Seinäjoen
ammattikorkeakoulun
julkaisusarja

C

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Hilkka Niemelä

# OHJELMOINNIN PERUSRAKENTEET

Seinäjoen ammattikorkeakoulun julkaisusarja
C. Oppimateriaaleja 9

Hilkka Niemelä

# OHJELMOINNIN PERUSRAKENTEET

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Seinäjoki 2015

**Seinäjoen ammattikorkeakoulun julkaisusarja**
Publications of Seinäjoki University of Applied Sciences

A. **Tutkimuksia** Research reports
B. **Raportteja ja selvityksiä** Reports
C. **Oppimateriaaleja** Teaching materials

**SeAMK julkaisujen myynti:**
Seinäjoen korkeakoulukirjasto
Kalevankatu 35, 60100 Seinäjoki
puh. 020 124 5040 fax 020 124 5041
seamk.kirjasto@seamk.fi

ISBN 978-952-7109-21-2 (verkkojulkaisu)
ISSN 1797-5581 (verkkojulkaisu)

# TIIVISTELMÄ

Niemelä, Hilkka. 2015. Ohjelmoinnin perusrakenteet. Seinäjoen ammattikorkeakoulun julkaisusarja C. Oppimateriaaleja 9, 63 s.

Dokumentissa esitellään ohjelmoinnin perusrakenteet: muuttujien käsittely, kontrollirakenteet, metodit, taulukot sekä merkkien ja merkkijonojen käsittely. Perusrakenteet ovat samankaltaisia kaikissa ohjelmointikielissä. Tietoa käsitellään muuttujien avulla. Ohjelman etenemistä säädetään kontrollirakenteilla, joita ovat valinta-, toisto- ja siirtorakenteet. Ohjelmia pilkotaan pienempiin, itsenäisesti toteutettaviin osiin metodien avulla. Tietojoukkojen käsittelyyn käytetään kokoelmia, joiden perusrakenteena on taulukko. Ohjelmointikielenä esimerkeissä käytetään C#-kieltä.

**Avainsanat:** ohjelmointi, muuttujat, kontrollirakenteet, metodit

**Yhteystiedot:** Hilkka Niemelä, Seinäjoen ammattikorkeakoulu, SeAMK Tekniikka
PL 412, 60101 Seinäjoki, hilkka.niemela@seamk.fi

# ABSTRACT

Niemelä, Hilkka. 2015. Basic structures of programming. Publications of Seinäjoki Polytechnic C. Teaching materials 9, 63 p.

In this document the basic structures of programming are presented: variables, control structures, methods and arrays as well as characters and strings. The basic structures are similar in all programming languages. The information is processed with the help of variables. The progress of the program is adjusted with control structures such as selection statements, iteration statements and jump statements. Programs are broken down into smaller parts with the help of methods. A method is a code block that contains a series of statements. Collections are used to manage groups of related objects. The basic structure of a collection is an array. Multiple variables of the same type can be stored in an array data structure. In the examples C#-language is used as a programming language.

**Keywords:** programming, variables, control structures, methods

**Contact information:** Hilkka Niemelä, Seinäjoki University of Applied Sciences,
P.O. Box 412, 60101 Seinäjoki, Finland, hilkka.niemelal@seamk.fi

# SISÄLLYS

# 1 JOHDANTO

Tietokone on rakennettu noudattamaan yksinkertaisia bitteinä esitettyjä komentoja. Näitä koneen osaamia komentoja kutsutaan konekieleksi. Konekieli on ihmisen kirjoittamana virhealtista ja työlästä, siksi lähes kaikki ohjelmointi tapahtuu nykyisin korkeamman tason kielillä ohjelmointia tukevan sovelluskehittimen avustuksella.

Koska kone osaa vain omaa konekieltään, on ihmisen kirjoittama ohjelma käännettävä koneen kielelle ennen kuin se voidaan suorittaa. Tarvitaan siis kääntäjä (ohjelma), jolla ohjelma käännetään joko etukäteen tai suorituksen aikana koneen ymmärtämään muotoon.

Tässä julkaisussa käytetään esimerkkikielenä C#-ohjelmointikieltä. C#-kieliset ohjelmat käännetään ensin välikielisiksi käännökseksi, joka ei ole sidoksissa mihinkään käyttöjärjestelmään tai prosessoriin. Varsinaisen ajon aikana tapahtuu lopullinen käännös käytössä olevan järjestelmän konekielelle.

Ohjelmoinnin perusrakenteet noudattavat samaa logiikkaa ohjelmointikielestä riippumatta. Tietoa käsitellään muuttujien avulla. Muuttujat ovat nimettyjä tiedon väliaikaisia tallennuspaikkoja. Ohjelmoija voi käyttää tietoa ohjelmassaan viittaamalla tarvittavan tiedon nimeen. Muuttujien käsittelyn perusteet on esitetty luvussa 2.

Ohjelmoinnin kontrollirakenteita ovat ehtolauseet, toistolauseet ja siirtorakenteet. Ehtolauseiden avulla ohjelmoija voi haarauttaa toimintoja siten, että tietyn ehdon voimassa ollessa suoritetaan tietyt lauseet ja toisaalta jonkin muun ehdon voimassa ollessa jotkin toiset lauseet. Toistorakenteiden avulla kone saadaan toistamaan samaa toimenpidettä annettujen rajoitusten perusteella. Siirtorakenteiden avulla voidaan keskeyttää toiston suoritus tai ohjelman suoritus voidaan siirtää jatkumaan osoitteistuksen perusteella jostain toisesta kohdasta. Kontrollirakenteet on käsitelty luvassa 3. Metodit esitellään luvussa 4. Metodit ovat pienehköjä ohjelman palasia, joilla on jokin rajattu ja määritelty tehtävä. Metodeita muodostetaan sellaisista toiminnoista, joita tarvitaan usein ja missä tahansa kohtaa ohjelmaa. Metodit nimetään ja niitä kutsutaan nimellä. Kutsun mukaan voidaan liittää annetun tehtävän suorittamiseen tarvittavat lähtötiedot eli parametrit. Metodista käytetään eri ohjelmointikielissä myös nimiä aliohjelma, proseduuri tai funktio.

Samankaltaisten tietojen joukkoa käsitellään kokoelmien avulla. Tässä dokumentissa on lyhyesti käsitelty kokoelmien perusrakennetta eli taulukkoa luvussa 5. Taulukoista esitellään niiden perustaminen ja läpikäyminen.

Luvussa 6 esitellään lyhyesti merkkien ja merkkijonojen käsittely. Esimerkein näytetään tyyppien määrittelyt ja keskeisimmän metodit niiden käsittelyssä.

Tässä julkaisussa ohjelmoinnin perusrakenteet esitetään käyttäen esimerkkinä C#-kieltä ja sovelluskehittimenä VisualStudio-ohjelmistoa. Tarvittavat välineet ovat kenen tahansa ladattavissa VisualStudion lataussivustolta (Microsoft 2015c).

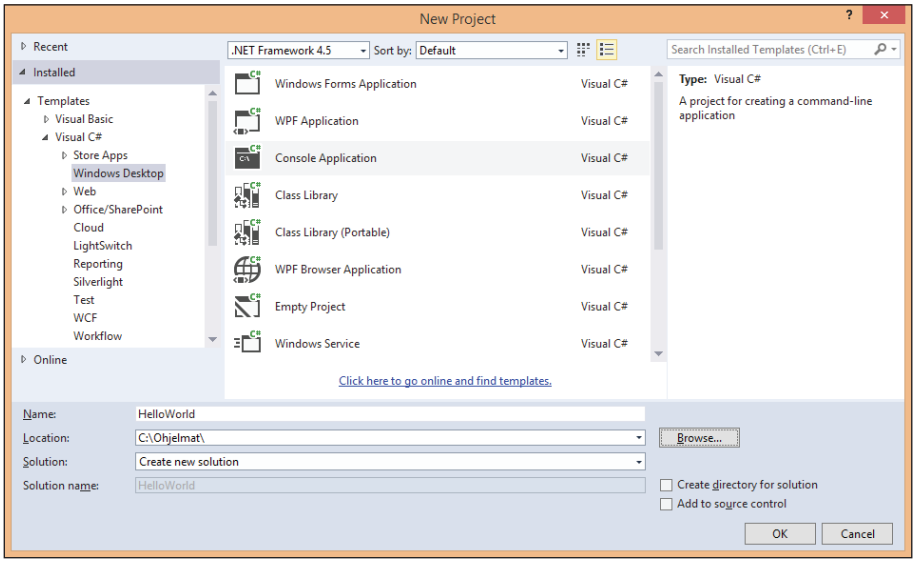
Sivustolta voi ladata joko sovelluskehittimen täyden version, jota voi käyttää ilmaiseksi 90 päivää tai ns. Express-version, jonka käyttöaika ei ole rajoitettu. Tässä julkaisussa esitetyt ohjelmat ovat toteutettavissa molemmilla versioilla. Express-version voi ladata edellä mainitulta sivustolta nimellä Express (versio) for Windows Desktop.

C#-kielen rakenne ja käyttö on esitetty seikkaperäisesti Microsoftin sivustolla (Microsoft 2015b). Kieli on laaja, tässä dokumentissa on käsitelty kielestä vain perusydin. Verkosta on löydettävissä ja ladattavissa runsaasti lisämateriaalia myös vastaalkajalle.

# 2 TIEDON KÄSITTELY MUUTTUJIEN AVULLA

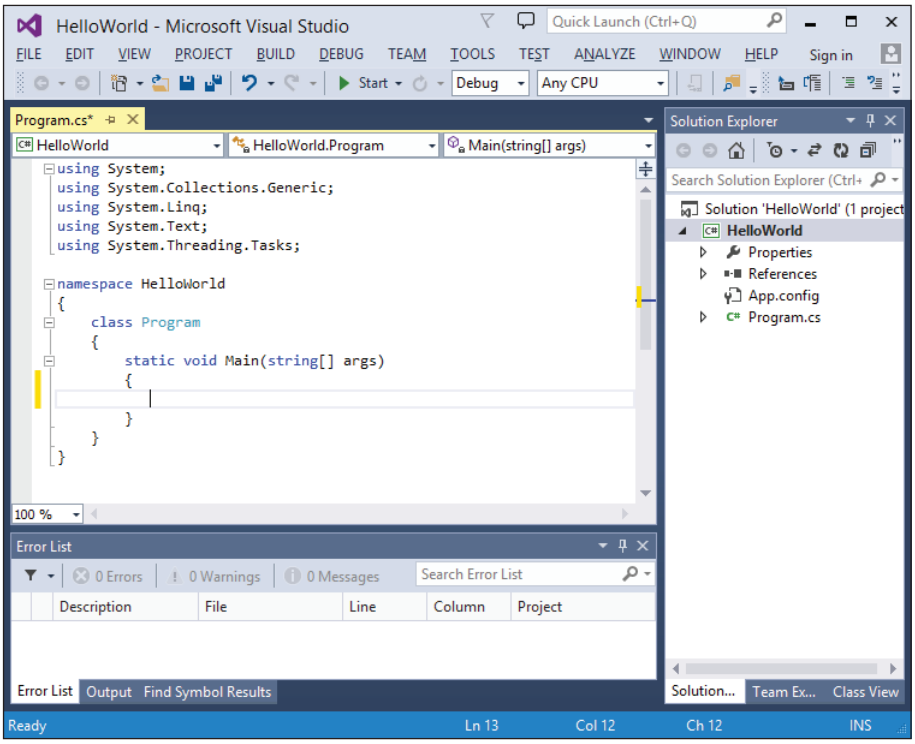
## 2.1 Ohjelman aloittaminen

Tutustutaan ohjelman tekemiseen perinteisellä Hello world-ohjelmalla. VisualStudiossa ohjelmat ohjelmoidaan projekteihin eli ensin VisualStudiolla luodaan uusi projekti: File – New project…. Valitaan käytettävä kieli (Visual C#), ohjelmaympäristö (Windows Desktop) ja mallipohjaksi Console Application. Eli kerrotaan kehittimelle, että se pohjustaa C#-kielisen ohjelman Windows Desktop-ympäristöön ja ohjelman käyttöliittymä perustuu konsolilla annettuihin komentoihin. Nimetään projekti (HelloWorld) ja määritetään kansio, johon ohjelma tallentuu. (kuvio 1).




KUVIO 1. Projektin perustaminen

Valitse Installed/Templates/Visual C#/Windows Desktop.
Valitse projektin tyypiksi Console Application.
Kirjoita ohjelman nimi Name-kohtaan.
Määrittele kansio, johon ohjelma talletetaan Location-kohtaan.
Ota rasti pois ruudusta Create directory for solution (ei välttämätöntä).
Paina OK-painiketta.




KUVIO 2. VisualStudion perustama ohjemaprojekti

VisualStudio-kehitin luo ohjelmaa varten valmiin pohjan (kuvio 2), johon tarvittavat lauseet lisätään. Konsoli-käyttöliittymällä varustettuun ohjelmaan tarvitaan VisualStudiossa vain kolmea ikkunaa: varsinainen ohjelmointialusta, SolutionExplorer ja ErrorList-ikkunat. Niiden näkyvyyttä säädetään View-valikon komennoilla.

Ohjelmaan, joka tervehtii konsolitulostuksena maailmaa sanoilla Hello world!, on kirjoitettava ainoastaan tulostuksen aikaansaava komento Main-metodiin. Komennossa kehotetaan kirjoittamaan rivi tekstillä Hello world! konsolille.

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello world!");
        }
    }
}
```

```
using System;
```

Using-avainsanalla kerrotaan, minkä nimiavaruuden sisältämiä luokkia C#-kielen kirjastosta halutaan koodissa käyttää. Using-määreellä saadaan koodin kirjoitusasu lyhemmäksi. Esimerkiksi yllä olevassa olisi ilman using-määrettä pitänyt kirjoittaa System.Console.WriteLine(...). Tässä ohjelmassa vain ensimmäistä using-määrettä tarvitaan, muut voi poistaa.

```
namespace HelloWorld
```

-määreellä ohjelmaluokka saadaan kuulumaan nimettyyn nimiavaruuteen, johon sitten voitaisiin viitata using-lauseella.

```
class Program
```

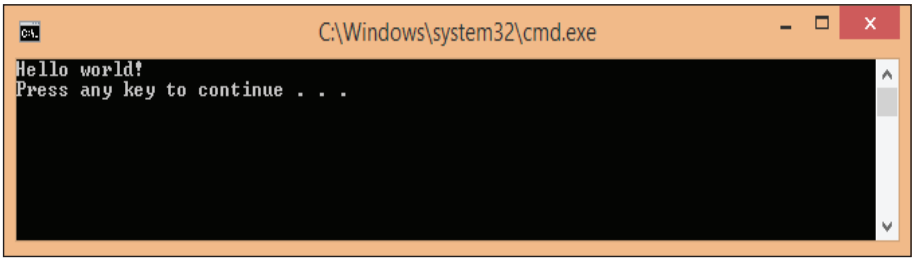
Kaikki C#-ohjelmat kirjoitetaan luokkaan. Kehitin nimeää luokan automaattisesti nimellä Program. Ohjelman nimi voidaan muuttaa Solution Explorer-ikkunassa rename-operaatiolla. Nimeämisessä pitää olla tarkkana, että tiedostolle jää tyypiksi .cs, esimerkiksi Hello.cs.

