Dizajn programskih jezika

— Osnovna svojstva programskih jezika —

Milena Vujošević Janičić

Matematički fakultet, Univerzitet u Beogradu

Sadržaj

1 Uvod

Svojstva programskih jezika

- Šta je ono što čini dva programska jezika sličnim?
- Na osnovu čega neki jezik pripada nekoj paradigmi?
- Šta je ono što čini dva programska jezika različitim?
- Na osnovu čega neki jezik ne pripada nekoj paradigmi?

- Šta je zajedničko za sve programske jezike?
- Šta je ono što je specifično za svaki programski jezik?
- Šta je ono što je specifično za neki programski jezik?

Svojstva programskih jezika

- Postoje različita svojstva programskih jezika koja ćemo izučavati:
  - Leksika
  - Sintaksa
  - Semantika
  - Imena, doseg, povezanost
  - Kontrola toka, podrutine
  - Tipovi
  - Kompajlirani/Interpretirani jezici
  - Izvršavanje

2 Leksika, sintaksa, semantika

2.1 Leksika

Leksika

- Programski jezici moraju da budu precizni
- Leksika je podoblast sintakse koja se bavi opisivanjem osnovnih gradivnih elemenata jezika.
- U okviru leksike, definišu se reči i njihove kategorije
- U programskom jeziku, reči se nazivaju lekseme, a kategorije tokeni.
- U prirodnom jeziku, kategorije su imenice, glagoli, pridevi
- U programskom jeziku, tokeni mogu da budu identifikatori, ključne reči, operatori...

Leksika

- `a = 2 * b + 1` — lekseme su a, =, 2, *, +, 1 i b, a njima odgovarajući tokeni su identifikator (a i b), operator =, operator *, operator +, celobrojni literali 2 i 1.
- Neki tokeni sadrže samo jednu reč, a neki mogu sadržati puno različitih reči
- Programski jezik C sadrži više od 100 različitih tokena: 44 različite ključne reči, identifikatore, celobrojne vrednosti, realne vrednsoti, karakterske konstante, stringovske literale, dve vrste komentara, 54 operatora...
- Drugi moderni programski jezici (npr Ada, Java) imaj sličan nivo kompleksnosti tokena

Leksika

- Razlikujemo ključne reči i identifikatore: identifikator ne može biti neka od ključnih reči, npr ne možemo da napravimo promenljivu koja bi se zvala if ili while
- Postoje ključne reči koje zavise od konteksta, engl. *contextual keywords* koje su ključne reči na određenim specifičnim mestima programa, ali mogu biti identifikatori na drugim mestima.
- Na primer, u C# -u reč yield može da se pojavi ispred break ili return, na mestima na kojima identifikator ne može da se pojavi. Na tim mestima, ona se interpretira kao ključna reč, ali može da se koristi na drugim mestima i kao identifikator.

Leksika

- C# 4.0 ima 26 takvih konteksno zavisnih ključnih reči, C++11 ih takođe ima dosta.
- Većina je uvedena revizijom postojećeg jezika sa ciljem da se definiše novi standard: sa velikim brojem korisnika i napisanog koda, vrlo je verovatno da se neka reč već koristi kao identifikator u nekom postojećem programu. Uvođenjem kontekstualne ključne reči, umesto obične ključne reči, smanjuje se rizik da se postojeći program neće kompilirati sa novom verzijom standarda.

Leksika

- Reči su obično definišu regularnim izrazima
- Na primer, `[a-zA-Z_][a-zA-Z_0-9]*`
- Regularnim izrazima se definišu reči, dok se konačnim automatima prepoznaju ispravne reči.
- Generisanje je bitno programerima, a prepoznavanje kompajlerima
- Leksikom programa obično se bavi deo prevodioca koji se naziva leksički analizator: on dodeljuje ulaznim rečima odgovarajuće kategorije, što je bitno za dalji proces prevođenja.

