SEINÄJOEN AMMATTIKORKEAKOULU Luentomoniste

Tekniikan yksikkö

Hilkka Niemelä

18.9.2014

**OHJELMOINNIN PERUSTEET 1**

**SISÄLLYS**

[1 JOHDANTO 3](#_Toc273016196)

[2 JOHDATUS C#-KIELEEN 4](#_Toc273016197)

[2.1 C#-perusteet 4](#_Toc273016198)

[2.1.1 Ohjelmaesimerkki 4](#_Toc273016199)

[2.1.2 Kommentointi 7](#_Toc273016200)

[2.1.3 Muuttujat ja tyyppimäärittelyt 8](#_Toc273016201)

[2.1.4 Nimet ja operaattorit 11](#_Toc273016202)

[2.1.5 Konsolilta lukeminen ja konsolille tulostaminen 12](#_Toc273016203)

[2.1.6 Harjoituksia lukuun 2 14](#_Toc273016204)

[3 KONTROLLIRAKENTEET 16](#_Toc273016205)

[3.1 Valintarakenteet 16](#_Toc273016206)

[3.1.1 if-lause 16](#_Toc273016207)

[3.1.2 Harjoituksia if-lauseesta 19](#_Toc273016208)

[3.1.3 switch-case lause 22](#_Toc273016209)

[3.1.4 Harjoituksia switch-case lauseesta 23](#_Toc273016210)

[3.2 Toistorakenteet 24](#_Toc273016211)

[3.2.1 for-lause 24](#_Toc273016212)

[3.2.2 foreach-lause 25](#_Toc273016213)

[3.2.3 while-lause 26](#_Toc273016214)

[3.2.4 do-while lause 27](#_Toc273016215)

[3.2.5 Harjoituksia toistorakenteista 27](#_Toc273016216)

[For-lause: 27](#_Toc273016217)

[3.3 Siirtorakenteet 30](#_Toc273016218)

[3.3.1 Keskeytys break 30](#_Toc273016219)

[3.3.2 Metodin keskeytys return 30](#_Toc273016220)

[3.3.3 Väliinjättö continue 31](#_Toc273016221)

[3.3.4 Hyppylause goto 31](#_Toc273016222)

[3.3.5 Harjoituksia siirtorakenteista 32](#_Toc273016223)

[4 METODIT 33](#_Toc273016224)

[4.1 Metodin määrittely 33](#_Toc273016225)

[4.2 Parametrit 35](#_Toc273016226)

[4.2.1 Arvoparametrit 35](#_Toc273016227)

[4.2.2 Viittausparametrit 38](#_Toc273016228)

[4.2.3 Ulostuloparametrit 39](#_Toc273016229)

[4.2.4 Vaihtuvamittainen parametrilista 40](#_Toc273016230)

[4.2.5 Harjoituksia metodeista 40](#_Toc273016231)

[5 TAULUKOT 43](#_Toc273016232)

[5.1 Miksi taulukoita tarvitaan 43](#_Toc273016233)

[5.2 Taulukkoharjoituksia 46](#_Toc273016234)

[6 MERKIT JA MERKKIJONOT 48](#_Toc273016235)

[6.1 Merkit 48](#_Toc273016236)

[6.2 Merkkijonot 48](#_Toc273016237)

[6.3 Harjoituksia merkeistä ja merkkijonoista 50](#_Toc273016238)

# JOHDANTO

Tietokone on rakennettu noudattamaan yksinkertaisia komentoja, jotka käsittelevät koneen komponenteissa olevia bittien jonoja. Näitä koneen ’osaamia’ komentoja kutsutaan konekieleksi. Konekieli on ihmisen kirjoittamana virhealtista ja työlästä, siksi lähes kaikki ohjelmointi tapahtuu nykyisin ns. lausekielellä 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.

C# on Microsoftin kehittämä lausekieli. C#-kieli julkaistiin vuonna 2000. Kieli on kehitetty C++:n ja Javan pohjalta. C#-sovellukset laaditaan .NET Framework kehykseen. .NET Framework on yhteinen alusta useille ohjelmointikielille. Virallisia NET-ohjelmointikieliä ovat C#, Visual C++, VB.NET, JScript.NET ja Visual J#. Sovellus voi muodostua eri ohjelmointikielillä ohjelmoiduista komponenteista.

Common Language Specification CLS määrittelee joukon sääntöjä, joita kaikkien .NET-kielien on noudatettava (<http://msdn.microsoft.com/en-us/library/12a7a7h3.aspx>).

Tähän liittyy myös Common Type System CTS, joka määrittelee kaikki .NETin tukemat tietotyypit (<http://msdn.microsoft.com/en-us/library/zcx1eb1e.aspx>). Näitä määrittelyjä tarvitaan, jotta eri ohjelmointikielillä ohjelmoidut osat ymmärtävät toisiaan. Kääntäjät, jotka noudattavat CLS määrityksiä, luovat objekteja, jotka voivat toimia yhdessä.

.NET Frameworkin yhteinen perusluokkakirjasto BCL (Base Class Library) on kaikille .NETin kielille yhteinen. (<http://msdn.microsoft.com/en-us/library/zcx1eb1e.aspx>) Visual Studio-kehittimestä kirjastoon pääsee käsiksi esimerkiksi Help – Contents - .NET Development - .NET Framework SDK – Class Library Reference.

Kaikki .NET-kääntäjät generoivat välikoodia eli lähdekieliset ohjelmat käännetään välikielisiksi assemblyiksi: Microsoft Common Intermediate Language MCIL = CIL = IL (vrt. Javan bytekoodi). IL ei ole sidoksissa käyttöjärjestelmään eikä prosessoriin eli NET-kääntäjät eivät generoi natiivia koodia.

CLR Common Language Runtime on .NETin ajonaikainen suoritusympäristö (vrt. JVM tulkki), joka kääntää CIL-tavukoodin ajonaikaisesti konekielelle (JIT-käännös Just In Time) prosessorin suoritettavaksi koodiksi. NET ei siis ole tulkkaava ympäristö, vaan koodi käännetään aina natiiviin muotoon ennen ajoa. CLR voidaan toteuttaa eri käyttöjärjestelmille. CLR ei ota kantaa käytettyyn kieleen, se voi olla mikä tahansa.

