# OBJEKTNO ORIJENTISANO PROGRAMIRANJE

## - projektni zadatak za školsku 2019/2020. 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 međuformu, 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 Fon Nojmanove arhitekture, a koja se zasniva na kontroli toka. Kod protočne paradigme, izvršavanje operacija je uslovljeno isključivo zavisnostima po podacima. Iako nije široko prihvaćena kao Fon Nojmanova arhitektura, protočna paradigma prirodnija je za rešavanje problema koji se mogu opisati grafom operacija zavisnih po podacima. U nastavku teksta je dat opis koncepata, podeljen u nekoliko manjih sekcija.

### *Operacije na hardverskom nivou*

***Token*** ima jedinstveno ime, realnu vrednost i identifikator operacije koja je proizvela vrednost.

***Apstraktna operacija (Operation)*** 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. Operand se operaciji prosleđuje kao token preko ulaznog porta (*input port*). Izvršavanje operacije ne može da počne sve dok se na svim njenim ulaznim portovima ne pojave operandi (tokeni), odnosno dok se ne završe operacije od kojih ona 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 raspoloživi na ulaznim portovima, operacija može istog trenutka (ali i kasnije) da započne svoje izvršavanje. Na početku izvršavanja operacija uklanja tokene na ulaznim portovima. Sve operacije koje u nekom trenutku ispunjavaju uslov za izvršavanje, treba da se izvršavaju paralelno. Kada se završi, operacija pravi token sa rezultatom.

***Aritmetička operacija (ArithmeticOperation)*** ima najviše dva ulazna operanda i proizvodi jedan izlazni rezultat. Sabiranje je binarna aritmetička operacija čiji rezultat je zbir vrednosti operanada sa kašnjenjem *Ta*. Množenje je binarna aritmetička operacija čiji je rezultat proizvod vrednosti operanada sa kašnjenjem *Tm*. Stepenovanje je binarna aritmetička operacija čiji je levi operand promenljiva koja se stepenuje, a drugi je eksponent. Stepenovanje je desno asocijativna operacija, dok za sabiranje i množenje redosled grupisanja nije bitan. Ova operacija proizvodi rezultat sa kašnjenjem *Te*. Postavljanje vrednosti promenljive je binarna operacija čiji je levi operand promenljiva, a desni operand je realna vrednost. Operacija proizvodi rezultat sa kašnjenjem koje je definisano za upis u memoriju *Mw*.

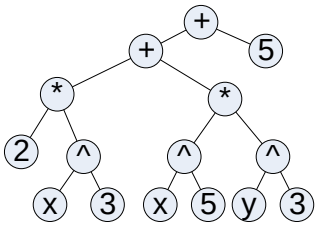
### *Izrazi i naredbe na softverskom nivou*

***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. *singe assignment rule*). Primer ulaznog programa je dat u prilogu.

***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 u prilogu. U svakom redu prevedenog fajla nalazi se specifikacija jedne operacije u troadresnom formatu. Svaka operacija ima jedinstveni identifikator posle kog se navodi operator, token rezultata, a zatim jedan ili dva operanda, u zavisnosti od operacije. Delovi operacije su

razdvojeni po jednim znakom razmaka. Predvideti dve strategije grupisanja podizraza: jednostavna strategija (*SimpleCompilation*) koja podizraze grupiše u redosledu u kom su navedeni u programu i napredna (*AdvancedCompilation*) koja podizraze uređuje tako da se minimizuje kašnjenje i dobije što ranija aktivacija za svaku operaciju. U svakom slučaju prevod mora da garantuje logičku ispravnost programa. Strategija je data u konfiguracionom fajlu kao u primeru datom u prilogu.

| | |
|---|---|
| Za izaz oblika:<br>2*x^3 + x^5*y^3 + 5<br>Postfiksni oblik izraza je:<br>2 x 3 ^ * x 5 ^ y 3 ^ * + 5 +<br>*Može se dobiti postorder obilaskom sintaksnog stabla - proveriti!* | Sintaksno stablo:<br>  |

