**DOKUMENTACIJA**

Programski prevodioci - predmetni zadatak

**Osnovni podaci**

|  |  |  |
| --- | --- | --- |
| Broj indeksa | Ime i prezime | Tema |
| SW 48/2019 | Jovan Tomić | miniTS |

**Korišćeni alati**

|  |  |
| --- | --- |
| Naziv | Verzija |
| Flex | 2.6.4-6 |
| Bison | 2:3.0.4.dfsg-1build1 |

## Evidencija implementiranog dela

U sklopu projektnog zadatka realizovane su sledeće funkcionalnosti TypeScript jezika (za svaku od stavki urađene su leksika i sintaksa, dok je za prve četiri stavke urađen i deo semantike):

* Definisanje funkcija. Podrazumevaju se funkcije sa nula, jednim ili više parametara.
* Deklarisanje i definisanje promenljivih.
* Dodela vrednosti za promenljive. Podrazumeva se i *inline* dodela.
* Pozivi funkcija
* *If* iskaz
* Relacioni i aritmetički izrazi
* *Return* izraz sa i bez vrednosti
* *While* i *do while* petlje
* *For* petlje (regularna, *for ... in, for ... of). Break* i *continue.*
* Inkrement i dekrement operatori (i pre- i post- verzije).
* *Switch case* iskaz
* *Try catch* iskaz
* Enumeracije
* Interfejsi

## Detalji implementacije

Definisanje funkcija

Obzirom da TypeScript nudi fleksibilnost prilikom deklarisanja funkcija, to je moguće uraditi na više načina. Dva glavna načina jesu definisanje imenovanih i anonimnih funkcija. Prilikom definisanja, moguće je (ali ne i neophodno) specificirati povratnu vrednost funkcije.

Definisanje imenovane funkcije:

<deklarator> <identifikator> = *function*(<lista\_parametara>) [:<tip\_povratne\_vrednosti>] <telo>

Definisanje anonimne funkcije:

*function* <identifikator> (<lista\_parametara>) [:<tip\_povratne\_vrednosti>] <telo>

Kao deklarator mogu da se koriste ključne reči *let, var* i *const.*

Pravila za identifikator su da mora da započne slovnim karakterom, a potom može da sadrži alfanumeričke karaktere kao i karaktere ‘$’ i ‘\_’.

U okviru liste parametara, navodi se nula, jedan ili više parametara, i to na sledeći način: <identifikator>: <tip>.

Tipovi koje podržava miniTS su: *number, bigint, boolean, string, any, void, undefined*.

U okviru semantičke analize, obezbeđena je jedinstvenost identifikatora koji se nalaze na istom nivou (ovo znači da je moguće definisati funkciju unutar funkcije sa istim identifikatorom), kao i nemogućnost pozivanja funkcije koja prethodno nije definisana.

Nešto više o tipovima

Number tip podrazumeva brojčane vrednosti. One mogu biti u različitim formatima (dekadni, binarni, oktalni, heksadecimalni), a format se specificira opštim oznakama koje su prisutne i u drugim programskim jezicima (npr. 123, 0x1AFF, 0o127, 0b1011...).

Bigint je specijalna vrsta celih brojeva, koja specifično služi za skladištenje veoma velikih vrednosti. Označava se sa karakterom ‘n’ nakon brojnog dela (npr. 1231238973228773n).

String tip predstavlja niz karaktera. Označava se tako što se niz karaktera okruži jednostrukim navodnicima, dvostrukim navodnicima ili backtick karakterima (`).

Any tip označava da vrednost može biti praktično bilo kog tipa. Preporuka je da se koristi samo kada smo veoma sigurni u ono što radimo.

Undefined tip označava nedefinisanu vrednost. Slično tome, void predstavlja nedostatak vrednosti, dok boolean može da poprimi vrednosti true i false.

Deklarisanje i definisanje promenljivih

Prilikom deklarisanja promenljivih nije neophodno specificirati tip povratne vrednosti. Definisanje vrednosti moguće je izvršiti naknadno, ali je takođe moguće to uraditi i prilikom same deklaracije. Sintaksa za deklaraciju je:

<deklarator> <identifikator> [: <tip\_povratne\_vrednosti>];

Prilikom dodele vrednosti, koristi se standardni operator ‘=’.

U okviru semantičke analize, obezbeđena je jedinstvenost identifikatora koji se nalaze na istom nivou. Takođe, nije moguće dodeliti vrednost nedeklarisanoj promenljivoj.