C#-kielellä kirjoitettu ohjelma

MCIL-assembly

c#-kääntäjä

Tulokset

CLR

# JOHDATUS C#-KIELEEN

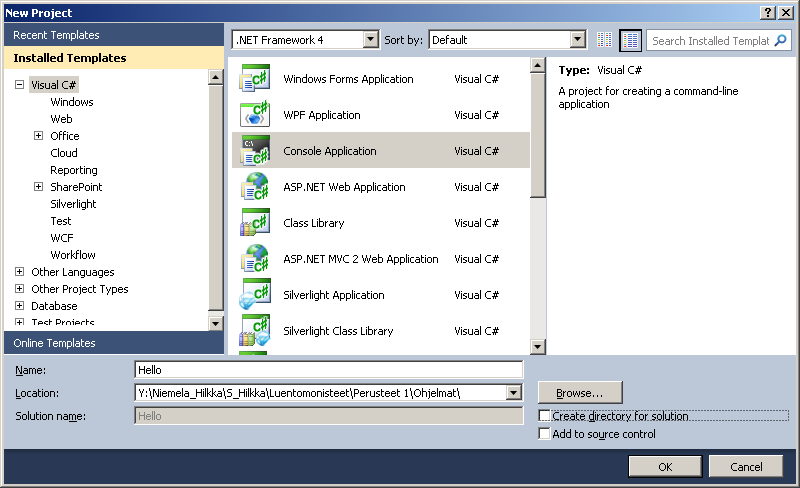
## C#-perusteet

### Ohjelmaesimerkki

Kätevintä C#-ohjelman kehittäminen on VisulStudion työkaluilla.

Tutustutaan ohjelman tekemiseen perinteisellä Hello world-ohjelmalla. VisualStudiossa ohjelmat ohjelmoidaan projekteihin eli ensin VisualStudiolla luodaan uusi projekti. Valitaan käytettävä templatekieli ja templateksi Console Application (eli kerrotaan kehittimelle, että se pohjustaa C#-kielisen ohjelman, jonka käyttöliittymä perustuu konsolilla annettuihin komentoihin). Nimetään projekti (Hello) ja määritetään sijainti.

1. File – New project…



1. Valitse Installed Templates-ikkunasta pohjakieleksi Visual C#,   
   keski-ikkunasta pohjaksi Console Application,   
   kirjoita ohjelman nimi Name-kohtaan,   
   määrittele kansio, johon ohjelma talletetaan Location-kohtaan  
   ja ota rasti pois ruudusta Create directory for solution  
   OK

VisualStudio-kehitin luo ohjelmaa varten valmiin pohjan, johon tarvittavat lauseet lisätään. Ohjelmaan, joka tervehtii konsolitulostuksena maailmaa sanoilla Hello world, on kirjoitettava ainoastaan tulostuksen aikaansaava rivi (Console.WriteLine("Hello world!");).

using System;

using System.Collections.Generic;

using System.Text;

namespace Hello

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello world!");

}

}

}

using System; Using-avainsanalla kerrotaan, minkä nimiavaruuden sisältämiä luokkia ym. halutaan koodissa käyttää. Using-määreellä saadaan koodin kirjoitusasu lyhyemmäksi. Esimerkiksi yllä olevassa olisi ilman using-määrettä pitänyt kirjoittaa System.Console.Writeline(…). Tässä ohjelmassa vain ensimmäinen using-määre tulee käyttöön, muut voisi poistaa.

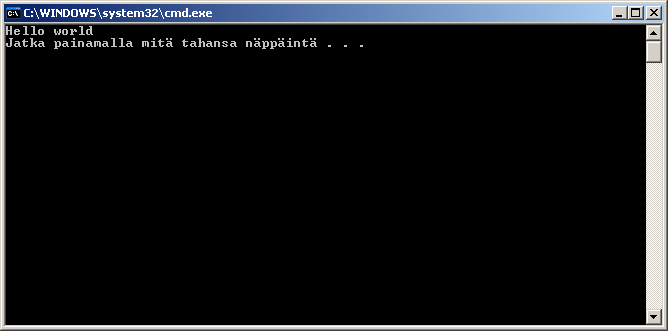
namespace Hello -määreellä luokka saadaan kuulumaan nimettyyn nimiavaruuteen, johon sitten voidaan viitata using-lauseella.

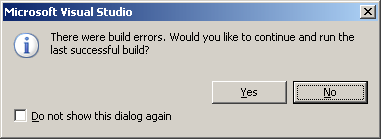
class Program Kaikki C#-ohjelmat kirjoitetaan luokkaan. Kehitin nimeää luokan automaattisesti nimellä Program. Ohjelman nimi voidaan muuttaa Solution Explorer-ikkunassa rename-operaatiolla.

static void Main(string[] args) on varsinainen pääohjelma, josta käynnistettävän ohjelman suoritus aloitetaan ja jonne ohjelmoija kirjoittaa suoritettavat lauseet.

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.

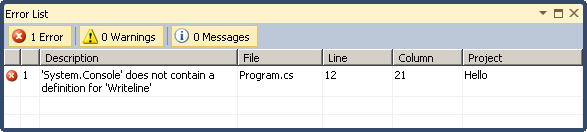
Ohjelma voidaan VisualStudiossa kääntää ensin välikielelle ja sitten erikseen suorittaa tai käännös ja suoritus voidaan käynnistää yhtä aikaa. Pelkän käännöksen voi tehdä esimerkiksi Build-valikosta komennolla Build Hello. Ohjelma ajetaan Build-valikon Start without Debugging-komennolla (CTRL-F5). Ohjelma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



Jos ohjelman lauseissa on kielioppivirheitä, kääntäjä antaa niistä ilmoituksen:  


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

ErrorList-ikkunaan ruudun alalaitaan tulostuu kaikki havaitut käännösvirheet.



Tässä kääntäjä kertoo, että sellaista määritystä kuin Writeline ei ole olemassa po. WriteLine. Virheselostetta kaksoisnapauttamalla kohdistus siirtyy suoraan virheen esiintymispaikkaan.

Sovelluksesta muodostuu useita erilaisia tiedostoja. Suoraan projektikansioon on sijoitettu ohjelmoimamme **Program.cs**. C#-ohjelmien tiedostotunniste on siis cs. Projektikansiossa on lisäksi

**Hello.sln**, joka sisältää tietoja solutioniin kuuluvista projekteista. Sovellusta uudelleen esille otettaessa juuri \*.sln-tiedosto avataan File-Open Project-komennolla. Tiedosto Hello.suo sisältää käyttäjän VS-asetukset. Tiedosto Hello.csproj sisältää projektin asetukset. Hello.csproj.user sisältää käyttäjän projektikohtaiset asetukset. Ohjelma käännetään bin\debug-kansioon eli siellä oleva Hello.exe on ohjelmamme välikielinen versio, joka voidaan ajaa CLR:llä. Kansioon obj talletetaan kääntäjän väliaikaiset tiedostot ja Properties-kansiossa AssemblyInfo.cs-tiedostossa sijaitsevat sovelluksen attribuutit, jotka kääntäjä liittää muodostettavaan sovellukseen.

### 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.

Loppurivi saadaan kommentiksi merkitsemällä //

Usean rivin kommentti alkaa merkeillä /\* ja päättyy merkkeihin \*/

Kommentit eivät mitenkään vaikuta ohjelman toimintaan, vaan ne ovat vain vihjeitä ohjelman sisällöstä ja toimintatavoista koodin lukijalle.