```
static void Main(string[] args)
```

Main on varsinainen pääohjelma, josta käynnistettävän ohjelman suoritus aloitetaan. Ohjelmoija kirjoittaa ohjelmassa suoritettavat lauseet Main-metodiin.

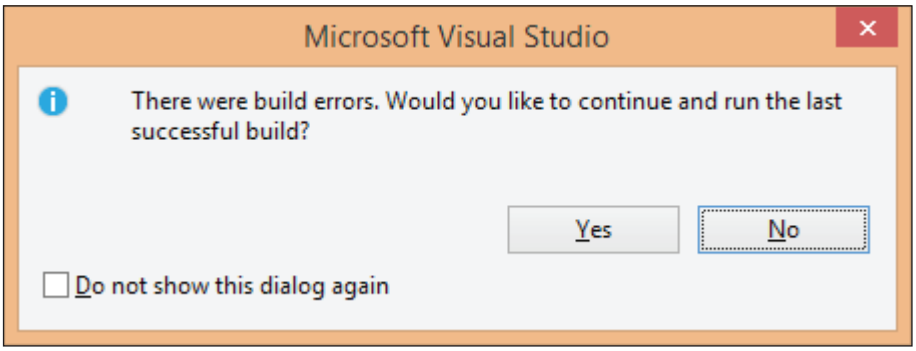
Huomaa lohkomerkit { }. Aaltosulut muodostavat parit, joiden sisälle lohkon asiat kirjoitetaan. Lohkot ovat sisäkkäisiä tai rinnakkaisia, ne eivät voi leikata toisiaan. Esitetyssä ohjelmassa on olemassa sisäkkäiset namespace-, class- ja main-lohkot.

Ohjelma voidaan VisualStudiossa kääntää ensin välikielelle ja sitten erikseen suorittaa tai käännös ja suoritus voidaan käynnistää yhtä aikaa. Pelkän käännöksen voi tehdä Build-valikosta komennolla Build Solution. Ohjelma ajetaan Debug-valikon Start without Debugging-komennolla (CTRL-F5). Ohjelma ajetaan erillisessä Console-ikkunassa, joka jää näkyviin kunnes painetaan jotain näppäintä (kuvio 3).




KUVIO 3. Ohjelma ajetaan konsoli-ikkunassa

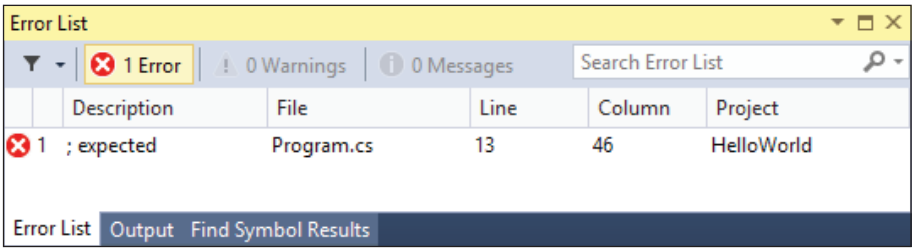
Jos ohjelman lauseissa on kielioppivirheitä, kääntäjä antaa niistä ilmoituksen (kuvio 4). Esimerkiksi, jos lauseen lopusta jää puolipiste pois:




KUVIO 4. Ilmoitus kielioppivirheestä

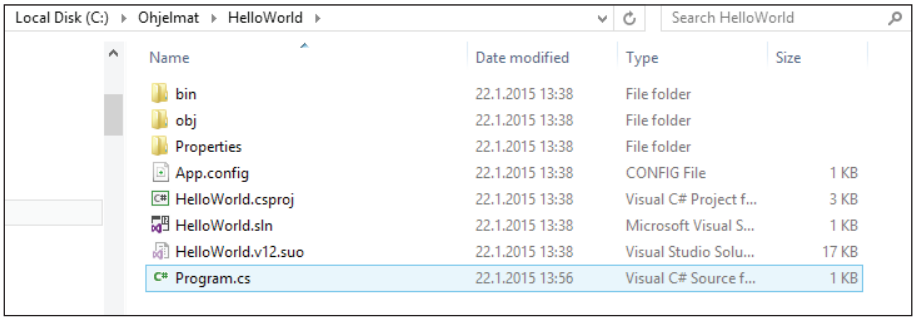
Tähän vastataan No-painikkeella, jolloin suoritus keskeytyy ja havaitut virheet voi korjata.

ErrorList-ikkunaan ruudun alalaitaan tulostuu kaikki havaitut käännösvirheet (kuvio 5).




KUVIO 5. Listaus virheistä Error List –ikkunassa

Tässä kääntäjä kertoo, puolipiste puuttuu riviltä 13. Virheselostetta kaksoisnapauttamalla kohdistus siirtyy suoraan virheen esiintymispaikkaan.




KUVIO 6. Projektin tiedostoja

Sovelluksesta muodostuu useita erilaisia tiedostoja (kuvio 6). Suoraan projektikansioon on sijoitettu ohjelmoimamme Program.cs. C#-ohjelmien tiedostotunniste on siis cs. Projektikansiossa on lisäksi HelloWorld.sln, joka sisältää tietoja sovellukseen kuuluvista projekteista. Sovellusta uudelleen esille otettaessa juuri *.sln-tiedosto avataan File-Open Project-komennolla. Tiedosto HelloWorld.suo sisältää käyttäjän VisualStudio-asetukset. Tiedosto HelloWorld.csproj sisältää projektin asetukset. HelloWorld.csproj.user sisältää käyttäjän projektikohtaiset asetukset. Ohjelma käännetään bin\debug-kansioon eli siellä oleva HelloWorld.exe on ohjelmamme välikielinen versio, joka voidaan ajaa kaksoisnapautuksella. Kansioon obj talletetaan kääntäjän väliaikaiset tiedostot ja Properties-kansiossa AssemblyInfo.cs-tiedostossa sijaitsevat sovelluksen attribuutit, jotka kääntäjä liittää muodostettavaan sovellukseen.

## 2.2 Kommentointi

Ohjelma on hyvä kommentoida, jotta sitä on helpompi myöhemmin ymmärtää. Käytännössä yleensä ohjelmaa ylläpitää joku muu kuin ohjelman alkuperäinen laatija, joten kommentointi on tärkeää. Kommentit eivät mitenkään vaikuta ohjelman toimintaan, vaan ne ovat vain vihjeitä ohjelman sisällöstä ja toimintatavoista koodin lukijalle.

Loppurivi saadaan kommentiksi merkitsemällä //
Usean rivin kommentti alkaa merkeillä /* ja päättyy merkkeihin */

Lisätään edelliseen ohjelmaan alkuun kommentti, mitä ohjelma tekee ja kuka sen on laatinut sekä kommentoidaan pääohjelman toimintaa. Lisäksi ohjelmasta on poistettu turhat using-lauseet.

```
using System;
/*
 * Sovellus tervehtii maailmaa tekstillä Hello world!
 * tekijä: Hilkka Niemelä
 */

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            // Tervehdyksen tulostus konsolille
            Console.WriteLine(”Hello world!”);
        }
    }
}
```

## 2.3 Muuttujat ja tyyppimäärittelyt

Jotta tietokoneella voi teettää laskutoimituksia, on ohjelmoijan nimettävä muistipaikat, joihin kone sijoittaa operandit ja lopputulokset (tehdään tilanvaraus). Näitä muistipaikkoja kutsutaan muuttujiksi. Voidaan siis määritellä, että käytetään muuttujia, joiden nimet ovat eka ja toka ja näiden summaa varten määritellään muuttuja summa. Annetaan ekalle ja tokalle arvot. Tämän jälkeen voidaankin käskeä koneen laskemaan ekan ja tokan summan lauseella:

summa = eka + toka;

Aivan näin yksinkertainen tilanne ei kuitenkaan ole. Myös muuttujien tyyppi on määriteltävä etukäteen. Tyyppi tarkoittaa, minkälaista tietoa muuttujaan aiotaan sijoittaa. Erilaisia tyyppejä ovat esimerkiksi kokonaisluku, liukuluku (desimaaliluku) ja merkkijono.

Tehdään uusi ohjelma nimeltään Yhteenlasku eli luodaan uusi projekti, jolle annetaan nimeksi Yhteenlasku.

```
class Program
{
        static void Main(string[] args)
        {
            // määritellään kolme kokonaisluku-muuttujaa
            int eka, toka, summa;
            //sijoitetaan yhteenlaskettaville alkuarvot
            eka = 10;
            toka = 50;
            // lasketaan summa
            summa = eka + toka;
            //tulostetaan vastaus
            Console.WriteLine(”Ekan ja tokan summa on {0} ”,
                              summa);
            Console.WriteLine(”Luvun {1} ja {2} summa on {0} ”,
                              summa,eka,toka);
        }
}
```

int-määreellä esiteltyyn muuttujaan saa sijoittaa vain negatiivisia tai positiivisia kokonaislukuja.

Muuttujaan tehtävät sijoitukset ilmaistaan yhtäsuuruus = -merkillä.

Tulostuslauseessa sellaisenaan tehtävät tulostukset sijoitetaan lainausmerkkien väliin. Kun halutaan koneen tulostavan muuttujan sisällön, kirjoitetaan muuttujan paikalle tulostuslauseeseen **aalto**sulkujen sisään parametrinumero. Lauseessa voi olla useita parametreja. Parametrien numerointi alkaa aina nollasta. Järjestyksessä parametrien ei tarvitse esiintyä eli lauseessa voi olla ensimmäisenä mainittu esimerkiksi parametri {2}, mutta silloin lauseessa on aina esiinnyttävä myös parametrit {0} ja {1}. Lainausmerkin jälkeistä pilkkua seuraavat muuttujat, joiden sisältö tulee parametrien paikalle. Muuttujien on oltava järjestyksessä: nollan sijaan tuleva, ykkösen sijaan tuleva jne.

Taulukossa 1 on esitetty listaus C#-kielessä olevista perusmuuttujatyypeistä.

TAULUKKO 1. C#-kielen perusmuuttujatyypit (Microsoft 2015a).

| C# Type | .NET Framework type (aliasnimi) |
|---|---|
| bool (true, false) | **System.Boolean** |
| byte (8 bit, etumerkitön, 0 - 255) | **System.Byte** |
| sbyte (8 bit, etumerkillinen, -128 - 127) | **System.SByte** |
| char (16 bit unicode merkki) | **System.Char** |
| decimal (128 bit, vakiotunnus m, arvoalue $7.9*10^{28}$) | **System.Decimal** |
| double (64 bit, vakiotunnus d, arvoalue $1.7*10^{308}$) | **System.Double** |
| float (32 bit, vakiotunnus f, arvoalue $3.4*10^{38}$) | **System.Single** |
| int (32 bit, kokoanaisluku) | **System.Int32** |
| uint (u = etumerkitön) | **System.UInt32** |
| long (64 bit, kokoanaisluku) | **System.Int64** |
| ulong (u = etumerkitön) | **System.UInt64** |
| short (16 bit, kokoanaisluku) | **System.Int16** |
| ushort (u = etumerkitön) | **System.UInt16** |
| string (merkkijono) | **System.String** |

Taulukossa olevista tyypeistä kokonaislukuja edustavat **byte, sbyte, short, ushort, int, uint, long ja ulong**. Ohjelmoinnin tekniikan kannalta ne ovat toistensa kaltaisia, erona on se, kuinka iso luku muuttujaan mahtuu.

**byte** on etumerkitön kokonaisluku (käytännössä siis ei-negatiivinen), joka esitetään kahdeksalla bitillä. Lukualue, joka sillä voidaan esittää on 0 – 255, joka tulee kaksijärjestelmästä. Kahdeksan bittiä voidaan esittää $2^8$:lla (=256) eri tavalla. Koska nolla otetaan mukaan, lukualueeksi tulee 0 – 255.

**sbyte** on 8 bitin kokonaisluku, joka voi olla negatiivinenkin eli yksi bitti käytetään etumerkin esittämiseen, jolloin lukualueeksi jää -128 – 127.

**short** on 16 bitillä esitettävä etumerkillinen kokonaisluku ja **ushort** vastaava etumerkitön.

**int** on etumerkillinen 32 bitillä esitettävä kokonaisluku ja **uint** vastaava etumerkitön. **long** on etumerkillinen 64 bitillä esitettävä kokonaisluku ja **ulong** vastaava etumerkitön.

Liukulukuja on kolmea kokoluokkaa **float** (32 b), **double** (64 b) ja **decimal** (128b).

**bool** on totuusarvo. Totuusarvoiseen muuttujaan voidaan sijoittaa arvot **true** tai **false**, joko suorana sijoituksena tai loogisen lausekkeen arvona.

**string**-tyyppiseen muuttujaan voidaan sijoittaa merkkijono. Merkkijonovakiot ympäröidään aina lainausmerkein. String on näistä erikoistapaus, se on viitemuuttuja, mutta sitä voidaan käyttää arvomuuttujan tapaan.

```
string nimi = “Uolevi”;
```

**char** voi sisältää yhden merkin. Merkit muodostuvat kahden tavun mittaisesta Unicode-koodatusta mistä tahansa kielen merkistä (suomi, venäjä, arabia,...) tai erikoismerkistä (piste, pilkku,..). Koska merkki on 16 bitin koodi, on se samalla myös merkin kokonaislukukoodi. Merkkivakiot ympäröidään heittomerkeillä.

```
char merkki = ‘a’;
char välimerkki = ‘!’;
```

Muuttujatyyppiin mahtuvan pienimmän tai suurimman arvon saa selville MinValue- ja MaxValue-arvoilla.

```
byte tavu = byte.MaxValue;
double desimaaliluku = double.MinValue;
```

Jos muuttujaan yrittää sijoittaa suuremman arvon kuin siihen mahtuu, aiheutuu virhe. Tilanteesta riippuen virhe syntyy joko heti käännösvaiheessa tai suoritusvaiheessa virheellisenä toimintana.

## 2.4 Nimet ja operaattorit

**Nimet** ovat ohjelmoijan valitsemia nimiä esimerkiksi muuttujille ja luokille. C#-kielessä nimet voivat sisältää numeroita ja kirjaimia, mutta nimen pitää alkaa kirjaimella. Nimet ovat case sensitive eli esimerkiksi muuttuja A ja a ovat erinimisiä. Microsoft suosittelee, että muuttujien nimet alkavat pienellä kirjaimella (camel notation) ja muut isolla kirjaimella (Pascal notation).

**Vakiot** ovat koodiin kirjoitettuja muuttujan arvoja, esimerkiksi:

int luku = 16;
string vastaus = ”torstai”;

Merkkijonovakiot (eli merkkijonon ’sisältö’) kirjoitetaan aina lainausmerkeissä. Vakioluvut kirjoitetaan sellaisenaan. Jos numeroita käsitellään merkkijonona, esimerkiksi postinumero (sillä ei lasketa mitään), sijoitetaan tällainen numero **string**-tyyppisen muuttujan arvoksi: **string** postinumero = ”60320”;

Myös muuttuja voidaan määritellä vakioksi avainsanalla **const**, eli tällaisen muuttujan arvoa ei voi muuttaa. **const int** jouluaatto = 24;

**Operaattorit** ovat komentoja, joilla käsitellään muuttujia ja vakioita.

Aritmeettisia operaattoreita ovat:
+ yhteenlasku
- vähennyslasku
* kertolasku
/ jakolasku
% jakojäännös

## 2.5 Konsolilta lukeminen ja konsolille kirjoittaminen

Konsolille tulostamiseen ja lukemiseen käytetään .NET Frameworkin System-nimiavaruuden luokan Console metodeita Write/WriteLine ja Read/ReadLine.

Lukeminen:

| | |
|---|---|
| Read( ) | lukee seuraavan **merkin** |
| ReadLine( ) | lukee seuraavan **rivin** |

Kirjoittaminen:

| | |
|---|---|
| Write(...) | kirjoittaa suluissa määritellyt tulosteet |
| WriteLine(...) | kirjoittaa määritellyt tulosteet **ja tekee rivinvaihdon** |

```
Console.WriteLine("Minkä ikäinen olet?");
/* kirjoitetaan suluissa oleva lause ja vaihdetaan riviä (=Line-
operaatio)*/

int ika = int.Parse(Console.ReadLine());
/* Lause on sijoituslause, jonka oikea puoli ratkaistaan ensin.
Luetaan konsolilta rivi ReadLine-metodilla. ReadLine-metodi
lukee merkkejä, esimerkiksi merkit 2 ja 3. Koska ne halutaan
sijoittaa kokonaislukumuuttujaan, ne on muutettava luvuksi 23.
Muunnos tehdään muuttujatyypin Parse-metodilla. Lopuksi syntynyt
kokonaisluku sijoitetaan muuttujan ika arvoksi. */

Console.Write("Mikä on nimesi? ");
/* Kirjoitetaan suluissa oleva merkkijono. Kohdistin jää samalle
riville! (ei ole Line-operaatiota)
*/

string nimi = Console.ReadLine();
/*  Luetaan merkit konsolilta ja sijoitetaan ne merkkijono-
tyyppisen muuttujan arvoksi. Parse-metodilla tehtävää muunnosta
ei tarvita, koska luettu on valmiiksi merkkejä, joita voi
sijoittaa string-tyyppisen muuttujan arvoksi.
*/

int c = Console.Read();
/* Luetaan yksittäinen merkki ja sijoitetaan se
kokonaisluku-tyyppisen muuttujan arvoksi. */
Console.WriteLine(c);
/* Tulostetaan annetun merkin kokonaislukukoodi ja
tehdään perään rivinvaihto */
```

Jos käyttäjältä kysytään lukua, luetaan se ReadLine-metodilla ja luettu merkkijono muutetaan oikeaan muotoon muuttujatyypin Parse-metodilla.