For petlje

Postoje tri vrste for petlji u TypeScript jeziku. To su klasična for petlja koja se sastoji iz tri dela (od kojih su sva tri opciona), ali i for ... of i for ... in petlja. Prva se koristi za iteraciju kroz elemente niza, dok se druga koristi za iteraciju kroz atribute objekta. U okviru miniTS projekta, realizovana je sintaksa za sve tri vrste petlji.

Unarni inkrement i dekrement operatori

Unarni inkrement i dekrement operatori služe da brojčanu vrednost povećaju ili smanje za 1. miniTS podržava i prefiksnu i postfiksnu notaciju kada je u pitanju sintaksa. Korišćenjem postfiksne notacije, prvobitno se iskoristi vrednost promenljive, a potom se inkrementira/dekrementira, dok se kod prefiksne notacije prvo primene operacije, a potom se iskoristi vrednost. miniTS sintaksno podržava ove operacije kao zasebne iskaze, ali i u okviru brojnih izraza.

Switch case iskaz

Switch case iskaz karakteriše prisustvo switch(x) dela, gde xpredstavlja promenljivu na osnovu čijih vrednosti će se realizovati različiti delovi koda iz *case* blokova. Promenljiva može biti bilo kog od navedenih tipova, sem *void* i *undefined*. Potrebno je na kraju svakog *case* bloka staviti i *break* iskaz, kako se ne bi izvršio i kod koji se nalazi u neposredno sledećem *case* bloku. Opciono, na kraju može da se nađe i *default* blok, koji se izvršava ukoliko vrednost promenljive nije pokrivena ni jednim *case* blokom.

Try catch iskaz

Try catch mehanizam se koristi za obradu delova koda u kojima možemo da očekujemo pojavu izuzetaka. U okviru miniTS, po uzoru na originalni TypeScript, neophodno je specificirati try deo, kao i jedan (ili oba) od preostala dva, u koje spadaju catch i finally. Catch deo obuhvata kod koji želimo da se izvrši nakon što se desi greška. Moguće je specificrati konkretnu grešku, ali je takođe moguće i izostaviti identifikator, te će biti obuhvaćena svaka greška. Finally blok se izvršava u svakom slučaju.

Enumeracije

Enumeracije u regularnom TS i miniTS poseduju određene razlike u odnosu na enumeracije u ostalim programskim jezicima. Naime, moguće je identifikatorima unutar enumeracije dodeliti vrednosti tipa number, što je slučaj i u većini drugih jezika, ali takođe je moguće dodeliti i vrednosti tipa string. Štaviše, moguće je i kombinovati tipove unutar jedne enumeracije, ali to nije od preteranog značaja za praktičnu primenu.

Interfejsi

Poput enumeracija, interfejsi u regularnom TS i u miniTS imaju neke zanimljiive karakteristike koje nisu tipično prisutne u ostalim programskim jezicima. Pored klasičnog deklarisanja tipova i identifikatora polja, moguće je specificirati i potpise funkcija koje bi implementacija trebala da poseduje. To je moguće učiniti na čak tri različita načina.

Prvi je anoniman način, bez identifikatora, a sintaksa je sledeća:

(<lista\_parametara>): <tip\_povratne\_vrednosti>;

Naredni način uvodi i identifikator:

<identifikator> (<lista\_parametara>): <tip\_povratne\_vrednosti>;

Poslednji način je uz korišćenje tzv. arrow notacije:

<identifikator>: (<lista\_parametara>): <tip\_povratne\_vrednosti>;

Funkcionalnosti koje nisu posebno navedene u okviru detalja implementacije ne razlikuju se u odnosu na standardne načine implementacije, tako da nisu posebno objašnjavane.

## Ideje za nastavak

Kada je u pitanju leksika i sintaksa, miniTS bi mogao da se proširi pojedinim naprednim funkcionalnostima, kao što su identifikatori pristupa, klase, nasleđivanje i brojni drugi elementi TypeScript jezika. Međutim, za osnovne potrebe, možda bi bilo bolje fokusirati se na produbljivanje već postojećih funkcionalnosti, pre svega semantičkom analizom i generisanjem koda. Naredni koraci prilikom razvoja definitivno bi morali da uključe semantičku analizu elemenata koji su implementirani. Kada je u pitanju generisanje asemblerskog koda za pojedine, manje komplekse funkcionalnosti, ono bi bilo u domenu izvodljivog sa trenutnim znanjem. Međutim, pretpostavka je da generisanje asemblerskog koda za komplikovanije stavke, kao što su npr. interfejsi, prevazilazi gradivo ovog predmeta.

## Literatura

* PDF dokumenti korišćeni na predavanjima i vežbama
* <https://www.tutorialspoint.com/typescript/>
* <https://www.typescriptlang.org/docs/>
* <https://www.tutorialsteacher.com/typescript/>