DOKUMENTACIJA

Programski prevodioci - predmetni zadatak

Osnovni podaci

Broj indeksa	Ime i prezime	Tema
SW 54/2019	Milan Sekulić	MiniJS

Korišćeni alati

Naziv	Verzija
Flex, Bison, Hipsim	Verzija sa vježbi

Evidencija implementiranog dela

MiniJS kompajler je funkcionalna i sintaksna modifikacija (i proširenje) MiniC kompajlera sa vježbi. Za svaku od sljedećih funkcionalnosti odrađeni su koraci vezani za sintaksu, semantiku, kao i generisanje asemblerskog koda.

Definisanje funkcija

U JavaScript programskom jeziku postoji više načina definisanja i korišćenja funkcija. U MiniJS kompajleru su implementirana tri načina definisanja funkcija:

Standard:

```
function func1() {
  // body
}
```

Arrow syntax

```
const func2 = () => {
  // body
}
const func21 = a => {
  // body
}
const func22 = (a, b) => {
  // body
}
```

Function expression syntax

```
const func31 = function(a, b) {
  // body
}
```

Pored same sintakse definisanja funkcija, omogućeno je definisanje funkcija sa **više parametara**.

JavaScript deklaracije

MiniJS dozvoljava upotrebu ***let*** i ***const*** ključnih riječi za deklarisanje promjenljivih.

Množenje

Dozvoljena je operacija množenja.

Postincrement i postdecrement iskazi

Implementirani su postinkrement (++) i postdekrement (--) **iskazi**.

Petlje

Omogućena je upotreba ***while*** i ***for*** petlji uz korišćenje ***break*** i ***continue*** ključnih riječi, koje omogućuju kontrolu toka petlje. Ove ključne riječi funkcionišu i unutar ugniježdenih petlji.

Ternarni operator

Omogućena je upotreba ternarnog operatora uz ograničenje da kondicionalni dio istog mora biti smješten unutar zagrada, za razliku od onog u JavaScript programskom jeziku.

Detalji implementacije

U ovom segmentu su opisani detalji i ograničenja implementacije svakog od zadataka, kao i generalne informacije o implementaciji kompajlera.

Generalno

MiniJS je prilično ograničen jezik. Pravljen je po uzoru na MiniC kompajler i samim tim je naslijedio dosta ograničenja. Takođe, određena ograničenja postoje zbog činjenice da se MiniJS prevodi u asemblerski jezik za koji posjedujem prilično skromno znanje.

Jedini tip koji se može koristiti u MiniJS jesu brojevi (označeni kao *number*) koji funkcionišu kao *signed int*. Struktura je ograničena tako da se sve osim globalnih promjenljivih piše unutar funkcija.

Definisanje funkcija

Pojednostavljena sintaksa definisanja funkcija je implementirana tako da omogućava definisanje funkcije ključnom riječi function, kao i pojednostavljeno definisanje arrow funkcije. Ukoliko arrow funkcija ima samo jedan parametar, moguće ga je navesti bez zagrada.

Takođe, implementacija **podržava više parametara**, naime obrtanjem rednih brojeva parametara i smještanjem te vrijednosti u atr1 kolonu tabele simbola omogućeno je pristupanje parametrima unutar funkcije.

JavaScript deklaracije

Za razliku od promjenljivih deklarisanih upotrebom ključne riječi ***let,*** promjenljive deklarisane upotrebom ključni riječi ***const*** moraju biti inicijalizovane u momentu deklaracije i nije dozvoljeno ponovo im dodjeljivati ili mijenjati vrijednost.

Takođe, MiniJS proširuje funkcionalnost MiniC kompajlera tako što dozvoljava da se inicijalizuje vrijednosti promjenljivih tokom njihove deklaracije. Dozvoljava i deklarisanje **globalnih promjenljivih**, ali isključivo korišćenjem *let* ključne riječi, s obzirom da se radi samo o deklaraciji bez inicijalizacije vrijednosti.

