O programskih jezikih na splošno

- programski jeziki so eno od glavnih orodij v računalništvu
- poznamo jih na tisoče, a samo peščica je takih, ki jih uporablja veliko število programerjev
- pri tem predmetu se bomo učili **osnovne principe**, po katerih delujejo programski jeziki
- kaj bomo delali: spoznavali koncepte in prijeme, ki jih srečamo v programskih jezikih
- česa ne bomo delali: prevajalnikov, poglobljenega programiranja v tem ali onem jeziku

Osnovno načelo dobrega načrtovanja programskega jezika: programski jezik je orodje, ki programerju omogoča, da na čim bolj neposreden način poda natančna navodila, kako naj računalik opravi neko nalogo.

Zakaj imamo toliko programskih jezkov?

1. Ker se programski jeziki sproti prilagajajo razvoju računalniške tehnologije in potrebam programerjev.
2. Ker se razvijajo novi programerski koncepti in tehnike.
3. Ker niso vsi stili programiranja enako primerni za reševanje vseh nalog.

Anatomija programskega jezika

Ko govorimo o programskem jeziku, obravnavamo več njegovih sestavnih delov:

- sintaksa: pravila, kako se piše kodo (na primer "vsak oklepaj mora imeti svoj zaklepaj")
- statična semantika: preverjanje, ali je program smiselen (na primer "spremenljivka i ni nikjer deklarirana")
- dinamična semantika: kako se program izvede
- denotacijska semantika: matematični pomen programa
- analiza programa: preverjanje, da deluje pravilno, optimizacija

Programski jezik nima nujno vseh teh komponent, čeprav vsaj sintakso in dinamično semantiko vedno imamo.

Aritmetični izrazi

Ogledali si bomo preproste aritmetične izraze: cela števila z operacijama `*` in `+` in spremenljivkami. To bi lahko bil majhen košček resnega programskega jezika.

Sintaksa

Sintaksa pove, kakšne izraze in programe lahko pišemo v programskem jeziku.

Konkretna sintaksa aritmetičnih izrazov

Programer zapiše program kot niz znakov, na primer:

```
"y + 8 * (5 + 7 * x)"
```

Ta oblika je primerna za človeka, a ni primerna za obdelavo z računalnikom, saj ne odraža strukture izraza.

Abstraktna sintaksa aritmetičnih izrazov

Izraze lahko opišemo s podatkovno strukturo drevo. To je oblika, primerna za obdelavo, ni pa primerna za človeka.

Prednosti:

1. Iz drevesa je takoj razvidna struktura programa ali izraza.
2. Drevo ne vsebuje nepomembnih komponent (na primer presledkov in oklepajev).

Drevesa opišemo kot podatkovno strukturo. Kako to naredimo, je odvisno od programskega jezika, v katerih implementiramo aritmetične izraze.

Slovnica in slovnična pravila

Pravila, ki opisujejo, kako tvorimo izraze ali drevesa, se imenujejo **slovnična pravila** ali **gramatika**. Poznamo več načinov, kako podamo pravila, mi si bomo ogledali poenostavljeno verzijo t.i. oblike BNF, ki delno določa konkretno sintakso:

```
⟨izraz⟩ ::= ⟨aditivni-izraz⟩ EOF

⟨aditivni-izraz⟩ ::= ⟨multiplikativni-izraz⟩ | ⟨aditivni-izraz⟩ + ⟨multiplikativni-izraz⟩

⟨multiplikativni-izraz⟩ ::= ⟨osnovni-izraz⟩ | ⟨multiplikativni-izraz⟩ * ⟨osnovni-izraz⟩

⟨osnovni-izraz⟩ ::= ⟨spremenljivka⟩ | ⟨številka⟩ | ( ⟨aditivni-izraz⟩ )

⟨spremenljivka⟩ ::= [a-zA-Z]+

⟨številka⟩ ::= -? [0-9]+
```

Glede na pravila, je dani izraz veljaven ali ne. Primer:

```
x * (5 + 8)
```

Iz konkretne v abstraktno sintakso

Konkretno sintakso predelamo v abstraktno sintakso s postopkom *razčlenjevana* (angl. *parsing*), ki ima dve fazi:

- **leksična analiza:** niz razbijemo niz gradnikov (angl. *token*)
- **razčlenjevanje** (angl. **parsing**): niz gradnikov predelamo v drevo

Leksična analiza odstrani nebistvene znake, kot so presledki in prehodi v novo vrsto, pogosto tudi komentarje.

Za aritmetične izraze so osnovni gradniki:

- `EOF` poseben gradnik, ki pomeni "konec"
- `PLUS` znak za seštevanje
- `KRAT` znak za množenje
- `SPREMENLJIVKA(x)` spremenljivka
- `ŠTEVILKA(n)` številka
- `OKLEPAJ` in `ZAKLEPAJ`

Primer: `x * (5 + 8)` nam da niz gradnikov

```
SPREMENLJIVKA(x) KRAT OKLEPAJ ŠTEVILKA(5) PLUS ŠTEVILKA(8) ZAKLEPAJ
```

Ta niz nam da ustrezno drevo.

