["]

{Char Ch} = {Printable} - ['']

stringLiteral = '"'( {String Ch} | '\'{Printable} )\* '"'

charLiteral = '' ( {Char Ch} | '\'{Printable} )''

!Declaration

!-----------

<declarations> ::= <declaration><declarations>

|

<declaration> ::= <functionDeclaration>

|<functionPrototype>

|<variableDeclaration>

!Function declaration

!--------------------

<functionPrototype>::=<functionID>'('<typeList>')'';'

|<functionID>'('<params>')'';'

|<functionID>'('')'';'

<functionDeclaration>::=<functionID>'('<params>')'<block>

|<functionID>'('<idList>')'<block>

|<functionID>'('')'<block>

<params>::=<params>','<param>

|<param>

<param>::=!<type> <array> id

|<type><id>

<typeList>::=<type>','<typeList>

|<type>

<idList>::=id','<idList>

|id

<functionID>::=<type>id

|id

!Variable declaration

!--------------------

<variableDeclaration>::=<type><var>';'

<var> ::=id

|id '='<opIf>

<array> ::= '[' <expression> ']'

| '[' ']'

|

!Types

!-----

<type>::=string

|int

|dec

!Statements

!----------

<statement>::=<variableDeclaration>

|if'('<expression>')'<statement>

|if'('<expression>')'<thenStatement>else<statement>

|while'('<expression>')'<statement>

|for'('<argument>';'<argument>';'<argument>')'<statement>

|<normalStatement>

<thenStatement>::=if'('<expression>')'<thenStatement>else<thenStatement>

|while'('<expression>')'<thenStatement>

|for'('<argument>';'<argument>';'<argument>')'<thenStatement>

|<normalStatement>

<normalStatement>::=do<statement>while'('<expression>')'';'

|<block>

|<expression>';'

|return<expression>';'

|';'

<argument> ::=<expression>

|

<block>::='{'<statementList>'}'

<statementList>::=<statement><statementList>

|

!Expressions

!-----------

<expression> ::=<expression>','<opSwap>

|<opSwap>

<opSwap>::=<opSwap>'<=>'<opAssign>

|<opAssign>

<opAssign>::=<opIf>'='<opAssign>

|<opIf>

<opIf> ::= <opOr>'?'<opIf>':'<opIf>

|<opOr>

<opOr>::=<opOr>'|'<opXor>

|<opXor>

<opXor>::=<opXor>'^'<opAnd>

|<opAnd>

<opAnd>::=<opAnd>'&'<opEq>

|<opEq>

<opEq>::=<opEq>'=='<opCmp>

|<opEq>'!='<opCmp>

|<opCmp>

<opCmp>::=<opCmp>'<'<opAdd>

|<opCmp>'>'<opAdd>

|<opCmp>'<>'<opAdd>

|<opCmp>'<='<opAdd>

|<opCmp>'>='<opAdd>

|<opAdd>

<opAdd>::=<opAdd>'+'<opMul>

|<opAdd>'-'<opMul>

|<opMul>

<opMul>::=<opMul>'\*'<opExp>

|<opMul>'/'<opExp>

|<opMul>'%'<opExp>

|<opExp>

<opExp>::=<opExp>'\*\*'<opUnary>

|<opUnary>

<opUnary>::= '!'<opUnary>

|'++'<opUnary>

|'--'<opUnary>

|<opFetch>'--'

|<opFetch>'++'

|<opFetch>

<opFetch>::=<value> !Čia buvo planuojama realizuoti masyvus - palikta, nes šalinimo metu galėjo atsirasti klaidų, kurių testavimas

<value>::=intLiteral

|decLiteral

|stringLiteral

|id'('<expression>')'

|id'('')'

|id

|'('<expression>')'

# Apribojimai

Interpretatorius yra sukurtas Python kalba, taigi veikia tik sistemose, kuriose įdiegta Python. Kūrimui naudota Python versija – 3.6.4, atgalinis suderinamumas nėra garantuojamas.

Kalbos kūrimo pradžioje buvo planuota galimybė kurti ir naudoti masyvus, bet galutinėje kalbos versijoje jie nėra realizuoti.

# Naudojimosi instrukcijos

Python paleisti dogescript.py failą, kaip parametrą pateikiant vykdytiną failą (numatytasis failo plėtinys - .doge, bet plėtinys nėra tikrinamas – veikia ir su kitų plėtinių failais, kurių turinys – tekstas.)

Failai ir jų paskirtys:

dogescript.py – Pagrindinis programos modulis.

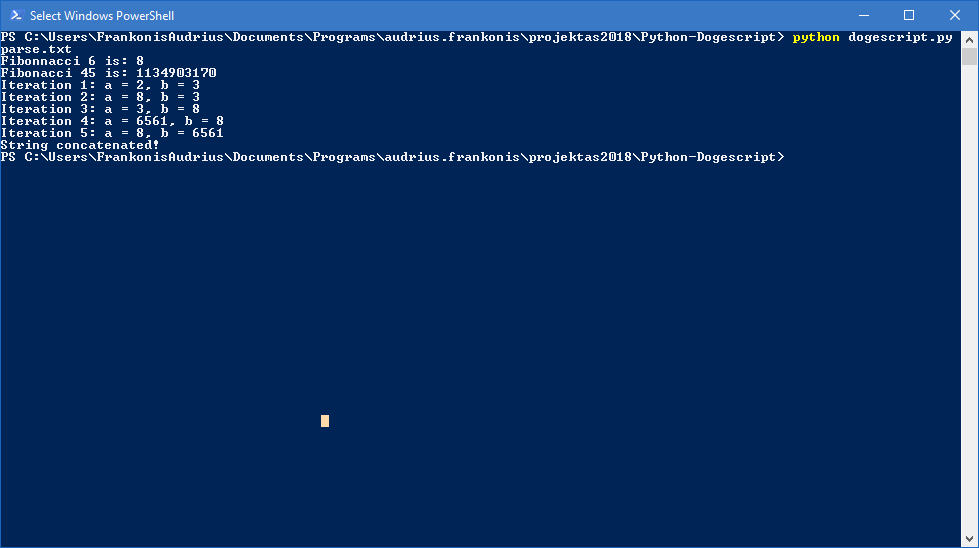
enums.py – Enum tipo kintamųjų, naudojamų keliuose failuose, saugykla.