```
int i = int.Parse(Console.ReadLine());
double joku = double.Parse(Console.ReadLine());
byte tavu = byte.Parse(Console.ReadLine());
ushort lyhyt = ushort.Parse(Console.ReadLine());
```

## Tulostuksen muotoilu

```
Console.WriteLine("{0} on {1} vuotias.", nimi, ika);
/*  Tulostettava merkkijonovakio kuvataan lainausmerkeissä.
Niihin kohtiin tulostuslausetta, joissa halutaan tulostuvan
muuttujan sisältö tai laskutoimituksen lopputulos merkitään
parametrinumero aaltosulkeissa. Numerointi alkaa nollasta.
Parametrien sijaan tulostuvat muuttujat ja laskutoimitukset
luetellaan tulostuslauseen sulkevan lainausmerkin jälkeen
pilkuilla erotettuna. Muuttujat on oltava järjestyksessä: ensin
0-parametrin sijaan tuleva, sitten 1-parametrin sijaan tuleva
jne.
*/

// sama tulostus voidaan toteuttaa myös lauseella
Console.WriteLine(nimi + " on " + ika + " vuotias.");
/* Muuttujat ja sellaisenaan tulostettava teksti vuorottelevat.
Vuoron vaihtuessa väliin lisätään +-merkki. Teksti ympäröidään
lainausmerkein, muuttujat ja laskutoimitukset kirjoitetaan ilman
niitä.
*/
```

## Yleisimpiä muotoilukomentoja

Rivinvaihto saadaan aikaan merkinnällä \n (new line).

Joitakin merkkejä käytetään erikoismerkkeinä tulostuksen yhteydessä. Muuhun käyttöön varattu merkki saadaan tulostettua merkitsemällä sen eteen \-merkin. Esimerkiksi lainausmerkki on varattu osoittamaan vakiomerkkijonon alkamista ja päättymistä. Jos pitää saada tulostettua pelkkä lainausmerkki, sen eteen sijoitetaan kenoviiva: \" .

Luvun tulostus halutulla desimaalimäärällä ilmaistaan parametrinumeron jälkeisellä merkinnällä

```
Console.WriteLine("{0:F2}",luku);
/* luku halutaan tulostaa kahdella
desimaalilla, F on lyhennys sanasta fixed */
```

Valuutan muotoilu aikaansaadaan merkinnällä C, mutta merkin koodauksen vuoksi konsolilla näkyy €-merkin tilalla ?-merkki.

```
// Päiväyksen tulostus Console.WriteLine("Tänään on " +
                                DateTime.Now.ToShortDateString());
```

## 2.6 Esimerkkejä muuttujien käsittelystä

**Esimerkki 1. Pinta-alojen laskenta**

```
using System;
/* Ohjelma laskee ja tulostaa  pinta-alat:
Pinta-ala neliöstä.
Ympyrän pinta-ala.
Suorakulmion pinta-ala.
Kolmion pinta-ala.
Lopuksi ohjelma tulostaa kaikkien kuvioiden yhteenlasketun pinta-
alan.
 */

namespace PintaAlat
{
    class Program
    {
        static void Main(string[] args) // pääohjelma
        {
            // neliö
            double sivu,nAla;
            Console.WriteLine(”Anna neliön sivun pituus”);
            // double.Parse muuntaa konsolilta luetut merkit
            // desimaaliluvuksi
           sivu =double.Parse( Console.ReadLine());
           nAla = sivu * sivu;
           Console.WriteLine(”Neliön pinta-ala on {0}”, nAla);

            // suorakulmio
           double sivu1, sivu2, sAla;
           Console.WriteLine(”Anna suorakulmion sivut omilla
                            + riveillään”);
           sivu1 = double.Parse(Console.ReadLine());
           sivu2 = double.Parse(Console.ReadLine());
           sAla = sivu1 * sivu2;
           Console.WriteLine(”Suorakulmion pinta-ala on {0}”,
                             sAla);

            // ympyrä
           double säde, yAla;
           Console.WriteLine(”Anna ympyrän säde”);
           säde = double.Parse(Console.ReadLine());
           yAla = Math.PI * Math.Pow(säde, 2);
            // f2 tarkoittaa, että tulos pyöristetään kahdelle
            // desimaalille
           Console.WriteLine(”Ympyrän pinta-ala on {0:f2}”,
                             yAla);

            // kolmio
           int kanta, korkeus;  // kokonaislukuvaraus
           double kAla;
           Console.WriteLine(”Anna kolmion kanta ja korkeus ”
                            +”omilla riveillään”);
```

```
            kanta = int.Parse(Console.ReadLine());
            korkeus = int.Parse(Console.ReadLine());

/* c# tekee kokonaisjaon, jos sekä jakaja että jaettava ovat
kokonaislukuja toinen pitää muuttaa desimaaliseksi esim. 2 => 2.0
tai 1.0*jaettava */
            kAla = kanta * korkeus /2.0;
            kAla = 1.0 *kanta * korkeus / 2;
            Console.WriteLine(”Kolmion pinta-ala on {0}”, kAla);

            // kaikki yhteensä
            Console.WriteLine(”Kaikki yhteensä {0:f2}”, nAla +
                                sAla + yAla + kAla);
        }
    }
}
```

**Esimerkki 2. Matkaan kuluvan ajan laskenta**

```
using System;
/* Ohjelma laskee, kuinka kauan matka jollekin
 * paikkakunnalle kestää, kun ajan laskemiseen käytetään
 * keskinopeutta. Aika tulostetaan kokonaisina tunteina
 * ja minuutteina.
 * */
namespace Matka
{
    class Program
    {
        static void Main(string[] args)
        {
            double matka, aika, nopeus;
            int tunnit, minuutit;

                Console.Write(“\nMatka? “);
                matka = double.Parse(Console.ReadLine());
                Console.Write(“Nopeus? “);
                nopeus = double.Parse(Console.ReadLine());

                aika = matka / nopeus;

                tunnit = (int)aika; // tipauttaa desimaalit pois
                minuutit = (int)((aika - tunnit) * 60);

                Console.WriteLine(”Matka kestää {0} tuntia {1} ”
                            +”minuuttia”, tunnit,minuutit);
        }
    }
}
```

**Esimerkki 3. Jakojäännöslaskutoimitus**

```
using System;
/* Ohjelma kysyy, kuinka suuri ryhmä on
 * tulossa Gondoli-hissiin. Yhteen hissiin mahtuu 8 henkilöä.
 * Ohjelma tulostaa, montako täyttä hissiä tulee ja montako
 * henkilöä on viimeisessä hississä,
 * kun hissit täytetään järjestyksessä täyteen.
 */
namespace Hissi
{
    class Program
    {
        static void Main(string[] args)
        {
            int ryhmänKoko, hissinKoko = 8, täydetHissit,
                viimeisessä;

                 Console.Write(”\nRyhmän koko? ”);
                ryhmänKoko = int.Parse(Console.ReadLine());

                // tekee kokonaisjaon
                // koska jaettava ja jakaja ovat kokonaislukuja
                täydetHissit = ryhmänKoko / hissinKoko;

                // % on jakojäännös merkki
                // vastaus on kuinka paljon kokonaisjaosta jää yli

                viimeisessä = ryhmänKoko % hissinKoko;
                Console.WriteLine(”{0} hengen ryhmästä tulee {1} ”
                + ”täyttä hissiä, viimeiseen hissiin jää {2} ”
                + ”henkilöä”, ryhmänKoko, täydetHissit,
                    viimeisessä);
         }
    }
}
```

**Esimerkki 4. Merkkijonon käsittely**

```
using System;
/* Ohjelma kysyy käyttäjän etu- ja sukunimen erikseen ja tulostaa
koko nimen.  */
namespace Nimi
{
    class Program
    {
        static void Main(string[] args)
        {
            string etunimi, sukunimi, kokonimi;
            Console.Write("Etunimi? ");
            etunimi = Console.ReadLine();
            Console.Write("Sukunimi? ");
            sukunimi = Console.ReadLine();
       // vaihtoehtoisia tulostustapoja
            kokonimi = etunimi + " " + sukunimi;
            Console.WriteLine(etunimi + " " + sukunimi);
            Console.WriteLine(kokonimi);
            Console.WriteLine("{0} {1}", etunimi, sukunimi);
        }
    }
}
```

# 3 KONTROLLIRAKENTEET

Harvoin jos koskaan käytännön sovellusohjelma voidaan laatia siten, että kone suorittaa sen alusta loppuun edeten komentojen esiintymisjärjestyksessä. Useimmiten kone on ohjattava tarkistamaan joidenkin ehtojen voimassaolo ja sen pohjalta päättelemään, mitkä komennot seuraavaksi suoritetaan. Tällaisia lauseita kutsutaan valintarakenteiksi.

## 3.1 Valintarakenteet

C#-kielessä valintarakenteita ovat if-lause ja switch-case –lause.

### 3.1.1 if-lause

**if-lausekkeen** rakenne on seuraava:

```
if ( looginen-lauseke)
  {
     MitäTehdäänLausekkeet;
   }
else
  {
     MitäTehdäänLausekkeet;
   }
```

Lauseen aloittaa avainsana **if**. Sen jälkeen suluissa on looginen lauseke, joka saa arvokseen joko **true** tai **false**. Jos looginen lauseke saa arvokseen true, kone suorittaa komennot, jotka on kirjoitettu välittömästi seuraavaan lohkoon. Lohkon lausekkeet suoritettuaan kone ohittaa else-lohkon eli sen lohkon lauseet jätetään suorittamatta.

Jos looginen lauseke saa arvokseen **false**, hypätään välittömästi seuraava lohko ohi ja suoritetaan else-lohkon lauseet. Else-lohko ei ole pakollinen. Jos else-lohko puuttuu ja loogisen lausekkeen arvoksi tulee false, ei tehdä mitään.

if-lauseita voi myös ketjuttaa eli testataan vuoronperään erilaisia vaihtoehtoja. Kun oikea vaihtoehto löytyy, suoritetaan sen kohdalta MitäTehdäänLausekkeet ja koko loppuosa ohitetaan.

```
if ( looginen-lauseke)
  {
    MitäTehdäänLausekkeet;
  }
else if ( looginen-lauseke)
  {
    MitäTehdäänLausekkeet;
  }
else
  {
    MitäTehdäänLausekkeet;
  }
```

## Loogiset lausekkeet

Loogisia lausekkeita rakennetaan vertailuista, loogisista operaatioista ja toisista loogisista lausekkeista.

Vertailuoperaatioita:

| | |
|---|---|
| > | suurempi kuin |
| >= | suurempi tai yhtäsuuri |
| < | pienempi kuin |
| <= | pienempi tai yhtäsuuri |
| == | yhtäsuuri |
| != | erisuuri |

Kaikki vertailut tuottavat totuusarvon **true** tai **false**. Loogisen lausekkeen tuloksen voi sijoittaa totuusarvoisen muuttujan arvoksi. Totuusarvoinen muuttuja määritellään määreellä **bool**.

Vertailuja joudutaan usein ketjuttamaan eli tarkistamaan, onko samanaikaisesti useita erilaisia ehtoja voimassa. Vertailuja ketjutetaan loogisilla operaatiomerkeillä:

| | |
|---|---|
| && | ja-operaatio |
| \|\| | tai-operaatio |
| ^ | xor-operaatio |
| ! | ei-operaatio |

**Esimerkki 1. Joko-tai tilanne**

```
/*
 * Ohjelmalla testataan, onko tivolin laitteeseen tuleva
 * henkilö riittävän pitkä
 */

using System;
namespace IfLause
{
    class Program
    {
        static void Main(string[] args)
        {
            int pituus, raja = 120;
            Console.Write("Kuinka pitkä olet (cm)? ");
            pituus = int.Parse(Console.ReadLine());

            // testataan onko pituutta riittävästi
            if (pituus < raja) // looginen lauseke
            {
                Console.WriteLine("Et voi käyttää laitetta!");
            }
            else
            {
                Console.WriteLine("Tervetuloa laitteeseen!");
            }
        }
    }
}
```

**Esimerkki 2. Vertailuketju**

```
/*
 * Ohjelmalla testataan, onko pakkanen, nollakeli, lämpöasteita
vai helle
 */

using System;
namespace IfLause
{
    class Program
    {
        static void Main(string[] args)
        {
            double asteet;
            Console.Write("Kuinka paljon ulkona on asteita? ");
            asteet = double.Parse(Console.ReadLine());

            // testataan asteiden määrä alhaalta ylöspäin
            if (asteet < -1) // looginen lauseke
            {
                Console.WriteLine("Ulkona on pakkasta!");
            }
            else if (asteet < 1)
            {
                Console.WriteLine("Ulkona on nollakeli!");
            }
            else if (asteet < 24)
            {
                Console.WriteLine("Ulkona on lämpöasteita!");
            }
            else // Huom. ei tarvitse if-lausetta
            {
                Console.WriteLine("Ulkona on helle!");
            }
        }
    }
}
```

**Esimerkki 3. Loogiset vertailut**

```
using System;
namespace LoogisetLausekkeet
{
    class Program
    {
        static void Main(string[] args)
        {
            bool onkoTotta;
            int a = 6, b = 7;

            onkoTotta = (a > b);  // false
            onkoTotta = (a < b);  // true
            onkoTotta = (a == b);  // false
            onkoTotta = (a >= b);  // false
            onkoTotta = (a <= b);  // true
            onkoTotta = (a != b);  // true
        }
    }
 }
```

**Esimerkki 4. Loogisten vertailujen ketjuttaminen**

```
using System;

namespace LoogisetLausekkeet
{
    class Program
    {
        static void Main(string[] args)
        {
            bool onkoTotta;
            int a = 6, b = 7, c = -3;

            onkoTotta = (a > b && b > c);  // false
            onkoTotta = (a > b || b > c);  // true
            onkoTotta = (!(a > b) && b > c);  // true
            onkoTotta = (a > b ^ b > c);  // true
              onkoTotta = (a > c ^ b > c);  // false
            onkoTotta = !(a > b && b > c);  // true
        }

    }
}
```

**Esimerkki 5. Kolmion suorakulmaisuuden tarkistaminen**

```
using System;
/* Ohjelma, joka tulostaa, onko kolmio suorakulmainen, kun sen
kateettien ja
 * hypotenuusan pituudet annetaan . */
namespace Kolmio
{
    class Program
    {
        static void Main(string[] args)
        {
                double kateettiA, kateettiB, hypotenuusa;

                 Console.WriteLine(”Anna kateettien pituudet”);
                kateettiA = double.Parse(Console.ReadLine());
                kateettiB = double.Parse(Console.ReadLine());
                 Console.WriteLine(”Anna hypotenuusan pituus”);
                hypotenuusa = double.Parse(Console.ReadLine());

                if (hypotenuusa * hypotenuusa ==
                   kateettiA * kateettiA + kateettiB * kateettiB)
                  {
                 Console.WriteLine(”Kolmio on suorakulmainen”);
                }
                else
                {
                 Console.WriteLine(”Kolmio ei ole ”
                                      +”suorakulmainen”);
                }
        }
    }
}
```

**Esimerkki 6. Arvosanan määräytyminen pistemäärän perusteella**

```
using System;
/* Ohjelma tulostaa kokeesta saadun pistemäärän perusteella
kokeen arvosanan.
 * Arvosanarajat ovat
0 - 10:       0
11 - 20:      1
21 - 30:      2
31 - 40:      3
41 - 50:      4
51 - 60:      5
 */
namespace Arvosanat
{
    class Program
    {
        static void Main(string[] args)
        {
              double pistemäärä;
              Console.Write("Pistemäärä? ");
              pistemäärä = double.Parse(Console.ReadLine());

              if (pistemäärä < 0)
                  Console.WriteLine("Virheellinen pistemäärä!");
              else if (pistemäärä <= 10)
                  Console.WriteLine("Arvosana on 0");
              else if (pistemäärä <= 20)
                  Console.WriteLine("Arvosana on 1");
              else if (pistemäärä <= 30)
                  Console.WriteLine("Arvosana on 2");
              else if (pistemäärä <= 40)
                  Console.WriteLine("Arvosana on 3");
              else if (pistemäärä <= 50)
                  Console.WriteLine("Arvosana on 4");
              else if (pistemäärä <= 60)
                  Console.WriteLine("Arvosana on 5");
              else
                  Console.WriteLine("Virheellinen pistemäärä!");
         }
    }
}
```

### 3.1.2 switch-case lause

**switch-case** lause toimii siten, että sen avulla on mahdollista haarautua yhteen lukuisasta määrästä erilaisia vaihtoehtoja. Haarautuminen perustuu lausekkeesta tai muuttujasta saatuun arvoon.

**switch-case** lause on muotoa:
**switch** (arvon saava lauseke)
{
 **case** arvo1: TeeJotain1;
 **break**;
 **case** arvo2: TeeJotain2;
 **break**;
 **case** arvo3: TeeJotain3;
 **break**;
 **case** arvo4: TeeJotain4;
 **goto case** arvo2;
 **default**: TeeJotain;
 **break**;
}

**case** on aina lopetettava joko break- tai goto komentoon, suoritus ei automaattisesti siirry seuraavaan caseen. **goto**-komennolla suorituksen saa siirtymään annettuun osoitteeseen. **default**-haara on niitä arvoja varten, joille ei ole omaa osumaa.

