Haskell

Kieli on nimetty loogikko Haskell Curryn mukaan, ja ensimmäinen julkaisu oli vuonna 1990. Kielen ‘standarditoteutus’ on GHC, joka kääntää ohjelman kohdearkkitehtuurin konekoodiksi.

- Puhtaasti funktionaalinen ohjelmointikieli

Ohjelmien käytös hyvin ennustettavaa ja kääntäjä voi varmistaa ohjelman oikeellisuuden. Käytännössä varsin usein jos ohjelma suostuu kääntymään se myös toimii oikein. Sivuvaikutteiset funktiota (esim IO) voi käsitellä vain spesiaalin Monadi -tyypin sisällä.

- Arvosovitus & Suojukset (Pattern matching & Guards)

Arvosovituksen avulla voi määritellä funktiolle tietyn arvon joillekin staattisille parametreille (ks. esimerkin fib). Suojuksilla taas määritellään arvo tietylle lauseelle, esimerkiksi `sign x | x < 0 = “negative”`

- Laiska arviointi

Kieli käyttää laiskaa arviointia, jolloin lauseiden arvot arvioidaan vasta silloin, kun niitä käytetään. Yksi käyttökohde tälle on äärettömän kokoisten listojen rakentaminen

- Tyyppipäättely

Kieli käyttää Hindley-Milner -tyyppipäättelyä, josta johtuen se on vahvasti ja staattisesti tyypitetty, mutta tyyppiannotaatiot toimivat usein pelkkänä dokumentaationa ohjelmoijalle, sillä kääntäjä osaa tulkita tyypit itse.

Haskell on suosittu akateemisissa piireissä, mutta käyttöä teollisuudessa on myös. Kielen funktionaalisuus voi vähentää virheiden määrää esimerkiksi rinnakkaislaskennassa, mutta joidenkin ongelmien ratkaiseminen tehokkaasti voi olla huomattavasti haastavampaa.

Haskell - esimerkkiohjelma

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)

fibList Int -> [Int]
fibList n = take n (generateList 1 1)

generateList :: Int -> Int -> [Int]
generateList a b = a : generateList b (a + b)

main :: IO ()
main = do
    putStrLn "Montako Fibonaccin lukua haluat
generoida?"
    howMany <- getLine
    putStrLn "Palautetaanko viimeinen luku (1) vai
kaikki luvut (2)?"
    method <- getLine
    if (read metodi) == 1
        then printFib fib howMany
        else printFib fibList howMany
    where
        printFib f m = putStrLn $ show $ f $ read m

-- esimerkkejä

fib 10 = 55
fibList 10 = [1,1,2,3,5,8,13,21,34,55]
```

COBOL

COBOL eli “Common Business-Oriented Language” kehitettiin vuonna 1959 kaupallis-hallinnollisiin sovelluksiin pääpainona erityisesti syötteiden ja tulosteiden käsittely**.** Perinteisesti COBOL on käännetty kieli, eikä se käytä välikieltä. Sille on kuitenkin olemassa erilaisia tulkkeja ja muita toteutuksia.

- Sisennyksien puute

Aluksi COBOLissa käytettiin reikäkorteista perua olevaa sisennystä, jossa koodi piti sijoittaa oikeille sarakkeille. Nykyään COBOL ei vaadi lainkaan sisennystä.

- Staattinen lohkonäkyvyys

DATA-osiossa globaaleiksi määritellyt muuttujat ovat näkyvissä koko ohjelmassa sekä aliohjelmissa. Ylätasoilla määriteltyjä ohjelmia voidaan kutsua sisemmiltä tasoilta, jos ne on määritelty käyttäen common-piirrettä.

- Englannin kieltä muistuttava syntaksi

COBOLin syntaksi on tehty muistuttamaan Englannin kieltä, jotta sitä olisi mahdollisimman helppo työstää ja lukea. Näin koodia voivat ymmärtää ja lukea muutkin kuin ohjelmoijat.

- Heikosti ja staattisesti tyypitelty

COBOLissa muuttujien tyypit määritellään staattisesti, mutta tyyppiturvallisuus on heikkoa, paitsi käytettäessä osoittimia ja viitteitä.

COBOLin tyypillisimmät käyttökohteet ovat kaupallis-hallinnollisten sovellusten eräajot sekä transaktioiden hallinta. COBOLin syntaksin helppolukuisuus on yksi sen suurimmista vahvuuksista vaikkakin COBOLille tyypillinen koodi on kuitenkin usein monoliittistä, ja näin ollen sen syntaksista on hankala nähdä mitä koodissa tapahtuu syvemmällä tasolla. COBOLilla ei niinkään ohjelmoida uusia ohjelmia, vaan sitä käytetään jo tehtyjen ohjelmien käyttöön ja ylläpitoon.

Cobol - esimerkkiohjelma

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE-PROGRAM.

DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  WS-NUM PIC S9(1).
    01  WS-STR PIC X(15).

PROCEDURE DIVISION.
    DISPLAY 'Give a number:'.
    ACCEPT WS-NUM.
    DISPLAY 'Give a word:'.
    ACCEPT WS-STR.

    IF WS-NUM > 1 THEN
        CALL 'NESTED-PROGRAM' USING BY CONTENT WS-NUM,BY REFERENCE
WS-STR
    ELSE
        DISPLAY 'Give a positive number.'
    END-IF.
    DISPLAY WS-NUM.
    DISPLAY WS-STR.
    STOP RUN.

IDENTIFICATION DIVISION.
PROGRAM-ID.  NESTED-PROGRAM.

DATA DIVISION.
    LINKAGE SECTION.
    01  LS-NUM PIC S9(1).
    01  LS-STR PIC X(15).

PROCEDURE DIVISION USING LS-NUM, LS-STR.
    PERFORM PRINT-PARA UNTIL LS-NUM < 1.
    MOVE 'New value' TO LS-STR.
    EXIT PROGRAM.
    PRINT-PARA.
        DISPLAY LS-STR.
        SUBTRACT 1 FROM LS-NUM.
END PROGRAM NESTED-PROGRAM.
END PROGRAM EXAMPLE-PROGRAM.
```

11.12.2019 Suoranta Aleksi, Sarapalo Joonas, Karttunen Jarkko, Heikkinen Juuso