C#

- Historia

  Tanskalainen Anders Hejlsberg kehitti C# työskennellessään Microsoftilla. Hejlsberg oli aikaisemmin toiminut Delphi-ohjelmointikielen kehityksessä, ja nykyään hän kehittää Typescriptiä. Microsoftin tavoite oli luoda ohjelmointikieli Javan haastajaksi. Microsoft julkaisi C# vuonna 2002 yhdessä .Net-sovelluskehyksen ja CLR kanssa. Kieli muistutti alussa Javaa, mikä saattoi johtua siitä, että C# suunnittelussa on käytetty ECMA-standardia, jonka yhtenä tarkoituksena oli luoda yksinkertainen, moderni oliokieli [CS_HIST]. Ohjelmointikieltä kehittää ja ylläpitää Microsoft ja 2019 helmikuussa C# oli neljänneksi suosituin ohjelmointikieli. [lähde]

- Tyyli

  C# on staattisesti ja vahvasti tyypitetty oliokieli. Muuttujien nimeämisessä noudatetaan camelcase-käytäntöä. Näkyvyysalue on staattinen ja määräytyy kaarisulkeiden perusteella. C# kaikki arvot ovat olioita ja ne perivät System.Object -luokan. Oliot jaetaan arvotyyppeihin ja viittaustyyppeihin. Arvotyyppiset muuttujat ovat skalaareja, viittaustyyppeihin kuuluvat rakenteiset tyypit. C# viittaustyyppejä ovat luokat, taulukot, delegaatit ja rajapinnat [MS_REF_T]. Viiteparametreissä tieto talletetaan kekoon ja arvoparametreissä tieto talletetaan pinoon. Näiden kahden muuttujan välillä kääntäjä suorittaa eksplisiittisiä ja implisiittisiä tyyppimuunnoksia.

- Ympäristö

  .Netissä ajonaikainen ympäristö on pakotettu, joka takaa turvallisen ajoympäristön, sillä ohjelmoija joutuu noudattamaan määrättyjä turvallisuus- ja tarkkaavaisuusmääräyksiä. .Net löytyy esiasennettuna uudemmista Windowsin versioista. C#:ia voidaan käyttää myös Mono-kääntäjällä muissa ympäristössä. Visual Studio on yleisesti käytössä C#-kehityksessä, jonka yhteydessä kieltä kutsutaan termillä Visual C#. Visual Studiossa on *ratkaisuja (solution)* ja *projekteja*. Yksi ratkaisu voi sisältää useampia projekteja ja jokaisen projektin täytyy kuulua johonkin ratkaisuun. Itse projekti sisältää ohjelman riippuvuudet ja niitä hallitaan NuGet-paketinhallintajärjestelmällä. Ohjelmassa riippuvuuksia tuodaan käyttöön *using*-lauseella.

- Rakenne

  C# luokan yleinen ohjelma-arkkitehtuuri koostuu nimiavaruudesta, luokan jäsenistä ja luokan jäsenten sisällöstä eli metodeista ja lauseista (Namespace -> classes members -> method -> statements). C# on polymorfinen kieli eli se tukee dynaamista sekä staattista sidontaa. C# käyttää staattista näkyvyysaluetta eli jokaisella muuttujalla on oma näkyvyysalueensa. C# on litteä kieli, sillä metodin sisälle voi määritellä apufunktion mutta apufunktio ei voi peittää ylemmän lohkon kenttiä.

- Hyvää ja huonoa

  C# on tyyppiturvallinen kieli eli ohjelmoija voi luottaa, ettei dynaamisia tyyppivirheitä synny. Toisaalta C# tukee myös dynamic-tyyppisiä muuttujia, joiden avulla voidaan hyödyntää dynaamista tyypitystä. C# on myös vuosien saatossa tuonut paljon lisää toiminnallisuuksia ohjelmointikieleen, esimerkiksi “nullable”-tyypin joka sallii muuttujan arvona nullin. Tällöin kääntäjä tyyppitarkastaja varoittaa null-arvon väärästä käytöstä.

