**DOKUMENTACIJA**

Programski prevodioci - predmetni zadatak

**Osnovni podaci**

|  |  |  |
| --- | --- | --- |
| Broj indeksa | Ime i prezime | Tema |
| SW22/2019 | Teodor Sakal Francišković | Proširenje miniC jezika: lambda funkcije, aritmetički operatori (množenje i deljenje) |

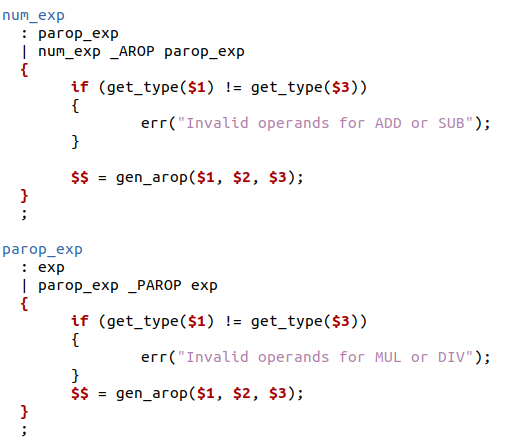
**Korišćeni alati**

|  |  |
| --- | --- |
| Naziv | Verzija |
| Bison, Flex, Hipsim | Verzija korišćena na vežbama 😊 |

## Evidencija implementiranog dela

* **Množenje i deljenje** – sintaksa, semantika, generisanje koda
* **Deklaracija lambda funkcije** – sintaksa, semantika, generisanje koda
* **Poziv lambda funkcije** – sintaksa, semantika, generisanje koda
* **Dodela vrednosti lambda funkcije nekoj promenljivoj** – sintaksa, semantika, generisanje koda
* **Lambda funkcija kao povratna vrednost (u *return* *statement-*u)** – sintaksa, semantika, generisanje koda

## Detalji implementacije i nedostaci

* **Množenje i deljenje**
  + Sintaksa, semantika, generisanje koda:
    - Dodate su oznake **\*** i **/**, za koje se *yylval.i* vrednost postavlja na MUL i DIV respektivno, a vraća se token \_PAROP.
    - Proširen je ***num\_exp*** na sledeći način: 
    - Ovime je postignuto da se operacije množenja i deljenja uvek prve izvrše, s obzirom da imaju prioritet. Funkcija **gen\_arop** je kod za generisanje koda koji je bio dat u početnoj verziji micko.y fajla, ali je prebačen u codegen.h/.c fajl.
    - Množenje i deljenje se može izvršiti između dva **exp**-a, koji su istog tipa.
  + Test primeri
    - Sintaksa, semantika:
      * U folderu **semantic\_tests/mul\_div** se nalaze test primeri za proveru sintakse i semantike množenja i deljenja.
      * test-ok1.mc, test-ok2.mc, test-ok3.mc reprezentuju rad sa \_AROP, pa \_PAROP, a zatim i jednih i drugih operacija, respektivno.
      * test-synerr1.mc, test-synerr2.mc, test-synerr3.mc reprezentuju sintaksne greške, gde fale operandi pri množenju i deljenju.
      * test-semerr1.mc, test-semerr2.mc, test-semerr3.mc reprezentuju semantičke greške u radu sa različitim tipovima operanada pri svim aritmetičkim operacijama implementiranih u micko.y fajlu.
    - Generisanje koda:
      * U folderu **code\_generation\_tests/mul\_div** se nalaze test primeri za generisanje koda gde se upotrebljuju operacije množenja i deljenja.
      * Postoji 5 fajlova, oblika test-okX.mc, gde je X broj između jedan i pet, gde se vrše različite operacije među operandima, koristeći i pozive funkcija, kao i zagrade.
* **Lambda funkcije**
  + Deklaracija lambda funkcije
    - Sintaksa:
      * Dodati su tokeni **lambda**, čija je povratna vrednost \_LAMBDA, token ’**:**’, čija je povratna vrednost \_COLON, kao i token ’**,**’ čija je povratna vrednost \_COMMA.
      * Sintaksa deklaracije lambda funkcije je sledećeg oblika: **\_LAMBDA \_ID \_ASSIGN lambda\_exp \_SEMICOLON**, odnosno, u samom kodu *“lambda x = int a, int b: a + b;”*. Pojam *lambda\_exp* je predstavljen kao više deklaracije više parametara, dvotačke i nekim numeričkim izrazom sastojanim od tih pojmova.
      * U tabeli simbola su pamćena dva nova tipa podataka:
        + **LAMBDA\_FUN**

*Name* je ime lambda funkcije.