```
static void Main(string[] args)
{
   Console.Write("Anna luku! ");
   int luku = int.Parse(Console.ReadLine());
   switch (luku)
   {
         case 1: Console.WriteLine("Annoit ykkösen");
                 break;
         case 2: Console.WriteLine("Annoit kakkosen");
                 break;
         case 3: Console.WriteLine("Annoit kolmosen");
                 break;
         case 4: Console.WriteLine("Annoit nelosen");
                 break;
         default: Console.WriteLine("Annoit muun kuin 1 - 4");
                 break;
    }
}
```

Valinnan perusteena oleva lauseke voi saada muunkin tyyppisen arvon kuin kokonaisluvun.

```
int reds = 0, blues = 0, greens = 0, other = 0;
Console.Write("Anna väri (red,blue,green,other color)! ");
string color = Console.ReadLine();
switch (color)
{
     case "red": reds++; // reds = reds + 1;
                 break;
     case "blue": blues++;
                 break;
     case "green": greens++;
                   break;
     default: other++;
              break;
}
```

Lausekkeen ei myöskään tarvitse olla yksinkertainen muuttuja, vaan mikä tahansa arvon saava lauseke kelpaa.

```
Console.Write("Anna luku! ");
int luku = int.Parse(Console.ReadLine());
switch (luku*10/2-luku)
{
        case 4: Console.WriteLine("Annoit ykkösen");
                        break;
        case 8: Console.WriteLine("Annoit kakkosen");
                        break;
        case 12: Console.WriteLine("Annoit kolmosen");
                        break;
        case 16: Console.WriteLine("Annoit nelosen");
                        break;
        default: Console.WriteLine("Annoit muun kuin 1 - 4");
                        break;
}
```

## 3.2 Toistorakenteet

Toisto tarkoittaa sitä, että samat lausekkeet toistetaan useampaan kertaan. Toistoja voi olla kiinteä määrä tai loogisella lausekkeella testattava tilanteesta riippuva määrä.

Toistorakenteita C#-kielessä ovat for-, foreach- ,while- ja do-while –lauseet.

### 3.2.1 for-lause

**for**-lause soveltuu hyvin tilanteeseen, jossa toistojen määrä on etukäteen tiedossa joko absoluuttisesti tai muuttujan arvona. Esimerkiksi tiedetään, että kuukausia on 12 ja viikonpäiviä 7. Jos jokaista kuukautta tai viikonpäivää kohden on tehtävä tietyt toimenpiteet, ne voidaan toistaa for-lausetta käyttäen. Aivan samoin, jos ennen toiston alkua on selvitettävissä toistojen määrä, se sijoitetaan muuttujan arvoksi ja toistoja suoritetaan muuttujan ilmoittama määrä. Monimutkaisempiakin toistoja for-lauseella on mahdollista totetuttaa, mutta muut toistorakenteet soveltuvat niihin tilanteisiin paremmin.

**for**-lausetta käytetään myös lukualueiden läpikäyntiin.

**for**-lauseen muoto:
**for** (laskurin alkutilan asetus; jatkamisehto; laskurin kasvatus tai vähennys)
{
    toistettavat lauseet;
}

Toistokertojen laskurina toimii yleensä int-tyyppinen muuttuja, joka nimetään yleisimmin i:ksi. Yleisin lähtöarvo muuttujalla on 0, mutta mikä tahansa alkuarvo on mahdollinen.

Toistoehto on looginen lauseke, jolla määritellään minkälaisilla laskurin arvoilla toistoa jatketaan.

Laskurin kasvatuksella/vähentämisellä määritellään, minkä verran laskurin arvoa kasvatetaan/vähennetään jokaisen toistokerran jälkeen.

**Esimerkki 1. Viiden luvun summa**

```
static void Main(string[] args)
{
      // lasketaan viiden luvun summa
       int summa = 0, luku;
       for (int i = 0; i < 5; i++)
       {
           // toistettava lauseet
            Console.Write("Anna luku ");
            luku = int.Parse(Console.ReadLine());
            summa = summa + luku; // summa += luku;
        }
            Console.WriteLine("Summa on " + summa);
  }
```

```
int i = 0;
/* määritellään toistomuuttuja ja annetaan sille alkuarvoksi 0
tämä osa lausekkeesta tehdään vain kerran lauseeseen tullessa */
i < 5;
/* toistoa jatketaan niin kauan kun lauseke saa arvon true -
tehdään aina ennen toistettavien lauseiden suoritusta */
i++
/* laskurin arvon kasvatus (sama kuin i = i + 1) - tehdään aina
toistettavien lauseiden suorituksen jälkeen */
```

Rutiinin eteneminen:

```
          a)-vain kerran                b)          d)
for    (   int i = 0;            i < 5;      i++     )

       {   c)
           // toistettava lauseet
            Console.Write("Anna luku ");
            luku = int.Parse(Console.ReadLine());
            summa = summa + luku;
        }
```

Kun kone suorittaa for-lauseen, se lauseeseen tullessaan tekee kohdan a) eli asettaa laskurin alkuarvon.

Seuraavaksi alkaa **toistorutiini**:

1. Tarkistetaan toistoehdon voimassa olo eli suoritetaan kohta b)
2. Jos toistoehto saa arvon true, suoritetaan lauseet c), jos arvoksi tulee false, kone jatkaa for-lauseen sulkevan aaltosulun jälkeisestä lauseesta.
3. Toistettavien lauseiden suorituksen jälkeen palataan aina kohtaan d) eli kasvatetaan laskuria.
4. Laskurin kasvatuksen jälkeen toistorutiini jatkuu kohdasta b).

**Esimerkki 2. Lukujen summa**

```
static void Main(string[] args)
{
    // lukujen summa
    int summa = 0, luku, lkm;
    Console.Write("Monenko luvun summan haluat laskea? ");
    lkm = int.Parse(Console.ReadLine());
    for (int i = 0; i < lkm; i++)
    {
              Console.Write("Anna luku ");
              luku = int.Parse(Console.ReadLine());
              summa = summa + luku;
     }
    Console.WriteLine("Summa on {0}" , summa);
 }
```

Esimerkissä toistojen määrä riippuu käyttäjän ilmoittamasta luvusta.

**Esimerkki 3. Lukualueen läpikäynti**

```
/* Ohjelma tulostaa muunnostaulukon tuumista senteiksi halutulta
väliltä.
Muunnostaulukko:
1”     2.54 cm
2”     5.08 cm
:        :
*/
static void Main(string[] args)
{
   int alku, loppu;

    Console.WriteLine(”Anna tuuma alueen alku ja loppu”);
    alku = int.Parse(Console.ReadLine());
    loppu = int.Parse(Console.ReadLine());

    for (int i = alku; i <= loppu; i++)
         // i:n sisältönä on tuumamäärä
    {
         /* tuuman merkki täytyy tulostaa \-merkin kanssa, muuten
         se tulkitaan tulostuksen kuvauksen loppumerkiksi \t on
         sarkainväli */
         Console.WriteLine(“{0}\”\t{1:f2} cm”, i, i * 2.54);
     }
}
```

### 3.2.2 foreach-lause

**foreach**-lauseella voidaan läpikäydä taulukko tai kokoelma. Lauseen käytöstä on esimerkkejä luvussa 5.

### 3.2.3 while-lause

**while**-lause on sopiva tapauksissa, joissa toisto riippuu jostain muusta kuin laskurin arvosta. Se voi riippua käyttäjän vastauksesta tai jonkin muuttujan tai useiden muuttujien arvojen kehittymisestä. Toki myös laskurin käyttö on **while**-lauseessa mahdollista.

**while**-lauseen muoto:

```
while (toistoehto)
{
    toistettavat lauseet;
}
```

Toistoehto on jokin looginen lauseke, jonka arvo tulee jossain vaiheessa päätyä arvoon false, jolloin toiston suoritus keskeytyy ja kone jatkaa whilen lopettavan aaltosulun jälkeisestä lauseesta.

**Esimerkki. Lukujen summa**

```
static void Main(string[] args)
{
   // lukujen summa, summaus jatkuu kunnes käyttäjä
   // antaa luvun 0
   int summa = 0, luku = 1; /* huom. annetaan luvulle
   sellainen alkuarvo, että toisto alkaa */
   while (luku != 0)
   {
         Console.Write("Anna luku ");
         luku = int.Parse(Console.ReadLine());
         summa = summa + luku;
    }
         Console.WriteLine("Summa on " + summa);
 }
```

**while**-lause on ns. alkuehtoinen toistolause eli toiston tarve tarkistetaan joka kerta ennenkuin toistettavat lauseet suoritetaan. Saattaa olla, että toistettavia lauseita ei suoriteta kertaakaan, jos toistoehto saa heti alussa arvokseen false. Lauseiden suorituksen jälkeen kone palaa aina testaamaan toistoehdon voimassaolon. Toisto päättyy, kun toistoehto saa arvokseen false.

```
       a)
while (luku != 0)
{      b)
     Console.Write("Anna luku ");
     luku = int.Parse(Console.ReadLine());
     summa = summa + luku;
 }
```

Rutiini kiertää järjestyksessä a), b), a), b), a), .. . , a)

## 3.2.4 do-while lause

do-while on ns. loppuehtoinen toistolause eli toistettavat lauseet suoritetaan ainakin kerran. Toistoehdon voimassaolo tarkistetaan aina toistettavien lauseiden suorituksen jälkeen, ei ennen niiden suoritusta kuten for- ja while-lauseissa.

**do-while**-lauseen rakenne:

```
do
{
   toistettavat lauseet;
} while (toistoehto);
```

**Esimerkki. Lukujen summa**

```
static void Main(string[] args)
{
       /* lukujen summa, summaus jatkuu kunnes käyttäjä antaa
luvun 0 */
       int summa = 0, luku ; /* huom. alkuarvoa luku-
       muuttujassa ei tarvita */
       do
       {
              Console.Write("Anna luku ");
              luku = int.Parse(Console.ReadLine());
              summa = summa + luku;
       } while (luku != 0);
       Console.WriteLine("Summa on " + summa);
}
```

**do-while**-lause on kätevä sellaisissa tapauksissa, että tietyt toimenpiteet tehdään ainakin kerran ja sitten, jos ehto on voimassa toimenpiteitä toistetaan tarvittava määrä.

## 3.3 Siirtorakenteet

Siirtorakenteet tarkoittavat lauseita, joilla koodin suorittamisen etenemiseen tulee jonkinlainen poikkeus normaalista.

### 3.3.1 Keskeytys break

Toistolause voidaan keskeyttää **break**-lauseella. Jos toistoja on useita sisäkkäin, keskeytetään niistä lähin.

**for-rakenteen keskeytys:**

```
for (int i = 0; i < 10; i++)
{
      if (i == 5) break;  // Toisto päättyy, kun i
                          // saavuttaa arvon 5
      Console.WriteLine(i);
}
```

**while-rakenteen keskeytys:**

```
int i = 0;
while (i < 10)
{
      if (i == 5) break;  // Toisto päättyy, kun i
                          // saavuttaa arvon 5
      Console.WriteLine(i);
      i++;
 }
```

**do-while -rakenteen keskeytys:**

```
int i = 0;
do
{
     if (i == 5) break;     // Toisto päättyy, kun i
                            //saavuttaa arvon 5
     Console.WriteLine(i);
     i++;
} while (i < 10);
```

Kaikissa edellä esitetyissä ohjelmissa toiston suoritus lopetetaan break-lauseella, vaikka varsinainen toistoehto olisi edelleen voimassa.

### 3.3.2 Metodin keskeytys return

Metodin suoritus päättyy return-lauseeseen. Jos pääohjelmassa on return-lause, ohjelman toiminta lopetetaan. Aiheesta enemmän metodien yhteydessä.

### 3.3.3 Väliinjättö continue

Toistossa voidaan jättää väliin loput toiston komennot ja siirrytään seuraavan toiston alkuun.

**for-esimerkki:**

```
for (int i = 0; i < 10; i++)
{
    if (i == 5) continue;   // Toistossa tulostetaa muut
                            //paitsi i:n arvo 5
    Console.WriteLine(i);
}
```

**while-esimerkki:**

```
int i = 0;
while (i < 10)
{
    if (i == 5)
    {
        i++;
        continue;    // Toistossa tulostetaa muut
                     //paitsi i:n arvo 5
    }
    Console.WriteLine(i);
    i++;
 }
```

**do-while -esimerkki:**

```
int i = 0;
do
{
    if (i == 5)
    {
          i++;
           continue;        //Toistossa tulostetaa muut
                            //paitsi i:n arvo 5
     }
     Console.WriteLine(i);
     i++;
 } while (i < 10);
```

### 3.3.4 Hyppylause goto

Koodin suorituksen voi siirtää annettuun osoitteeseen **goto**-lauseella. Lausetta on käytettävä harkitusti, ettei koodista tule vaikealukuista.