hannibal.py – Lekserio modulis.

peter.py – Parserio modulis.

stdfunc.py – Standartiniai metodai, kurie reikalingi visoms programoms (rašymas į konsolę, failą, sistemos funkcijos kvietimas.) Įkeliamas prieš kodo interpretavimą, taigi visos funkcijos gali būti perrašomos¡

Naudojimo pavyzdys, kai ir interpretatorius, ir interpretuojamas failas yra darbiniame aplanke:



pav. 1 Kodo vykdymo pavyzdyz

# Pavyzdinis kodas ir jo vykdymo rezultatai

Failas: parse.doge

int a = 1;

int fibRec(int i){

if(i <= 0){

return 0;

}

else if (i <= 2){

return 1;

} else {

return fibRec(i - 1) + fibRec(i-2);

}

}

int fibLoop(int i){

if (i <= 0){

return 0;

} else if (i <= 2){

return 1;

} else {

int prev = 1;

int next = 1;

int j;

for(j = 3; j <=i; j++){

int tmp = prev + next;

prev = next;

next = tmp;

}

return next;

}

}

int reverse(int i){

int rez = 0;

while (i > 0){

rez = rez \* 10 + i%10;

i = i/10;

}

return rez;

}

int main(){

printConsole("Fibonnacci 6 is: " + fibRec(6));

printConsole("Fibonacci 45 is: " + fibLoop(45));

int a = 2;

int b = 3;

int i = 0;

while (a < 10000){

i++;

printConsole("Iteration " + i + ": a = " + a + ", b = " + b);

if (a > b){

a <=> b;

} else {

a = a\*\*b;

}

}

printConsole("String " + "concatenated!");

printFile("File printing again!!!!", "test.txt");

appendFile("Appending too! >>test.txt && echo \"HACKS!!!\"> hacks.txt ", "test.txt");

appendFile("Newlines working!", "test.txt");

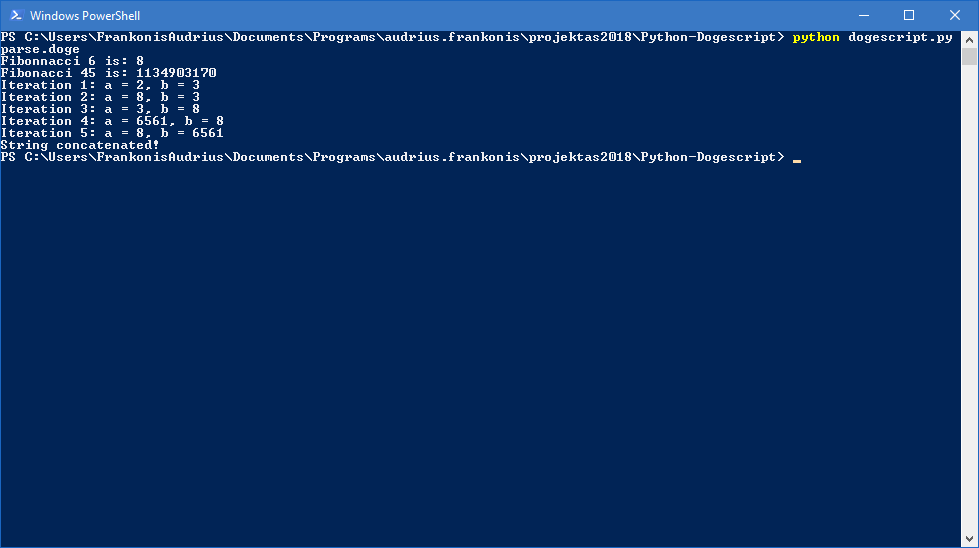
int number = 123456789;

int rev = reverse(number);

appendFile("Reverse of " + rev + " is " + rev, "test.txt");

}

Vykdymas ir rezultatai konsolėje:



pav. Vykdymo rezultatai

Rezultatai (text.txt):

File printing again!!!!

Appending too! >>test.txt && echo "HACKS!!!"> hacks.txt

Newlines working!

Reverse of 987654321 is 987654321

# Išvados

Sukurta sėkmingai interpretuojama C stiliaus kalba. Išmoktas lekserio, parserio kūrimas, išorinių komandų kvietimas. Nerealizuotas masyvų palaikymas, nepalaikomas funkcijų užklojimas.

# Literatūra

1. Paskaitų medžiaga „Moodle“
2. <http://goldparser.org/doc/index.htm>
3. <http://goldparser.org/grammars/files/C-ANSI.zip>
4. <https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form>
5. <https://www.youtube.com/playlist?list=PLBOh8f9FoHHiKx3ZCPxOZWUtZswrj2zI0>