

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Programavimo kalbų teorija (P175B124)
Projektinis darbas – “Koridorius”

Atliko:

IFF-1/6 gr. studentai

Daugardas Lukšas

Domantas Bieliūnas

Lukas Kuzmickas

Priėmė:

doc. Sajavičius Svajūnas

lekt. Fyleris Tautvydas

KAUNAS 2023

TURINYS

1.	PIRMASIS KOMPILIATORIAUS KŪRIMO PROJEKTINIO (SAVARANKIŠKO) DARBO ATSISKAITYMAS	3
1.1	Kalbos idėja ir pavadinimas.	3
1.2	Kalbos savybės, palaikomos struktūros bei duomenų tipai (t.y. ataskaitoje pridėtas jūsų kalbos kodas).	3
1.3	Baziniai ir palaikomų kalbos konstrukcijų pavyzdžiai (pvz.: sąlygos sakiniai, ciklai, , kt.) (t.y. kodas kuriamoje kalboje).	3
1.4	Unikali savybė ir jos naudojimo kodo pavyzdys (jūsų programavimo kalboje parašytas pavyzdys).	4
1.5	Pasirinkti darbo įrankiai (ir pasirinkimo priežastys) bei darbui naudojama programavimo kalba.	6
1.6	Kodo pavyzdžiai.	6
2.	ANTRASIS KOMPILIATORIAUS/INTERPRETATORIAUS KŪRIMO PROJEKTINIO (SAVARANKIŠKO) DARBO ATSISKAITYMAS	7
2.1	Kuriamos kalbos ir komandos pavadinimas	7
2.2	Išvardinti pasirinkti naudojami įrankiai (jeigu įrankiai buvo pakeisti, tai dėl kokios priežasties)	7
2.3	Pavaizduotas įrankių naudojimas (kaip naudojant įrankius yra gaunamas galutinis kompiliatoriaus/interpretatoriaus exe/jar/etc.).	7
2.4	Pateikta visa/dalis kuriamos kalbos gramatikos ARBA naudojamų įrankių konfigūracinio failo pavyzdys (kuriame matyti visas/dalis kuriamos kalbos aprašymas)	8
3.	GALUTINIS (3 ETAPO) PROJEKTINIO DARBO ATSISKAITYMAS	11
3.1	Suformuota užduotis.	11
3.2	Esminės kalbos savybės (palaikomos struktūros, tipai, sintaksė ir t.t).	11
3.3	Kūrimo priemonės.	12
3.4	Apribojimai (veikia tik X aplinkoje, Y - nėra realizuota ir pan.).	13
3.5	Galutinė kalbos gramatika (ar įrankių konfigūracijos svarbiausios dalys).	13
3.6	Kaip sukompiliuoti ir paleisti?	16
3.7	Pavyzdinis kodas kalboje ir gauti rezultatai vykdant.	17
	ŠALTINIAI	24

1. Pirmasis kompiliatoriaus kūrimo projektinio (savarankiško) darbo atsiskaitymas

1.1 Kalbos idėja ir pavadinimas.

Kalbos idėja - programavimo kalba, kuri savo sintakse ir duomenų tipais yra panaši į C programavimų kalbų šeimą. Unikali kalbos savybė padeda kaupti buvusių funkcijų rezultatus. Mūsų kalbos pavadinimas - „Funcitive”.

1.2 Kalbos savybės, palaikomos struktūros bei duomenų tipai (t.y. ataskaitoje pridėtas jūsų kalbos kodas).

Kalbos savybės:

- Sintaksė.
- Duomenų tipai.

Palaikomos struktūros:

Masyvas – daugelis vienodo tipo duomenų.

Duomenų tipai (kiekvienas jų gali būti masyvas):

Int – sveikieji skaičiai

Float – realieji skaičiai

Char - simboliai

1.3 Baziniai ir palaikomų kalbos konstrukcijų pavyzdžiai (pvz.: sąlygos sakiniai, ciklai, , kt.) (t.y. kodas kuriamoje kalboje).

