DOKUMENTACIJA

Programski prevodioci - predmetni zadatak

Osnovni podaci

Broj indeksa	Ime i prezime	Tema
SW47-2019	Bane Gerić	Kreiranje leksera, parsera i AST-a pomoću ANTLR alata

Korišćeni alati

Naziv	Verzija
ANTLR	4.9.3
InteliJ	2022.1.2

Evidencija implementiranog dela

Kreirana je gramatika miniC# jezika po ugledu na miniC gramatiku. U odnosu na miniC gramatiku dodato je još da fajl mora početi sa *namespace*. Taj namespace može da sadrži 0 ili više klasa, a svaka klasa ima listu atributa i listu funkcija. Atributi i funkcije dodatno mogu da imaju *accessor* (public, private ili protected). Dakle, implementirana je gramatika miniC jezika sa pomenutim dodacima. Za sve pomenuto je implementirana leksička, sintaksna i semantička analiza. Ovo sve je implementrano uz pomoć ANTLR alata i Java programskog jezika.

Detalji implementacije

Gramatika jezika kreirana je uz pomoć pomentog ANTLR alata. Predstavlja proširenje gramatike miniC jezika sa elemntima C# jezika i nazvana je miniC#. Na Slika 1 je prikazan dio gramatike koji je dodat, odnosno dio sa klasama i namespace-om.

```
program             : namespace
                    ;

namespace           : NAMESPACE ID LBRACKET class_def* RBRACKET
                    ;

class_def           : ACCESSOR? CLASS ID LBRACKET class_body RBRACKET
                    ;

class_body          : attribute_list function_list
                    ;
```

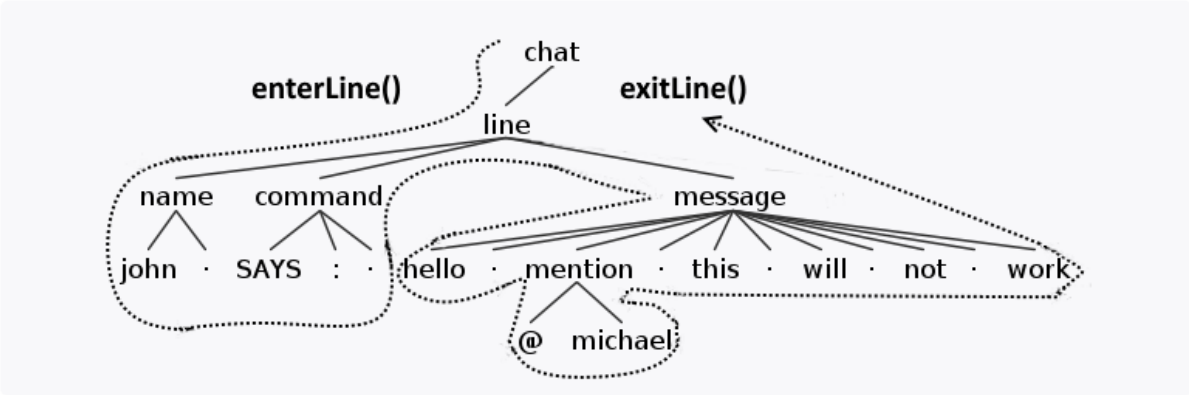
Slika 1. Dio gramatike sa namespace-om i klasama

Kao što može da se vidi, jedan namespace može da sadrži 0 ili više klasa, a svaka klasa može da ima accessor, a pored njega ima definiciju i tijelo klase. Tijelo klase se sastoji iz liste atributa, koji takođe mogu da imaju accessore, kao i liste funkcija. Svaka fukncija unutar jedne klase mora biti jedinstvena, dok se nazivi funkcija unutar različitih klasa mogu preklapati.

Mora da postoji tačno jedna *main* metoda, koja može biti nazvana malim ili velikim početnim slovom. Što se tiče tipova podataka podržani su *int* i *long* (u miniC-u je *uint* umjesto long-a).

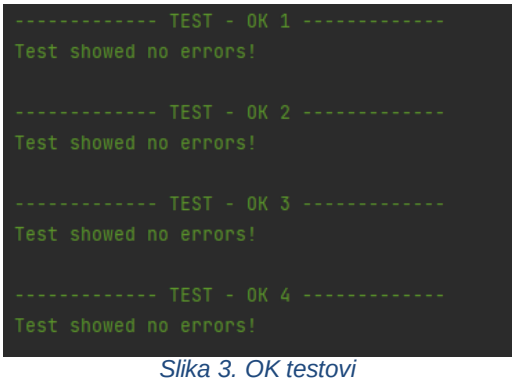
Što se tiče implementacije tabele simbola (*SymbolTable*), ona je implementirana kao java klasa koja sadrži listu svih redova (red je zasebna klasa - *SymTabElement*). Pored liste redova posjeduje i metode za manipulaciju tabelom. Te metode su implementirane po uzoru na metode iz miniC jezika i prilagođene javi.