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

/\*

\* Sovellus tervehtii maailmaa tekstillä Hello world

\* tekijä: Hilkka Niemelä

\* 1.9.2010

\*

\*/

using System;

namespace Hello

{

class Program

{

static void Main(string[] args)

{

// Tervehdyksen tulostus konsolille

Console.WriteLine("Hello world");

}

}

}

### 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 muuttujaa 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.

using System;

namespace 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 = -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 lauseessa on aina esiinnyttävä myös parametrit {0} ja {1}. Lainausmerkin jälkeistä pilkkua seuraavat muuttujat, joiden sisältö tulee parametrien paikalle. Niiden on oltava järjestyksessä: nollan sijaan tuleva, ykkösen sijaan tuleva jne.

Taulukko C#-kielessä olevista perusmuuttujatyypeistä:

|  |  |
| --- | --- |
| **C# Type** | **.NET Framework type (aliasnimi)** |
| [**bool**](http://msdn.microsoft.com/library/en-us/csref/html/vcrefthebooltype.asp) **(true, false)** | **System.Boolean** |
| [**byte**](http://msdn.microsoft.com/library/en-us/csref/html/vclrfbyte_pg.asp) **(8 bit, etumerkitön, 0 - 255)** | **System.Byte** |
| [sbyte](http://msdn.microsoft.com/library/en-us/csref/html/vclrfsbyte.asp) (8 bit, etumerkillinen,  -128 - 127) | **System.SByte** |
| [**char**](http://msdn.microsoft.com/library/en-us/csref/html/vclrfchar_pg.asp) **(16 bit unicode merkki)** | **System.Char** |
| [decimal](http://msdn.microsoft.com/library/en-us/csref/html/vcrefthedecimaltype.asp) (128 bit, vakiotunnus m, arvoalue 7.9\*1028) | **System.Decimal** |
| [**double**](http://msdn.microsoft.com/library/en-us/csref/html/vclrfdouble_pg.asp) **(64 bit, vakiotunnus d, arvoalue 1.7\*10308)** | **System.Double** |
| [float](http://msdn.microsoft.com/library/en-us/csref/html/vclrffloat_pg.asp) (32 bit, vakiotunnus f, arvoalue 3.4\*1038) | **System.Single** |
| [**int**](http://msdn.microsoft.com/library/en-us/csref/html/vclrfint_pg.asp) **(32 bit, kokoanaisluku)** | **System.Int32** |
| [uint](http://msdn.microsoft.com/library/en-us/csref/html/vclrfuintpg.asp) (u = etumerkitön) | **System.UInt32** |
| [long](http://msdn.microsoft.com/library/en-us/csref/html/vclrflong_pg.asp) (64 bit, kokoanaisluku) | **System.Int64** |
| [ulong](http://msdn.microsoft.com/library/en-us/csref/html/vclrfulongpg.asp) (u = etumerkitön) | **System.UInt64** |
| [short](http://msdn.microsoft.com/library/en-us/csref/html/vclrfshort.asp) (16 bit, kokoanaisluku) | **System.Int16** |
| [ushort](http://msdn.microsoft.com/library/en-us/csref/html/vclrfushortpg.asp) (u = etumerkitön) | **System.UInt16** |
| [**string**](http://msdn.microsoft.com/library/en-us/csref/html/vclrfstring.asp) **(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 28 :lla (=256) eri tavalla. Koska nolla otetaan mukaan, lukualueeksi tulee 0 – 255.

**sbyte** on 8 bitin kokonaisluku, joka voi olla negatiivinenkin eli yksi bitti ’hukkaantuu’ 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 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 joko heti käännösvaiheessa tai suoritusvaiheessa virheellisenä toimintana.

### Nimet ja operaattorit

**Nimet** ovat ohjelmoijan valitsemia nimiä esimerkiksi muuttujille ja luokille 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).

**Varattuja sanoja** kielessä ovat seuraavat:

|  |  |  |  |
| --- | --- | --- | --- |
| [abstract](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfabstract.asp) | [event](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfeventpg.asp) | [new](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfnewoppg.asp) | [struct](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvcrefstructtypes.asp) |
| [as](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfas.asp) | [explicit](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfexplicit.asp) | [null](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfnull.asp) | [switch](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftheswitchstatement.asp) |
| [base](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfbasepg.asp) | [extern](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfexternpg.asp) | [object](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvcgrfobject.asp) | [this](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthispg.asp) |
| [bool](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvcrefthebooltype.asp) | [false](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrffalse.asp) | [operator](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfoperator.asp) | [throw](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthethrowstatement.asp) |
| [break](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthebreakstatement.asp) | [finally](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthetryfinallystatementx.asp) | [out](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfout.asp) | [true](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftrue.asp) |
| [byte](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfbyte_pg.asp) | [fixed](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrffixed.asp) | [override](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfoverridepg.asp) | [try](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthetrycatchstatement.asp) |
| [case](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftheswitchstatement.asp) | [float](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrffloat_pg.asp) | [params](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfparams.asp) | [typeof](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftypeofpg.asp) |
| [catch](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthetrycatchstatement.asp) | [for](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftheforstatement.asp) | [private](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfprivatepg.asp) | [uint](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfuintpg.asp) |
| [char](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfchar_pg.asp) | [foreach](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftheforeachstatement.asp) | [protected](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfprotectedpg.asp) | [ulong](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfulongpg.asp) |
| [checked](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfchecked.asp) | [goto](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthegotostatement.asp) | [public](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfpublicpg.asp) | [unchecked](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfunchecked.asp) |
| [class](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvcreftheclasstype.asp) | [if](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftheifstatement.asp) | [readonly](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfreadonlypg.asp) | [unsafe](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfunsafe.asp) |
| [const](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfconstpg.asp) | [implicit](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfimplicit.asp) | [ref](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfref.asp) | [ushort](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfushortpg.asp) |
| [continue](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthecontinuestatement.asp) | [in](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftheforeachstatement.asp) | [return](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthereturnstatement.asp) | [using](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfusing.asp) |
| [decimal](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvcrefthedecimaltype.asp) | [int](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfint_pg.asp) | [sbyte](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfsbyte.asp) | [virtual](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfvirtualpg.asp) |
| [default](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftheswitchstatement.asp) | [interface](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvcreftheinterfacetype.asp) | [sealed](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfsealed.asp) | [volatile](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfvolatile.asp) |
| [delegate](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvcrefthedelegatetype.asp) | [internal](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfinternalpg.asp) | [short](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfshort.asp) | [void](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvcgrfvoid.asp) |
| [do](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthedostatement.asp) | [is](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfispg.asp) | [sizeof](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfsizeofpg.asp) | [while](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfthewhilestatementx.asp) |
| [double](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfdouble_pg.asp) | [lock](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrflockstatement.asp) | [stackalloc](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfstackalloc.asp) |  |
| [else](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrftheifstatement.asp) | [long](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrflong_pg.asp) | [static](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfstaticpg.asp) |  |
| [enum](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvcreftheenumerationtypes.asp) | [namespace](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfnamespace.asp) | [string](http://blog.project84.net/ct.ashx?id=c8bdc67b-8d4c-477f-9f8a-0ecf803c00c4&url=http%3a%2f%2fmsdn.microsoft.com%2flibrary%2fen-us%2fcsref%2fhtml%2fvclrfstring.asp) |  |