Palaikomos bazinės kalbos konstrukcijos:

Kintamųjų priskyrimas

```
Int i = 12;  
Char c = 'a';  
Float f = 12.69;
```

Masyvai

```
Int [] Integers = { 1, 2, 3 };
```

```
Integers[0]; // grąžintų 1
```

Sąlygos sakiniai (if,else,switch)

```
If ( ) { }  
else { }  
Switch ( ){  
    Case :  
    break;  
    Case:  
    break;  
    Default:  
}
```

Ciklai(for,while)

```
for ( ; ; ) {  
}  
  
while( ){  
}
```

Funkcijos:

```
phoonk void foo () {  
    print("bar");  
}  
foo();
```

1.4 Unikali savybė ir jos naudojimo kodo pavyzdys (jūsų programavimo kalboje parašytas pavyzdys).

Automatinis funkcijos kintamasis

Tarkime turime funkciją aprašyta tokiu pavidalu:

```
phoonk int multiply(int a, int b){
```

```
    return a * b;
}
```

Mūsų kalbos unikalioji savybė padaro taip, kad kiekvieną kartą įvykdžius funkciją, jei funkcija nėra void tipo, galime gauti prieš tai grąžintų funkcijų rezultatus betkurioje kodo vietoje, globaliųjų kintamųjų pagalba. Tereikia iškvieisti kintamąjį su tokiu pačiu pavadinimu kaip funkcija.

Pavyzdžiui, turint omeny praeitą pavyzdį:

```
multiply(2, 2);
print(multiply[0]);
```

Ir įvykdžius šią programinio kodo dalį gautume rezultatą lygu 4.

Svarbu paminėti, jog rezultatai saugomi masyvo principu, tad norint pasiekti paskutiniausią elementą, turime žinoti tuo metu esančio masyvo dydį/ilgį. Siekdami išspręsti šią problemą, pridėjome dar vieną savybę į mūsų kalbą. Norint gauti n -ąjį elementą nuo galo, tereikia parašyti į indekso vietą neigiamą šio skaičiaus reikšmę. Pavyzdžiui, norint pasiekti paskutinį elementą:

```
multiply[-0]
```

Taip pat norėjome šiek tiek pakeisti kaip mūsų kalba priims masyvų režius, užtikrindami, kad nebūtų galima iš jų išlipti. Jeigu turime masyvą su n elementų, ir prašome programos grąžinti $n + 1$ elementą, tai atliekame operaciją $(n + 1) \% n$ ir gauname elemento indeksą, kurį programa grąžins. Pavyzdžiui:

```
int [] arr = {1, 2}; // nustatome, kad masyvas turėtų du elementus
arr[5]; // programa grąžins - 2
```

Turime pastebėti, kad šiek tiek sudėtingesnis atvejis atsiranda kuomet įvedame per didelį neigiamą skaičių. Tarkim įvedame tokį programinį kodą:

```
int [] arr = {1, 2};
arr[-5];
```

Lygiai taip pat paskaičiuojam liekaną su masyvo ilgiu, šiuo atveju gautume –1, o gražinę antrą elementą nuo pabaigos, gautume 1.

1.5 Pasirinkti darbo įrankiai (ir pasirinkimo priežastys) bei darbui naudojama programavimo kalba.

Slack – platforma skirta efektyviai komunikacijai.

Jira – lengva darbo dokumentacija, organizacinis aspektas bei efektyvesnis darbo pasiskirstymas ir komunikacija.

Bitbucket – kodo dokumentacija ir nuoseklesnis darbo užtikrinimas.

Java - naudotume kaip programavimo kalbą pagrindui, ant jos būtų statomas kompiliatorius.

1.6 Kodo pavyzdžiai.

```
// Paprasčiausios funkcijos pavyzdys
phoonk void foobar (int x) {
    if (x % 3 == 0 && x % 5 == 0)
        print "FooBar";
    else if (x % 5 == 0)
        print "Foo";
    else if (x % 3 == 0)
        print "Bar";
    else
        print "NOTFOOBAR";
}

// Funkcija, kuri parodo mūsų unikaliąją savybę
phoonk int power(int base, int exponent) {
    int result = base;
    for (int i = 1; i < exponent; i++)
    {
        result = result * result;
    }

    return result;
}

// Pavyzdys, kaip galima iškviešti funkciją
foobar(97);

// Unikalių savybės pavyzdžiai
// Komentaruose parodome numatomą išvestį ar funkcijos rezultata
int x0 = power(2, 4); // 16
int x1 = power;      // 16
int x2 = power[-0];  // 16
int x3 = power[0];   // 16

power(2, 5); // 32
int x4 = power[-0]; // 32
int x5 = power[0];  // 16
int x6 = power[-1]; // 16

power(2, 6); // 64
int x7 = power[-0]; // 64
int x8 = power[-1]; // 32
```

```

int x9 = power[-2];    // 16
int x10 = power[-3];   // 64
int x11 = power[-4];   // 32

int x12 = power[0];    // 16
int x13 = power[1];    // 32
int x14 = power[2];    // 64
int x15 = power[3];    // 16
int x16 = power[4];    // 32