Množenje

Množenje je implementirani tako da radi sa pozitivnim i negativnim brojevima. Jednostavno se vrši sabiranje (ili oduzimanje u slučaju negativnog drugog operanda) prvog operanda onoliko puta kolika je apsolutna vrijednost drugog.

Postincrement i postdecrement iskazi

Postinkrement i postdekrement su implementirani samo u formi iskaza. Takođe, postinkrement se kao takav koristi u definisanju for petlje.

Petlje

Petlje su implementirane po uzoru na petlje sa vježbi, uz dodatak ključnih riječi ***break*** i ***continue***. Ove ključne riječi su implementirane kao iskazi koji se mogu bilo kada napisati. Kada se naiđe na iskaz, u slučaju da se nalaze unutar neke petlje, na tu petlju će se i primijeniti. Ako se pak nalaze van petlje, ništa se ne dešava. Takođe, pamćenjem trenutne petlje i prethodne petlje *break* i *continue* ključne riječi se mogu koristiti unutar ugniježdenih petlji. U asemblerskom kodu se labele za sve petlje označavaju sa *@loop_*, kako bi ključne riječi radile unutar obje vrste petlji.

Ograničenje for petlje leži u tome da svaka for petlja mora definisati novu promjenljivu broja iteracije i isključivo je dozvoljeno inkrementiranje te promjenljive. Takođe biće prijavljena greška beskonačne petlje u slučaju da se petlje ponove previše puta.

Ternarni operator

Ternarni operator je ograničen tako što se uslov mora pisati unutar zagrada zbog zamršenog kružnog konflikta koji nastaje zbog načina kako su izrazi definisani.

Ideje za nastavak

S obzirom na prirodu JavaScript jezika postoji veliki broj interesantnih mogućih proširenja ovog kompajlera. Naime, navešću samo neka od njih.

Script?

Jezik je trenutno ograničen tako da se sav kod izvršava unutar funkcija i započinje unutar main() funkcije. Jedna prednost skriptovnih jezika jeste da se korišćenjem njih kod može pisati i izvršavati bez spomenute krute C-ovske funkcijske strukture. Razmišljao sam o rješavanju ovog problema, ali zbog toga što nisam bio siguran kako sam Hipsim generiše i analizira asemblerski kod, nisam se upuštao u to.

Funkcije

JavaScript koristi funkcije na svaki mogući način. Funkcije, koje mogu biti i **anonimne**, se mogu prosljeđivati drugim funkcijama kao parametri, vraćati kao povratne vrijednosti, smještati u promjenljive itd.

U sintaksi MiniJS kompajlera, u slučaju arrow sintakse ili sintakse funkcijskih izraza stvara se privid da se sama referenca na funkciju smješta unutar neke promjenljive. To u implementaciji pak nije slučaj, odnosno ta ‘promjenljiva’ služi samo da *definiše ime funkcije*.

Imao sam na umu proširiti semantiku tako da se promjenljive koje ‘čuvaju referencu na funkciju’ definisane ključnom riječi *let* mogu redefinisati (za razliku od *const*) i možda da se reference na druge funkcije mogu slobodno dodjeljivati promjenljivim. To bih vjerovatno pokušao čuvajući neku vrstu reference u tabeli simbola, ali nisam siguran koliko bih uspješan bio u tome.

Operacije

Naravno, pored samog množenja bilo bi interesantno implementirati i druge operacije, na primjer dijeljenje, ostatak pri dijeljenju (%), kvadriranje (**) i tako dalje.

Takođe, za sve implementirane operacije ne bi bilo pretjerano teško dodati i operatore dodjele (+=, -=, *=, /=. %=, **=).

Nizovi

Iako sam JavaScript podržava nizove visokog nivoa koji mogu sadržati različite vrste objekata unutar njih, bilo bi dobro za MiniJS implementirati nizove niskog nivoa, čija dužina bi se definisala pri njihovoj deklaraciji i koji bi sadržali isključivo brojeve.

Literatura

- *https://www.programiz.com/javascript/online-compiler/*