```
static void Main(string[] args)
{
   int i = 0;
   do
   {
      if (i == 5) goto loppu;      // Toisto päättyy, kun i
                                   //saavuttaa arvon 5
      Console.WriteLine(i);
      i++;
   } while (i < 10);

   loppu: Console.WriteLine("Loppu");
 }
```

# 4 METODIT

## 4.1 Metodin määrittely

Metodit ovat nimettyjä lohkoja, jotka voidaan suorittaa kutsumalla metodia sen nimellä mistä kohtaa ohjelmaa tahansa. Metodeihin ohjelmoidaan sellaiset toimenpiteet, jotka toistuvat usein samankaltaisina. Ohjelmointikielten kirjastoissa on valmiina tuhansia yleiskäyttöisiä metodeja, joita ohjelmoijat voivat käyttää omissa ohjelmissaan. Tämän dokumentin esimerkeissä on tähän mennessä käytetty Console-luokan metodeja WriteLine, Write ja ReadLine.

Metodi on muotoa:
**määreitä arvotyyppi nimi**(parametrit)
{
SuoritettavatLauseet;
}

Määreitä on useita erilaisia. Ensimmäisenä määritellään **näkyvyys** eli onko metodi yksityinen vai julkinen. Yksityinen tarkoittaa sitä, että muut ohjelmaluokat eivät voi kutsua sitä. Määreenä käytetään sanaa **private**. Julkinen metodi on sellainen, jota mikä tahansa ohjelma voi kutsua, jos se on saatavilla. Määresanana on **public**. Jos näkyvyys määrettä ei anneta lainkaan, on se oletusarvoisesti private. Aikaisemmin käytetyt julkiset metodit WriteLine ja ReadLine ovat ohjelmakirjastossa määritelty määreellä public.

Seuraava määre määrittelee, onko metodi luokan metodi vai olion metodi. Tässä dokumentissa käsitellään pelkästään luokan metodeja. Jos metodi on luokan metodi, julkisuusmäärettä seuraa avainsana **static**. Olion metodilla static-määre puuttuu.

**Arvotyyppi** määrittelee, minkä tyyppisen tiedon metodi palauttaa kutsujalleen. Tyypit ovat samat kuin muuttujatyypit. Jos metodi ei varsinaisesti palauta mitään tietoa, merkitään sen arvotyypiksi **void**.

Metodin nimen ohjelmoija voi itse keksiä. Ainakin julkisten metodien nimet kirjoitetaan isolla alkukirjaimella.

Parametrit ovat muuttuja-määrittelyitä. Parametri-muuttujien välityksellä metodille tuodaan lähtötietoja tehtävän suorittamista varten tai välitetään muuttujia, joihin tulokset tallennetaan.

**Esimerkki 1. Hello-tervehdys metodilla**

```
class Program
    {
        static void Main(string[] args)
        {

        }
       static void Tervehdi()
        {
            Console.WriteLine("Hello there!");
        }
    }
```

Esimerkissä on ohjelmoitu metodi nimeltään Tervehdi. Se on luokkametodi (`static`), se ei palauta mitään tietoa (`void` ) ja toiminnoksi sille on määritelty Console-luokan julkisen WriteLine-metodin kutsu. Kun ohjelma käynnistetään tällaisenaan, se ei tee mitään. Ohjelman suoritus alkaa aina Main-metodista ja siitä suoritetaan ne lauseet, jotka Main-metodissa on määritelty. Esimerkissä Main-metodissa ei ole lauseita, siksi ohjelma ei tee mitään. Main-metodiin on lisättävä Tervehdi-metodin kutsu.

**Esimerkki 2. Hello-tervehdys kutsuen metodia**

```
class Program
    {
        static void Main(string[] args)
        {
            Program.Tervehdi();
        }
        static void Tervehdi()
        {
            Console.WriteLine("Hello there!");
        }
    }
```

Nyt Main-metodissa kutsutaan Tervehdi-metodia. **Metodia kutsutaan metodin nimellä**. Nimi muodostuu metodin määrittelevän luokan nimestä ja itse metodin nimestä. Jos kutsutaan oman luokan metodia, luokan nimeä ei ole pakko mainita. Ylläolevassa esimerkissä riittäisi pelkästään kutsu: Tervehdi(). Sulkuihin kirjoitetaan välitettävät parametrit. Parametrit ovat tietoja/ohjeita tehtävän suorittamiseksi. Tervehdi-metodilla ei ole parametreja, joten kutsussa sulut ovat tyhjät. Sulut on kuitenkin kirjoitettava.

Tervehdi-metodia voi kutsua, miten monta kertaa tahansa ja mistä kohtaa tahansa Program-luokan sisällä.

**Esimerkki 3. Hello-tervehdys metodin kutsut**

```
class Program
    {
        static void Main(string[] args)
        {
            Tervehdi();
            Kuulumiset();
            Tervehdi();
        }
        static void Tervehdi()
        {
            Console.WriteLine("Hello there!");
        }
        static void Kuulumiset()
        {
            Console.WriteLine("Mitä kuuluu?");
        }
    }
```

Esimerkissä on ohjelmoitu toinen metodi Kuulumiset, jota myös kutsutaan pääohjelmasta. Metodien kirjoitusjärjestyksellä ei ole väliä. Ne sijoitetaan luokkaan rinnakkain (metodien lohkosulut eivät voi mennä ristiin). Kirjoitusjärjestyksellä ja suoritusjärjestyksellä ei ole mitään tekemistä keskenään. Suoritus määräytyy kutsujärjestyksestä. Metodit voivat kutsua myös toisiaan eli aina kutsuja ei ole Main-metodi.

**Esimerkki 4. Metodit kutsuvat toisiaan**

```
class Program
    {
        static void Main(string[] args)
        {
            Tervehdi();
        }
        static void Tervehdi()
        {
            Console.WriteLine("Hello there!");
            Kuulumiset();
        }

        static void Kuulumiset()
        {
            Console.WriteLine("Mitä kuuluu?");
        }
    }
```

Ohjelmassa Main-metodi kutsuu Tervehdi-metodia, joka puolestaan kutsuu Kuulumiset-metodia.

## 4.2 Parametrit

Lähes kaikissa metodeissa tarvitaan parametreja. Parametreilla ohjataan metodin suoritusta toimimaan juuri siinä tilanteessa halutulla tavalla. Aikaisemmin käytettyihin WriteLine-metodien kutsuihin on liitetty aina parametreja eli ne merkkijonot, jotka on haluttu metodien niissä tilanteissa kirjoittavan konsolille. Sama WriteLine-metodi osaa kirjoittaa konsolille siis millaisen merkkijonon tahansa, jonka se saa parametrinaan. Tällainen ominaisuus tekee metodista hyvin kätevän. Tarvitsee ohjelmoida metodi vain kertaalleen ja sen toimintaa ohjataan parametrein.

C#-kielessä on kolmenlaisia parametreja metodin toiminnan säätelemiseksi: arvoparametrit, viittausparametrit ja ulostuloparametrit.

### 4.2.1 Arvoparametrit

Arvoparametri tarkoittaa sitä, että metodille välitetään jokin arvo (luku, merkkijono, olio), jota metodi tarvitsee tehtävänsä suorittamiseen. Parametreja voi olla useita. Arvoparametri voidaan välittää joko muuttujan sisältönä tai vakioarvona.

**Esimerkki 1. Vakion käyttö arvoparametrina**

```
static void Main(string[] args)
{
     Tervehdi("Maija");
}
static void Tervehdi(string kuka)
{
     Console.WriteLine("Hello there {0}!", kuka);
}
```

Esimerkissä on Tervehdi-metodin määrittelyä muutettu siten, että parametreihin on merkitty yksi string-tyyppinen parametri, jolle on annettu nimi kuka. Määrittely aiheuttaa sen, että aina kun Tervehdi-metodia kutsutaan, on parametrisulkujen sisään kirjoitettava merkkijonovakio tai merkkijonotyyppisen muuttujan nimi, joka sisältää merkkijonon. Esimerkissä kutsu on suoritettu käyttäen merkkijonovakiota "Maija".

Tervehdi-metodin määrittelyssä varataan tila arvoparametria kuka varten. Kun metodia kutsutaan, kutsun mukana tuleva parametrin arvo sijoitetaan metodin tilanvaraukseen. kuka-muuttujan arvo on nyt Tervehdi-metodin käytettävissä.

Ei ole kovin fiksua kirjoittaa ohjelmaa, joka osaa tervehtiä pelkästään Maija-nimisiä käyttäjiä. Ohjelmaa kannattaakin korjata siten, että se osaa tervehtiä ketä tahansa käyttäjää.

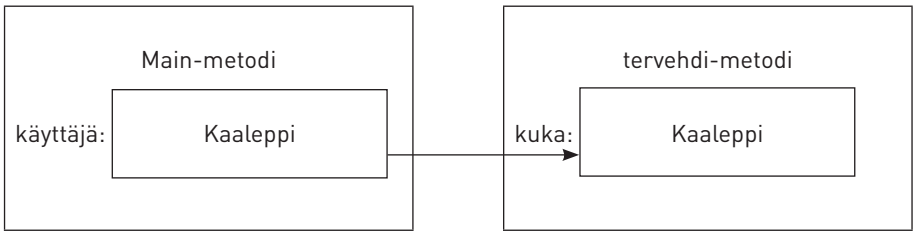
**Esimerkki 2. Muuttujan käyttö arvoparametrina**

```
class Program
    {
        static void Main(string[] args)
        {
            string käyttäjä;
            Console.Write("Mikä on nimesi? ");
            käyttäjä = Console.ReadLine();
            Tervehdi(käyttäjä);
        }
        static void Tervehdi(string kuka)
        {
            Console.WriteLine("Hello there "+kuka+"!");

        }
    }
```

Nyt ohjelmassa ensin kysytään käyttäjän nimi Main-metodin muuttujan käyttäjä sisällöksi. Tämän muuttujan sisältö sitten välitetään Tervehdi-metodille, jossa se sijoitetaan muuttujan kuka arvoksi. Tapahtuu tavallaan muuttujan arvon kopiointi metodin nimiavaruudesta toisen metodin nimiavaruuteen. Kopioitava muuttuja ja se muuttuja, johon kopioidaan voivat olla saman nimisiä tai erinimisiä.

Jokainen metodi muodostaa oman nimiavaruuden ja niiden toimintaa voidaan kuvata seuraavasti kuviossa 7 esitetyllä tavalla.




KUVIO 7. Arvoparametrin välitys kutsuvalta metodilta kutsutulle metodille

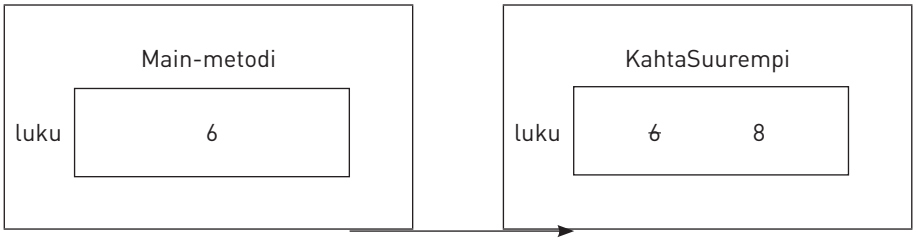
**Esimerkki 3. Arvoparametrina käytetyn muuttujan arvo ei muutu**

```
class Program
    {
        static void Main(string[] args)
        {
              int luku = 6;
              KahtaSuurempi(luku);
              KahtaSuurempi(10);
              Console.WriteLine(luku); // tulostuu 6

         }
        static void KahtaSuurempi(int luku)
        {
              luku += 2;
              Console.WriteLine(luku); // tulostuu 8,
                // toisella kutsulla tulostuu 12
        }
    }
```

Jos arvoparametrina välitettyä muuttujan arvoa muutetaan kutsutussa metodissa, muutokset eivät vaikuta alkuperäiseen muuttujaan. Arvoparametrilla välitetään vain muuttujan sen hetkinen arvo metodin käyttöön. Sillä ei ole merkitystä, vaikka kutsutussa metodissa muuttujasta käytettäisiin samaa nimeä, ne ovat silti kaksi eri asiaa ja kaksi eri tilanvarausta.

Siinä metodissa, jossa kutsu sijaitsee ei kutsussa esiintyvän muuttujan arvo muutu, vaan muuttujan arvon kopio välitetään kutsuttuun metodiin (kuvio 8).




KUVIO 8. Arvoparametrina välitetyn muuttujan arvo ei muutu kutsuvassa metodissa, vaikka arvoa muutetaan kutsutussa metodissa.

Esimerkissä Main-metodissa on määritelty muuttuja luku ja sijoitettu sinne arvo 6. Seuraavaksi kutsutaan metodia KahtaSuurempi, jonka parametriksi tarvitaan kokonaisluku. Tarvittavaksi kokonaisluvuksi esimerkissä sijoitetaan Main-metodin muuttujan luku **arvo**. Muuttujan arvo on 6 ja se siirtyy kutsun mukana metodin KahtaSuurempi muistialueelle. KahtaSuurempi-metodissa on määritelty, että kutsun mukana tuleva kokonaisluku sijoitetaan siellä luku-nimisen parametrin (muuttujan)

arvoksi. Metodissa luku-parametrin arvoa kasvatetaan kahdella. Kasvatus tapahtuu KahtaSuurempi-metodin puolella eikä sillä ole vaikutusta Main-metodin luku-muuttujan arvoon, vain KahtaSuurempi-metodin luku-muuttujan arvo vaihtuu.

Parametreja voi olla useita samassa metodissa.

**Esimerkki 4. Monta parametria**

```
class Program
    {
        static void Main(string[] args)
        {
            int luku1 = 7;
            double luku2 = 4.5;
            Kerro(luku1, luku2);
        }
        private static void Kerro(int kertoja,
                                  double kerrottava)
        {
            Console.WriteLine(kertoja * kerrottava);
        }
    }
```

Kerro-metodissa on kaksi parametria: ensimmäinen on integer-parametri ja toinen on double-parametri. Kerro-metodia voi kutsua vain siten, että parametrisulkujen sisällä ensin mainitaan integer-arvo ja sitten double-arvo. Kutsussa parametrien arvot on siis oltava samassa järjestyksessä kuin metodin määrittelyssä tehdyt parametrien tilanvaraukset.

Kutsun mukana lähtevä luku1:sen arvo sijoittuu Kerro-metodin puolella kertojan arvoksi ja luku2:sen arvo kerrottavan arvoksi.

## Arvonpalautus

Metodi voidaan ohjelmoida palauttamaan yhden **arvon**. Jos metodi ei palauta mitään arvoa, esiintyy määrittelysana void. Jos metodi palauttaa merkkijonon, palautusarvoksi kirjoitetaan **string**, jos palautetaan kokonaisluku palatutusarvoksi merkitään **int** jne.

**Esimerkki 5. Arvon palauttava metodi**

```
class Program
    {
        static void Main(string[] args)
        {
            int luku1 = 7;
            double luku2 = 4.5;
            double tulo = Kerro(luku1, luku2);
            Console.WriteLine("Tulo on {0}", tulo);
        }
        static double Kerro(int kertoja, double kerrottava)
        {
              double tulo = kertoja * kerrottava;
              return tulo;
        }
     }
```

Esimerkissä metodi Kerro on määritelty palauttamaan double-arvon. Koodissa arvo palautetaan return-lauseella. Jos metodi on määritelty arvon palauttavaksi, siihen on kirjoitettava return-lause/lauseet palauttamaan luvatun tyyppinen arvo.

Kutsuvassa metodissa palautusarvo siepataan yksinkertaisimmillaan vastaavan tyyppisen muuttujan arvoksi tekemällä muuttujaan sijoitusoperaatio metodin kutsulle. Toinen vaihtoehto on kirjoittaa metodin kutsu johonkin lausekkeeseen siihen kohtaan, jossa palautusarvoa käytetään.

Esimerkiksi lauseiden

```
double tulo = Kerro(luku1, luku2);
Console.WriteLine("Tulo on {0}" , tulo);
```

sijaan voisi olla

```
Console.WriteLine("Tulo on {0}", Kerro(luku1, luku2));
```

Metodiin voidaan tähän tapaan ohjelmoida vain yhden arvon palautuksen.

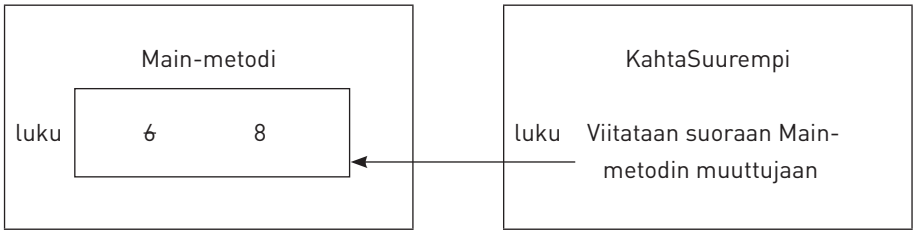
## Viittausparametrit

Viittausparametri tarkoittaa sitä, että metodi käsittelee **suoraan** sitä muuttujaa, jonka kutsuja antaa metodille parametriksi. Parametrin arvoa ei siis kopioida, vaan viitataan suoraan alkuperäiseen muuttujaan. Viittausparametri ilmaistaan **ref** avainsanalla, joka pitää mainita sekä kutsussa, että kutsuttavan metodin määrittelyssä. Muuttujalla

on **oltava arvo jo kutsuttaessa** ja muuttujan on oltava sellainen, jolle voidaan asettaa arvo. Siten **const** avainsanalla määritelty vakiomuuttuja ei kelpaa viittausparametriksi.

**Esimerkki 6. Viittausparametrin käyttö**

