**MikroJava kompajler**

Elektrotehnički fakultet Univerziteta u Beogradu  
Katedra za računarsku tehniku i informatiku  
  
Predmet: Programski prevodioci 1  
Školska godina: 2024/2025  
Ispitni rok: ?

Sadržaj

[1. O projektu 4](#_Toc206336630)

[1.1. Ukratko o projektu 4](#_Toc206336631)

[1.2. Ključne karakteristike 4](#_Toc206336632)

[1.2.1. Osobine jezika 4](#_Toc206336633)

[1.2.2. Komponente kompajlera 4](#_Toc206336634)

[1.2.3. Implementacioni detalji 4](#_Toc206336635)

[2. Pokretanje 5](#_Toc206336636)

[3. Pomoćne klase 5](#_Toc206336637)

[3.1. MySymboltableVisitor 5](#_Toc206336638)

[3.2. StructExtended 5](#_Toc206336639)

[3.3. TabExtended 6](#_Toc206336640)

[3.4. SetHandler 6](#_Toc206336641)

[3.5. TVFEntry 6](#_Toc206336642)

[3.6. TVF 6](#_Toc206336643)

[3.7. TVFHandler 7](#_Toc206336644)

[4. Testiranje 7](#_Toc206336645)

# O projektu

Kompajler za MikroJava programski jezik, razvijan kao deo kursa Programski prevodioci 1 na Elektrotehničkom fakultetu Univertiteta u Beogradu.

## Ukratko o projektu

Ovaj kompajler služi da prevodjenje MikroJava izvornog koda u bajtkod pogodan za tumačenje MikroJava virtuelne mašine. MikroJava je pojednostavljen jezik sličan javi, njegove karakteristike su sledeće:

* **Podržava skalarne tipove**: int, char, bool
* **Podržava tipove referenci**: nizove, skupove, klase i interfejse
* **Objektno je orijentisan**: podržani su nasleđivanje, polimorfizam i preklapanje metoda
* **Dozvoljene kontrolne strukture**: if-else i do-while
* **Ugrađeni neklasni statički metodi**: print, read, ord, chr, len, add, addAll

## Ključne karakteristike

### Osobine jezika

* Statička promenljive i metodi
* Automatski objekti
* Klasno nasleđivanje i implementacija interfejsa
* Preklapanje metoda
* Skupovi celobrojnih vrednosti i nizovi
* Tabela vertuelnih funkcija za polimorfizam
* Osnovne ulazno/izlazne operacije

### Komponente kompajlera

* Lekser – leksička analiza (src/spec/mjlexer.flex) izvršava se uz pomoć JFlex alata
* Parser – sintaksna analiza (src/spec/mjparser.cup) izvršava se uz pomoć cup\_v10k alata
* Semantički analizator – semantička analiza, uz priloženu tabelu simbola
* Kod generator – generisanje koda, izvršava se uz pomoć mj-runtime klase

### Implementacioni detalji

* Proverava opšte kontekstne uslove
* Proverava kompatibilnost tipova podataka
* Određivanje opsega važenja i životnog veka
* Kreiranje Tabele virtuelnih funkcija za svaku klasu

# Pokretanje

Pre pokretanja moramo biti sigurni da imamo verziju jave 1.8, kao i Apache Ant alat za pokretanje targeta.

Projekat pokrećemo tako što:

1. Pokrenemo repackage target kojim ćemo izgenerisati klase za svaku smenu unutar naše gramatike, koje dalje koristi parser prilikom obilaska sintaksnog stabla
2. Podesimo izvorni fajl (u kojem će se nalaziti kod koji prevodimo) i podesimo odgovarajuću putanju do njega unutar Compiler klase (tests/rs.ac.bg.etf.pp1/Compiler.java)
3. Pokrenemo main metodu klase Compiler
4. Pokrenemo runObj target koji će iz generisanog bajtkoda unutar objektnog fajla izgenerisati asembli instrukcije i izvršiti program
5. Napomena: ukoliko je potrebno i to zahteva program (koristi se read instrukcija) treba podesiti input.txt fajl unutar input foldera, čitanje iz njega predstavlja ulazni tok podataka

# Pomoćne klase

Kako bismo rasteretili kod unutar takozvanih „božanskih klasa“ uvede su pomoćne klase koje logički izdvajaju pojedine funkcionalnosti programa:

## MySymboltableVisitor

Ova klasa proširuje SymbolTableVisitor klasu tako što uvodi ispise za tipove koji nisu definisani unutar osnovne klase: ispit bool tipa, skupova, kao i klasa i interfejsa

Ova klasa se koristi za ispis kreiranih objektnih čvorova nakon obilaska sintaksnog stabla i semantičke analize.

## StructExtended

Kako osnovna klasa Struct ne podržava reference koje mogu biti klasnog tipa ili tipa niza mi želimo proširenje ovog skupa na tipove skupova i interfejsa.

Takođe, uz ovu modifikaciju dolazi do problema unutar metoda unutar klase Struct koje pozivaju isReferenceType pa je potreba za uvodjenjem nove klase jasna. Takodje, sama logika provere equal se može značajno pojednostaviti (ukoliko su u pitanju reference osim nizova) proverom samih adresa referenci, jer u samom programu ne pravimo duplikate objektnih čvorova.