*Kind* je LAMBDA\_FUN.

*Type* je povratni tip lambda funkcije.

*Atr1* je broj parametara lambda funkcije.

*Atr2* je redni broj lambda funkcije u kodu.

* + - * + **LAMBDA**

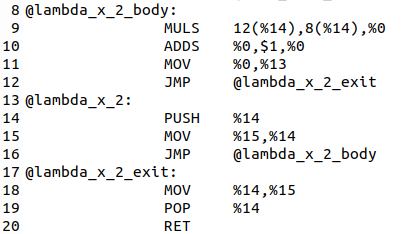
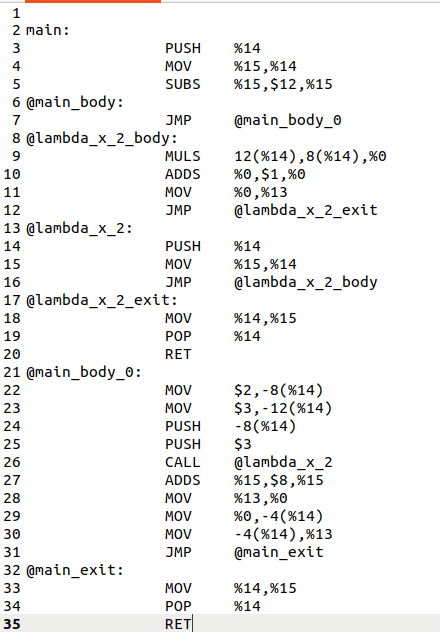
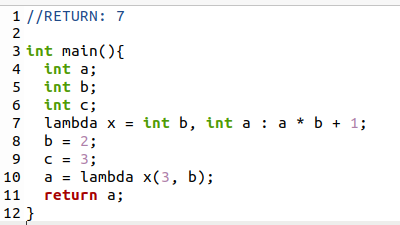
*Name* je ime lambda parametra.

*Kind* je LAMBDA.

*Type* je tip lambda parametra.

*Atr1* je redni broj lambda funkcije kojoj parametar pripada.

*Atr2* je redni broj parametra u lambda funkciji.