**Vakiot**  ovat koodiin kirjoitettuja muuttujan arvoja esim. 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:

[+](http://msdn.microsoft.com/library/en-us/csref/html/vclrfplusoperator.asp)  yhteenlasku

[-](http://msdn.microsoft.com/library/en-us/csref/html/vclrfminusoperator.asp)   vähennyslasku

[\*](http://msdn.microsoft.com/library/en-us/csref/html/vclrftimesoperator.asp)   kertolasku

[/](http://msdn.microsoft.com/library/en-us/csref/html/vclrfdivideoperator.asp)   jakolasku

[%](http://msdn.microsoft.com/library/en-us/csref/html/vclrfremainderoperator.asp) jakojäännös

### Konsolilta lukeminen ja konsolille tulostaminen

Konsolille tulostamiseen ja lukemiseen käytetään .NET Frameworkin System-nimiavaruuden luokkaa Console.

Konsolilta lukemiseen käytetään merkkejä lukevia metodeja:

Read( ) lukee seuraavan **merkin** Unicode-koodin (kokonaisluvun)

ReadLine( ) lukee seuraavan **rivin**

Kirjoittamiseen käytetään metodeja:

Write(…) kirjoittaa 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)  
muuhun käyttöön varattu merkki saadaan tulostettua merkitsemällä sen eteen \-merkin (\” \\)  
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.

// Muotoillaan tulostusta.

Console.WriteLine("MG:n {0} maksaa {1:C} {2} kuukaudelta.\n", "studio", 499.0, 3);

// {1:C} muotoilee listan toisen parametrin näyttämään paikallista valuuttaa.

// Päiväyksen tulostus

Console.WriteLine("Tänään on " + DateTime.Now.ToShortDateString());

### Harjoituksia lukuun 2

**Ensimmäiset ohjelmat**

1. Ohjelmoi ohjelma Hello, joka tervehtii maailmaa sanoilla Hello world!
2. Muuta Hello-ohjelmaa siten, että se tervehtii maailmaa myös suomeksi ja ruotsiksi.
3. Tee uusi projekti ja anna sille nimeksi Tänään. Ohjelmoi lauseet, jolla ohjelma tulostaa, mikä viikonpäivä tänään on, mitä kuukautta ja kuinka monetta päivää eletään ja onko nyt aamu- vai iltapäivä. (Konkarit voivat ohjelmoida ohjelman katsomaan tiedot koneen kellosta DateTime, aloittelijat kirjoittavat tiedot tämänhetkisen tilanteen mukaan).
4. Tee uusi projekti PintaAloja. Ohjelmoi lauseet, jotka laskevat ja tulostavat seuraavanlaiset pinta-alat:  
   Neliön pinta-ala, kysy sivun pituus käyttäjältä.  
   Ympyrän pinta-ala, kysy säde käyttäjältä  
   Kolmion pinta-ala, kysy kanta ja korkeus käyttäjältä.  
   Lopuksi ohjelma tulostaa kaikkien kuvioiden yhteenlasketun pinta-alan.

**Yleiskäyttöiset, laskentaa sisältävät ohjelmaharjoitukset**

1. **Tee ohjelma, joka laskee ja tulostaa taksimatkan hinnan. Hintaan vaikuttavat lähtöhinta ja ajettu kilometrimäärä sekä kilometrihinta=0.5.**
2. Tee projekti Matka, jonka ohjelma laskee, kuinka kauan matka jollekin paikkakunnalle kestää, kun ajan laskemiseen käytetään keskinopeutta. Aika tulostetaan kokonaisina tunneina ja minuutteina.
3. Tee projekti Korotukset ja ohjelmoi ohjelma, joka pyytää luvun ja potenssin. Ohjelma korottaa luvun annettuun potenssiin ja tulostaa vastaukset.
4. Tee projekti Bensa ja ohjelmoi siihen ohjelma, joka laskee ja tulostaa tankatun bensiinin hinnan.   
   Tulostusmalli:   
   Tankattiin 50 litraa, litrahinta on 1.283.   
   Yhteensä bensa maksaa 64.15 E.
5. Tee projekti Asteet ja ohjelmoi sinne ohjelma, joka osaa muuntaa Fahrenheit-asteet Celsius-asteiksi (C = 5 \* ( F – 32 ) / 9.0 ). Tarkistus: 32F = 0C  
   - Fahrenheit-asteet kysytään käyttäjältä
6. **Tee projekti Gondoli. Ohjelmoi ohjelma, joka kysyy, kuinka suuri ryhmä on tulossa Gondoli-hissiin. Yhteen hissiin mahtuu 8 henkilöä. Ohjelma tulostaa, montako henkilöä on viimeisessä hississä, kun hissit täytetään järjestyksessä täyteen. Käytä jakojäännös-laskutoimitusta.**
7. Tee ohjelma, joka laskee kuinka kauan mäkihyppääjä on ’lennossa’, kun nopeutena käytetään hyppyriltä lähtönopeutta. Nopeus mitataan kilometreina tunnissa, lopputulos ilmoitetaan sekunteina.
8. **Tee ohjelma, joka tulostaa käteen jäävän nettopalkan, kun käyttäjältä kysytään bruttopalkka ja veroprosentti**
9. Tee ohjelma Nimi, joka kysyy käyttäjän etu- ja sukunimen erikseen ja tulostaa koko nimen.
10. Laadi ohjelma Onnenluvut, joka tulostaa onnenlukusi ja onnenkertoimesi. Onnennumero on jakojäännös (kokonaisluku), joka saadaan, kun pituuden ja painon tulo jaetaan luvulla 21. Onnenkerroin (desimaaliluku) on onnennumero jaettuna luvulla 4.3. Ohjelma kysyy ensin käyttäjän nimen ja muut tarvittavat tiedot lukujen laskentaa varten. Lopuksi ohjelma tulostaa lopputuloksen seuraavassa muodossa:  
      
    Matti Meikäläinen  
    Onnennumero: 15   
    Onnenkerroin: 3.7
11. Tee ohjelma, joka laskee ja tulostaa, kuinka paljon kauemmin köröttelijältä kuluu aikaa tietyn matkan ajamiseen kuin kaaharilta. Laskelma perustuu keskinopeuden käyttöön.  
    Ohjelma tulostaa esimerkiksi Helsingin ja Seinäjoen välisestä matkasta:  
      
    Välillä Seinäjoki - Helsinki köröttelijällä meni aikaa 1 tunti 5 minuuttia enemmän kuin kaaharilla.  
    Kaaharilta meni aikaa 3 tuntia 25 minuuttia.  
    Köröttelijältä meni aikaa 4 tuntia 30 minuuttia.
12. Tee ohjelma, joka tulostaa **yhdellä** tulostuslauseella:  
    Kalle: ”En kyllä tee mitään.”  
    Liisa: ”Ja varmana teet, tai näytän taivaan merkit ?/\*\!!”