```

2. Antrasis kompiliatoriaus/interpretatoriaus kūrimo projektinio (savarankiško) darbo atsiskaitymas

2.1 Kuriamos kalbos ir komandos pavadinimas

Mūsų kalbos pavadinimas - „Functive”. Komandos pavadinimas – Koridorius.

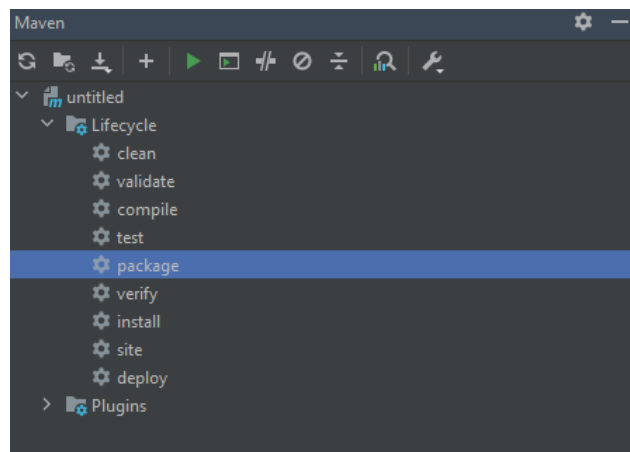
2.2 Išvardinti pasirinkti naudojami įrankiai (jeigu įrankiai buvo pakeisti, tai dėl kokios priežasties)

Kompiliatorius kūrimui naudosime ANTLR4. Keičiame programavimo kalbą į Java, kadangi daugiau medžiagos, kaip naudotis ANTLR4.

2.3 Pavaizduotas įrankių naudojimas (kaip naudojant įrankius yra gaunamas galutinis kompiliatoriaus/interpretatoriaus exe/jar/etc.).

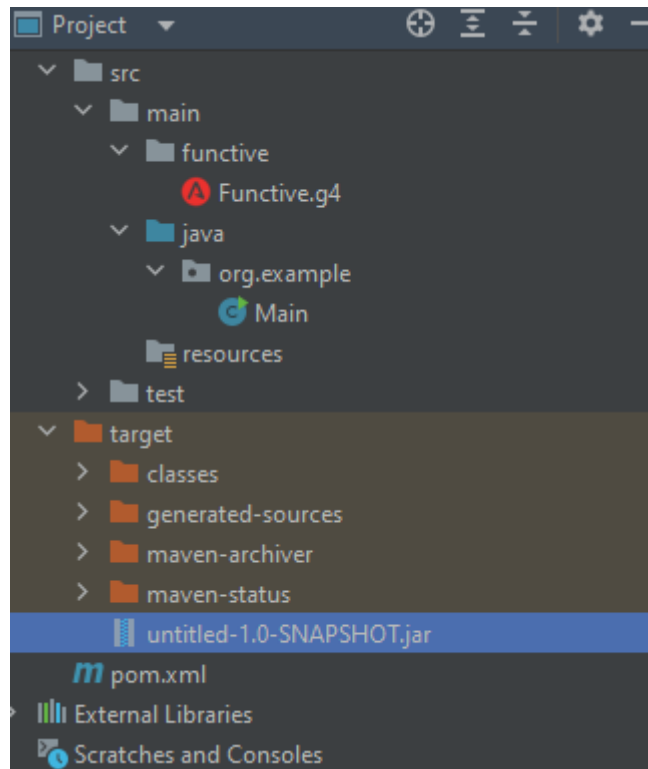
Kuriame kompiliatorių, todėl pavaizduosime, kaip sukursime galutinį failą. Kadangi dirbame ir naudojame kompiliatoriaus kūrimui Java, todėl mums reikia sugeneruoti .jar failą.

Naudodami Maven, galime sukurti savo kodui jar failą ir per konsolinę sąsają (pvz. Gitbash arba cmd). Pasirenkame savo projekta -> Lifecycle -> package (du kartus paspaudžiame).



