

## OBJEKTNO ORIJENTISANO PROGRAMIRANJE

- projektni zadatak za školsku 2015/2016. godinu -

Implementirati na programskom jeziku C++ simulator mašine sa protočnom obradom podataka (engl. *data flow*). Potrebno je implementirati prevodilac koji prevodi jednostavne programe naredbi u međufornu, koju mašina sa protočnom obradom, koja se simulira, može da izvrši.

### **Opis arhitekture protočne obrade (*data flow architecture*).**

Paradigma protočne obrade daje potpuno drugačiji koncept izvršavanja programa za razliku od standardne Von Neumanove arhitekture, a koja se zasniva na kontroli toka. Kod protočne paradigme, izvršavanje operacija je uslovljeno isključivo zavisnostima po podacima. Iako nije široko prihvaćena kao Von Neumanova arhitektura, protočna paradigma prirodnija je za rešavanje problema koji se mogu opisati grafom operacija zavisnih po podacima. U nastavku teksta je dat opis koncepta, podeljen u nekoliko manjih sekcija.

### **Operacije na hardverskom nivou**

**Apstraktna operacija (*Operation*)** može da ima proizvoljno mnogo operanada, ima jedinstven identifikator, može da se izvrši i proizvede rezultat. Operacija daje rezultat sa kašnjenjem  $T$ , koje je specifično za svaku konkretnu operaciju. Operacija dobija vrednost jednog operanda preko jednog ulaznog porta (*input port*). Vrednost operanda može se dobiti zadavanjem konstante, promenljive ili kao rezultat neke druge operacije (zavisnost operacija po podacima). Izvršavanje operacije ne može da počne sve dok se na njenim ulaznim portovima ne pojave vrednosti svih operanada (dok se ne završe sve operacije od kojih operacija zavisi po podacima). Ukoliko operacija ne zavisi od drugih operacija po podacima, najraniji trenutak za početak njenog izvršavanja je početak programa. Kada su svi ulazni operandi neke operacije raspoloživi na njenim ulaznim portovima, operacija se može izvršiti; svojim izvršavanjem operacija „konzumira“ sve svoje ulazne operande koji time više nisu „raspoloživi“ za neko novo izvršavanje iste operacije. Sve operacije koje ispunjavaju uslov da se izvrše izvršavaju se paralelno. Operacija svoj rezultat šalje preko ulaznih portova vezanih operacija zavisnih od nje. Operacije se povezuju u mrežu tako što se jednoj operaciji se može definisati ulazni port druge operacije kojoj prva treba da se prosledi svoj rezultat.

**Aritmetička operacija (*Arithmetic Operation*)** ima najviše dva ulazna operanda i proizvodi jedan izlazni rezultat. Sabiranje je binarna, levo asocijativna aritmetička operacija čiji rezultat je zbir vrednosti operanada ( $T=5ns$ ). Množenje je binarna, levo asocijativna aritmetička operacija čiji je rezultat proizvod vrednosti operanada ( $T=8ns$ ). Stepenuvanje je binarna aritmetička operacija čiji je levi operand promenljiva koja se stepenuje, a drugi je eksponent ( $T=10ns$ ). Postavljanje vrednosti promenljive je binarna operacija čiji je levi operand promenljiva, a desni operand je realna vrednost ( $T=3ns$ ).

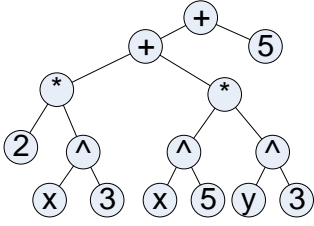
### **Izrazi i naredbe na softverskom nivou**

**Izraz (*Expression*)** čine konstante i promenljive povezane operatorima  $+$ ,  $*$ ,  $^$ . Specijalni oblici izraza su konstanta i promenljiva. Naredba dodele vrednosti omogućava dodelu vrednosti izraza na desnoj strani promenljivoj na levoj strani. Prioriteti operatora od najmanjeg ka najvećem su redom  $+$ ,  $*$ ,  $^$ .

**Program** se piše u tekstualnom fajlu i čini ga niz naredbi dodele vrednosti. Svaka naredba dodele vrednosti se piše u jednoj liniji teksta. Svaka promenljiva je predstavljena samo jednim slovom. Za jednu promenljivu sme postojati samo jedna dodela vrednosti (tzv. pravilo samo jedne dodele, engl. *single assignment rule*). Smatrati da će taj uslov biti ispoštovan u test primerima, pa nije neophodno programirati takvu proveru. Sintaksa ulaznog programa je data u prilogu specifikacije.

**Prevodilac (*Compiler*)** prevodi napisani program. Izlaz prevodioca je tekstualni fajl čije ime je isto kao ime ulaznog fajla, s tim što mu je ekstenzija *.imf* (*intermediate form*). Sintaksa fajla prevoda je data na kraju specifikacije. U svakom redu prevedenog fajla nalazi se po jedna operacija sa svojim

operandima. Za prevođenje složenih izraza preporučuje se korišćenje koncepta postfiksno zapisa izraza i/ili sintaksnog stabla.

