

**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/INTERPRETATORIAUS KŪRIMO PROJEKTINIO (SAVARANKIŠKO) DARBO ATSISKAITYMAS	3
1.1	Kalbos idėja ir pavadinimas, komandos pavadinimas ir jos nariai.	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).	4
1.4	Unikali savybė ir jos naudojimo kodo pavyzdys (jūsų programavimo kalboje parašytas pavyzdys).	5
1.5	Pasirinkti darbo įrankiai (ir pasirinkimo priežastys) bei darbui naudojama programavimo kalba.	6
1.6	Kodo pavyzdžiai.	7
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.).	8
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	8
4.	ŠALTINIAI	8

# 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 programavimo kalbų šeimą. Unikali kalbos savybė padeda kaupti buvusių funkcijų rezultatus. Mūsų kalbos pavadinimas - „Funcative”.



1 pav. Kalbos logotipas.

## 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  
String - simbolių rinkinys/masyvas

### 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;  
String s = "asd";  
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 unikali savybė padaro taip, kad kiekvieną kartą įvykdžius funkciją, jei funkcija nėra void tipo, galime gauti prieš tai grąžintų funkcijų rezultatus betkuriame kodo vietoje, globaliųjų kintamųjų pagalba. Tereikia iškviešti 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 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:

```
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:

```
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 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 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);

// 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

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

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

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

### **3. Galutinis (3 etapo) projektinio darbo atsiskaitymas**

### **4. Šaltiniai**