2 pav. Maven naudojimas

Po viso šito laukiame, kol bus sukurtas jar failas mūsų kodui. Jis dažniausiai kuriamas target/ direktorijoje.



3 pav. Jar generavimas

Galime savo sukompiliuotą kodą paleisti naudodami komandą (java -cp ##.jar (package)). Taip gauname savo kodą.

2.4 Pateikta visa/dalis kuriamos kalbos gramatikos ARBA naudojamų įrankių konfigūracinio failo pavyzdys (kuriame matyti visas/dalis kuriamos kalbos aprašymas)

Funcative.g4

```
grammar Funcative;

// Main rule
program: statement* EOF;

// Statement
statement
: varDeclaration ';'
| arrayDeclaration ';'
| assignment ';'
| ifStatement
| switchStatement
```



```

| forLoop
| whileLoop
| functionDeclaration
| functionCall ';'
| print ';'
| returnStatement ';'
;

// Variable Declaration
varDeclaration: (TYPE | 'String') IDENTIFIER '=' expression;

// Array Declaration
arrayDeclaration: TYPE ('[' ']') IDENTIFIER '=' '{' expressionList '}';

// Assignment
assignment: IDENTIFIER '=' expression;

// If statement
ifStatement: 'if' '(' expression ')' '{' statement* '}' ('else' '{' statement* '}')?;

// Switch statement
switchStatement: 'switch' '(' expression ')' '{' caseStatement* defaultStatement? '}';

caseStatement: 'case' expression ':' statement*;

defaultStatement: 'default' ':' statement*;

// For loop
forLoop: 'for' '(' (assignment | varDeclaration) ';' expression ';' expression ')' '{'
statement* '}';

// While loop
whileLoop: 'while' '(' expression ')' '{' statement* '}';

// Function Declaration
functionDeclaration: 'phoonk' (TYPE | 'void') IDENTIFIER '(' (TYPE IDENTIFIER (',' TYPE
IDENTIFIER)*)? ')' '{' statement* '}';

// Function Call
functionCall: IDENTIFIER '(' expressionList? ')';

// Print
print: 'print' '(' (expression | STRING_LITERAL) ')';

// Return statement
returnStatement: 'return' expression?;

// Expression
expression
: '(' expression ')'
| '!' expression
| '-' expression
| expression ('*' | '/' | '%' | '+' | '-' | '<' | '<=' | '>' | '>=' | '=' | '!=' | '&&' | '||') expression
| INT_LITERAL
| FLOAT_LITERAL
| CHAR_LITERAL
| IDENTIFIER ('[' expression ']')?
| functionCall

```

```

;

expressionList: expression (',' expression)*;

// Tokens
TYPE: ('Int' | 'Float' | 'Char');
IDENTIFIER: [a-zA-Z_] [a-zA-Z_0-9]*;
INT_LITERAL: [0-9]+;
FLOAT_LITERAL: [0-9]+ '.' [0-9]+;
CHAR_LITERAL: '\'' . '\'';
STRING_LITERAL: '"' .*? '"';
LINE_COMMENT: '//' ~[\r\n]* -> skip;
BLOCK_COMMENT: '/*' .*? '*/' -> skip;
WS: [ \t\r\n]+ -> skip;

```

3. Galutinis (3 etapo) projektinio darbo atsiskaitymas

3.1 Suformuota užduotis.

Kalbos idėja - programavimo kalba, kuri savo sintakse ir duomenų tipais yra panaši į C programavimų kalbų šeimą. Unikali kalbos savybė padeda kaupti buvusių funkcijų rezultatus – masyvo indeksas gali būti neigiamas. Mūsų kalbos pavadinimas - „Functive”.

3.2 Esminės kalbos savybės (palaikomos struktūros, tipai, sintaksė ir t.t).

Kalbos savybės:

- Sintaksė.
- Duomenų tipai.
- Neigiamas masyvo indeksas.

Palaikomos struktūros:

Masyvas – daugelis vienodo tipo duomenų.