  Lähteet:

  [MS_REF_T]: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types

  [CS_HIST]: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history

Jere Lahelma, Vili Lipo, Ollipekka Väänänen

APL

- APL:n tausta
  Kenneth E. Iverson aloitti APL:n suunnittelun 50-luvun lopulla Harvardin yliopistossa. APL kehitettiin alun perin matemaattiseksi notaatioksi, jota kirjoitettiin käsin. Kielen suunnittelija Kenneth Iverson sai Turing-palkinnon 1979 kielen kehittämisen johdosta. APL:n monikäyttöisten operaattoreiden ansiosta koodirivejä säästyy, ja ohjelmista tulee tiiviimpiä. Iverson on suunnitellut myös toisen ohjelmointikielen: J. APL on vaikuttanut ainakin S-kieleen ja sen kautta R-kieleen ja MATLAB-ohjelmistoon.
- Kielen piirteet
  APL käyttää dynaamista tyypitystä. Kielen oletustyyppi on taulukko. Taulukot voivat sisältää numeroita, merkkejä, ja taulukoita. Alkuperäisessä toteutuksessa ei ollut mitään kontrollirakenteita kuten silmukoita tai ehtolauseita. Kielessä on goto-lause, johon on mahdollista yhdistää ehtoja nokkelilla laskuilla, näin pakon vaatiessa voi toteuttaa ehdollista suorittamista ja toistoa. Lausekkeiden evaluoinnissa perinteisiä laskujärjestyssääntöjä ei noudateta ja se tapahtuu oikealta vasemmalle. Evaluointijärjestystä voi kuitenkin ohjata sulkeilla.
- Oma merkistö
  Tyypillinen APL-ohjelma koostuu APL:n omista yhden merkin operaattoreista ja funktioista, jotka käsittelevät yksi- tai moniulotteisia taulukoita. Tehokas APL-ohjelmointi vaatii siis oman näppäimistöasettelun, jotta erikoismerkkien kirjoittaminen on sujuvaa. Merkistön tarkoituksena oli luoda kokonaan uusi matemaattinen notaatio, joten kaikki merkit olivat uusia [ULMANN].
- Käyttökohteet
  APL soveltuu matemaattiseen laskentaan, johon se oli alun perin suunniteltu, korvaamaan perinteinen matemaattinen notaatio taulukko laskennassa. APL mahdollistaa myös luontevan ohjelmointityylin, joka muistuttaa ihmisen matemaattista ajattelua [WHY_APL].
- APL:n funktiot
  APL:n operaattorit ja funktiot jaetaan niladisiin (ei parametreja), monadisiin (yksi parametri) ja dyadisiin (kaksi parametria) niiden käytön mukaan. APL funktioita voi toteuttaa käyttämällä määrittämällä sen otsakelauseella tai sitomalla anonyymin funktion nimeen. Anonyymien funktioiden muodollisia parametreja merkitään aina samoilla merkeillä, vasen on ⍺ ja oikea ⍵. Anonyymit funktiot ovat erittäin kompakteja, mutta niiden toimintaa voi olla vaikeaa seurata.
- Taulukko-operaatiot
  Lähes kaikki APL-operaatiot kohdistuvat taulukoihin. Tämä mahdollistaa ohjelmoinnin ilman tarvetta explisiittiselle iteroinnille. Koska APL-tulkki toteuttaa standardiin kuuluvat operaatiot se voi vapaasti muuttaa niiden sisäistä rakennetta, jolloin toteutukset voivat sisältää rinnakkaista laskentaa ilman, että ohjelmoija on tästä tietoinen. APL-kieltä on mahdollista kääntää ajettavaksi näytönohjaimille.
- Hyvää ja Huonoa
  APL-ohjelmat ovat hyvin tiiviitä. Kielen puolustajien mielestä tämä on hyvä asia, koska APL-ohjelmassa on vähemmän rivejä ylläpidettäväksi kuin muilla kielillä kirjoitetussa. APL:llän vastustajat taas sanovat, että APL-ohjelman lähdekoodi on hyvin vaikeaselkoista ja toisen ajatuksia on lähes mahdotonta ymmärtää koodista yksinään. APL-kielisissä projekteissa suositaan runsasta kommentointia.