***Mašina (Machine)*** može da učita i izvršava program iz zadatog .imf fajla *exec(string file):void*. Sadrži rezervoar (*pool*) operacija koje se izvršavaju (*executing*), rezervoar operacija koje čekaju izvršavanje (*waiting*), kao i rezervoar operacija koje su završile (*completed*). Program se izvršava tako što se učitaju operacije, spremne se ubacuju u rezervoar za izvršavanje, dok se ostale stavljaju u rezervoar za čekanje. Mašina preuzima token rezultata svake završene operacije i ažurira portove operacija koje zavise od tog rezultata. Ako operacija vrši upis u memoriju, portovi zavisnih operacija dobijaju token tek kada se završi upis podatka. Mašina stopira distribuciju tokena sve dok se vrednost ne upiše u memoriju. Mašina izvršava program sve dok se rezervoar za čekanje ne isprazni. Ima definisan fajl zapisnika izvršavanja (*execution log*) u kom se zapisuju sva izvršavanja operacija i upisi/čitanja iz memorije i fajl sa vrednostima promeljivih iz memorije nakon izvršavanja. Oba fajla treba da imaju isto ime kao ulazni fajl prevedenog programa, s tim što je ekstenzija fajla zapisnika izvršavanja .log, a ekstenzija fajla sa vrednostima promenljivih .mem. Zapis izvršavanja operacije sadrži njen identifikator, trenutak početka izvršavanja, kao i trenutak završetka, kao što je dato u primeru u prilogu.

***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 *VarNotAvailableException*. Memorija dozvoljava maksimalno *Nw* paralelnih upisa. Sme postojati samo jedna memorija u sistemu.

Napisati glavni program koji preko argumenata komandne linije prihvata ulazni fajl programa i konfiguracioni fajl (videti prilog), poziva prevodioca, koji program prevede u .imf fajl, a zatim pozove mašinu da učita prevedeni fajl i izvrši operacije. Ostali fajlovi se prave na osnovu imena ulaznog fajla programa, sa odgovarajućim ekstenzijama u istom direktorijumu gde se ulazni fajl nalazi.

## *Prilog*

U sledećoj tabeli prikazano je izvršavanje programa na osnovu zadatih konfiguracionih parametera. Redosled operacija u zapisniku se pravi na sledeći način: operacije se najpre uređuju rastuće po: vremenu početka, zatim vremenu završtka, i na kraju rednim brojem.

<table>
<tr><th>Konfiguracija</th><th>Program</th><th>Prevod (.imf)</th><th>Trag izvršavanja u .log fajlu</th></tr>
<tr>
<td rowspan="3">Ta = 5<br>Tm = 10<br>Te = 15<br>Tw = 20<br>Nw = 1<br>compilation = simple</td>
<td>x = 2<br>y = 3<br>z = 2*x^3 + x^5*y^3 + 5</td>
<td rowspan="3">[1] = x 2<br>[2] = y 3<br>[3] ^ t1 x 3<br>[4] * t2 2 t1<br>[5] ^ t3 x 5<br>[6] ^ t4 y 3<br>[7] * t5 t3 t4<br>[8] + t6 t2 t5<br>[9] + t7 t6 5<br>[10] = z t7</td>
<td rowspan="3">[1] (0-20)ns<br>[3] (20-35)ns<br>[5] (20-35)ns<br>[2] (20-40)ns<br>[4] (35-45)ns<br>[6] (40-55)ns<br>[7] (55-65)ns<br>[8] (65-70)ns<br>[9] (70-75)ns<br>[10] (75-95)ns</td>
</tr>
<tr><td>Vrednosti promenljivih (.mem)</td></tr>
<tr><td>x = 2<br>y = 3<br>z = 885</td></tr>
</table>

## *Tehnički zahtevi*

Simulator može 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 mašine sa protočnom obradom podataka 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. Studentima se preporučuje da dobro analiziraju implementaciju zadate biblioteke, pre nego što počnu implementaciju simulatora.

## *Napomene*

1. Ukoliko u zadatku nešto nije dovoljno jasno definisano, treba usvojiti razumnu pretpostavku i na temeljima te pretpostavke nastaviti izgrađivanje rešenja.
2. Test primeri potrebni za izradu simulatora su na sajtu predmeta.
3. 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;

21. 12. 2019. godine *Sa predmeta*