2.2 Sintaksa

Sintaksa

- Sintaksa programskog jezika definiše strukturu izraza, odnosno načine kombinovanja osnovnih elemenata jezika u ispravne jezičke konstrukcije.
- Formalno, sintakse se opisuju kontekstno slobodnim gramatikama, a prepoznaju parserima (potisni automat)

Sintaksa

- Na primer, sledeća jednostavna gramatika definiše jednostavne aritmetičke izraze

```
<exp> ::= <exp> "+" <exp>
<exp> ::= <exp> "*" <exp>
<exp> ::= "(" <exp> ")"
<exp> ::= "a"
<exp> ::= "b"
<exp> ::= "c"
```

- Program u ovom jeziku je proizvod ili suma od 'a', 'b' i 'c', na primer ispravan izraz je a*(b+c)

Sintaksa

- Sintaksa modernih programskih jezika je često veoma slična
- Veliki broj jezik je nastao pod uticajem C-a
- Ključne reči su uglavnom preuzete iz engleskog jezika
- Često programski jezici koji pripadaju istoj paradigmi imaju sličnu sintaksu (često ali ne uvek, npr Lisp i Haskell imaju potpuno različite sintakse)
- Sintaksa aritmetičkih izraza je često veoma slična

Sintaksa - Ezoterični programski jezici

- Ezoterični programski jezici (eng. *esolang*) su programski jezici dizajnirani da testiraju granice dizajna programskih jezika.
- Njihova sintaksa je obično veoma kompleksna i potpuno drugačija
- Ovi jezici imaju čudne ciljeve: da budu maksimalno nerazumljivi, da budu što manji, da budu što teži za kompilaciju i debagovanje, da budu zabavni, da budu šaljivi...
- Njihov cilj nije da budu upotrebljivi, korisni niti da rešavaju neki konkretan problem, već da zabave i da ispitaju granice i mogućnosti programskih jezika
- Pored drugačije sintakse, često imaju i potpuno neočekivanu semantiku

Sintaksa - Ezoterični programski jezici

- Primer programa u programskom jeziku AsciiDots

```
/#$<.
*-[+]
\#1/
```

- Primer programa u programskom jeziku Befunge

```
"dlroW olleH">:v
              ^,_@
```

- Primer programa u programskom jeziku Brainfuck:

```
++++++++++[>+++++++>++++++++++>+++<<<-]>++.>+.+++++++
..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.
```

Sintaksa - Ezoterični programski jezici

- Primer programa u programskom jeziku LOLCODE:

```
HAI
CAN HAS STDIO?
VISIBLE "HAI WORLD!"
KTHXBYE
```

- Primer programa u programskom jeziku Rockstar:

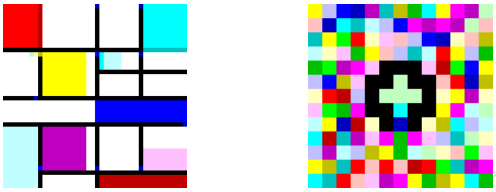
```
Put the whole of your heart into my hands
```

- Primer programa u programskom jeziku Unlambda:

```
`.!`.d`.l`.r`.o`.w`. `.,`.o`.l`.l`.e`.Hi
```

Sintaksa - Ezoterični programski jezici

- Primeri programa u programskom jeziku Piet:



Piet program koji štampa reč ”Piet” (levo) i ”Hello World” (desno).

2.3 Neformalna semantika

Semantika

- Semantika pridružuje značenje ispravnim konstrukcijama na nekom programskom jeziku.
- Semantika programskog jezika određuje značenje jezika.
- Obično je značajno teže definisati nego sintaksu.

Semantika

- Semantika može da se opiše formalno i neformalno, često se zadaje samo neformalno.
- Uloga neformalne semantike je da programer može da razume kako se program izvršava pre njegovog pokretanja.
- Na primer, semnatika nardbe `if(a<b) a++;` neformalno se opisuje sa „ukoliko je vrednost promenljive a manja od vrednosti promenljive b, onda uvećaj vrednost promenljive a za jedan"

2.4 Formalna semantika

Uloga formalne semantike

- Formalna semantika omogućava formalno rezonovanje o svojstvima programa
- Na primer, formalna semantika nekog jezika, zajedno sa modelom programa (tranzicionim sistemom) omogućava ispitivanje različitih uslova ispravnosti/korektnosti tog programa
- Različitim programskim jezicima prirodno odgovaraju različite semantike

Semantika

- Formalan opis značenja jezičkih konstrukcija
  - Operaciona semantika
  - Denotaciona semantika
  - Aksiomska semantika

Semantika

- Kompajler prevodi kod na mašinski kod u skladu sa zadatom semantikom jezika.
- Tokom kompilacije, vrši se proveravanje da li postoji neka semantička greška, tj situacija koja je sintaksno ispravna ali za konkretne vrednosti nema pridruženo značenje zadatom semantikom.
- Neki aspekti semantičke korektnosti programa se mogu proveriti tokom prevođenja programa — na primer, da su sve promenljive koje se koriste u izrazima definisane i da su odgovarajućeg tipa),
- Neki aspekti semantičke korektnosti se mogu proveriti tek u fazi izvršavanja programa — na primer, deljenje nulom, pristup elementima niza van granica...