**Lisätehtäviä**

1. Tee ohjelma, jolla voit laskea kauanko korkeintaan kestää avata matkalaukku, jonka numerolukon yhdistelmä on unohtunut. Ohjelmalle annettavat lähtötiedot ovat numerolevyjen määrä ja yhden levyn valintamahdollisuuksien määrä sekä arvio siitä kauanko yhden numerosarjan kokeilu vie aikaa.
2. Tee ohjelma, joka laskee kauanko tasoristeyksessä palaa punainen valo, jos se kytkeytyy päälle, kun veturi on 200 metrin päässä tulossa ja kytkeytyy pois, kun viimeinen vaunu on 50 metrin päässä ohittanut tasoristeyksen.
3. **Tee ohjelma, joka laskee kuinka korkealle pesäpallo nousee syöttäjällä, kun pallon lähtönopeus mitataan. Vihje: käytä potentiaali- ja liike-energian kaavaa mgh = ½ mv2 (m = massa, g = 9,81m/s2, h = korkeus, v = nopeus m/s).**
4. Tee ohjelma, joka laskee tehtävän 14 kaaharin ja köröttelijän matkasta: Köröttelijä lähtee tiettyyn kellonaikaan ajamaan matkan toisesta päästä. Ohjelman pitää laskea, koska kaaharin täytyy lähteä liikkeelle, jotta he tapaisivat matkan puolivälissä.

# 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.

## Valintarakenteet

## 

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

### 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 loikkaa else-lohkon ohi eli sen lohkon lauseet jätetään suorittamatta.

Jos looginen lauseke saa arvokseen **false**, loikataan välittömästi seuraavan lohkon yli 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 if** ( looginen-lauseke)

{

MitäTehdäänLausekkeet;

}

**else**

{

MitäTehdäänLausekkeet;

}

Esimerkki 1.

/\*

\* 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.

/\*

\* 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 enää testiä, kaikki muut arvot toteuttavat tämän

{

Console.WriteLine("Ulkona on helle!");

}

}

}

}

**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**.

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

}

}

}

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

&& ja-operaatio

|| tai-operaatio

^ xor-operaatio

! ei-operaatio

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 > b && b > c); // true

}

}

}

if-lauseella on lyhempi merkintätapa:

i == 0 ? a = b; b = a;

on sama kuin:  
if (i == 0)

a = b;

else

b = a;

### Harjoituksia if-lauseesta

1. **Tee ohjelma, joka tulostaa kahdesta annetusta luvusta, ovatko ne yhtä suuria vai erisuuria.  
   - int**
2. **Lisää edelliseen ominaisuus, jonka avulla ohjelma tulostaa, kumpi luvuista oli suurempi, jos ne eivät olleet yhtä suuria.**

1. Tee ohjelma, joka tulostaa, onko kolmio suorakulmainen, kun sen kateettien ja hypotenuusan pituudet annetaan ( kateettien (a ja b)neliöiden summa on sama kuin hypotenuusan (c) neliö c\*c = a\*a + b\*b).
2. Porrasverokortti: Ohjelma kysyy palkan ja ennakonpidätysprosentin ja sille tulorajan, lisäksi kysytään ennakonpidätysprosentti tulorajan ylittävälle tulolle. Ohjelma tulostaa käteen jäävän osuuden.
3. Ohjelma laskee vuosikilometrikorvauksen määrän. Korvaus määräytyy siten, että ekalta 5000 kilometriltä korvaus on 43 senttiä ja ylimeneviltä 30 senttiä.
4. **Ohjelma tulostaa kokeesta saadun pistemäärän perusteella kokeen arvosanan. Arvosanarajat ovat   
   0 – 10: 0  
   11 – 20: 1  
   21 – 30: 2  
   31 – 40: 3**
5. **Ohjelmoi ohjelma, joka määrittelee kuulotutkimuksen perusteella henkilön kuuloaistin tason. Määrittely tapahtuu seuraavilla raja-arvoilla:   
   Kuulokynnys (dB) Kuulonalenema  
   < 25 ei alenemaa  
   25–40 lievä  
   41–55 kohtalainen  
   56–70 keskivaikea  
   71–90 vaikea  
   > 90 erittäin vaikea**
6. **Ohjelmoi yksikertainen laskukone, joka osaa laskea kahdesta luvusta joko summan, erotuksen, tulon, potenssiin korotuksen tai osamäärän. Laskukoneelle annetaan kaksi lukua ja lukujen jälkeen operaattori (+, -, \*,^, /). Esimerkiksi 4 \* 5 => vastaus on 20**
7. Ohjelmoi painoindeksiohjelma. Indeksi saadaan, kun paino(kg) jaetaan pituuden (m) neliöllä. Ohjelma tulostaa painoindeksin ja kommentoi ali- tai ylipainosta. Jos indeksi on alle 18,5 on alipainoinen, jos se on korkeintaan 25 on normaalipainoinen, jos se on korkeintaan 30, on lievästi ylipainoinen, jos indeksi on korkeintaan 35, on vakavasti ylipainoinen ja suuremmat arvot merkitsevät sairaalloista ylipainoa. Indeksi tulostetaan yhdellä desimaalilla. Ohjelman on toimittava nikottelematta antoi käyttäjä pituutensa sitten metreissä tai senteissä.
8. Tee ohjelma, joka tulostaa onko käyttäjän antama luku   
   parillinen vai pariton  
   nollaa suurempi, mutta pienempi kuin 150  
   lukujen 45 ja 65 väliltä
9. Tee ohjelma, joka kysyy käyttäjältä kolme kokonaislukua a, b ja c, sekä desimaaliluvut p, q ja r.   
   Ohjelma tulostaa, onko totta vai tarua:  
   joko a on 7 ja b on negatiivinen tai q on suurempi kuin r:n ja p:n erotus  
   r < -1.7 tai r > 23.1, c:n ja a:n summa on parillinen ja b ei ole 538  
   joko a on 7 tai b on c, mutta ei molemmat
10. Tee ohjelma, joka pyytää kolme lukua ja tulostaa ne suuruusjärjestyksessä.
11. **Tee ohjelma, joka laskee asunnonvälittäjän perimän palkkion. Palkkio on suuruudeltaan 5% asunnon hinnasta, kuitenkin vähintään 2000€ ja enintään 15000€.**