  Lähteet
[ULMANN] http://www.vaxman.de/publications/apl_slides.pdf
[WHY_APL] http://www.aplborealis.com/whyapl.html

Tyypillinen APL-toteutus funktiolle FizzBuzz:

```
FizzBuzz←{(⍵'Fizz' 'Buzz' 'FizzBuzz')[{+/1 2×0=3 5|⍵}⍵]}
```

Ymmärrettävä vaihtoehtototeutus:

```
∇ULOS←FizzBuzz LUKU;ARVOT;JAKOJAANNOS;INDEKSI
ARVOT←(LUKU 'Fizz' 'Buzz' 'FizzBuzz')
JAKOJAANNOS←3 5|LUKU
INDEKSI←+/1 2×0=JAKOJAANNOS
ULOS←ARVOT[INDEKSI]
∇
```

```
Kutsu:
FizzBuzz¨1 2 3 5 10 15 16
1 2   Fizz    Buzz    Buzz    FizzBuzz   16
```

Funktiossa siis otetaan jakojäännös vektorille 3 5 luvusta LUKU.
Jakojäännös operaatio | palauttaa 2 alkioisen taulukon, jossa on jakojäännökset.
Jakojäännöksiä verrataan tämän jälkeen nollaan. Tämä vertailu palauttaa vektorin josssa on ykkösiä ja nollia.
Nämä kerrotaan vektorilla 1 2, josta otetaan summa.
Nyt tätä lukua voi käyttää indeksinä palautusarvotaulukolle.

Iteratiivinen Fibonacci:

```
fiboiter←{⍵⍴1↓↑+.×/⍵/⊂2 2⍴1 1 1 0}
```

Funktiossa luodaan 2x2 matriisi jossa on rivit 1 1 ja 10. Siitä luodaan oikean parametrin antaman määrän mukaan kopioita, joista suoritetaan matriisilaskuoperaatioita, siten että ylimmälle riville jää fibonaccin luvut w ja w+1. Sitten loppu funktio vain valitsee matriisista ensimmäisen rivin toisen alkion ja asettaa sen 1x1 taulukkoon.

Rekursiivinen Fibonacci:

```
fibonacci←{0 1{⍵=0:⍺⍴⍺ ⋄ (1↓⍺,+/⍺)∇ ⍵-1}⍵}
```

Tämän sisäfunktio laskee siis rekursiivisesti parametrina välitettävän vektorin summan ja lisää sen vektoriin ja tiputtaa ensimmäisen alkion pois. Kun oikea parametri on 0 palautetaan vektorin viimeinen arvo. Koska tämä funktio on häntärekursio, niin se on rekursiivisuudestaan huolimatta paljon tehokkaampi Dyalog-APL tulkilla kuin ylempi iteratiivinen versio.

C# FizzBuzz

```
using System;

namespace FizzBuzz
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 1; i <= 100; i++)
            {
                if (i % 3 == 0 && i % 5 == 0)
                {
                    Console.WriteLine("FizzBuzz");
                }
                else if (i % 3 == 0)
                {
                    Console.WriteLine("Fizz");
                }
                else if (i % 5 == 0)
                {
                    Console.WriteLine("Buzz");
                }
                else
                {
                    Console.WriteLine(i);
                }
            }
            Console.ReadKey();
        }
    }
}
```

C# iteratiivinen fibonacci

```
using System;

namespace Fibonacci
{
    class Program
    {
        public static int Fibonacci(int n)
        {
            int a = 0;
            int b = 1;

            for (int i = 0; i < n; i++)
            {
                int temp = a;
                a = b;
                b = a + temp;
            }

            return a;
        }
        static void Main(string[] args)
        {
            for (int i = 0; i < 15; i++)
            {
                Console.WriteLine(Fibonacci(i));
            }

            Console.ReadKey();
        }
    }
}
```