Primer: `x * ((5 + 8` nam da niz gradnikov

```
SPREMENLJIVKA(x) KRAT OKLEPAJ OKLEPAJ ŠTEVILKA(5) PLUS ŠTEVILKA(8)
```

Ta niz ni veljaven in ne določa drevesa. Javimo sintaktično napako.

Iz abstraktne v konkretno sintakso

Če imamo drevo in bi ga radi prikazali kot niz, to naredimo tako, da ga pretvorimo rekurzivno. Pojavi se vprašanje, kako vstaviti oklepaje. To boste ugotavljali na vajah.

Operacijska semantika

Kakšen je postopek, s katerim izračunamo vrednost izraza? Podali bomo dva osnovna načina.

Še prej uvedemo pojem *okolja* (angl. runtime environment), v katerem bomo izračunali izraz. V primeru aritmetičnih izrazov okolje poda vrednosti spremenljivk. Je torej preslikava iz imen spremenljivk v njihove vrednosti.

Semantika velikih korakov

Podamo pravila za relacija

$\eta \mid e \hookrightarrow n$

kjer je η okolje, e je izraz in n celo število. Zgornji izraz preberemo takole: "V okolju η se izraz e evalvira v število n."

Na primer, pričakujemo, da velja

$[x \mapsto 3, y \mapsto 2, z \mapsto 5] \mid x + 2 * y \hookrightarrow 7$

Pravila za računanje izrazov podamo kot pravila sklepanja. Pravilo sklepanja zapišemo takole:

$$\frac{P_1 \quad P_2 \quad \cdots \quad P_i}{S}$$

To preberemo: Če smo že dokazali predpostavke $P_1, P_2, \cdots, P_i$, potem sledi tudi sklep S.

Na primer:

$$\frac{x > 0 \qquad y < 0}{x \cdot y < 0}$$

Preberemo: "če je x pozitiven in y negativen, potem je $x \cdot y$ negativen."

Lahko se zgodi, da pravilo nima predpostavk:

$$\frac{}{S}$$

Takemu pravilu pravimo tudi *aksiom*, saj pove da S velja. Primer aksioma je zakon refleksivnosti za enakost:

$$\frac{}{x = x}$$

Pravila za semantiko velikih korakov se glasijo:

Dogovori:

- η je okolje
- n je število
- e, e_1, ... so izrazi

Pravila:

$$\frac{\eta(x) = n}{\eta \mid x \hookrightarrow n}$$

$$\frac{}{\eta \mid n \hookrightarrow n}$$

$$\frac{\eta \mid e_1 \hookrightarrow n_1 \qquad \eta \mid e_2 \hookrightarrow n_2 \qquad n_1 \cdot n_2 = n}{\eta \mid e_1 * e_2 \hookrightarrow n}$$

$$\frac{\eta \mid e_1 \hookrightarrow n_1 \qquad \eta \mid e_2 \hookrightarrow n_2 \qquad n_1 + n_2 = n}{\eta \mid e_1 + e_2 \hookrightarrow n}$$

Pozor, v pravilu za seštevanje znak + nad črto pomeni matematično operacijo seštevanje, pod črto pa je to del sintakse aritmetičnih izrazov, se pravi + je samo simbol v izrazu. Pri pravilu za množenje te težave nismo imeli, ker smo matematično množenje označili z $\cdot$, množenje kot simbol pa z $*$.

Semantika malih korakov

Lahko pa podamo posamezne korake, ki nas pripeljejo do rešitve, podamo relacijo (pozor, puščico $\hookrightarrow$ smo spremenili v puščico $\mapsto$):

$$\eta \mid e \mapsto e'$$

ki pove, kako naredimo en osnovni korak v računanju.

Pravila se glasijo:

```
 η(x) = n
----------
η | x ↦ n


      η | e₁ ↦ e₁'
---------------------
η | e₁ + e₂ ↦ e₁' + e₂


      η | e₂ ↦ e₂'
---------------------
η | n₁ + e₂ ↦ n₁ + e₂'


  n₁ + n₂ = n
---------------
 η | n₁ + n₂ ↦ n


      η | e₁ ↦ e₁'
---------------------
η | e₁ * e₂ ↦ e₁' * e₂


      η | e₂ ↦ e₂'
---------------------
η | n₁ * e₂ ↦ n₁ * e₂'


  n₁ · n₂ = n
-----------------
 η | n₁ * n₂ ↦ n
```

Primer

```
[x ↦ 3, y ↦ 2, z ↦ 5]  |  x + 2 * y  ↦  3 + 2 * y
```

Korake lahko nadaljujemo, dokler gre:

1. `[x ↦ 3, y ↦ 2, z ↦ 5] | x + 2 * y ↦ 3 + 2 * y`
2. `[x ↦ 3, y ↦ 2, z ↦ 5] | 3 + 2 * y ↦ 3 + 2 · 2`
3. `[x ↦ 3, y ↦ 2, z ↦ 5] | 3 + 2 · 2 ↦ 3 + 4`
4. `[x ↦ 3, y ↦ 2, z ↦ 5] | 3 + 4 ↦ 7`

Primer

Izvajanje se lahko tudi zatakne, na primer, če spremenljivka nima vrednosti:

```
[x ↦ 3]  |  x + 2 * y  ↦  3 + 2 * y
```

Naslednjega koraka ni, ker ne moremo uporabiti nobenega od pravil, ki so na voljo.