Duomenų tipai (kiekvienas jų gali būti masyvas):

Int – sveikieji skaičiai

Float – realieji skaičiai

String – simboliai

Palaikomos bazinės kalbos konstrukcijos:

Kintamųjų priskyrimas

```
Int i = 12;  
String ab = "adssadasd";  
Float f = 12.69;
```

Masyvai

```
Int [] Integers = [1, 2, 3];
```

```
Integers[0]; // grąžintų 1
```

Sąlygos sakiniai (if,else,switch)

```
If ( ) { }  
else { }  
Switch ( ){  
    Case :  
    break;  
    Case:  
    break;  
    Default:  
}
```

Ciklai(for,while)

```
for ( ; ; ) {  
}  
  
while( ){  
}
```

Funkcijos:

```
phoonk void foo () {  
    print("bar");  
}  
  
foo();
```

3.3 Kūrimo priemonės.

Kompiliatorius kūrimui naudosime ANTLR4. Naudojame JAVA programavimo kalbą ir deriname viską su ANTLR4 plugin. Repozitorijai naudojome Bitbucket, “version control” Git.

3.4 Apribojimai (veikia tik X aplinkoje, Y - nėra realizuota ir pan.).

Viskas programavimo kalboje yra realizuota, neturime apribojimų.

3.5 Galutinė kalbos gramatika (ar įrankių konfigūracijos svarbiausios dalys).

grammar funcative;

// Main rule

program: statement* EOF;

// Statement

statement:

- varDeclaration ';'
| arrayDeclaration ';'
| arrayAssignment ';'
| arrayAccessAssignment ';'
| assignment ';'
| ifElseIfElseStatement
| switchStatement
| forLoop
| whileLoop
| functionDeclaration
| functionCall ';'
| print ';'
| returnStatement ';'
| breakStatement ';;'

// Variable Declaration

varDeclaration: (TYPE | 'String') IDENTIFIER ('=' expression)?;

// Array Declaration

arrayDeclaration:

TYPE ('[' ']') IDENTIFIER ('=' '[' expressionList ']')?;

// Assignment

```

arrayAccessAssignment: IDENTIFIER arrayAccess '=' expression;
arrayAccess: '[' expression ']';
arrayAssignment: IDENTIFIER '=' '[' expressionList ']';
assignment: IDENTIFIER '=' expression;

// arrayAccess is optional, because it means that we are specifying the index of the array

// If statement
ifElseIfElseStatement:
    ifStatement elseifStatement* elseStatement?;
ifStatement: 'if' '(' expression ')' block;
elseifStatement: 'else' 'if' '(' expression ')' block;
elseStatement: 'else' block;

// Switch statement
switchStatement:
    'switch' '(' expression ')' '{' caseStatement* defaultStatement? '}';

caseStatement: 'case' expression ':' statement*;
breakStatement: 'break';

defaultStatement: 'default' ':' statement*;

// For loop
forLoop: 'for' '(' forControl ')' block;
forControl: (varDeclaration | assignment) ';' expression ';' assignment;

// While loop
whileLoop: 'while' '(' expression ')' block;

// Function Declaration
functionDeclaration:
    'phoonk' (TYPE | 'void') IDENTIFIER '(' parameters? ')' block;
parameters: parameter (',' parameter)*;
parameter: TYPE IDENTIFIER;

// Function Call
functionCall: IDENTIFIER '(' arguments? ')';

```

```

arguments: expression (',' expression)*;

// Block
block: '{' statement* '}';

// Print
print: 'print' expression;

// Return statement
returnStatement: 'return' expression?;

// Expression
expression:
    expression '+' expression          # addExpression
  | expression '-' expression          # subtractExpression
  | expression '*' expression          # multiplyExpression
  | expression '/' expression          # divideExpression
  | expression '%' expression          # modulusExpression
  | expression '<' expression           # lessThanExpression
  | expression '>' expression           # greaterThanExpression
  | expression '<=' expression          # lessThanEqualExpression
  | expression '>=' expression          # greaterThanEqualExpression
  | expression '==' expression         # equalExpression
  | expression '!=' expression         # notEqualExpression
  | expression '&&' expression          # andExpression
  | expression '||' expression         # orExpression
  | expression '[' expression ']'      # arrayAccessExpression
  || expression '.' IDENTIFIER         # objectAccessExpression // we don't have objects
  | '(' expression ')'                 # parenthesisExpression
  | literal                           # literalExpression
  | IDENTIFIER                         # identifierExpression
  | functionCall                       # functionCallExpression;

literal: INTEGER | FLOAT | STRING | BOOLEAN;

expressionList: expression (',' expression)*;

// Tokens