* + - * Dodate su dve nove funkcije u **symtab.h/.c** fajlovima, **lookup\_lambda\_symbol** i **lookup\_lambda\_function**, a razlika u odnosu na **lookup\_symbol** funkciju jeste ta što ove dve funkcije pored poređenja atributa *name* i *kind*, vrše poređenje i po atributu *atr1*.
    - Semantika:
      * Nemoguće je definisati dve lambda funkcije sa istim nazivom i sa istim brojem parametara. Znači, lambda funkcija x sa dva parametra, i lambda funkcija x sa 3 parametra je validna semantika.
      * Nemoguće je definisati parametre sa istim nazivom u okviru jedne lambda funkcije. Znači, ne mogu postojati dva parametra *a* u jednoj lambda funkciji.
      * Parametri jedne lambda funkcije ne mogu biti različitog tipa, jer aritmetičke operacije sa različitim tipovima nisu podržane u micko.y fajlu. Znači, parametri *unsigned a* i *int b* se ne mogu naći u okviru parametara jedne lambda funkcije.
      * Nemoguće je koristiti neki parametar u okviru lambda funkcije, a da taj parameter prethodno nije deklarisan. Znači, ako je jedini parametar lambda funkcije *a,* a u telu lambda funkcije se nalazi izraz *a + b*, biće ukazano na semantičku grešku.
      * Podržan je rad sa više parametara u okviru lambda funkcije. O nedostacima ove funkcionalnosti će više biti rečeno kasnije.
    - Generisanje koda:
      * Generisanje koda deklarisanja jedne lambda funkcije će biti prikazano na sledećem primeru: 
      * Labele vezane za lambda funkcije se sastoje iz tri dela:
        + *lambda\_X\_Y*,
        + *lambda\_X\_Y\_body*,
        + *lambda\_X\_Y\_exit*, gde X predstavlja naziv lambda funkcije, a Y broj parametara lambda funkcije.
      * Lambda funkcije suštinski funkcionišu veoma slično kao obične funkcije micka kada je u pitanju generisanje koda. Jedina razlika je u tome što lambda funkcije nisu definisane pre *main* funkcije, pa je potrebno navigirati korektno kroz njih pomoću JMP-ova.
      * Napravljena je i modifikacija u generisanju koda vezanog za *main* funkciju, gde je implementirano JMP-ovanje na različite labele oblika *main\_body\_redniBroj,* u slučaju kreiranja više lambda funkcija u jednom fajlu. Više detalja o prethdno spomenutoj modifikaciji se može videti u test primerima.
      * Funkcija **gen\_sym\_name** u fajlu codegen.h/.c je takođe modifikovana, kako bi se mogli generisati lambda parametri na korektan način.
    - Test primeri
      * U folderu **semantic\_tests/lambda** se nalaze svi primeri vezani za lambda funkcije. U ovom pasusu će biti samo reči o delu vezanom za definisanje lambda funkcija.
      * U test-okX.mc fajlovima, gde je X vrednost između jedan i tri, se nalaze korektni načini definisanja lambad funkcija.
      * U test-synerr1.mc fajlu nedostaje tip jednog od parametara lambda funkcije, dok u test-synerr2.mc fajlu je dodeljen tip lambda funkciji.
      * U test-semerr1.mc fajlu su definisane dve lambda funkcije sa istim nazivom i sa istim brojem parametara, u test-semerr2.mc fajlu je definisana lambda funkcija sa dva ista parametra, u test-semerr4.mc fajlu se u lambda funkciji koristi parametar koji nije prethodno definisan, dok se u test-semerr7.mc prosleđuju dva parametra različitog tipa, pa sabiranje u telu lambda funkcije ne može da prođe.
  + Poziv lambda funkcije
    - Sintaksa:
      * Dodat je novi pojam **lambda\_call**, koji je oblika \_LAMBDA \_ID \_LPAREN lambda\_arguments \_RPAREN, odnosno, primer toga je a = “lambda x(1, 2)”;.
      * lambda\_arguments jeste skup od jednog ili više lambda argumenta, gde je jedan lambda argument zapravo num\_exp.
    - Semantika:
      * Nemoguće je pozvati lambda funkciju koja prethodno nije deklarisana.
      * Nemoguće je pozvati lambda funkciju i proslediti joj netačan broj argumenata.
      * Nemoguće je proslediti argument koji nije istog tipa kao lambda funkcija.
    - Generisanje koda:
      * Najznačajnije detalje generisanja koda ćemo proći kroz sledeći primer: 
      * Na slikama iznad se nalaze sadržaji fajlova test-ok1.mc iz foldera code\_generation\_tests/lambda i test-ok1.asm, respektivno.
      * Glavna modifikacija koja je morala biti urađena, u odnosu na već implementirane funkcije u micko.y fajlu, jeste *push-*ovanje argumenata pri pozivu lambda funkcije. S obzirom da u originalnoj verziji nije implementiran rad sa funkcijom sa više parametara, sada je bilo potrebno proširiti logiku rada sa više parametara kod lambda funkcija. Problem je nastajao u tome što se prvi argument lambda poziva *push*-ovao prvi, dok bi u samom izvršavanju tela lambda poziva taj parametar postao poslednji (znači, ako su postojala 3 argumenta u lambda pozivu, vrednost prvog argumenta bi bila dodeljena trećem). Prethodno spomenuti problem nije nastajao pri radu sa (lambda) funkcijama sa jednim argumentom iz očiglednih razloga. Ideja za rešavanje ovog problema jeste da se prvobitno MOV-uju sve vrednosti, a nakon toga PUSH-uju u suprotnom smeru, kako bi prvi argument lambda poziva bio PUSH-ovan poslednji, odnosno skinut prvi sa *stack*-a. Ova modifikacija je stvorila novi problem, o kojem će više biti rečeno ubrzo.