```
class Program
    {
        static void Main(string[] args)
        {
            int luku = 6;   // alkuarvon asetus
                            // viiteparametri-muuttujalle
            KahtaSuurempi(ref luku);      // avainsana ref
            Console.WriteLine(luku);      // tulostuu 8
        }
       static void KahtaSuurempi(ref int luku)
                            // viittausparametri
         {
            luku += 2;
            Console.WriteLine(luku); // tulostuu 8
        }
   }
```



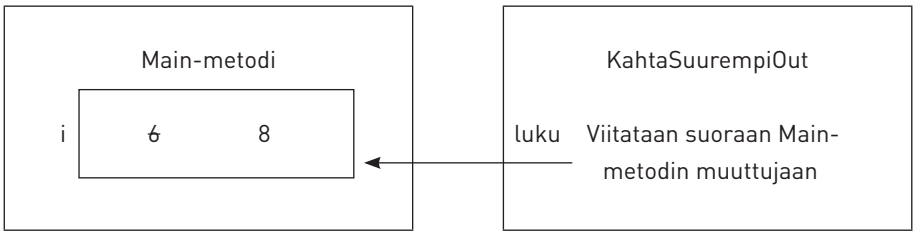

KUVIO 9. Viittausparametrin arvo muuttuu kutsuvassa metodissa, jos sitä muutetaan kutsutussa metodissa.

## 4.2.2 Ulostuloparametrit

Ulostuloparametri toimii kuten viittausparametri, mutta sillä **ei tarvitse olla alkuarvoa ennen kutsua**. **Kutsutussa metodissa ulostuloparametrille on asetettava alkuarvo**. Parametria ei kopioida, vaan käsitellään suoraan alkuperäistä muuttujaa. Ulostuloparametri merkitään **out**-avainsanalla ja avainsana on mainittava sekä kutsussa että metodin parametrien määrittelyssä.

**Esimerkki 7. Ulostuloparametrin käyttö**

```
static void Main(string[] args)
{
    int i;
    KahtaSuurempiOut(out i);      // paramerilla ei ole
                                  // alkuarvoa
            Console.WriteLine(i); // tulostaa 8
}
static void KahtaSuurempiOut(out int luku)
// i:n alias-nimenä tällä puolella on luku
{
    luku = 6; // alkuarvon asettaminen
    luku += 2;
    Console.WriteLine(luku); // tulostaa 8
}
```




KUVIO 10. Ulostuloparametrissa käsitellään kutsujan muuttujaa

### 4.2.3 Vaihtuvamittainen parametrilista

Avainsanalla **params** voidaan määritellä vaihtuva määrä parametreja.

**Esimerkki. Vaihtuva määrä parametreja**

```
static void Main(string[] args)
{
       VaihtuvaMaaraParametreja(5, 6, 7,8);
       //parametrien lukumäärä on vapaasti valittavissa
       VaihtuvaMaaraParametreja(1);
}
private static void VaihtuvaMaaraParametreja(params int[]
taulukko)
{
    for (int i = 0; i < taulukko.Length; i++)
       Console.WriteLine(taulukko[i]);
}
```

# 5 TAULUKOT

## 5.1 Miksi taulukoita tarvitaan

Hyvin usein on tilanne, ettei etukäteen voi päätellä, kuinka monta muuttujaa tullaan tarvitsemaan tietojen käsittelemistä varten. Tai muuttujia tarvittaisiin niin monta, että olisi kovin vaivaloista määritellä ja nimetä ne kaikki.

Esimerkiksi kurssille voi ilmoittautua maksimissaan 50 henkilöä. Näistä tarvittaisiin lista eli kaikkien nimet olisi pidettävä tallessa. Pitäisi siis määritellä 50 string-tyyppistä muuttujaa ja kysyä niihin sisältö. Entä, jos kurssille tulisikin vain 10 henkilöä – ohjelma kyselisi siitä huolimatta kaikki 50 nimeä.

```
static void Main(string[] args)
        {
            string nimi1,nimi2,nimi3,...., nimi50;
            // nimien kyseleminen
            Console.Write("Anna nimi: ");
            nimi1 = Console.ReadLine();
            Console.Write("Anna nimi: ");
            nimi2 = Console.ReadLine();
            Console.Write("Anna nimi: ");
            nimi3 = Console.ReadLine();
                :
            Console.Write("Anna nimi: ");
            nimi50 = Console.ReadLine();
            // tässä välissä nimiä käsitellään jotenkin
            // nimien tulostus
            Console.WriteLine(nimi1);
            Console.WriteLine(nimi2);
            Console.WriteLine(nimi3);
                :
            Console.WriteLine(nimi50);
        }
```

Edellä olevaan ohjelmaan saisi toki if-lauseilla parannusta (jos käyttäjä antaa tyhjän nimen, lopetetaan kysely goto-lauseella), mutta edelleen yhtä työlästä ohjelman kirjoittaminen silti olisi. Ja entä, jos käsiteltäviä tietoja olisikin 1000 tai 10000. Näin tätä asiaa ei voi ratkaista.

Ongelmaan on keksitty ratkaisuksi taulukko, joka on kokoelma saman tyyppisiä muuttujia. Ei tarvitse keksiä kuin yhden nimen ja erilliset muuttujat on indeksoitu (numeroitu). Indeksointi alkaa aina nollasta ja taulukon viimeinen indeksi on taulukon koko-1.

**Esimerkki 1. Taulukon määrittely ja solujen käyttö**

Määritellään taulukko nimeltään luvut, johon voi tallettaa kokonaislukuja. Taulukon koko on kymmenen solua eli se sisältää kymmenen kappaletta kokonaislukumuuttujia.

```
int[] luvut = new int[10];
```

| 45 | 25 | -8 | 22 | 0 | 8 | 70 | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Taulukon soluihin (muuttujiin) viitataan hyväksikäyttäen indeksinumeroa.

```
luvut[0] = 45;
luvut[1] = 25;
luvut[6] = luvut[0] + luvut[1];
```

Nyt taulukon muuttujia voi käsitellä samalla nimellä indeksointia hyväksikäyttäen.

Nyt edellä esitetty nimien kysely ja uloskirjoitus on helppo ohjelmoida joustavaksi.

**Esimerkki 2. Taulukon käyttö nimien keruussa**

```
class Program
    {
        static void Main(string[] args)
        {
            string[] nimet = new string[50];

            for (int i = 0; i < 50; i++)
            {
                Console.Write("Anna nimi: ");
                nimet[i] = Console.ReadLine();
                if (nimet[i] == "") break;
            }
            for (int i = 0; i < 50; i++)
            {
                if (nimet[i] == "") break;
                Console.WriteLine(nimet[i]);
            }
        }
    }
```

Taulukkoa käydään läpi toistolausetta käyttäen solu kerrallaan. Ohjelmoijan on huolehdittava, ettei ohjelma viittaa sellaiseen soluun, jota ei ole olemassa. Sellaisessa tapauksessa ohjelman suoritus keskeytyy virheeseen.

## 5.2 Taulukon käsittelyyn soveltuvia metodeja

Kun taulukosta on etsittävä tiettyä sisältöä, käytetään Contains-metodia. Jos edellä määritellystä taulukosta halutaan etsiä nimeä Aaro, kirjoitetaan komento:

```
if (nimet.Contains(”Aaro”))…
```

Taulukon koko saadaan selville integer-tyyppisestä ominaisuudesta **Length**

```
int koko = nimet.Length;
```

Läpikäyntiin tarvittava for-lause voidaan kirjoittaa Length-ominaisuuteen perustuen, jolloin vältetään mahdollinen taulukon ulkopuolelle viittaaminen:

```
for (int i = 0; i < nimet.Length; i++)
```

Tähän liittyen taulukon viimeinen indeksinumero on arvoltaan Length-1.

Taulukon läpikäyntiin on olemassa hyvä toistorakenne **foreach**. Rakenne on tavallista for-lausetta parempi siinä mielessä, että sitä käyttäen ei koskaan pysty viittaamaan taulukon ulkopuolelle, vaan rakenne itse valvoo rajojen sisällä pysymisen.

Rakenne on muotoa:

**foreach** ( kokoelmanTyyppi muuttujanNimi **in** kokoelma)
{
    vuorossa olevan muuttujan käsittely muuttujanNimi perusteella;
}

**foreach**-lausekkeen sulkujen sisällä määritellään muuttuja, joka on samaa tyyppiä kuin taulukon sisältö ja nimetään se. **in**-avainsanan jälkeen kerrotaan, mitä kokoelmaa läpikäydään. Rakenne poimii nimetystä kokoelmasta muuttujan kerrallaan käsittelyyn, jonka ohjelmoija ohjelmoi aaltosulkujen sisään.

**Esimerkki 1. Taulukon läpikäynti foreach-rakenteella**

```
int lkm = 0;
string[] nimet = { ”Anne”, ”Mika”, ”Beata”, ”Aaro”,”Mika” };
foreach (string nimi in nimet)
{
    if (nimi == “Aaro”)
    {
         lkm++;
    }
}
Console.WriteLine(“Löytyi {0} Aaroa”, lkm);
```

Taulukon sisällön voi antaa jo määrittelylauseessa. Sisältö kirjoitetaan aaltosulkujen sisään. Jos sisältö on merkkijonoja, ne merkkijonovakioiden merkintätavan mukaan annetaan lainausmerkeissä, merkit puolestaan heittomerkkien sisällä.

```
int[] luvut = { 5, 78, 89, 2, -3, 7 };
double[] desimaaliluvut = { 4.6, 2.33, 7.89, 1.9 };
string[] nimet = { “Matti”, “Maija”, “Ulla”, “Tero” };
char[] merkit = {‘a’,’b’,’5’,’&’,’=’};
```

Esimerkissä luvut-taulukosta muodostuu 6-soluinen taulukko, joka saa määritellyt alkuarvot. Desimaaliluvut-taulukko on nelisoluinen samoin kuin nimet-taulukko ja merkit-taulukko on viisisoluinen.

Taulukot voivat olla myös kaksi- tai useampiulotteisia. Ulotteisuus ilmaistaan riittävällä määrällä pilkkuja määrittelylauseessa.

Määritellään kaksiulotteinen kokonaislukutaulukko, jossa on neljä riviä ja neljä saraketta.

```
int[,] matriisi = new int[4,4];
matriisi[0, 0] = 14;
matriisi[0,1] = 6;
```

**Esimerkki 2. Myös moniulotteinen taulukko voidaan läpikäydä foreach-rakenteella.**

```
string[,] nimet2 = new string[3, 2];
nimet2[0, 0] = “Maija”;
nimet2[0, 1] = “Meikäläinen”;
nimet2[1, 0] = “Matti”;
nimet2[1, 1] = “Meikäläinen”;
foreach (string nimi in nimet2)
{
   Console.WriteLine(nimi);
}
```

# 6 MERKIT JA MERKKIJONOT

## 6.1 Merkkien esittäminen

Merkki tarkoittaa sekä näkyvää merkkiä kuten kirjainta, numeroa tai erikoismerkkiä (esim. piste, pilkku, dollari..) että kirjoitusasua vailla olevaa merkkiä kuten välilyönti ja rivinvaihtomerkki.

Merkkien koodaamiseen bittien avulla on olemassa useita erilaisia merkistöjä. Vanha paljon käytetty merkistö on ASCII-koodi, jossa merkki muodostetaan kahdeksan bitin avulla. ASCII-koodattu merkki mahtuu siten yhteen tavuun. ASCII-koodin puutteena on, että sillä pystytään esittämään vain 256 ($2^8$) erilaista merkkiä. Näin ollen kansalliset erikoismerkit koodataan milloin mitenkin ja esimerkiksi skandinaavisten kirjainten kanssa on ongelmia siirryttäessä merkistöstä toiseen.

**Unicode**-merkistössä kullekin merkille varataan 16 bittiä, jolloin merkistöllä voidaan esittää 65536 erilaista merkkiä. Unicode-koodattu merkki vie siten tilaa kaksi tavua koneen muistia. Unicode-merkistöä käytettäessä myös kansallisille merkeille on varattu oma pysyvä koodi eikä esimerkiksi skandinaaviset kirjaimet aiheuta enää ongelmia.

Merkkien aakkosjärjestys määrittyy merkin koodin bittikombinaation perusteella. Char on C#-kielessä varattu muuttujatyypiksi merkkitiedolle. Muuttujaan mahtuu 16 bittiä eli yhdessä char-tyyppisessä muuttujassa on tasan yksi merkki. Merkkien aakkosjärjestystä, erisuuruutta ja samuutta toistensa suhteen voi testata normaalein luvuista tutuin operaation (<, >, !=, =). Itseasiassa char-tyyppinen muuttuja on kokonaisluku ja sen sisällön voi suoraan sijoittaa 16-bittiseen tai sitä suurempaan kokonaislukutyyppiseen muuttujaan.

```
int koodi = merkki;
```

Merkkivakio on ympäröitävä heittomerkein:

```
char merkki = ’t’;
merkki = ’&’;

char merkki;
Console.Write(“Anna merkki! “);
merkki = char.Parse(Console.ReadLine());
```

Char-tyyppiselle tiedolle on olemassa valmiina useita sen käsittelyyn sopivia metodeja.

Is-alkuisilla metodeilla voidaan tutkia täyttääkö merkki, jonkin ehdon:

```
if (char.IsDigit(merkki))  // onko reaaliluku
if (char. IsNumber (merkki))  // onko numero
if (char.IsLetter(merkki)) // onko kirjain
if (char.IsLetterOrDigit(merkki))  // onko numero tai kirjain
```

To-alkuisilla metodeilla merkkiä voidaan muokata:

```
// merkki muutetaan pieneksi kirjaimeksi
char uusiMerkki = char.ToLower(merkki); char uusiMerkki =
// merkki muutetaan isoksi kirjaimeksi
char.ToUpper(merkki);
// merkki muutetaan merkkijonoksi
string merkkijono = merkki.ToString();
```

## 6.2 Merkkijonot

Merkkijono muodostuu nollasta tai useammasta merkistä. Merkkijonon pituudella tarkoitetaan merkkijonossa olevien merkkien lukumäärää. Nollan mittaista merkkijonoa voidaan kutsua myös tyhjäksi merkkijonoksi. Yhden merkin mittainen merkkijono on eriasia kuin yksi merkki. Merkkijono-muuttujan tyyppi on string ja merkin tietotyyppi on char. Merkkijonon pituus saadaan selville ominaisuudella Length.

```
int pituus = merkkijono.Length;
```

Merkkijonovakio ilmaistaan lainausmerkein ympäröitynä.

```
string merkkijono = ”Tämä on merkkijonovakio”;
```

Merkkijonon merkit voidaan ajatella numeroiduksi merkkijonon alusta alkaen siten, että ensimmäisen merkin indeksi (numero) on nolla toisen merkin indeksi on ykkönen jne. Aivan kuten taulukossakin merkkijonon viimeisen merkin indeksi on merkkijonon pituus miinus yksi. Merkkijonosta voidaan poimia erikseen merkki indeksillä viittaamalla.

```
// poimitaan merkkijonon toinen merkki
char yksittainenMerkki = merkkijono[1];
```

Sen sijaan merkin vaihto ei onnistu indeksoimalla

```
merkkijono[2] = 'a';
```

Kahden merkkijonon samankaltaisuutta tai erilaisuutta voidaan testata operaatoreiden == ja != avulla.

```
string merkkijono = "Tämä on merkkijonovakio";
string jono2 = "abc";
if (merkkijono == jono2)... tai if (merkkijono != jono2)...
```

Kahden merkkijonon aakkosjärjestyksen voi testata CompareTo-metodilla seuraavasti.

```
if (jono2.CompareTo(merkkijono) < 0)
   Console.WriteLine("jono2 on aakkosissa ennen merkkijonoa");
else if (jono2.CompareTo(merkkijono) > 0)
   Console.WriteLine("jono2 on aakkosissa merkkijonon jälkeen");
else if (jono2.CompareTo(merkkijono) == 0)
   Console.WriteLine("jono2 ja merkkijono ovat saman sisältöisiä");
```

Isoiksi ja pieniksi kirjaimiksi merkkijono voidaan muokata ToUpper- ja ToLower-metodien avulla.