Semantika

- Semantičke provere se mogu podeliti na statičke (provere prilikom kompilacije) i dinamičke (provere prilikom izvršavanja programa).
- Statički se ne može pouzdano utvrditi ispunjenost semantičkih uslova, tako da je moguće da se neke greške ne utvrde, iako prisutne, kao i da se neke greške pogrešno utvrde iako nisu prisutne i da rezultiraju nepotrebnim proverama prilikom izvršavanja programa.
- Različitim programskim jezicima odgovaraju izvršni programi koji sadrže različite nivoe dinamičkih provera ispravnosti.

3 Imena, povezanost, doseg

3.1 Imena

Imena

- Ime — string koji se koristi za predstavljanje nečega (promenljive, konstante, operatora, tipova...)
- Prvi programski jezici imali su imena dužine jednog karaktera (po uzoru na matematičke promenljive). Fortran I je to promenio dozvoljavajući šest karaktera u imenu.
- Fortran 95 i kasnije verzije Fortrana dozvoljavaju 31 karakter u imenu, C99 nema ograničenja za interna imena, ali samo prvih 63 je značajno. Eksterna imena u C99 (ona koja su definisana van modula i o kojima mora da brine linker) imaju ograničenje od 31 karaktera. Imena u Javi, C# i Adi nemaju ograničenja dužine i svi karakteri su značajni. C++ ne zadaje limit dužine imena, ali implementacije obično zadaju.

Imena

- Imena u većini programskih jezika imaju istu formu - slova, cifre i podvlaka
- Podvlaka se sve češće zamenjuje kamiljom notacijom
- U nekim jezicima imena moraju da počnu specijalnim znacima (npr PHP ime mora da počne sa $)
- Imena su najčešće case sensitive, što može da stvara probleme

Imena

- Specijalne reči se korsite da učine program čitljivijim - imenuju akcije koje treba da se sprovedu ili sintaksno odvajaju delove naredbi i programa. U većini jezika specijalne reči su rezervisane reči koje ne mogu da budu predefinisane od strane programera.
- Ključne reči su najčešće rezervisane reči, ali ne moraju to da budu, kao na primer kontestno zavisne ključne reči

- Nije dobro ako jezik ima preveliki broj rezervisanih reči
- Na primer, COBOL ima oko 300 rezervisanih reči, u koje spadaju i count, length, bottom, destination...

3.2 Promenljive

Promenljive

- Promenljiva: apstrakcija memorijskih jedinica (ćelija), ime promenljive je imenovanje memorijske lokacije
- Karakteristke promenljivih: ime, adresa, vrednost, tip, životni vek, doseg
- Imena promenljivih su najčešća imena u programu, ali nemaju sve promenljive imena (tj nemaju sve meorijske lokacije imena, na primer, memorijske lokacije eksplicitno definisane na hipu kojima se pristupa preko pokazivača)

Adresa

- Adresa promenljive: adresa fizičke memorije koja je pridružena promenljivoj za skladištenje podataka.
- Promenljiva može imati različite fizičke lokacije prilikom istog izvršavanja programa (npr različiti pozivi iste funkcije rezultuju različitim adresama na steku). Adresa se često naziva l-vrednost
- Moguće je da postoje više promenljivih koje imaju istu adresu - aliasi
- Aliasi pogoršavaju čitljivost programa i čine verifikaciju programa težom.
- Aliasi: unije u C/C++-u, dva pointera koji pokazuju na istu memorijsku lokaciju, dve reference, pointer i promenljiva...
- Aliasi se u mnogim jezicima kreiraju kroz parametre poziva potprograma

Tip

- Tip promenljive određuje opseg vrednosti koje promenljiva može da ima kao i operacije koje se mogu izvršiti za vrednosti tog tipa
- Osnovni tipovi, niske, korisnički definisani prebrojivi tipovi (enum, subrange), nizovi, asocijativni nizovi, strukture, torke, liste, unije, pokazivači i reference

Vrednost

- Vrednost promenljive je sadržaj odgovarajuće memorijske ćelije, često se naziva r-vrednost
- Da bi se pristupilo r-vrednosti mora najpre da se odredi l-vrednost, što ne mora da bude jednostavno, jer zavisi od pravila dosega.