**Lisätehtäviä**

1. Kaveruksilla on yhteinen auto, jossa toisen omistusosuus on 1/3. Auton kustannukset jaetaan siten, että hankittavat lisävarusteet maksetaan omistusosuuksien mukaan ja huolto- ja polttoainekulut ajokilometrien mukaan. Tee ohjelma, jolla voit laskea kaverusten kirjanpidon perusteella, miten vuoden aikana kertyneet kustannukset jakautuvat, kumpiko on velkaa toiselle ja kuinka paljon.
2. **Tee ohjelma, joka vertaa kahden autovuokraamon hinnoittelua auton vuokrauksessa. Hinnoittelu perustuu vuorokausihintaan ja ajokilometreihin. Esimerkiksi: toisessa vuokraamossa auton vuorokausihinta on 110€ vapaalla ajomäärällä. Toisessa vuorokausihinta on 50€ plus 0,60€ kilometri. Ohjelman on tulostettava molempien autojen kokonaisvuokrakulut annetussa tapauksessa ja suositeltava jompaakumpaa vuokraamoa.**

* Kysy käyttäjältä vuorokausien määrä ja arvioidut kilometrit
* Laske vuokra vuokraamossa A ja B (double vuokraA, vuokraB)
* Vertaa laskettuja vuokria ja kerro kumpi oli halvempi

1. Tee ohjelma, joka laskee valtionveron määrän. Vero määräytyy seuraavasti:

|  |  |  |
| --- | --- | --- |
| Alaraja, euroa | Vero alarajan kohdalla, euroa | Vero alarajan ylittävältä osalta, prosenttia |
| 12 600 | 8 | 8,5 % |
| 20 800 | 705 | 19,0 % |
| 34 000 | 3 213 | 23,5 % |
| 62 000 | 9 793 | 31,5 % |

Käyttäjältä kysytään vuositulo. Tee muuttuja veroa varten. Laske sitten vero if-lauseissa.

1. Eräästä kurssista pääsee läpi, jos pistemäärä ylittää tietyn raja-arvon. Kokonaispistemäärän perusteella määräytyy lopullinen arvosana. Lisäksi edellytyksenä on, että ykkösosion kokeesta ja kakkososion kokeesta on saatava tietty minimipistemäärä eli läpi ei pääse, jos jompikumpi pistemäärä on liian matala, vaikka kokonaispisteet riittäisivätkin. Vielä kolmososio suoritetaan hyväksytty/hylätty periaatteella ja siitä on saatava hyväksytty merkintä. Laadi ohjelma, jolla voi määritellä opiskelijan arvosanan.

### switch-case lause

**switch-case** lause toimii siten, että sen avulla on mahdollista haarautua yhteen lukuisasta määrästä erilaisia vaihtoehtoja. Haarautuminen perustuu yhteen 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 ’valu’ 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 jonkun muun kuin ykkösestä neloseen");

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 jonkun muun kuin ykkösestä neloseen");

break;

}

### Harjoituksia switch-case lauseesta

1. Tee ohjelma Tervehdys, joka kysyy käyttäjän nimen ja sukupuolen. Sukupuolesta riippuen ohjelma tervehtii käyttäjää käyttäen puhutteluna sukupuoleen liittyvää sanaa esimerkiksi: Tervehdys herra Turunen.
2. **KOTITEHTÄVÄ 18.9. Erääseen esitykseen myydään lippuja siten, että eläkeläinen saa aikuisten lipusta 20%:n alennuksen, opiskelija 40%:n alennuksen ja lapsi 60%: alennuksen. Laadi ohjelma, joka tulostaa ostettavan lipun hinnan. Käytä ratkaisussa switch-case rakennetta. Normaali aikuisten lippu maksaa 22 euroa. Kysy käyttäjältä onko hän opiskelija, eläkeläinen, lapsi tai joku muu.**
3. Ohjelmoi tehtävän 24 laskukone käyttäen switch-case rakennetta.

## 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.

### 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.

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;

}

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 looginen 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) b) c)

for ( int i = 0; i < 5; i++ )

{ d)

// 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 d), jos arvoksi tulee false, kone jatkaa for-lauseen sulkevan aaltosulun jälkeisestä lauseesta.
3. Toistettavien lauseiden suorituksen jälkeen palataan aina kohtaan c) eli kasvatetaan laskuria.
4. Laskurin kasvatuksen jälkeen toistorutiini alkaa alusta.

Esimerkki 2.

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 " + summa);

}

Esimerkissä toistojen määrä riippuu käyttäjän ilmoittamasta luvusta.

### foreach-lause

**foreach**-lauseella voidaan läpikäydä taulukko tai kokoelma. Palataan siihen asiaan taulukkojen käsittelyn yhteydessä ([linkki](#foreach))

### 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.

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 lähtee liikkeelle

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 ennen kuin 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)

### 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.

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.

### Harjoituksia toistorakenteista

### For-lause:

1. Tee ohjelma, joka tulostaa kaikki positiiviset sataa pienemmät kolmella jaolliset luvut.  
   Testaa ja muuta sen jälkeen ohjelmaa siten, että se tulostaa kolmella jaolliset mielivaltaiselta lukuväliltä.
2. Tee ohjelma, joka tulostaa muunnostaulukon tuumista senteiksi halutulta väliltä. Yksi tuuma on 2.54 cm.  
   Ohjelmoi seuraavanlainen tulostus.  
   Muunnostaulukko:  
   1” 2.54 cm  
   2” 5.08 cm  
   : :
3. Tee ohjelma, joka tulostaa halutun luvun (esimerkissä luku 6) kertotaulun 12 saakka  
   Tulostusmalli esimerkiksi luvusta 6:  
   1 \* 6 = 6  
   2 \* 6 = 12  
    :  
   12 \* 6 = 72
4. Tee ohjelma, joka laskee viikon aikana kertyneet kuntopisteet. Pisteet lasketaan siten, että kävelykilometrista saa yhden pisteen, juoksukilometrista kaksi, hiihtokilometrista 3 ja uintikilometrista 5 pistettä. Päivää kohden hyväksytään vain yksi suoritus. Lopuksi ohjelma tulostaa saadut kuntopisteet.   
   Esimerkiksi:  
   Kävely: 20  
   Juoksu: 6  
   Hiihto: 0  
   Uinti: 5
5. Tee ohjelma, joka kysyy käyttäjältä ensin lukujen lukumäärän. Sen jälkeen kysytään luvut, ohjelma etsii annetuista luvuista pienimmän ja tulostaa sen lopuksi. Lisää myös suurimman etsintä ja lukujen summa.
6. Tee ohjelma, joka tulostaa karkausvuodet halutulta vuosilukuväliltä. Vuosi on karkausvuosi, jos se on jaollinen 4:llä, mutta ei saa olla sadalla jaollinen, poikkeus: 400:lla jaolliset ovat kuitenkin karkausvuosia. Vihje: Ohjelmoi laskurimuuttuja läpikäymään jokaisen vuoden vuosilukuväliltä. Tarkista jokaisesta vuodesta karkausvuosi-sääntö.
7. Tee ohjelma, joka tulostaa halutulla merkillä halutun kokoisen suorakulmion.
8. Tee opetusohjelma, joka testaa käyttäjän yhdestä kymmeneen kertotaulutaitoja. Ohjelma kysyy ensin, montako laskutehtävää esitetään. Seuraavaksi ohjelma kyselee halutun määrän tehtäviä. Aina käyttäjän vastauksen jälkeen ohjelma vastaa, oliko tehtävä oikein vai väärin. Lopuksi ohjelma tulostaa oikeiden ja väärien vastausten lukumäärän.  
   Satunnaisen luvun väliltä 1 – 10 saa lauseilla:  
   Määrittele ohjelman alussa satunnaislukugeneraattorin luonti:  
   Random gen = new Random();  
   Kun tarvitset satunnaislukua, arvo se seuraavalla tavalla:  
   int luku = gen.Next(10)+1;

