Ada

Ada on Yhdysvaltain puolustusministeriön käyttöön 1980 kehitetty vahvasti ja staattisesti tyypitetty käännettävä kieli. Ada on alusta asti tarkoitettu hyvin turvalliseksi ja se on nykypäivänä käytössä järjestelmissä joissa turvallisuus ja vakaus ovat kriittisiä, kuten lennonohjauksessa ja satelliiteissa.

- Ilmaisuvoimainen ja vahva tyyppijärjestelmä
    - Nimiekvivalenssi
    - Mahdollisuus määritellä hyvin monenlaisia alityyppejä, kuten kokonaislukujen Integer alityyppi Natural, siis luonnolliset luvut
    - Tuki olio-ohjelmoinnille
- Aito, syvä ja tarkkaan määritelty lohkorakenne
    - Staattinen näkyvyys
    - Tietyt kielen rakenteet, esimerkiksi proseduurit sisältävät erillisen “määrittelyalueen”
    - Aliohjelmat ja tyypit täytyy määritellä määrittelyalueella
    - Myös muuttujat täytyy esitellä määrittelyalueella, missä ne voidaan myös määritellä
- Parametreille voi määritellä erilaisia moodeja
    - In-moodissa olevien parametrien arvoja ei ole mahdollista muuttaa
    - In out-moodissa aliohjelma voi muuttaa parametrin arvoa
    - Out-moodin parametrit toimivat muuten in out-parametrien tavoin, mutta lisäksi kertoo ohjelmoijalle, että aliohjelma ei välitä parametrin alkuarvosta vaan ylikirjoittaa sen ennen sen lukemista

Ada on nykypäivänä melko marginaalinen kieli. Kielen tarkka syntaksi ja vahva tyyppijärjestelmä tekevät siitä turvallisen ja luotettavan mutta myös monisanaisen ja monimutkaisen.

12.12.2019 Daniel Martinez, Jani Takkinen

```
with Ada.Text_IO;
with Ada.Integer_Text_IO;
with Ada.IO_Exceptions;

procedure Suurempi is
	package T_IO renames Ada.Text_IO;
	package I_IO renames Ada.Integer_Text_IO;

	subtype PieniLuku is Integer range 0..5;
	A: PieniLuku;
	B: PieniLuku;
begin
	T_IO.Put ("Syötä luvun A arvo: ");
	-- Luemme luvun ja talletamme arvon muuttujaan A.
	I_IO.Get (A);
	T_IO.Put ("Syötä luvun B arvo: ");
	I_IO.Get (B);
	if A > B then
		T_IO.Put_Line ("Luku A oli suurempi.");
	elsif B > A then
		T_IO.Put_Line ("Luku B oli suurempi.");
	else
		T_IO.Put_Line ("Luvut olivat yhtä suuret.");
	end if;
exception
	when Virhe : Constraint_Error =>
		T_IO.Put_Line("Liian pieni tai suuri luku!");
	when Virhe : Ada.IO_Exceptions.Data_Error =>
		T_IO.Put_Line("Syötä vain lukuja!");
end Suurempi;
```

Erlang

Erlang tai Erlang/OTP (Open Telecom Platform) on Ericssonin kehittämä ohjelmointikieli ja suoritusympäristö. Kehitettiin korvaamaan vanhentunut PLEX-ohjelmointikieli puhelinverkkolaitteiden ohjelmoinnnissa.
Ensimmäinen versio julkaistiin 1986 ja kieli on edelleen käytössä.

- Puhtaasti funktionaalinen kieli
    - Muuttujan voi sitoa vain kerran
    - Automaattinen häntärekursio jos mahdollista
- Prosessit
    - Suoritusympäristön sisäiset prosessit
    - Tyypillinen Erlang-järjestelmä koostuu supervisor- ja worker-prosesseista
    - Supervisor-prosessi valvoo worker-prosessien toimintaa ja voi esim. käynnistää kaatuneen prosessin uudestaan
    - Prosessien käyttö mahdollistaa...
- Suoritusaikaiset päivitykset
    - Järjestelmän koodia voi päivittää suoritusaikana
    - Mahdollistaa aina käynnissä olevien järjestelmien päivittämisen, hyödyllistä esim. tietoliikenneverkoissa
    - Jos prosessin koodi päivittyy, seuraavalla käynnistyskerralla prosessi käyttää uutta versiota.

Erlang sopii erityisen hyvin hajautettujen, vikasietoisten ja jatkuvasti toiminnassa olevien järjestelmien toteuttamiseen, joten sen suosio tuskin ainakaan vähenee tulevaisuudessa.
Toisaalta Erlang ei ole kovin yleiskäyttöinen kieli, se ei sovi kovin hyvin esimerkiksi graafisten ohjelmien toteuttamiseen tai suurten tietomäärien käsittelyyn. Kielen huono puoli on melko korkea oppimiskynnys.

```
-module(prosessit).
-export([run/0, ping/3, pong/0]).

ping(1, List, Pong_PID) ->
  Pong_PID ! {ping, loppu, self()},
  io:format("~w~n", [List]);

ping(N, List, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
     {pong, Num} ->
     io:format("Pong -> Ping, ~w~n", [Num]),
     ping(Num, [Num | List], Pong_PID)
  end.

pong() ->
  receive
     {ping, loppu, Ping_PID} ->
     loppu;
     {ping, Ping_PID} ->
     io:format("Ping -> Pong~n"),
     Ping_PID ! {pong, rand:uniform(10)},
     pong()
  end.

run() ->
  Pong_PID = spawn(?MODULE, pong, []),
  Ping_PID = spawn(?MODULE, ping, [100, [], Pong_PID]),
  {Pong_PID, Ping_PID}.
```