3.3 Doseg

Doseg

- Doseg određuje deo programa u kojem je vidljivo neko ime
- Pravila dosega mogu da budu veoma jednostavna (kao u skript jezicima) ili veoma kompleksna
- Doseg ne određuje nužno životni vek objekta

3.4 Životni vek

Životni vek

- Životni vek obično odgovara jednom od tri osnovna mehanizma skladištenja podataka
- Statički objekti — koji imaju životni vek koji se prostire tokom celog rada programa (npr globalne promenljive)
- Objekti na steku — živoni vek koji se alocira i dealocira LIFO principom, obično zajedno sa pozivom i završetkom rada podprograma.
- Objekti na hipu — koji se alociraju i dealociraju proizvoljno od strane programera, implicitno ili eksplicitno, i koji zahtevaju opštiji i skuplji sistem za upravljanje skladištenjem (memorijom)

3.5 Povezivanje

Povezivanje

- Povezivanje uspostavlja odnos između imena sa onim što to ime predstavlja.
- Vreme povezivanja je vreme kada se ova veza uspostavlja
- Rano vreme povezivanja — veća efikasnost, kasno vreme povezivanja — veća fleksibilnost.

Povezivanje

- Vreme povezivanja može biti (vreme - primer)
  - vreme dizajna programskog jezika (osnovni konstrukti, primitivni tipovi podataka...)
  - vreme implementacije (veličina osnovnih tipova podataka...)
  - vreme programiranja (algoritmi, strukture, imena promenljivih)
  - vreme kompiliranja (preslikavanje konstrukata višeg nivoa u mašinski kod)
  - vreme povezivanja (kada se ime iz jednog modula odnosi na objekat definisan u drugom modulu)
  - vreme učitavanja (kod starijih operativnih sistema, povezivanje objekata sa fizičkim adresama u memoriji)
  - vreme izvršavanja (povezivanje promenljivih i njihovih vrednosti)

4 Kontrola toka i tipovi

4.1 Kontrola toka

Kontrola toka

- Kontrola toka definiše redosled izračunavanja koje računar sprovodi da bi se ostvario neki cilj.
- Postoje različiti mehanizmi određivanja kontrole toka:
  1. Sekvenca — određen redosled izvršavanja
  2. Selekcija — u zavisnosti od uslova, pravi se izbor (if, switch)
  3. Iteracija — fragment koda se izvršava više puta
  4. Podrutine su apstrakcija kontrole toka: one dozvoljavaju da programer sakrije proizvoljno komplikovan kod iza jednostavnog interfejsa (procedure, funkcije, rutine, metodi, potprogrami)

Kontrola toka

- Postoje različiti mehanizmi određivanja kontrole toka:
  5. Rekurzija — definisanje izraza u terminima samog izraza (direktno ili indirektno)
  6. Konkurentnost — dva ili više fragmenta programa mogu da se izvršavaju u istom vremenskom intervalu, bilo paralelno na različitim procesorima, bilo isprepletano na istom procesoru
  7. Podrška za rad sa izuzecima —fragment koda se izvršava očekujući da su neki uslovi ispunjeni, ukoliko se desi suportno, izvršavanje se nastavlja u okviru drugog fragmenta za obradu izuzetka
  8. Nedeterminizam — redosled izvršavanja se namerno ostavlja nedefinisan, što povlači da izvršavanje proizvoljnim redosledom dovodi do korektnog rezultata

4.2 Tipovi

Tipovi

- Programski jezici moraju da organizuju podatke na neki način
- Tipovi pomažu u dizajniranju programa, proveri ispravnosti programa i u utvrđivanju potrebne memorije za skladištenje podataka
- Mehanizmi potrebni za upravljanjem podacima nazivaju se sistem tipova

Tipovi

- Sistem tipova obično uključuje
  - skup predefinisanih osnovnih tipova (npr int, string...)
  - mehanizam građenja novih tipova (npr struct, union)
  - mehanizam kontrolisanja tipova
    - pravila za utvrđivanje ekvivalentnosti: kada su dva tipa ista?
    - pravila za utvrđivanje kompatibilnosti: kada se jedan tip može zameniti drugim?
    - pravila izvođenja: kako se dodeljuje tip kompleksnim izrazima?
  - pravila za proveru tipova (statička i dinamička provera)

Tipovi

- Jezik je tipiziran ako precizira za svaku operaciju nad kojim tipovima podataka može da se izvrši
- Jezici kao što je asembler i mašinski jezici nisu tipizirani jer se svaka operacija izvršava nad bitovima fiksne širine
- Postoji slabo i jako tipiziranje
  - Kod jako tipiziranih jezika izvođenje operacije nad podacima pogrešnog tipa će izazvati grešku
  - Slabo tipizirani jezici izvršavaju implicitne konverzije ukoliko nema poklapanja tipova
  - Neki jezici dozvoljavaju eksplicitno kastovanje tipova

