**MikroJava kompajler**

Elektrotehnički fakultet Univerziteta u Beogradu  
Katedra za računarsku tehniku i informatiku  
  
Predmet: Programski prevodioci 1  
Školska godina: 2025/2026  
Ispitni rok: Januar

# Opis zadatka

Ovaj projekat predstavlja implementaciju jednostavnog kompajlera za jezik Mikrojava, izrađenog kao deo praktične obaveze u okviru kursa Programski prevodioci 1.

Projekat zahteva implementaciju kompajlera za programski jezik Mikrojava, koji ima četiri glavne faze:

1. **Leksička analiza** - prepoznavanje tokena (identifikatori, konstante, ključne reči, operatori)
2. **Sintaksna analiza** - parsiranje programa prema gramatici i izgradnja apstraktnog sintaksnog stabla
3. **Semantička analiza** - provera kontekstnih uslova i upotrebe simbola
4. **Generisanje koda** - prevodjenje u Mikrojava bajtkod

Priloženo rešenje projekta uključuje:

Nivo A

* **Leksička analiza**: Osnovni tokeni
* **Sintaksna analiza**: Osnovne konstrukcije (dodele, aritmetički izrazi, osnovni iskazi)
* **Semantička analiza**: Provera upotrebe simbola (konstante, globalne i lokalne promenljive)
* **Generisanje koda**: Podrška za osnovne iskaze i aritmetičke izraze

Nivo B

* **Sintaksna analiza**: Kontrolne strukture (if-else, do-while, break, continue)
* **Semantička analiza**: Provera funkcija, nizova i formalnih parametara
* **Generisanje koda**: Podrška za kontrolne strukture i pozive funkcija

Nivo C

* **Sintaksna analiza**: Unutrašnje klase, nasleđivanje, interfejsi
* **Semantička analiza**: Provera polimorfizma, nasleđivanja i interfejsa
* **Generisanje koda**: Podrška za objekte, nasleđivanje i virtuelne metode

Tehnički detalji

* Koristi JFlex za leksičku analizu
* Koristi AST-CUP za sintaksnu analizu
* Koristi tabelu simbola sa sajta predmeta
* Generišemo bajtkod za Mikrojava VM
* Implementiran oporavak od grešaka

# Opis komandi

## Generisanje Java koda alatima (JFlex i CUP)

Proces generisanja je automatizovan kroz Ant target-e:

**Leksička analiza (JFlex)**

*ant lexerGen*

* Generiše Yylex.java iz mjlexer.flex i smešta ga u src/rs/ac/bg/etf/pp1
* Koristi lib/JFlex.jar

**Sintaksna analiza (CUP)**

*ant parserGen*

* Generiše MJParser.java i AST klase iz mjparser.cup
* Opcije:
  + -ast rs.ac.bg.etf.pp1.ast - generiše AST klase u navedeni paket
  + -dump\_states - ispisuje stanja parsera (za debug)
  + -buildtree - omogućava izgradnju stabla

## Prevođenje generisanog koda

*ant compile*

* Kompajlira sve Java fajlove u src/rs/ac/bg/etf/pp1
* Koristi biblioteke:
  + JFlex.jar, cup\_v10k.jar (za parser/lekser)
  + log4j-1.2.17.jar (za logovanje)
  + symboltable.jar (tabela simbola)
  + mj-runtime-1.1.jar (MikroJava VM)

## Testiranje generisanog bajtkoda

Postoje target-ovi za disasembliranje i pokretanje bajtkoda:

**Disasembliranje (.obj fajla)**

*ant disasm*

* Pokreće rs.etf.pp1.mj.runtime.disasm na output/program.obj
* Prikazuje instrukcije MikroJava bajtkoda

**Debug izvršavanja**

*ant debugObj*

* Pokreće program u debug modu sa mj-runtime-1.1.jar
* Koristi ulaz iz input/input.txt

**Normalno izvršavanje**

*ant runObj*

* Pokreće program bez debug informacija

## Čišćenje generisanih fajlova

*ant delete*

* Briše generisane fajlove (osim izuzetih u <exclude> listi)

## Napomene

Target repackage menja reference na pakete u generisanom kodu

*ant repackage*

Prebacuje paket generisanog koda iz src.rs.ac.bg.etf.ppp1.ast u rs.ac.bg.etf.ppp1.ast

# Opis novouvedenih klasa

Pored klasa opisanih postavkom zadataka (dakle Compiler, SemanticAnalyzer i CodeGenerator) uvedeno je nekoliko klasa koje olakšavaju implementaciju samog rešenja

## ExtendedStruct

Implementacija same klase Struct ima nekoliko nedostataka:

1. Mogućnost ulaska u beskonačnu petlju u metodi equals

Problem nastaje ukoliko je kind == Class pa moramo pozvati Obj.equalsCompeteHash metodu, ona može biti problematična kako u njoj proveravamo da li su objekti jednaki equals metodom iz klase Obj.   
Unutar metode equals radimo sledeće:

* Poredimo kind polje – nema potencijalnih problema
* Poredimo name polje – nema potencijalnih problema
* Poredimo adr polje – nema potencijalnih problema
* Poredimo level polje – nema potencijalnih problema
* Pozivamo equalsCompleteHash za lokalne cvorove lokalnih simbola što može predstavljati problem ukoliko naiđemo na elemente koji su već poredjeni ranije, čime ulazimo u beskonačnu petlju

Rešenje je vrlo jednostavno – equals metoda će da poredi čvorove tipa Struct po adresi, jer svakako ne pravimo dvostruke čvorove za bilo koju vrstu Struct čvora, već je adresa za određeni tip jedinstvena.

Međutim ovo pravilo ne važi za nizove jer potencijalno možemo imati neograničeni broj vrsta nizova i prilikom kreiranja Obj čvora niza kreiramo i novi Struct čvor za niz. Pa je stoga u slučaju nizova potrebno proveriti elemType polja na jednakost (equals metodom).

1. Tip reference je nepotpun

Tipovi reference jesu Struct čvorovi sledeće vrste:

* Class
* Interface
* Array
* Set (korišćen Enum iz implementacije)

1. Kako je tip reference nepotpun to povlači lošu implementaciju metoda koje koriste equals
2. assignableTo metoda pored lošeg equals nema proveru za dodelu tipa nasleđene klase tipu roditeljske klase

Ukoliko element nije assignable to ne mora nužno značiti da se ne može odraditi operacija dodele u destinacioni Struct čvor, jer potencijalno naš čvor može proširivati taj interfejs ili klasu.

Implementacija je data u klasi ExtendedStruct koja je odrađena po projektnom uzorku Singleton.

## MySymbolTableVisitor

Ova klasa dodatno ispisuje tip Bool, tip Set, kao i klase i interfejse

## Klase za tabelu virtuelnih funkcija

### TVFEntry

Svrha:

* Predstavlja jedan unos u tabeli virtuelnih funkcija (VMT - Virtual Method Table).

Sadržaj:

* String methodName - Ime metode
* int methodAddress - Adresa metode u memoriji (offset u VMT)

Značaj:

* Čuva osnovne informacije o metodi potrebne za dinamičko vezivanje.

### TVF

Svrha:

* Implementira tabelu virtuelnih funkcija za jednu klasu

Kljucne funkcionalnosti:

* int memoryStartTVF - polje pokazivač na mesto u statičkoj memoriji odakle počinje tekući TVF
* List<TVFEntry> entries - niz ulaza – u ulaze stavljamo prvo sve metode koje nasledimo iz roditeljske klase, sa sve adresama, zatim dodajemo metode koje implementira naša klasa, ukoliko ime metode već postoji (što znači da je implementira i nadklasa) mi u taj ulaz upisujemo novo mesto u memoriji koje predstavlja implementaciju metode u našoj klasi (tekućoj)
* inheritFromParent() – nasledjuje metode iz roditeljske klase/interfejsa
* addEntry()/addMethod() – dodaje nove metode u tabelu
* putTVFInMemory() – generise bajtkod za smestanje TVF u staticku memoriju (koristi Code klasu)

Značaj:

* Omogućava polimorfizam kroz dinamičko vezivanje metoda.
* Čuva informacije o svim metodama klase (i nasleđenim)

### TVFHandler

Svrha:

* Upravlja svim tabelama virtuelnih funkcija u programu (odradjena po projektnom uzorku Singleton).

Kljucne funkcionalnosti:

* Map<Struct, TVF> tvfMap – mapa koja povezuje tip klase sa njenom TVF, u pitanju je ulancana hes mapa kako bismo elementima mogli da pristupimo u onom redosledu u kojem smo ih dodavali u TVFHandler.
* createTVF() – kreira novu TVF za datu klasu
* inheritMethods() – nasledjuje metoda iz roditeljske klase/interfejsa
* addClassMethods() – dodaje metode definisane u samoj klasi
* putAllTVFsInMemory() – Generise bajtkod za sve TVF u programu

Znacaj:

* Centralizovani menadzer za sve TVF u kompajleru
* Obavlja kompletnu pripremu virtuelnih metoda za izvrsavanje

# Opis testova

Pored testova prilozenih u postavci zadatka (koji prolaze) napravljen je niz testova da se proveri otpornost programa na semanticke greske i oporavak u slucaju greske leksera.