```

```

TYPE: 'int' | 'float' | 'boolean' | 'void' | 'string';
INTEGER: ('0' | [1-9][0-9]*);
FLOAT: [0-9]* '.' [0-9]+;
STRING: '"' ~[\r\n]"* '"';
BOOLEAN: 'true' | 'false';
IDENTIFIER: [a-zA-Z_] [a-zA-Z_0-9]*;
WS: [ \t\r\n]+ -> skip;

// Lexer rules
SINGLE_LINE_COMMENT: '/' ~[\r\n]* -> skip;

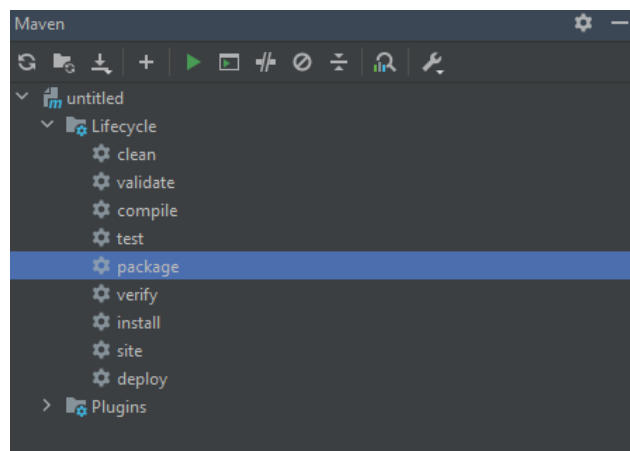
MULTI_LINE_COMMENT: '/*' .*? '*/' -> skip;

```

3.6 Kaip sukompiliuoti ir paleisti?

Naudojame Maven komandas – build package.

Sugeneruojame .jar failą ir jį galime executinti java -jar komandomis.



9 pav. Maven komandos.

Generuojama ANTLR4 su gramatika, naudodami komandą:

```
mvn clean antlr4:antlr4@build-parser
```

Kompiliatoriaus paleidimas (generuojamas executable failas):

```
mvn clean package
```

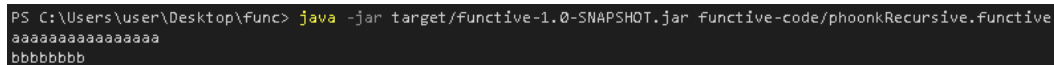
Kodo paleidimas per konsolinę sąsają (naudojamos reikalingos java bibliotekos):

```
java -jar target/functive-1.0.jar [directory-to-code-file]
```


3.7 Pavyzdinis kodas kalboje ir gauti rezultatai vykdam.

phoonkRecursive.functionive

```
phoonk string recursive(string a, int b){  
    if(b > 3){  
        return a;  
    }  
    b = b + 1;  
    return recursive(a + a, b);  
}  
print recursive("a", 0);  
print recursive("b", 1);
```



```
PS C:\Users\User\Desktop\func> java -jar target/functionive-1.0-SNAPSHOT.jar Functionive-code/phoonkRecursive.functionive  
aaaaaaaaaaaaaaaa  
bbbbbbbb
```

10 pav. phoonkRecursive.functionive

phoonkTest.functionive

```
int a = 1;  
int b = 2;  
  
print "original b: " + b;  
  
phoonk int add10(int c){  
    return c + 10;  
}  
  
b = add10(b);  
  
print "Should add 10 to b: " + b;  
  
phoonk void printHelloWorld(){  
    print "Hello World!";  
}
```

```

printHelloWorld();

phoonk string getFullName(string firstName, string lastName){
    return firstName + " " + lastName;
}

string fullName;
fullName = getFullName("Test", "Func");
print fullName;

string fullNameTest = getFullName("Test1", "Func2");
print fullNameTest;

print getFullName("test", "print");

phoonk void Fibonnacci(){
    int n = 10;
    print "starting fibonnaci test with n = " + n;

    int first = 1;
    int second = 0;
    int fibonnacci = 1;
    while(fibonnacci < n){
        int prevSecond = second;
        second = second + first;
        first = prevSecond;
        print "fibonnacci #" + fibonnacci + " = " + second;
        fibonnacci = fibonnacci + 1;
    }
}

Fibonnacci();

int [] testArray = [1,2,3,4,5];
print "first element: " + testArray[0];

phoonk int testPhoonkAnswer(int p){
    return p * 2;
}