**While tai do-while:**

1. Tee ohjelma, joka kyselee käyttäjältä lukuja. Ohjelma ilmoittaa, kuinka mones syötetyistä luvuista oli negatiivinen. Negatiiviseen lukuun törmättyään ohjelma lopettaa toimintansa ja tulostaa monesko se oli.
2. Tee tehtävän 44 kertotauluohjelmasta uusi versio, jossa käyttäjä saa päättää, koska lopetetaan. Ohjelma kysyy aina tehtävän suorituksen jälkeen Jatketaanko?
3. Tee ohjelma, joka pelaa kanssasi yksinkertaistettua venttiä. Molemmille jaetaan kaksi korttia. Lisäkortteja ei saa ottaa. Se voittaa, joka pääsee lähemmäksi lukua 21, yli ei saa mennä. Kone on jakaja => tasapelissä kone voittaa. Ohjelma tulostaa molempien kortit ja kumpi voitti. Peliä on voitava jatkaa ilman uudelleen käynnistystä. Lopuksi ohjelma tulostaa kummankin pelaajan voittojen määrän.
4. Ohjelmoi seuraava peli: Tietokone arpoo jonkin kokonaisluvun väliltä 0 – 9. Käyttäjä saa yrittää arvata korkeintaan kolme kertaa, minkä luvun kone arpoi. Jos arvaa ekalla kerralla, pelaaja voittaa 30e, jos toisella voittaa 20e ja kolmannella 10e. Jos ei arvaa, häviää 8e. Peliä voi jatkaa niin kauan kuin haluaa, kunhan kassassa on vähintään seuraavan pelin häviöön riittävä rahamäärä.

**Lisätehtäviä:**

1. Ohjelma piirtää \*-merkeillä halutun korkuisen kuusen: \*  
    \* \*  
    \* \* \*  
    \* \* \* \*
2. Laske ekologisen jalanjälkesi pisteet ja päiväkeskiarvo viime viikolta liikkumisen osalta. Käytä seuraavaa pisteytystä.

|  |  |
| --- | --- |
| Kävellen | 0 |
| Pyöräillen | 5 |
| Käytin joukkoliikennettä | 10 |
| Kuljin kimppakyydillä | 15 |
| Käytin yksityisautoa | 30 |

1. Koodaa ohjelma, joka etsii kahden ei-negatiivisen kokonaisluvun suurimman yhteisen tekijän (Euklideen algoritmi):  
   Luvut m ja n.  
   Jos n = 0, niin tulos on s.y.t. = m.   
   Jos r on jakolaskun m/n jakojäännös, aseta m = n, n = r, ja palaa kohtaan edelliseen kohtaan.
2. Koodaa ohjelma, joka voisi olla osa gps-paikantimen ohjelmistoa. Ohjelma laskee kuljetun matkan alkaen lähtökoordinaateista ja kulkien annettujen koordinaattien kautta päätepisteeseen. Kahden pisteen etäisyys lasketaan Pythacoraan lauseella.
3. Tee ohjelma, joka etsii kokeilemalla funktion f(x) = e-2x+2x2 pienimmän arvon.
4. Tee ohjelma, joka jakaa luvun tekijöihin.
5. Kehitä ventti-peliä lähemmäksi todellisia sääntöjä.

## Siirtorakenteet

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

### Keskeytys break

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

for (int i = 0; i < 10; i++)

{

if (i == 5) break; // Toisto päättyy, kun i saavuttaa arvon 5

Console.WriteLine(i);

}

Sama while-rakenteella.

int i = 0;

while (i < 10)

{

if (i == 5) break; // Toisto päättyy, kun i saavuttaa arvon 5

Console.WriteLine(i);

i++;

}

Sama do-while rakenteella.

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.

### Metodin keskeytys return

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

### Väliinjättö continue

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

for (int i = 0; i < 10; i++)

{

if (i == 5) continue; // Toistossa tulostetaa muut paitsi i:n arvo 5.

Console.WriteLine(i);

}

Sama while-lauseella

int i = 0;

while (i < 10)

{

if (i == 5)

{

i++;

continue; // Toistossa tulostetaa muut paitsi i:n arvo 5.

}

Console.WriteLine(i);

i++;

}

Sama do-while lauseella

int i = 0;

do

{

if (i == 5)

{

i++;

continue; //Toistossa tulostetaa muut paitsi i:n arvo 5.

}

Console.WriteLine(i);

i++;

} while (i < 10);

### 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");

}

### Harjoituksia siirtorakenteista

1. Ohjelmoi tehtävä 45 käyttäen **break**-lausetta.
2. Tee kertotaulun opetusohjelmasta sellainen versio, että virheellisen vastauksen jälkeen kone kysyy saman laskutoimituksen uudestaan. Kokeile ratkaisussa **goto**-lauseen käyttöä.
3. Tee ohjelma, joka arpoo lukuja väliltä 0 – 1000 käyttäjän haluaman määrän. Ohjelma tulostaa sellaiset luvut, jotka sijoittuvat välille 100 – 150. Käytä ratkaisussa **continue**-lausetta.
4. Ohjelma kyselee ihmisten mitattua lämpöä, kunnes annetaan arvo 0. Jos lämpöä on vähemmän kuin 37, hänet luokitellaan terveeksi. Jos kuume on välillä 37 – 38, hänelle annetaan yksi burana. Jos kuumetta on yli 38, annetaan kaksi buranaa. Lopuksi ohjelma tulostaa potilaiden yhteismäärän, terveiden lukumäärän ja sairaiden lukumäärän sekä tarvittujen buranoiden määrän.

# METODIT

## 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. Meidän 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. Käyttämämme julkiset metodit WriteLine ja ReadLine ovat ohjelmakirjastossa määritelty määreellä public. Jos julkisuusmäärettä ei ole, on metodi oletusarvoisesti private.

Seuraava määre määrittelee, onko metodi luokan metodi vai olion metodi. Aluksi ohjelmoimme 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.

class Jotain

{

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 mitään, siksi ohjelma ei tee mitään. Main-metodiin on lisättävä tervehdi-metodin kutsu.

Esimerkki 2.

class Program

{

static void Main(string[] args)

{

Tervehdi();

}

static void Tervehdi()

{

Console.WriteLine("Hello there!");

}

}

Nyt Main-metodissa kutsutaan tervehdi-metodia. **Metodia kutsutaan metodin nimellä** ja sulkuihin kirjoitetaan välitettävät parametrit. 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.