    - Test primeri
      * test-semerr3.mc demonstrira slučaj poziva lambda funkcije gde nisu svi parametri odgovarajućeg tipa, test-semerr5.mc demonstrira pozivanje nedefinisane lambda funkcije, dok test-semerr6.mc demonstrira prosleđivanje lošeg broja parametara lambda funkciji.
      * Testovi koji se odnose na generisanje koda, koji se nalaze u **code\_generation\_tests/lambda**, demonstriraju slučajeve kada je kreirano više lambda funkcija, kada je ista lambda funkcija pozvana nekoliko puta, kada su prosleđeni argumenti oblika nekog *num\_exp*-a (na primer, poziv funkcije), kada je kreirana lambda funkcija sa istim nazivom, ali različitim brojem argumenata, itd.
    - **Nedostatak 1**
      * Postoji problem pri prosleđivanju više od jednog *num\_exp*-a u pozivu lambda funkcije.
        + Zašto? – Prethodno spomenuto PUSH-ovanje argumenata tek nakon odrađenih svih MOVE-ova argumenata u registre stvara problem jer se iz prvog slobodnog registra pushuju vrednosti na stack, a u slučaju postojanja više *num\_exp*-ova, PUSH-uje se vrednost iz registra %0 (ili prvog slobodnog), onoliko puta koliko *num\_exp*-ova postoji.
        + Ideja za rešavanje problema: Potrebno je odraditi PUSH-ovanje registra nakon svakog MOV-a. Na linku <https://pastebin.com/0zdiK2ep> je dat kod, gde je pokušano da se reši problem. U pojmu **lambda\_call**, na 438. liniji koda, je pokušano da se iz globalnih listi, koje su popunjavane u **lambda\_argument**, na 543. liniji koda, izvuku jedan po jedan argument iz liste i da se izgeneriše sav potreban kod u slučaju da je argument **num\_exp** (detaljniji postupak će biti objašnjen u razgovoru uživo). Međutim, dati kod nije uspeo u potpunosti da odradi šta je zamišljeno, tako da je ovo ostavljeno za dalje razmatranje u budućim projektima.
    - **Nedostatak 2**
      * Postoji problem kada se pozove više (lambda) funkcija u jednoj liniji koda.
        + Zašto? – U slučaju postojanja dva lambda poziva u jednoj liniji koda, nakon što se izvrši prvi lambda poziv, vrednost se čuva u %0 (slobodnom registru). U izračunavanju vrednosti vezane za drugi lambda poziv, nova vrednost biva prebačena u registar %0. Recimo da želimo da saberemo vrednosti ova dva lambda poziva. Registar %0 sada ne sadrži više vrednost iz prvog lambda poziva, već sadrži vrednost drugog lambda poziva, dok će sledeći slobodan registar (%1), takođe sadržati tu vrednost, pa sabiranje neće biti korektno izvršeno.
        + Ideja za rešavanje problema: Teorijski, potrebno je izmeniti način na koji registri bivaju zauzimani, tako da ne dođe do poklapanja čuvanja vrednosti među registrima. Praktično, u fajlovima codegen.h/.c bi bilo potrebno izmeniti logiku funkcija **gen\_sym\_name**, **free\_reg**, **take\_reg**, tako da ne dolazi do poklapanja čuvanja vrednosti u registrima, dok bi takođe bilo potrebno obratiti pažnju da izmenjene funkcije korektno zauzimaju registre i za sve ostale moguće slučajeve korišćenja ovih funkcija.
  + Dodela vrednosti lambda funkcije nekoj promenljivoj
    - Sintaksa:
      * Sintaksa je identična kao pri bilo kojoj drugoj dodeli vrednosti nekom parametru, pogledati primer sintakse u prethodnom pasusu vezanu za sintaksu.
    - Semantika:
      * Nemoguće je dodeliti vrednost lambda funkcije nekoj promenljivoj koja nije istog tipa kao dodeljena lambda funkcija, odnosno njena vrednost.
    - Testovi
      * U gotovo svim testovima se može videti dodela vrednosti lambda funkcija nekoj promenljivoj.
  + Lambda funkcija kao povratna vrednost (u *return* *statement-*u)
    - Kako je pojam **lambda\_call** dodat u **num\_exp** kao jedan od mogućih pojmova, nije bilo nikakvih modifikacija van onoga što je spomenuto u prethodnim pasusima.
    - Testovi
      * test-ok7.mc demonstrira poziv lambda funkcije u **return statement**-u**.**
* **Finalne napomene**
  + U priloženom kodu su takođe dodavane razne globalne promenljive, koje su bile korišćene u izradi projekta. Za dublje razumevanje svake od njih je potrebno izučiti priložen kod.

## Ideje za nastavak

Pogledati prethodno spomenute nedostatke, obeležene crvenom bojom. Nad njima bi se mogao dalje razvijati ovaj projekat. Takođe bi se projekat mogao produbiti novim aritmetičkim operatorima, kao što su ceo deo, moduo, inkrementi, stepenovanje...

## Literatura

* <https://web.iitd.ac.in/~sumeet/flex__bison.pdf> - Korišćen za razvijanje logike oko operatora množenja i deljenja, konkretno strana 30.
* Razni sajtovi koji objašnjavaju izgled lambda funkcija u različitim programskim jezicima. Za samu implementaciju generisanja koda, inspiracija je delimično uzeta od već odrađenog dela za generisanje koda funkcija u micko.y fajlu.