DOKUMENTACIJA

Programski prevodioci - predmetni zadatak

Osnovni podaci

Broj indeksa	**Ime i prezime**	**Tema**
SW51/2019	Strahinja Popović	MiniPython

Korišćeni alati

Naziv	Verzija
Flex	2.6.4.
Bison	3.8.2.
GNU Make	4.3.

Evidencija implementiranog dela

1. Aritmetičke operacije sabiranja, oduzimanja, množenja i deljenja.
2. Relacione operacije <, >,<=, >=, ==, !=.
3. Negacija *“not”*.
4. Logički operatori *“and”* i *“or”.*
5. *Assignment statement* (a=2).
6. *Multi assignment statement* (a,b,c=1,2,3).
7. *Return statement* (return 123).
8. Definisanje funkcije bez ili sa vise parametara (def imeFunkcije(....):).
9. Poziv funkcije (imeFunkcije(.....)).
10. *Break statement*
11. *Continue statement*
12. P*ass statement*
13. *If, elif, else statement*
14. *While, else statement*
15. *Try, except, finally, else statement*
16. Unarni operator *“is”.*

Implementirana provera sintakse i semantike i generisanje ekvivalentnog asemblerskog koda za delove pod brojevima: 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14

Implementirana provera sintakse i semantike za delove pod brojevima: 4, 15, 16

Detalji implementacije

Kako bi Python skripta bila uspešno prevedena u asembler ona mora da sadrži definisanu funkciju pod imenom *“main”* koja će se prevesti u labelu *main* u asemblerskom kodu.

Kako se Python zasniva na indentaciji kao pokazitelj početka i kraja bloka, pri leksičkoj analizi bilo je potrebno generisati tokene _INDENT i _DEDENT. Pojavljivanje karaktera “\t” značilo je indentaciju, ali kako bi se odredilo koji token treba da se generiše, implementirano je brojanje *tab* karaktera i upoređivanje sa prošlim stanjem za svaki novi red. Kada bi prošlo stanje bilo manje od trenutnog generisao bi se token _INDENT, dok bi se u slučaju da je prošlo stanje bilo veće od trenutnog generisao token DEDENT. Ukoliko bi prošlo stanje bilo jednako sa sadašnjim ne bi se generisao ni jedan token od ova dva.

Aritemtičke, logičke i relacione operacije, unarni operator *“is”* i negacija *“not”* implementirani su tako da se poštuje međusobna asocijativnost. Implementirana je semantička provera tipa operanda koji učestvuje u operaciji. Tip podatka može da bude *UNKNOWN* što znači da se tip ne zna i tada se zanemaruje semantička provera pri vršenju operacija. To može da se desi samo kada se analizira operacija u kom učestvuje bar jedan parametar funkcije jer se njegov tip ne može odrediti.

Definisanje funkcija urađeno je na način da podržava parametre. Parametri mogu da budu sa predefinisanim vrednostima ili bez njih, ali samo tako da parametri sa predefinisanim vrednostima mogu da budu samo nakon parametara bez predefinisanih vrednosti. Za sve parametre je urađena sintaksa i semantička analiza, dok je generisanje asemblera implementirano samo za parametre bez predefinisanih vrednosti. Na izlazu iz funkcije iz tabele simbola brišu se svi parametri i varijable prvi put definisane u skripti.

Return statement je implementiran tako da može, a ne mora da se javi unutar tela funkcije. Ako se nađe van funkcije baciće se greška. *Return statement* može da se pozove sam ili da vrati jednu vrednost.

While petlja je urađena tako da je podržava ugnježdavanje. Takođe Python podržava *while-else* strukturu, tj. ukoliko se uslov za ulaz u petlju ne zadovolji prvi put izvršava se *else* deo petlje. *Break* i *continue* su podržani samo kada se pozivaju unutar neke petlje. Na izlasku iz *while* i *else* bloka brišu se sve varijable definisane unutar tog bloka.

If statement podržava sva tri slučaja, tj. *if, elif* i *else.* Strutktura ovog izraza podrazumeva da se na prvom mestu pojavljuje *if* blok, zatim ni jedan, jedan ili više *elif* blokova i na kraju jedan ili ni jedan *else* blok. Varijable definisane unutar svakog od blokova izraza brišu se iz tabele na svakom kraju bloka.

Try, except, finally, else izraz podržan je samo kod provere sintakse i semantike. Struktura ovog izrada podrazumeva da se na prvom mestu nalazi *try* blok, zatim jedan ili više *except* blokova i na kraju ili *else* ili *finally* blok.

Pass izraz je veoma jednostavan jer sem sintaksne provere ne zahteva ništa drugo da bi se uspešno pokrenula napisana Python skripta, kako semantička provera za izraz ne postoji, a asmeblerski ekvivalent je apsolutno ništa.

Ideje za nastavak

Python originalno podržava izvršavanje skripte bez *main* funkcije ili čak bez i jedne definisane funkcije. Ovaj projekat podržava sintaksnu i semantičku proveru tako napisanog koda, ali prevođenje Python koda u asemblerski moguće je samo ako se sve izrazi nalaze unutar definisanih funkcija ili postoji funkcija pod imenom *main* koja je ujedno i polazište izvrašavanja skripte. Ideja je da se omogući prevođenje Python skripte bez definisanja *main* funkcije kao i da je moguće pisati izraze van tela funkcija.

Pri testiranju rekurzivnog poziva funkcija utvrđeno je da u nekim slučajevima rezultat nije dobar, stoga potrebno je naći grešku i ukloniti je.

Trenutno nije podržano generisane asemblerskog koda za predefinisane parametre, ideja je da se omogući da pri pozivima funkcije ne budu direktno prosleđene vrednosti za parametre već da se preuzmu vrednosti koje su im dodeljene pri definisanju funkcije i njenih parametara.

Try except blok je trenutno podržan samo pri sintaksnoj i semantičkoj analizi. Ideja je da se u budućnosti podrži podizanje izuzetaka i kompletno generisanje ostalog asemblerskog koda za taj izraz.

Literatura

Korišćena je samo dokumentacija o Python gramatici kojoj se može pristupiti preko sledećeg linka: https://docs.python.org/3/reference/grammar.html