```
jono2.ToLower();
jono2.ToUpper();
```

IndexOf-metodilla saa selville, missä indeksissä jokin merkki on tai mistä indeksistä jokin merkkijono alkaa, myös etsinnän aloituskohdan voi määritellä. Jos merkkiä/merkkijonoa ei löydy, metodi palauttaa negatiivisen luvun.

```
int indeksi = merkkijono.IndexOf('a');
indeksi =  merkkijono.IndexOf("abc");
indeksi = merkkijono.IndexOf('b', 3);
```

Osan merkkijonosta voi irrottaa metodilla Substring. Metodin parametriksi määritellään mistä indeksistä lähdetään ja miten monta merkkiä poimitaan. Jos poimittavien merkkien määrää ei kerrota, otetaan kaikki merkit jonon loppuun saakka.

```
string pieniPala = merkkijono.Substring(0, 5);
```

Merkkijonoja voi liittää toisiinsa katenoimalla. Katenointimerkkinä toimii plus-merkki.

```
string merkkijonoja = ”ff” + ”Kiva” + ”99” + ” Jihuu”;
string lisää = ”!”;
string uusiMerkkijono = merkkijonoja+lisää;
```

Merkkijono on itseasiassa viittaustyyppi ja yksittäinen merkkijono on siten merkkijono-olion ilmentymä. C#-kieleen on kuitenkin tehty välineet, joilla perusmerkkijonojen käsittely on saatu näyttämään arvotyyppien käsittelyltä.

# 7 YHTEENVETO

Dokumentissa on käsitelty kaikkein keskeisimmät ohjelmointikielille yhteiset piirteet. Aluksi opiskeltiin tiedon prosessointi muuttujien avulla. Tuotiin esiin, miten tiedolle varataan muistitila (= muuttujan tyypitys), miten muuttuja nimetään ja miten muuttujia voi käyttää laskentaan sekä koneen ja käyttäjän välisen keskusteluun. Ohjelmassa tehtävien valintojen suorittamiseksi opittiin käyttämään ehto- ja valintalauseita: if ja swich-case.

Tietokoneohjelmilla tehdään paljon toistoa vaativia toimenpiteitä. Toistorakenteina käytettiin for-, while- ja do-while –lauseita. Taulukoiden yhteydessä vielä otettiin esiin esimerkin avulla foreach-toistolause. Kaikki toistorakenteet on mahdollista keskeyttää. Keskeytyslauseista käsiteltiin break-, continue-, return- ja goto-lauseet.

Metodit ovat erittäin keskeinen ohjelmarakenteen suunnitteluväline. Ohjelmat saadaan pilkottua pienempiin, hallittavampiin osiin metodien avulla. Metodien koodisisältö voi olla hyvinkin pieni, yksi tai kaksi lausetta. Pienien metodien käyttö on perusteltua esimerkiksi tehtävän toistuvuuden vuoksi. Kun tehtävä hoidetaan metodilla, se suoritetaan aina täsmälleen samalla tavalla, joten se myös näyttäytyy sekä ohjelmoijalle että ohjelman käyttäjälle samanlaisena. Metodeista opittiin käyttämään sekä arvon palauttavia että arvoa palauttamattomia metodeja. Edelleen opittiin käyttämään parametreja tietojen välittämiseen metodilta toiselle.

Lopussa tehtiin vielä lyhyt katsaus taulukoiden ja merkkien/merkkijonojen käsittelyyn. Taulukoilla saa käsiteltyä samankaltaisten tietojen joukkoa ja merkkijonoja voidaan käyttää tekstipohjaisen tiedon käsittelyyn.

Tämän julkaisun esimerkeissä pitäydyttiin merkkipohjaisen käyttöliittymän (=konsoli) käyttämiseen. Graafisen käyttöliittymän tekeminen ohjelmaan tuo paljon lisäopiskeltavaa. Konsolikäyttöliittymää käyttämällä pystyttiin keskittymään ohjelmoinnin ytimen esiin tuomiseen. Jatko-opintoina tässä dokumentissa esitettyihin ohjemointirakenteisiin voisikin suositella graafisen käyttöliittymän ohjelmointiin liittyviä asioita. Toinen etenemissuunta voisi olla oliot ja niiden hyödyntäminen ohjemoinnissa. Olioiden käsittely on kaikille olio-ohjelmointikielille periaatteiltaan samanlaista aivan kuten tässä julkaisussa esitetyt ohjelmoinnin perusrakenteetkin.

# LÄHTEET

Microsoft 2015a. Built-In Types Table (C# Reference). [Verkkosivu]. Microsoft Co. [Viitattu 30.1.2015]. Saatavana: http://msdn.microsoft.com/en-us/library/ya5y69ds.aspx

Microsoft 2015b. C# programming guide. [Verkkosivu]. Microsoft Co. [Viitattu 30.1.2015]. Saatavana: https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx

Microsoft 2015c. VisualStudio downloads. [Verkkosivu]. Microsoft Co. [Viitattu 30.1.2015]. Saatavana: http://www.visualstudio.com/en-us/downloads

# SEINÄJOEN AMMATTIKORKEAKOULUN JULKAISUSARJA - PUBLICATIONS OF SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

## A. TUTKIMUKSIA - RESEARCH REPORTS

1. Timo Toikko. Sosiaalityön amerikkalainen oppi. Yhdysvaltalaisen caseworkin kehitys ja sen yhteys suomalaiseen tapauskohtaiseen sosiaalityöhön. 2001.

2. Jouni Björkman. Risk Assessment Methods in System Approach to Fire Safety. 2005.

3. Minna Kivipelto. Sosiaalityön kriittinen arviointi. Sosiaalityön kriittisen arvioinnin perustelut, teoriat ja menetelmät. 2006.

4. Jouni Niskanen. Community Governance. 2006.

5. Elina Varamäki, Matleena Saarakkala & Erno Tornikoski. Kasvuyrittäjyyden olemus ja pk-yritysten kasvustrategiat Etelä-Pohjanmaalla. 2007.

6. Kari Jokiranta. Konkretisoituva uhka. Ilkka-lehden huumekirjoitukset vuosina 1970–2002. 2008.

7. Kaija Loppela. "Ryhmässä oppiminen - tehokasta ja hauskaa": Arviointitutkimus PBL-pedagogiikan käyttöönotosta fysioterapeuttikoulutuksessa Seinäjoen ammattikorkeakoulussa vuosina 2005-2008. 2009.

8. Matti Ryhänen & Kimmo Nissinen (toim.). Kilpailukykyä maidontuotantoon: toimintaympäristön tarkastelu ja ennakointi. 2011.

9. Elina Varamäki, Juha Tall, Kirsti Sorama, Aapo Länsiluoto, Anmari Viljamaa, Erkki K. Laitinen, Marko Järvenpää & Erkki Petäjä. Liiketoiminnan kehittyminen omistajanvaihdoksen jälkeen – Casetutkimus omistajanvaihdoksen muutostekijöistä. 2012.

10. Merja Finne, Kaija Nissinen, Sirpa Nygård, Anu Hopia, Hanna-Leena Hietaranta-Luoma, Harri Luomala, Hannu Karhu & Annu Peltoniemi.

Eteläpohjalaisten elintavat ja terveyskäyttäytyminen : TERVAS – terveelliset valinnat ja räätälöidyt syömisen ja liikkumisen mallit 2009 – 2011. 2012.

11. Elina Varamäki, Kirsti Sorama, Anmari Viljamaa, Tarja Heikkilä & Kari Salo. Eteläpohjalaisten sivutoimiyrittäjien kasvutavoitteet sekä kasvun mahdollisuudet. 2012.

12. Janne Jokelainen. Hirsiseinän tilkemateriaalien ominaisuudet. 2012.

13. Elina Varamäki & Seliina Päällysaho (toim.) Tapio Varmola – suomalaisen ammattikorkeakoulun rakentaja ja kehittäjä. 2013.

14. Tuomas Hakonen. Bioenergiaterminaalin hankintaketjujen kanttavuus eri kuljetusetäisyyksillä ja -volyymeilla. 2013.

15. Minna Zechner (toim.). Hyvinvointitieto: kokemuksellista, hallinnollista ja päätöksentekoa tukevaa? 2014.

16. Sanna Joensuu, Elina Varamäki, Anmari Viljamaa, Tarja Heikkilä & Marja Katajavirta. Yrittäjyysaikomukset, yrittäjyysaikomusten muutos ja näihin vaikuttavat tekijät koulutuksen aikana. 2014.

17. Anmari Viljamaa, Seliina Päällysaho & Risto Lauhanen (toim.). Opetuksen ja tutkimuksen näkökulmia: Seinäjoen ammattikorkeakoulu. 2014.

18. Janne Jokelainen. Vanhan puuikkunan energiakunnostus. 2014.

19. Matti Ryhänen & Erkki Laitila (toim.). Yhteistyö- ja verkostosuhteet: Strateginen tarkastelu maidontuotantoon sovellettuna. 2014.

20. Kirsti Sorama, Elina Varamäki, Sanna Joensuu, Anmari Viljamaa, Erkki K. Laitinen, Erkki Petäjä, Aapo Länsiluoto, Tarja Heikkilä & Tero Vuorinen. Mistä tunnet sä kasvajan - seurantatutkimus eteläpohjalaisista kasvuyrityksistä. 2015.

# B. RAPORTTEJA JA SELVITYKSIÄ - REPORTS

1. Seinäjoen ammattikorkeakoulusta soveltavan osaamisen korkeakoulu-tutkimus- ja kehitystoiminnan ohjelma. 1998.

2. Elina Varamäki - Ritva Lintilä - Taru Hautala - Eija Taipalus. Pk-yritysten ja ammattikorkeakoulun yhteinen tulevaisuus: prosessin kuvaus, tuotokset ja toimintaehdotukset. 1998.

3. Elina Varamäki - Tarja Heikkilä - Eija Taipalus. Ammattikorkeakoulusta työelämään: Seinäjoen ammattikorkeakoulusta 1996-1997 valmistuneiden sijoittuminen. 1999.

4. Petri Kahila. Tietoteollisen koulutuksen tilanne- ja tarveselvitys Seinäjoen ammattikorkeakoulussa: väliraportti. 1999.

5. Elina Varamäki. Pk-yritysten tuleva elinkaari - säilyykö Etelä-Pohjanmaa yrittäjämaakuntana? 1999.

6. Seinäjoen ammattikorkeakoulun laatujärjestelmän auditointi 1998–1999. Itsearviointiraportti ja keskeiset tulokset. 2000.

7. Heikki Ylihärsilä. Puurakentaminen rakennusinsinöörien koulutuksessa. 2000.

8. Juha Ruuska. Kulttuuri- ja sisältötuotannon koulutusselvitys. 2000.

9. Seinäjoen ammattikorkeakoulusta soveltavan osaamisen korkeakoulu. Tutkimus- ja kehitystoiminnan ohjelma 2001. 2001.

10. Minna Kivipelto (toim.). Sosionomin asiantuntijuus. Esimerkkejä kriminaalihuolto-, vankila- ja projektityöstä. 2001.

11. Elina Varamäki - Tarja Heikkilä - Eija Taipalus. Ammattikorkeakoulusta työelämään. Seinäjoen ammattikorkeakoulusta 1998–2000 valmistuneiden sijoittuminen. 2002.

12. Varmola T., Kitinoja H. & Peltola A. (ed.) Quality and new challenges of higher education. International Conference 25.-26. September, 2002. Seinäjoki Finland. Proceedings. 2002.

13. Susanna Tauriainen & Arja Ala-Kauppila. Kivennäisaineet kasvavien nautojen ruokinnassa. 2003.

14. Päivi Laitinen & Sanna Välisaari. Staphylococcus aureus -bakteerien aiheuttaman utaretulehduksen ennaltaehkäisy ja hoito lypsykarjatiloilla. 2003.

15. Riikka Ahmaniemi & Marjut Setälä. Seinäjoen ammattikorkeakoulu – Alueellinen kehittäjä, toimija ja näkijä. 2003.

16. Hannu Saari & Mika Oijennus. Toiminnanohjaus kehityskohteena pkyrityksessä. 2004.

17. Leena Niemi. Sosiaalisen tarkastelua. 2004.

18. Marko Järvenpää (toim.) Muutoksen kärjessä. Kalevi Karjanlahti 60 vuotta. 2004.

19. Suvi Torkki (toim.). Kohti käyttäjäkeskeistä muotoilua. Muotoilijakoulutuksen painotuksia SeAMK:ssa. 2005.

20. TimoToikko (toim.). Sosiaalialan kehittämistyön lähtökohta. 2005.

21. Elina Varamäki & Tarja Heikkilä & Eija Taipalus. Ammattikorkeakoulusta työelämään. Seinäjoen ammattikorkeakoulusta v. 2001–2003 valmistuneiden sijoittuminen opiskelun jälkeen. 2005.

22. Tuija Pitkäkoski, Sari Pajuniemi & Hanne Vuorenmaa (ed.). Food Choices and Healthy Eating. Focusing on Vegetables, Fruits and Berries. International Conference September 2nd – 3rd 2005. Kauhajoki, Finland. Proceedings. 2005.

23. Katariina Perttula. Kokemuksellinen hyvinvointi Seinäjoen kolmella asuinalueella. Raportti pilottihankkeen tuloksista. 2005.

24. Mervi Lehtola. Alueellinen hyvinvointitiedon malli – asiantuntijat puhujina. Hankkeen loppuraportti. 2005.

25. Timo Suutari, Kari Salo & Sami Kurki. Seinäjoen teknologia- ja innovaatiokeskus Frami vuorovaikutusta ja innovatiivisuutta edistävänä ympäristönä. 2005.

26. Päivö Laine. Pk-yritysten verkkosivustot – vuorovaikutteisuus ja kansainvälistyminen. 2006.

27. Erno Tornikoski, Elina Varamäki, Marko Kohtamäki, Erkki Petäjä, Tarja Heikkilä, Kirsti Sorama. Asiantuntijapalveluyritysten yrittäjien näkemys kasvun mahdollisuuksista ja kasvun seurauksista Etelä- ja Keski-Pohjanmaalla –Pro Advisor –hankkeen esiselvitystutkimus. 2006.

28. Elina Varamäki (toim.) Omistajanvaihdosnäkymät ja yritysten jatkuvuuden edistäminen Etelä-Pohjanmaalla. 2007.

29. Beck Thorsten, Bruun-Schmidt Henning, Kitinoja Helli, Sjöberg Lars, Svensson Owe and Vainoras Alfonsas. eHealth as a facilitator of transnational cooperation on health. A report from the Interreg III B project "eHealth for Regions". 2007.

30. Anmari Viljamaa, Elina Varamäki (toim.) Etelä-Pohjanmaan yrittäjyyskatsaus 2007. 2007.

31. Elina Varamäki - Tarja Heikkilä - Eija Taipalus – Marja Lautamaja. Ammattikorkeakoulusta työelämään. Seinäjoen ammattikorkeakoulusta v.2004–2005 valmistuneiden sijoittuminen opiskelujen jälkeen. 2007.

32. Sulevi Riukulehto. Tietoa, tasoa, tekoja. Seinäjoen ammatti-korkeakoulun ensimmäiset vuosikymmenet. 2007.

33. Risto Lauhanen & Jussi Laurila. Bioenergian hankintalogistiikka. Tapaustutkimuksia Etelä-Pohjanmaalta. 2007.

34. Jouni Niskanen (toim.). Virtuaalioppimisen ja -opettamisen Benchmarking Seinäjoen ammattikorkeakoulun, Seinäjoen yliopistokeskuksen sekä Kokkolan yliopistokeskuksen ja Keski-Pohjanmaan ammattikorkeakouun Averkon välillä keväällä 2007. Loppuraportti. 2007.