Tipovi

- Važan je trenutak kada se radi provera tipova
  - Za vreme prevođenja programa — statičko tipiziranje
  - Za vreme izvršavanja programa — dinamičko tipiziranje
- Statičko tipiziranje je manje sklono greškama ali može da bude previše restriktivno, dok je dinamičko tipiziranje sklonije greškama i teško za debagovanje ali fleksibilnije.

5 Prevođenje i izvršavanje

5.1 Kompilacija vs Interpretiranje

Kompilacija vs Interpretiranje

- Kompilirani jezici se prevode u mašinski kod koji se izvršava direktno na procesoru računara - faze prevođenja i izvršavanja programa su razdvojene.

- U toku prevođenja, vrše se razne optimizacije izvršnog koda što ga čini efikasnijim.
- Jednom preveden kod se može puno puta izvršavati, ali svaka izmena izvornog koda zahteva novo prevođenje.

Kompilacija vs Interpretiranje

- Interpretirani jezici se prevode naredbu po naredbu i neposredno zatim se naredba izvršava — faze prevođenja i izvršavanja nisu razdvojene već su međusobno isprepletene.
- Rezultat prevođenja se ne smešta u izvršnu datoteku, već je za svako naredno pokretanje potrebna ponovna analiza i prevođenje.
- Sporiji, ali prilikom malih izmena koda nije potrebno vršiti analizu celokupnog koda.
- Hibridni jezici — kombinacija prethodna dva.

Kompilacija vs Interpretiranje

- Teorijski, svi programski jezici mogu da budu i kompilirani i interpretirani.
- Moderni programski jezici najčešće pružaju obe mogućnosti, ali u praksi je za neke jezike prirodnije koristiti odgovarajući pristup.
- Nekada se u fazi razvoja i testiranja koristi interpretirani pristup, a potom se generiše izvršni kod kompilacijom.
- Prema tome, razlika se zasniva pre na praktičnoj upotrebi nego na samim karakteristikama jezika.

5.2 Izvršavanje

Izvršavanje

- Svaka netrivijalna implementacija jezika višeg nivoa intenzivno koristi rantajm biblioteke
- Rantajm sistem se odnosi na skup biblioteka od kojih implementacija jezika zavisi kako bi program mogao ispravno da se izvršava.
- Neke bibliotečke funkcije rade jednostavne stvari (npr podrška aritmetičkim funkcijama koje nisu implementirane u okviru hardvera, kopiranje sadržaja memorije...) dok neke rade komplikovanije stvari (npr upravljanje hipom, rad sa baferovanim I/O, rad sa grafičkim I/O...)

Izvršavanje

- Neki jezici imaju veoma male rantajm sisteme (npr C)
- Neki jezici koriste rantajm sisteme intenzivno
- Virtuelne mašine u potpunosti skrivaju hardver arhitekture nad kojom se program izvršava
- C# koristi rantajm sistem CLI, JAVA koristi JVM

6 Pitanja i literatura

Pitanja

- Koja su osnovna svojstva programskih jezika?
- Koji formalizam se koristi za opisivanje sintakse programskog jezika?
- Šta definiše semantkika programskog jezika?
- Koji su formalni okviri za definisanje semantike programskih jezika?
- Šta je ime?
- Šta je povezivanje?
- Koja su moguća vremena povezivanja?

Pitanja

- Šta je doseg?
- Šta je kontrola toka?
- Koji su mehanizmi određivanja kontrole toka?
- Šta je sistem tipova i šta on uključuje?
- Šta je tipiziranje i kakvo tipiziranje postoji?
- Kada se radi provera tipova?
- Koja je razlika između kompiliranja i interpretiranja?
- Šta je rantajm sistem?

Literatura

- Programming Language Pragmatics, Third Edition, 2009 by Michael L. Scott
- Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages (Pragmatic Programmers), 2010. by Bruce A. Tate
- Semantics with Applications: A Formal Introduction. Hanne Riis Nielson, Flemming Nielson. John Wiley & Sons, Inc. http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.pdf

Literatura

- Programming Language Pragmatics, Third Edition, 2009 by Michael L. Scott
- Deo materijala je preuzet od prof Dušana Tošića, iz istoimenog kursa.