Jedini slučaj kada je potrebna nešto složenija provera jeste referenca na nizove. Ovaj problem nastaje jer nismo ograničili skup tipa elemenata niza, pa potencijalno postoji neograničen broj mogućnosti da se nizovi kreiraju sa različitim tipom elementa. Zato je pored toga, što je Struct tipa niza potrebno pozvati equals metodu iz klase extended struct koja će proveriti jednakost tipova elemenata nizova.

Kako smo uveli pojam polimorfizma tako je potrebno redefinisati, ili u našem slučaju definisati novu metodu assignableTo koja se koristi prilikom dodele. Ukoliko su operandi operatora dodele reference takodje treba proveriti (u slučaju da tipovi nisu exsplicitno identični) da li je moguća implicitna konverzija iz tip desnog i tip levog operanda. Ovo radimo tako što proveravamo sve klase koje proširuje desni operand i sve interfejse koje isti implementira (primetiti da je moguća implementacija samo jednog interfejsa).

Kako se ova klasa samo svodi na pozive statičkih metoda, dovoljna je jedna instanca pa je urađena po projektnom uzorku Singleton.

## TabExtended

Kako postavka zadatka zahteva uvođenje novih ugrađenih tipova (bool i set) i ugrađenih metoda (vidljivih: add, addAll i nevidljivih: $printSet, $union), jasna je potreba za uvodjenjem klasa kojoj će se delegirati odgovornost kreiranja i ubacivanja odgovarajućih objektnih čvorova u tabelu simbola.

U ovoj klasi takođe se logika priložene Tab klase prilagođava logici programa, to jest formalni parametri metoda se svoj fpPos atribut postavljaju na jedan.

Pre svega ovoga, naravno potrebno je pozvati init metodu klase Tab koja inicijalizuje tabelu simbola.

Kako je dovoljno samo jednom inicijalizovati tabelu simbola ova klasa je odrađena po projektnom uzorku Singleton, gde se prilikom kreiranja statičkog objekta instance klase poziva init metoda unutar koje se poziva init metoda klase Tab, a nakon tog poziva se dodaju odgovarajući Objektni čvorovi tipova i metoda.

## SetHandler

Ovoj klasi delegirana je odgovornost kreiranja koda za operacije sa skupovima prilikom generisanja koda. Kako samo jednom želimo da popunimo memoriju meetodama koje su vezane za skupove, i ovu klasu kreiramo po projektnom uzorku Singleton gde se memorija puni metodama prilikom inicijalizacija statičke instance klase.

## TVFEntry

Ova klasa predstavlja apstrakciju jednog ulaza u tabelu virtuelnih funkcija i sastoji se od imena metoda i njegove adrese u memoriji.

## TVF

Ova klasa predstavlja listu ulaza (TVFEntry) i interfejs prema toj listi. Takodje u ovoj klasi se čuva memorijska lokacija otiska ove klase u statičkoj zoni memorije. Interfejs omogućava sledeće:

* Dohvatanje lokacije TVF-a u memoriji
* Nasleđivanje metoda roditeljske klase – ubacuje nove ulaze u TVF
* Pronalaženje ulaza unutar TVF-a u suprotnom vraća null
* Dodavanje novog ulaza u TVF
* Utiskivanje TVF-a u statičku memoriju (moguće uraditi samo jednom unutar jednog generisanja koda)

## TVFHandler

Služi za upravljanjem tabelama ulaza. Ova lista predstavlja ulančanu mapu (prilikom obilaska ključevi se vraćaju onim redom kojim su dodavani, ovo je korisno prilikom debagovanja programa i provere generisanih mikroinstrukcija) gde je ključ strukturni čvor korisnički definisane klase za koji se vezuje TVF koji predstavlja vrednost ove mape.

Ova klasa poziva metode klase TVF i omogućava da svaka klasa jednostavno koristi svoju tabelu simbola, bez da išta zna o postojanju ostalih tabela simbola.

Ova klasa takodje omogućava utiskivanje svih tabela simbola u memoriju.

Kako želimo postojanje samo jedne mape TVF-ova ova klasa je odrađena po projektnom uzorku Singleton.

# Testiranje

Testovi se nalaze unutar foldera tests i podeljeni su u nekoliko logičkih celina po podfolderima prema delovima kompajlera.

Važno je obratiti pažnju na to da su testovi namenjeni samo fazi u čijem se korenom direktorijumu nalaze, kao i fazama koje njoj prethode. Za faze koje su logički iza faze u korenom direktorijumu ponašanje je nedefinisano. Dakle, ako se na primer nalazimo u testu za semantičku analizu, izlazi su predviđeni za fazu semantičke analize, da bismo ih izvršili neophodno je proći kroz faze leksičke i sinaksne analize, ali faza generisanja koda tek sledi, pa ponašanje može biti nedefinisano ako se iz kompajliranja ova faza ne isključi.

Testovi su napravljeni tako da pokriju što je više moguće slučajeva (tačnih i netačnih).

Takodje, u folderu GenerisanjeKoda-TestPrimeri\_24\_25\_jan\_feb dostupni su zvanični testovi.