35. Heli Simon & Taina Vuorela. Ammatillisuus ammattikorkeakoulujen kielten- ja viestinnänopetuksessa. Oulun seudun ammattikorkeakoulun ja Seinäjoen ammattikorkeakoulun kielten- ja viestinnänopetuksen arviointi- ja kehittämishanke 2005–2006. 2008.

36. Margit Närvä - Matti Ryhänen - Esa Veikkola - Tarmo Vuorenmaa. Esiselvitys maidontuotannon kehittämiskohteista. Loppuraportti. 2008.

37. Anu Aalto, Ritva Kuoppamäki & Leena Niemi. Sosiaali- ja terveysalan yrittäjyyspedagogisia ratkaisuja. Seinäjoen ammattikorkeakoulun Sosiaali- ja terveysalan yksikön kehittämishanke. 2008.

38. Anmari Viljamaa, Marko Rossinen, Elina Varamäki, Juha Alarinta, Pertti Kinnunen & Juha Tall. Etelä-Pohjanmaan yrittäjyyskatsaus 2008. 2008.

39. Risto Lauhanen. Metsä kasvaa myös Länsi-Suomessa. Taustaselvitys hakkuumahdollisuuksista, työmääristä ja resurssitarpeista. 2009.

40. Päivi Niiranen & Sirpa Tuomela-Jaskari. Haasteena ikäihmisten päihdeongelma? Selvitys ikäihmisten päihdeongelman esiintyvyydestä pohjalaismaakunnissa. 2009.

41. Jouni Niskanen. Virtuaaliopetuksen ajokorttikonsepti. Portfoliotyyppinen henkilöstökoulutuskokonaisuus. 2009.

42. Minttu Kuronen-Ojala, Pirjo Knif, Anne Saarijärvi, Mervi Lehtola & Harri Jokiranta. Pohjalaismaakuntien hyvinvointibarometri 2009. Selvitys pohjalaismaakuntien hyvinvoinnin ja hyvinvointipalveluiden tilasta sekä niiden muutossuunnista. 2009.

43. Vesa Harmaakorpi, Päivi Myllykangas ja Pentti Rauhala. Seinäjoen ammattikorkeakoulu. Tutkimus-, kehittämis ja innovaatiotoiminnan arviointiraportti. 2010.

44. Elina Varamäki (toim.), Pertti Kinnunen, Marko Kohtamäki, Mervi Lehtola, Sami Rintala, Marko Rossinen, Juha Tall ja Anmari Viljamaa. Etelä-Pohjanmaan yrittäjyyskatsaus 2010. 2010.

45. Elina Varamäki, Marja Lautamaja & Juha Tall. Etelä-Pohjanmaan omistajanvaihdosbarometri 2010. 2010.

46. Tiina Sauvula-Seppälä, Essi Ulander ja Tapani Tasanen (toim.). Kehittyvä metsäenergia. Tutkimusseminaari Seinäjoen Framissa 18.11.2009. 2010.

47. Autio Veli, Björkman Jouni, Grönberg Peter, Heinisuo Markku & Ylihärsilä Heikki. Rakennusten palokuormien inventaariotutkimus. 2011.

48. Erkki K. Laitinen, Elina Varamäki, Juha Tall, Tarja Heikkilä & Kirsti Sorama. Omistajanvaihdokset Etelä-Pohjanmaalla 2006-2010 -ostajayritysten ja ostokohteiden profiilit ja taloudellinen tilanne. 2011.

49. Elina Varamäki, Tarja Heikkilä & Marja Lautamaja. Nuorten, aikuisten sekä ylemmän tutkinnon suorittaneiden sijoittuminen työelämään - seurantatutkimus Seinäjoen ammattikorkeakoulusta v. 2006-2008 valmistuneille. 2011.

50. Vesa Harmaakorpi, Päivi Myllykangas and Pentti Rauhala. Evaluation Report for Research, Development and Innovation Activitiesus. 2011.

51. Ari Haasio & Kari Salo (toim.). AMK 2.0 : Puheenvuoroja sosiaalisesta mediasta ammattikorkeakouluissa. 2011.

52. Elina Varamäki, Tarja Heikkilä, Juha Tall & Erno Tornikoski. Eteläpohjalaiset yrittäjät liiketoimintojen ostajina, myyjinä ja kehittäjinä. 2011.

53. Jussi Laurila & Risto Lauhanen. Pienen kokoluokan CHP -teknologiasta lisää voimaa Etelä-Pohjanmaan metsäkeskusalueelle. 2011.

54. Tarja Keski-Mattinen, Jouni Niskanen & Ari Sivula. Ammattikorkeakoulu-opintojen ohjaus etätyömenetelmillä. 2011.

55. Tuomas Hakonen & Jussi Laurila. Metsähakkeen kosteuden vaikutus polton ja kaukokuljetuksen kannattavuuteen. 2011.

56. Heikki Holma, Elina Varamäki, Marja Lautamaja, Hannu Tuuri & Terhi Anttila. Yhteistyösuhteet ja tulevaisuuden näkymät eteläpohjalaisissa puualan yrityksissä. 2011.

57. Elina Varamäki, Kirsti Sorama, Kari Salo & Tarja Heikkilä. Sivutoimiyrittäjyyden rooli ammattikorkeakoulusta valmistuneiden keskuudessa. 2011.

58. Kimmo Nissinen (toim.). Maitotilan prosessien kehittäminen: Lypsy-, ruokinta- ja lannankäsittely- sekä kuivitusprosessien toteuttaminen; Maitohygienian turvaaminen maitotiloilla; Teknologisia ratkaisuja, rakennuttaminen ja tuotannon ylösajo. 2012.

59. Matti Ryhänen & Erkki Laitila (toim.). Yhteistyö ja resurssit maitotiloilla : Verkostomaisen yrittämisen lähtökohtia ja edellytyksiä. 2012.

60. Jarkko Pakkanen, Kati Katajisto & Ulla El-Bash. Verkostoitunut älykkäiden koneiden kehitysympäristö : VÄLKKY-projektin raportti. 2012.

61. Elina Varamäki, Tarja Heikkilä, Juha Tall, Aapo Länsiluoto & Anmari Viljamaa. Ostajien näkemykset omistajanvaihdoksen toteuttamisesta ja onnistumisesta. 2012.

62. Minna Laitila, Leena Elenius, Hilkka Majasaari, Marjut Nummela, Annu Peltoniemi (toim.). Päihdetyön oppimista ja osaamista ammatti-korkeakoulussa. 2012.

63. Ari Haasio (toim.). Verkko haltuun! - Nätet i besittning!: Näkökulmia verkostoituvaan kirjastoon. 2012.

64. Anmari Viljamaa, Sanna Joensuu, Beata Taijala, Seija Råtts, Tero Turunen, Kaija-Liisa Kivimäki & Päivi Borisov. Elävästä elämästä: Kumppaniyrityspedagogiikka oppimisympäristönä. 2012.

65. Kirsti Sorama. Klusteriennakointimalli osaamistarpeiden ennakointiin: Ammatillisen korkea-asteen koulutuksen opetussisältöjen kehittäminen. 2012.

66. Anna Saarela, Ari Sivula, Tiina Ahtola & Antti Pasila. Mobiilisovellus bioenergiaalan oppismisympäristöksi Bioenergia-asiantuntijuuden kehittäminen työelämälähtöisesti -hanke. 2013.

67. Ismo Makkonen. Korjuri vs. koneketjuenergia puunkorjuussa. 2013.

68. Ari Sivula, Risto Lauhanen, Anna Saarela, Tiina Ahtola & Antti Pasila Bioenergia-asiantuntijuutta kehittämässä Etelä-Pohjanmaalla. 2013.

69. Juha Tall, Kirsti Sorama, Piia Tulisalo, Erkki Petäjä & Ari Virkamäki. Yrittäjyys 2.0. – menestyksen avaimia. 2013.

70. Anu Aalto & Salla Kettunen. Hoivayrittäjyys ikääntyvien palveluissa - nyt ja tulevaisuudessa. 2013.

71. Varpu Hulsi, Tuomas Hakonen, Risto Lauhanen & Jussi Laurila. Metsänomistajien energiapuun myyntihalukkuus Etelä- ja Keski-Pohjanmaan metsäkeskusalueella. 2013.

72. Anna Saarela. Nuoren metsänhoitokohteen ympäristön hoito ja työ-turvallisuus: Suomen metsäkeskuksen Etelä- ja Keski-Pohjanmaan alueyksikön alueella toimivien energiapuuyrittäjien haastattelu. 2014.

74. Elina Varamäki, Tarja Heikkilä, Juha Tall, Anmari Viljamaa & Aapo Länsiluoto. Omistajanvaihdoksen toteutus ja onnistuminen ostajan ja jatkajan näkökulmasta. 2013.

75. Minttu Kuronen-Ojala, Mervi Lehtola & Arto Rautajoki. Etelä-Pohjanmaan, Keski-Pohjanmaan ja Pohjanmaan hyvinvointibarometri 2012: ajankohtainen arvio pohjalaismaakuntien väestön hyvinvoinnin ja palvelujen tilasta sekä niiden muutossuunnista. 2014.

76. Elina Varamäki, Juha Tall, Anmari Viljamaa, Kirsti Sorama, Aapo Länsiluoto, Erkki Petäjä & Erkki K. Laitinen Omistajanvaihdos osana liiketoiminnan kehittämistä ja kasvua - tulokset, johtopäätökset ja toimenpide-ehdotukset. 2013.

77. Kirsti Sorama, Terhi Anttila, Salla Kettunen & Heikki Holma. Maatilojen puurakentamisen tulevaisuus: Elintarvikeklusterin ennakointi. 2013.

78. Hannu Tuuri, Heikki Holma, Yrjö Ylkänen, Elina Varamäki & Martti Kangasniemi. Kuluttajien ostopäätöksiin vaikuttavat tekijät ja oheispalveluiden tarpeet huonekaluhankinnoissa: Eväitä kotimaisen huonekaluteollisuuden markkina-aseman parantamiseksi. 2013.

79. Ismo Makkonen. Päästökauppa ja sen vaikutukset Etelä- ja Keski-Pohjanmaalle. 2014.

80. Tarja Heikkilä, Marja Katajavirta & Elina Varamäki. Nuorten ja aikuisten tutkinnon suorittaneiden sijoittuminen työelämään – seurantatutkimus Seinäjoen ammattikorkeakoulusta v. 2009–2012 valmistuneille. 2014.

81. Sari-Maarit Peltola, Seliina Päällysaho & Sirkku Uusimäki (toim.). Proceedings of the ERIAFF conference "Sustainable Food Systems: Multi-actor Co-operation to Foster New Competitiveness of Europe". 2014.

82. Sarita Ventelä, Heikki Koskimies & Juhani Kesti. Lannan vastaanotto-halukkuus kasvinviljelytiloilla Etelä- ja Pohjois-Pohjanmaalla. 2014.

83. Maciej Pietrzykowski & Timo Toikko (Eds.). Sustainable welfare in a regional context. 2014.

84. Janne Jokelainen. Log construction training in the Nordic and the Baltic Countries. PROLOG Final Report. 2014.

85. Anne Kuusela. Osallistava suunnittelun tiedonhankintaprosessi kolmannen iän asumisympäristötarpeiden kartoittamisessa: CoTHREE-projektin raportti. 2015

88. Ismo Makkonen. Bioöljyjalostamon investointiedellytykset Etelä-Pohjanmaan maakunnassa. 2014.

90. Anmari Viljamaa, Elina Varamäki, Arttu Vainio, Anna Korsbäck ja Kirsti Sorama. Sivutoiminen yrittäjyys ja sivutoimisesta päätoimiseen yrittäjyyteen kasvun tukeminen Etelä-Pohjanmaalla. 2014.

91. Elina Varamäki, Anmari Viljamaa, Juha Tall, Tarja Heikkilä, Salla Kettunen & Marko Matalamäki. Kesken jääneet yrityskaupat - myyjien ja ostajien näkökulma. 2014.

92. Terhi Anttila, Hannu Tuuri, Elina Varamäki & Yrjö Ylkänen. Millainen on minun huonekaluni? Kuluttajien huonekaluhankintoihin arvoa luovat tekijät ja markkinasegmentit. 2014.

93. Anu Aalto, Anne Matilainen & Maria Suomela. Etelä-Pohjanmaan Green Care -strategia 2015 - 2020. 2014.

94. Kirsti Sorama, Salla Kettunen & Elina Varamäki. Rakennustoimialan ja puutuotetoimialan yritysten välinen yhteistyö : Nykytilanne ja tulevaisuuden suuntaviivoja. 2014.

95. Katariina Perttula, Hillevi Eromäki, Riikka Kaukonen, Kaija Nissinen, Annu Peltoniemi & Anu Hopia. Kropsua, hunajaa ja puutarhan tuotteita: ruokakulttuuri osana ikäihmisten hyvää elämää. 2015

96. Heikki Holma, Salla Kettunen, Elina Varamäki, Kirsti Sorama & Marja Katajavirta. Menestystekijät puutuotealalla: aloittavien ja kokeneiden yrittäjien näkemykset. 2014.

97. Anna Saarela, Heikki Harmanen & Juha Tuorila. Happamien sulfaattimaiden huomioiminen tilusjärjestelyissä. 2014.

98. Erkki Kytönen, Juha Tall & Aapo Länsiluoto. Yksityinen riskipääoma pienten yritysten kasvun edistäjänä Etelä-Pohjanmaalla. 2015.

99. Eliisa Kallio, Juhani Suojaranta & Ari Sivula. Seinäjoen ammattikorkeakoulun Elintarvike- ja maatalouden yksikön työharjoitteluprosessin kehittäminen virtuaalimaatiloilla: oppimisympäristö työharjoittelun tukena. 2015

100. Tarja Heikkilä & Marja Katajavirta. Seinäjoen ammattikorkeakoulun opiskelijabarometri 2014. Tutkintoon johtavassa koulutuksessa olevien nuorten toisen ja valmistuvien vuosikurssien sekä aikuisopiskelijoiden tulokset. 2015.

101. Juha Tall, Elina Varamäki, Salla Kettunen & Marja Katajavirta. Perustamalla tai ostamalla yrittäjäksi - kokemukset yrittäjäuran alkutaipaleelta. 2015

102. Sarita Ventelä (toim.), Toni Sankari, Kaija Karhunen, Anna Saarela, Tapio Salo, Markus Lakso & Tiina Karsikas. Lannan ravinteet kiertoon Etelä- ja Pohjois-Pohjanmaalla: Hydro-Pohjanmaa -hankkeen loppujulkaisu 1. 2014.

# C. OPPIMATERIAALEJA - TEACHING MATERIALS

1. Ville-Pekka Mäkeläinen. Basics of business to business marketing. 1999.

2. Lea Knuuttila. Mihin työohjausta tarvitaan? Oppimateriaalia sosiaalialan opiskelijoiden työnohjauskurssille. 2001.

3. Mirva Kuni & Petteri Männistö & Markus Välimaa. Leikkauspelot ja niiden hoitaminen. 2002.

4. Kempas Ilpo & Bartens Angela. Johdatus portugalin kielen ääntämiseen: Portugali ja Brasilia. 2011.

5. Ilpo Kempas. Ranskan kielen prepositio-opas : Tavallisimmat tapaukset, joissa adjektiivi tai verbi edellyttää tietyn preposition käyttöä tai esiintyy ilman prepositiota. 2011.

6. Risto Lauhanen, Jukka Ahokas, Jussi Esala, Tuomas Hakonen, Heikki Sippola, Juha Viirimäki, Esa Koskiniemi, Jussi Laurila & Ismo Makkonen. Metsätoimihenkilön energialaskuoppi. 2014.

7. Jyrki Rajakorpi, Erkki Laitila & Mari Viljanmaa. Esimerkkejä maatalousyritysten yhteistyöstä: näkökulmia maitotilojen verkostoihin. 2014.

8. Douglas D. Piirto. Leadership : A lifetime quest for excellence. 2014.

**Seinäjoen ammattikorkeakoulu**
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Seinäjoen korkeakoulukirjasto
Kalevankatu 35, PL 97, 60101 Seinäjoki
puh. 020 124 5040 fax 020 124 5041
seamk.kirjasto@seamk.fi

ISBN 978-952-7109-21-2 (verkkojulkaisu)
ISSN 1797-5581 (verkkojulkaisu)