```

```

}

int test = testPhoonkAnswer(2); // 4 i masyva
//print test;

//print testPhoonkAnswer[0]; // 4

testPhoonkAnswer(5); // 10 i masyva

print testPhoonkAnswer[0]; // 4
print testPhoonkAnswer[1]; // 10

print "should return -4: " + testPhoonkAnswer(-2); // -4 i masyva

print "should return 10: " + testPhoonkAnswer[4]; // tik 3 elementai yra masyve

print "should return -4: " + testPhoonkAnswer[-1];
print "should return 10: " + testPhoonkAnswer[-2];
print "should return 4: " + testPhoonkAnswer[-3];
print "should return -4: " + testPhoonkAnswer[-4];
print "should return 10: " + testPhoonkAnswer[-2];

```

```

PS C:\Users\user\Desktop\Func> java -jar target/funcive-1.0-SNAPSHOT.jar funcive-code/phoonkTest.funcive
original b: 2
Should add 10 to b: 12
Hello World!
Test Func
Test1 Func2
test print
starting fibonnaci test with n = 10
fibonnacci #1 = 1
fibonnacci #2 = 1
fibonnacci #3 = 2
fibonnacci #4 = 3
fibonnacci #5 = 5
fibonnacci #6 = 8
fibonnacci #7 = 13
fibonnacci #8 = 21
fibonnacci #9 = 34
first element: 1
4
10
should return -4: -4
should return 10: 10
should return -4: -4
should return 10: 10
should return 4: 4
should return -4: -4
should return 10: 10

```

11 pav. phoonkTest.funcive.

printTest.funcive

```

print "";
print "123";
print "test";
print "test123";
print 123456;
print 1234.55;
print true;
print false;

int testPrint1 = 99;
float testPrint2 = 44.44;
string testPrint3 = "test string";
boolean testPrint4 = true;
boolean testPrint5 = false;
int [] testArrayPrint = [1,2,3,4,5,6];

print testPrint1;
print testPrint2;
print testPrint3;
print testPrint4;
print testPrint5;
print testArrayPrint;

print testPrint1 + " + " + testPrint1 + " = " + (testPrint1 + testPrint1);
print testPrint2 + " + " + testPrint2 + " = " + (testPrint2 + testPrint2);
print testPrint3 + " + " + testPrint3 + " = " + (testPrint3 + testPrint3);
print testPrint4 + " + " + testPrint4 + " = " + (testPrint4 + testPrint4);
print testPrint5 + " + " + testPrint5 + " = " + (testPrint5 + testPrint5);
print testPrint4 + " + " + testPrint5 + " = " + (testPrint4 + testPrint5);
// print testArrayPrint + " + " + testArrayPrint + " = " + (testArrayPrint + testArrayPrint); // throws error

print testPrint1 + " - " + testPrint1 + " = " + (testPrint1 - testPrint1);
print testPrint2 + " - " + testPrint2 + " = " + (testPrint2 - testPrint2);
// print testPrint3 + " - " + testPrint3 + " = " + (testPrint3 - testPrint3); // strings cannot be subtracted
// print testPrint4 + " - " + testPrint4 + " = " + (testPrint4 - testPrint4); // booleans cannot be subtracted
// print testPrint5 + " - " + testPrint5 + " = " + (testPrint5 - testPrint5); // booleans cannot be subtracted
// print testPrint4 + " - " + testPrint5 + " = " + (testPrint4 - testPrint5); // booleans cannot be subtracted

