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

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 - ,,Functive”.

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 funckiją, jei funkcija nėra void tipo, galime gauti prieš tai grąžintų funkcijų rezultatus betkurioje kodo vietoje, globaliųjų kintamųjų pagalba. Tereikia iškviesti 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 žinot 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 rėžius, 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 grąžinę 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 pavyzdziai.

```
// 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 unikaliają 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 iškviesti funkciją
foobar(97);

// Unikalios savybės pavyzdžiai
// Komentaruose parodome numatomą išvestį ar funkcijos rezultatą
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.

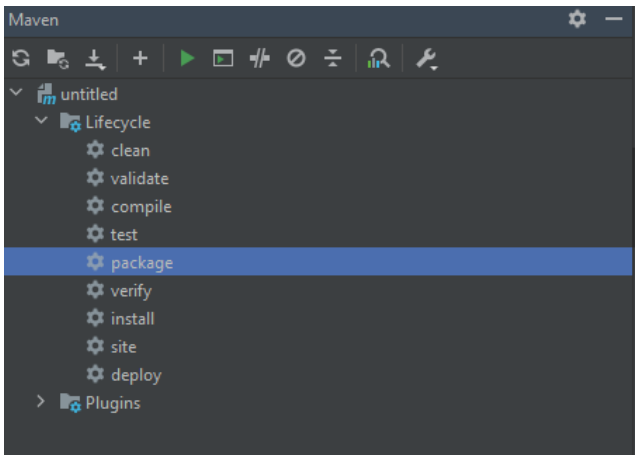
2.2 Išvardinti pasirinkti naudojami įrankiai (jeigu įrankiai buvo pakeisti, tai dėl kokios priežasties)

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.).

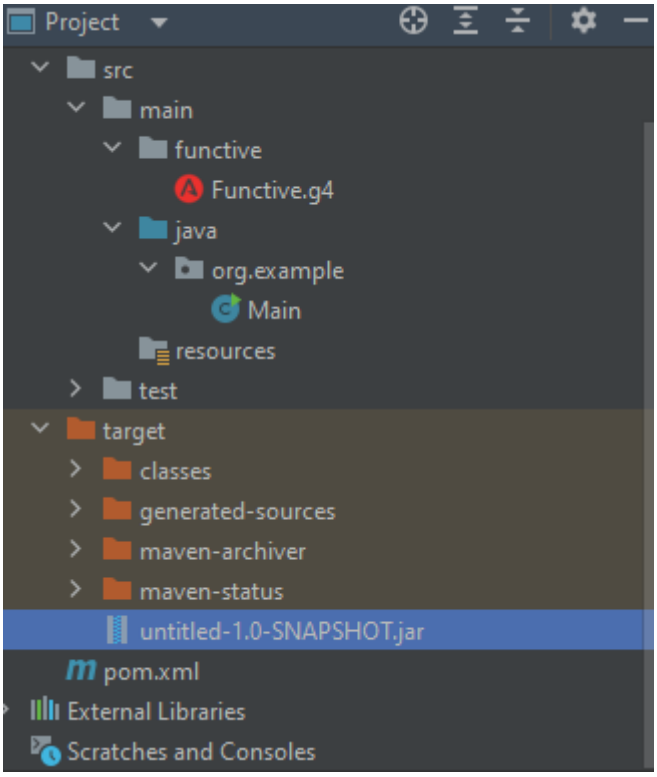
Kuriame kompiliatorių, todėl pavaizduosime, kaip sukursime galutinį failą. Kadangi dirbame ir naudojame kompiliatoriaus kūrimui Java, todėl mums reikia sugeneruoti .jar failą.

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)

Functive.g4

```
grammar Functive;

// 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

4. Šaltiniai