<p>Za izraz oblika:  <math>2 * x^3 + x^5 * y^3 + 5</math>          Postfiksni oblik izraza je:  <math>2 \ x \ 3 \ ^ \ * \ x \ 5 \ ^ \ y \ 3 \ ^ \ + \ 5 \ +</math>  <i>Može se dobiti postorder obilaskom sintaksnog stabla - proveriti!</i></p>	<p>Sintakšno stablo:</p> 
--	---

**Mašina (Machine)** može da učita i izvrši program iz zadanog .imf fajla (*exec(string file):void*). Program se izvršava tako što se učitaju operacije, napravi se graf operacija (zavisnosti po podacima), i operacije se rasporede u red za izvršavanje, tako što se prvo stavljaju sve one operacije čiji su svi operandi izračunati. Mašina promenljive čita/piše iz/u memorije. Mašina izvršava program sve dok se red operacija ne isprazni, pri čemu se mora ispoštovati da svaka operacija svoj rezultat daje tek nakon isteka vremena kašnjenja. Mašina treba da ostavlja trag izvršavanja u log fajlu, koji ima isto ime kao ulazni fajl prevedenog programa, samo što mu je ekstenzija .log. U log fajlu, mašina za svaku izvršenu operaciju treba da ispiše trenutak kada je stekla uslove da započne izvršavanje, kao i trenutak u kom se izračunavanje završilo. Sva vremena su u nanosekundama.

**Memorija (Memory)** omogućava pisanje i čitanje promenljivih. Može se postaviti vrednost promenljive (*set(string varName, double val)*), kao i pročitati vrednost promenljive (*get(string varName):double*). Ukoliko se čita promenljiva koja ne postoji u memoriji, prijavljuje se greška izuzetkom *VariableNotFoundException*. Sme postojati samo jedna memorija u sistemu. Pristup podacima u memoriji se izvršava trenutno bez kašnjenja.

Napisati glavni program koji preko argumenata komandne linije prihvata ulazni fajl programa, poziva prevodioca, koji program prevede u .imf fajl, a zatim pozove mašinu da učita prevedeni fajl i izvrši operacije.

## Prilog

Program	Prevod (.imf)	Trag izvršavanja u .log fajlu
$x = 2$	(1) = x 2	(1) 0ns/3ns
$y = 3$	(2) = y 3	(2) 0ns/3ns
$z = 2 * x^3 + x^5 * y^3 + 5$	(3) ^ x 3	(3) 3ns/13ns
	(4) * 2 (3)	(5) 3ns/13ns
	(5) ^ x 5	(6) 3ns/13ns
	(6) ^ y 3	(4) 13ns/21ns
	(7) * (5) (6)	(7) 13ns/21ns
	(8) + (4) (7)	(8) 21ns/26ns
	(9) + (8) 5	(9) 26ns/31ns
	(10) = z (9)	(10) 31ns/36ns
		Svaka operacija se naravno izvršava čim pre može.

U prevedenoj formi svaka operacija se zapisuje u posebnom redu. Za svaku operaciju se prvo navodi njen jedinstveni identifikator u zagradama, zatim operator i operandi. Ukoliko je operand neke operacije rezultat druge operacije, onda se taj operand zapisuje navođenjen identifikatora operacije u zagradama, kao što je prikazano u tabeli. Delovi operacije su razdvojeni po jednim znakom razmaka.

## **Tehnički zahtevi**

1. Programski sistem realizovati tako da bude detaljno komentarisano, modularan i lako proširiv novim operacijama.
2. Za obradu i oporavak od eventualnih grešaka MORA se koristiti mehanizam izuzetaka.
3. Simulator MORA biti implementiran korišćenjem jezgra simulatora diskretnih događaja (*simulation engine*), na koji je nadograđen kod za simulaciju protoka saobraćaja (TSS). Studentima NIJE DOZVOLJENO da menjaju izvorni kod jezgra simulatora, osim ako je to zaista neophodno, što će utvrditi i uraditi nastavnici. Dato jezgro (`simulationEngine.lib`) je dostupno na sajtu predmeta. Simulator digitalnih mreža treba implementirati kroz nadogradnju jezgra, a po uzoru na TSS. Izvorni kod biblioteke je dostupan studentima i namenjen je isključivo za analizu implementacije postojećih funkcionalnosti.

## **Napomene**

1. Ukoliko u zadatku nešto nije dovoljno jasno definisano, treba usvojiti razumnu pretpostavku i na temeljima te pretpostavke nastaviti izgrađivanje rešenja.
2. Prateća dokumentacija i test primeri potrebni za izradu simulatora su na sajtu predmeta.
3. Studentima se preporučuje da dobro analiziraju implementaciju zadate biblioteke, pre nego što počnu implementaciju simulatora.
4. Za implementaciju rešenja mora se koristiti razvojno okruženje MSVS, verzija 201x.
5. Za uspešno odbranjen projektni zadatak potrebno je na odbrani pokazati sledeće datoteke:
  - `oop_pr.cpp` – izvorni tekst glavnog programa na jeziku C++;
  - `oop_pr.vcproj` – informacije o projektu koji sadrži sve potrebno za glavni program;
  - `oop_pr.sln` – informacije o svim projektima relevantnim za projektni zadatak;
  - sve datoteke koje sadrže deklaracije i definicije korišćenih klasa;

12.12.2015. godine

*Sa predmeta*