Leksička i sintaksna analiza su interno već podržane na osnovu same gramatike, tako je samo odrađena semantička analiza. Ona je urađena upotrebom klase *Listener* koja nasleđuje klasu *miniCSharpBaseListener* i preklapa potrebne metode. Za svaki element gramatike se generišu dvije metode (*enter* i *exit*) na osnovu kojih se može uraditi semantička analiza. Posmatrajući AST na Slika 2 možemo vidjeti kako funkcionišu ove metode. Semantička analiza se obavlja obliazeći stablo po dubini.

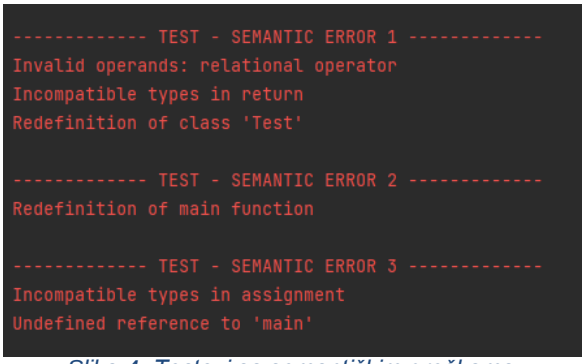


Slika 2. Enter i exit metode na AST

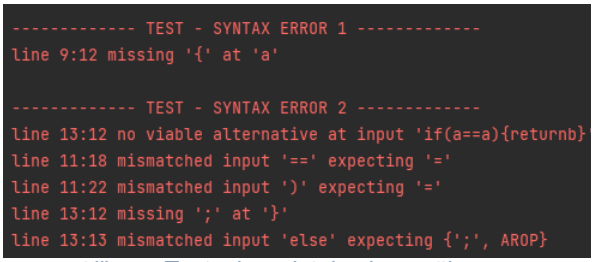
U projektu se nalazi 9 testnih primjera koji služe za testiranje rada parsera. Pokretanjem *main* metode pokreću se svi testovi, a njihovi rezultati budu ispisani u konzoli. Testovi koji su bez grešaka ispisani su u konzoli zelenom bojom, dok su testovi sa greškama ispisani crvenom bojom. Žutom bojom su prikazana upozorenja na potencijalne semantičke greške koje je program uočio tokom sintaksne analize, ali ne mora da znaži da one zaista postoje. Prvo se ispisuju semantičke greške, a na kraju sintaksne. Ovo je tako zato što ANTLR interno ispisuje sintaksne greške kao *System.err.print(...)*, ali one sadrže informacije o tome na kom tokenu je nastala greška kao i tačnu liniju greške u fajlu. Na sledećim slikama možemo vidjeti kako izgledaja konzola nakon pokretanja testova(Slika 3, Slika 4, Slika 5, Slika 6).



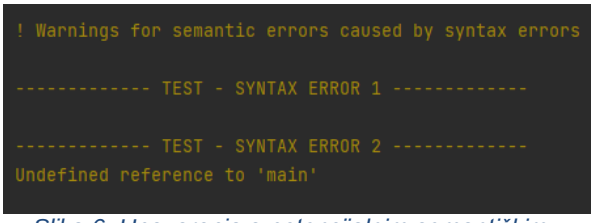
Slika 3. OK testovi



Slika 4. Testovi sa semantičkim greškama



Slika 5. Testovi sa sintaksnim greškama



Slika 6. Upozorenja o potencijalnim semantičkim greškama

Ideje za nastavak

U radu nije implementirano generisanje koda, pa bi to svakako bila jedna od prvih ideja za nastavak. Takođe, pošto su podržane klase bilo bi poželjno da se podrže i konstruktori za te klase. Zatim bi se mogle implementirati funkcije sa više paramatera, kao i *for* i *while* petlje. Nakon toga bi mogli da se podrže još neki tipovi podataka poput *string*-a ili *char*-a i tada bismo imali dosta kompletniji jezik.

Literatura

- https://www.antlr.org/
- https://tomassetti.me/antlr-mega-tutorial/#chapter30
- https://www.youtube.com/watch?v=p2gIBPz69DM
- https://github.com/antlr/antlr4/blob/master/doc/options.md
- https://tomassetti.me/listeners-and-visitors/