```

```

// print testArrayPrint + " - " + testArrayPrint + " = " + (testArrayPrint - testArrayPrint); // throws error

print testPrint1 + " * " + testPrint1 + " = " + (testPrint1 * testPrint1);
print testPrint2 + " * " + testPrint2 + " = " + (testPrint2 * testPrint2);
// print testPrint3 + " * " + testPrint3 + " = " + (testPrint3 * testPrint3); // strings cannot be multiplied
// print testPrint4 + " * " + testPrint4 + " = " + (testPrint4 * testPrint4); // booleans cannot be multiplied
// print testPrint5 + " * " + testPrint5 + " = " + (testPrint5 * testPrint5); // booleans cannot be multiplied
// print testPrint4 + " * " + testPrint5 + " = " + (testPrint4 * testPrint5); // booleans cannot be multiplied
// print testArrayPrint + " * " + testArrayPrint + " = " + (testArrayPrint * testArrayPrint); // throws error

print testPrint1 + " / " + testPrint1 + " = " + (testPrint1 / testPrint1);
print testPrint2 + " / " + testPrint2 + " = " + (testPrint2 / testPrint2);
// print testPrint3 + " / " + testPrint3 + " = " + (testPrint3 / testPrint3); // strings cannot be divided
// print testPrint4 + " / " + testPrint4 + " = " + (testPrint4 / testPrint4); // booleans cannot be multiplied
// print testPrint5 + " / " + testPrint5 + " = " + (testPrint5 / testPrint5); // booleans cannot be multiplied
// print testPrint4 + " / " + testPrint5 + " = " + (testPrint4 / testPrint5); // booleans cannot be multiplied
// print testArrayPrint + " / " + testArrayPrint + " = " + (testArrayPrint / testArrayPrint); // throws error

print testPrint1 + " % " + testPrint1 + " = " + (testPrint1 % testPrint1);
print testPrint2 + " % " + testPrint2 + " = " + (testPrint2 % testPrint2);
// print testPrint3 + " % " + testPrint3 + " = " + (testPrint3 % testPrint3); // Modulus arithmetics doesn't work on
strings
// print testPrint4 + " % " + testPrint4 + " = " + (testPrint4 % testPrint4); // Modulus arithmetics doesn't work on
booleans
// print testPrint5 + " % " + testPrint5 + " = " + (testPrint5 % testPrint5); // Modulus arithmetics doesn't work on
booleans
// print testPrint4 + " % " + testPrint5 + " = " + (testPrint4 % testPrint5); // Modulus arithmetics doesn't work on
booleans
// print testArrayPrint + " % " + testArrayPrint + " = " + (testArrayPrint % testArrayPrint); // throws error

```

```

PS C:\Users\User\Desktop\Func> java -jar target/function-1.0-SNAPSHOT.jar function-code/printTest.Functionive
123
test
test123
123456
1234.55
true
false
99
44.44
test string
true
false
[1, 2, 3, 4, 5, 6]
99 + 99 = 198
44.44 + 44.44 = 88.88
test string + test string = test stringtest string
true + true = true
false + false = false
true + false = true
99 - 99 = 0
44.44 - 44.44 = 0.0
99 * 99 = 9801
44.44 * 44.44 = 1974.9135
99 / 99 = 1
44.44 / 44.44 = 1.0
99 % 99 = 0
44.44 % 44.44 = 0.0

```

12 pav. phoonkPrint.functionive

switchTest.functionive

```
int a = 1;
```

```
int testSwitch = 0;
```

```
switch (a){ // should go to case 1 and default
```

```
case 1:
```

```
testSwitch = 1;
```

```
case 2:
```

```
testSwitch = 10;
```

```
case 3:
```

```
testSwitch = 100;
```

```
default:
```

```
testSwitch = testSwitch + 69;
```

```
}
```

```
print "testswitch: " + testSwitch;
```

```
a = 2;
```

```
testSwitch = 0;
```

```
switch (a){ // should go to case 2 and default
```

```
case 1:
```

```
testSwitch = 1;
```

```
break;
```

```
case 2:
```

```
testSwitch = 10;
```

```
case 3:
```

```
testSwitch = 100;
```

```

        break;
    default:
        testSwitch = testSwitch + 69;
    }
    print "testswitch: " + testSwitch;

a = 3;
testSwitch = 0;
switch (a){ // should go to case 3 and break
    case 1:
        testSwitch = 1;
        break;
    case 2:
        testSwitch = 10;
    case 3:
        testSwitch = 100;
        break;
    default:
        testSwitch = testSwitch + 69;
}
print "testswitch: " + testSwitch;

```

```

PS C:\Users\user\Desktop\Func> java -jar target/Function-1.0-SNAPSHOT.jar funcative-code/switchTest.function
testswitch: 70
testswitch: 79
testswitch: 100

```

13 pav. switchTest.function

Šaltiniai

1. ANTLR. *ANTLR* [online]. [no date] [viewed 4 May 2023]. Available from: <https://www.antlr.org/>
2. (2023, February 19). *antlr4/getting-started.md at master · antlr/antlr4*. GitHub. <https://github.com/antlr/antlr4>