Imperatív programozás

Programozási nyelvek szabályrendszere

Kozsik Tamás és mások

1 Programozási nyelvek definíciója

Programozási nyelv szabályai

- Lexikális
- Szintaktikus
- Szemantikus

Egy programozási nyelv definiálása során le kell rögzítenünk a nyelv szabályait, melyek meghatározzák, hogy mik a „jó programok'. A nyelv szabályait három fő kategóriába szokták osztani.

Lexikális szabályok

Milyen építőkockái vannak a nyelvnek?

- Kulcsszavak: `while`, `for`, `if`, `else` stb.
- Operátorok: `+`, `*`, `++`, `?:` stb.
- Zárójelek és elválasztó jelek
- Literálok: `42`, `123.4`, `44.44e4`, `"Hello World!"` stb.
- Azonosítók
- Megjegyzések

Case-(in)sensitive?

A nyelv lexikális szabályai azt határozzák meg, hogy a nyelvben mik a morfémák, azaz a legkisebb, már önálló jelentéssel bíró építőkockák. Ezek az építőkockák egy programozási nyelvben a kulcsszavak, az operátorok, a zárójelek, vesszők/pontosvesszők, literálok, azonosítók és megjegyzések.

Például C-ben az alábbi kulcsszavak vannak: Pythonban pedig az alábbiak:

Egy nagyon fontos lexikális kérdés még az, hogy a nyelv érzékeny-e a kis- és nagybetűkre. Vannak *case-insensitive* nyelvek, például az Ada, amelyek nem tesznek különbseget a kis- és nagybetűk között, így például elfogadják kulcsszónak a `while` és a `wHiLE` karaktersorozatot is. A C és Python nyelvek viszont *case-sensitive*-ek.

Azonosító

- Alfanumerikus
- Ne kezdődjön számmal
- Lehet benne `_` jel?

Jó

- `factorial`, `i`
- `computePi`, `open_file`, `worth2see`, `Z00`
- `__main__`

Rossz C-ben

- `2cents`
- `fifty%`

- `nőnemű` és $A\theta\acute{\eta}\nu\alpha$ (jók Pythonban)

Az azonosítók (identifier) a legtöbb nyelvben alfanumerikusak lehetnek, azaz betűkből és számjegyekből épülhetnek fel. Egy fontos szabály szokott lenni, hogy betűvel kell kezdődniük. Ezen kívül sok nyelvben nem használhatunk speciális jeleket vagy nem ASCII betűket azonosítókban.

A dupla aláhúzás jellel kezdődő azonosítókat C-ben és Pythonban speciális célokra használják, így az általunk írt deklarációkban nem szoktunk ilyen azonosítót megadni.

Szintaktikus szabályok

Hogyan építkezhetünk?

- Hogyan épül fel egy ciklus vagy egy elágazás?
- Hogyan néz ki egy alprogram? stb.

Backus-Naur form (Backus normal form) – BNF

```
<statement> ::= <expression-statement>
              | <while-statement>
              | <if-statement>
              | …

<while-statement> ::=  while (<expression>) <statement>

<if-statement> ::=  if (<expression>) <statement>
                   <optional-else-part>

<optional-else-part> ::= ""
                       | else <statement>
```

Szemantikus szabályok

Értelmes, amit építettünk?

- Deklaráltam a használt változókat? (C)
- Jó típusú paraméterrel hívtam a műveletet?

stb.

Típusellenőrzés

- A változókat, függvényeket a típusuknak megfelelően használtuk-e
- A nem típushelyes programok értelmetlenek

Statikus és dinamikus típusrendszer

- A C fordító ellenőrzi *fordítási időben* a típushelyességet
- Pythonban *futási időben* az interpreter vizsgálja ezt

Erősen és gyengén típusos nyelv

- Gyengén típusos nyelvben automatikusan konvertálódnak értékek más típusúra, ha kell
  - Eleinte kényelmes
  - De könnyen írunk mást, mint amit szerettünk volna
- A C-ben és Pythonban viszonylag szigorúak a szabályok (erősen típusosak)

A programozási nyelvek egy részénél a fordítóprogram már a fordítási időben minden egyes részkifejezésről el tudja dönteni, hogy az milyen típusú. Ezeket a nyelveket statikus típusrendszerrel rendelkezőnek nevezzük. Ennek vannak előnyei, hiszen a nyelv alaposabb ellenőrzéseket tud végrehajtani és optimálisabb kódot is tud generálni. Ilyen nyelv a Fortran, Algol, C, Pascal, C++, Java, C#, Go.

Más nyelveknél, legtöbbször az interpretált nyelveknél, egy változó idővel más típusú értékekre is hivatkozhat. Ilyenkor a fordító futási időben kezeli a típusinformációkat. Ezt dinamikus típusrendszer-nek nevezzük. Ilyen nyelv pl. a Python.

Mindez nem jelenti, hogy a dinamikus típusrendszer nem ellenőrizheti a típusok alkalmazását, sőt helytelen alkalmazás hibát okozhat. Azokat a nyelveket, ahol ilyen hibák előfordulnak erősen típusos-nak nevezzük, szemben a gyengén típusos nyelvekkel.

A C erősen típusos statikus típusrendszerrel rendelkező nyelv, a Python erősen típusos dinamikus típusrendszerű.

Statikus és dinamikus szemantikai szabályok

- Statikus: amit a fordító ellenőriz
- Dinamikus: amit futás közben lehet ellenőrizni

Eldönthetőségi probléma…

Összefoglalva

- Lexikális: mik az építőkockák?
- Szintaktikus: hogyan építünk struktúrákat?
- Szemantikus: értelmes az, amit felépítettünk?
  - statikus szemantikai szabályok
  - dinamikus szemantikai szabályok

2 Kitekintés későbbi tárgyakra

A fordítóprogramok részei

- Lexer: tokenek sorozata
- Parser: szintaxisfa, szimbólumtábla
- Szemantikus (pl. típus-) ellenőrzés

(vagy különböző szintű fordítási hibák)

Formális nyelvek

- Lexikális szabályok: reguláris nyelvtan
- Szintaktikus szabályok: környezetfüggetlen nyelvtan
- Szemantikus szabályok: környezetfüggő, vagy megkötés nélküli nyelvtan

Program szemantikája

A (nyelv szabályainak megfelelő) program jelentése

3 Pragmatika

A nyelv definíciója

- Lexika
- Szintaktika
- Szemantika
- Pragmatika

Pragmatika

Hogyan tudjuk hatékonyan kifejezni magunkat?

- Konvenciók
- Idiómák
- Jó és rossz gyakorlatok

stb.

Konvenció

általános vagy fejlesztői csoportra (cégre) specifikus

- kapcsos zárójelek elhelyezése
- névválasztás (pl. setter/getter)
- azonosítók írásmódja, nyelve
- kis- és nagybetűk