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.

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.

## Parametrit

Lähes kaikissa metodeissa tarvitaan parametreja. Parametreilla ohjataan metodin suoritusta toimimaan juuri siinä tilanteessa halutulla tavalla. Olemme kirjoittaneet jokaiseen WriteLine-metodin kutsuun jonkin parametrin eli sen merkkijonon, jonka olemme halunneet metodin siinä tilanteessa 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.

### 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.

class Program

{

static void Main(string[] args)

{

Tervehdi("Maija");

}

static void Tervehdi(string kuka)

{

Console.WriteLine("Hello there {0}!", kuka);

}

}

Esimerkissä on tervehdi-metodia muutettu siten, että parametreihin on merkitty yksi string-tyyppinen parametri, jolle on keksitty 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-metodissa varataan tila arvoparametria kuka varten. Kun metodia kutsutaan, kutsun mukana oleva 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.

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 arvon muuttujan arvon kopiointi metodin nimiavaruudesta toisen metodin nimiavaruuteen. Kopioitavan muuttujan ja sen muuttujan, johon kopioidaan voivat olla samannimisiä tai erinimisiä.

Jokainen metodi muodostaa oman nimiavaruuden ja niiden toimintaa voidaan kuvata seuraavasti:

Main-metodi

tervehdi-metodi

kuka:

Kaaleppi

Kaaleppi

käyttäjä:

Esimerkki 3.

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.

Main-metodi

KahtaSuurempi

luku

6

luku

6

8

Esimerkissämme 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 ’lähtee kutsun mukana metodin KahtaSuurempi tontille’. 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.

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 niin **int** jne.

Esimerkki 5.

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, se ei käänny ennen kuin return-lause/lauseet on metodiin ohjelmoitu.

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);

tilalla voisi olla

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

Metodiin voidaan tähän tapaan ohjelmoida vain yhden arvon palautus.

### 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.

class Program

{

static void Main(string[] args)

{

int luku = 6; // alkuarvon asetus viiteparametri-muuttujalle

KahtaSuurempi(ref luku); // kutsussa avainsana ref

Console.WriteLine(luku); // tulostuu 8

}

static void KahtaSuurempi(ref int luku) // viittausparametri

{

luku += 2;

Console.WriteLine(luku); // tulostuu 8

}

}

Main-metodi

KahtaSuurempi

luku

6 8

Viitataan suoraan Main-metodin muuttujaan

### 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 se on mainittava sekä kutsussa että metodin parametrien määrittelyssä.

class Program

{

static void Main(string[] args)

{

int i;

KahtaSuurempiOut(out i); //paramerilla ei tarvitse olla alkuarvoa

Console.WriteLine(i); // tulostaa 8

}

static void KahtaSuurempiOut(out int luku) // i:n alias-nimenä

// tällä puolella on luku

{

luku = 6; // alkuarvon asettaminen pakollinen kutsutussa metodissa

luku += 2;

Console.WriteLine(luku); // tulostaa 8

}

}

Main-metodi

KahtaSuurempiOut

i

6 8

Viitataan suoraan Main-metodin muuttujaan

luku

### Vaihtuvamittainen parametrilista

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

class Program

{

static void Main(string[] args)

{

VaihtuvaMaaraParametreja(5, 6, 7,8); // kutsun parametrien lukumäärä on

VaihtuvaMaaraParametreja(1); // vapaasti valittavissa

}

private static void VaihtuvaMaaraParametreja(params int[] taulukko)

{

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

Console.WriteLine(taulukko[i]);

}

# 

Palataan tähän taulukoiden käsittelyn jälkeen.

### Harjoituksia metodeista

1. Ohjelmoi metodi TulostaKahtena( ). Metodi kysyy käyttäjältä luvun ja tulostaa sen kaksinkertaisena.
2. Ohjelmoi metodi TulostaKatenoituna( ). Metodi kysyy käyttäjältä palan tekstiä ja tulostaa tekstin yhdellä rivillä itseensä katenoituna.
3. Ohjelmoi metodi TulostaKolmena(int luku). Metodi tulostaa parametrin kolminkertaisena.
4. Ohjelmoi metodi Virheilmoitus(string virhe). Metodi tulostaa tekstin:  
   Ohjelma on suorittanut virheen ---tähän parametrina tullut virhe---- ja se lopetetaan.  
   **Vihje:** Pääohjelmassa kysytään tulostettava virheteksti. Virheteksti välitetään kutsun mukana metodille Virheilmoitus, joka tulostaa sen annetun tekstin kanssa.
5. Ohjelmoi metodi SeiskallaJaollinen, joka tulostaa, onko parametrina saatu luku jaollinen seiskalla vai ei.
6. Ohjelmoi metodi PalautaKahtena( ). Metodi kysyy käyttäjältä luvun ja palauttaa sen kutsujalle kaksinkertaisena.
7. Ohjelmoi metodi Triplaa(int luku). Metodi palauttaa parametrin kolminkertaisena.
8. Ohjelmoi metodi Maksimi(double eka,double toka). Metodi palauttaa parametreistä ei-pienemmän.
9. Ohjelmoi metodi EkaSuurempi(double eka,double toka). Metodi palauttaa arvonaan true, jos eka on suurempi, muuten se palauttaa arvon false.
10. Ohjelmoi metodi Valuutta, joka muuntaa parametrina saadun valuutta summan toiseen valuuttaan parametrina saadun kurssin perusteella. Metodi palauttaa uuden summan.
11. Ohjelmoi metodi, joka tulostaa parametrina saamansa merkin parillisille riveille ja toisen parametrina saamansa merkin parittomille riveille. Rivien määrä ja pituus tulevat myös parametreina.
12. Ohjelmoi metodi, joka saa parametrinaan kokonaisluvun. Metodin tehtävä on aloittaa summaus nollasta ja edetä annettuun kokonaislukuun. Metodi tulostaa summan muodostumista.  
    0 + 1 = 1  
    1 + 2 = 3  
    3 + 3 = 6 jne..
13. Ohjelmoi metodi KorkoaKorolle(double saldo,double korko,int vuodet), joka laskee ja palauttaa uuden saldon. Uusi saldo lasketaan kaavalla saldo \*(1 + korko/100)vuodet . Käytä potenssiinkorotukseen Math-luokan metodia Pow.
14. Ohjelmoi metodi, joka saa parametrinaan tuotteen hinnan ja alv-prosentin. Metodi palauttaa pääohjelmalle alv:n sisältävän hinnan. Pääohjelma tulostaa nettohinnan, alv:n osuuden ja verollisen hinnan.
15. Tee ohjelma, joka osaa muuntaa Celsius-asteet Fahrenheit-asteiksi ja päinvastoin. Pääohjelma kirjoittaa valikon, josta toiminto käynnistyy:  
      
    Lämpöasteiden muunnos  
    1 = Celsiukset Fahrenheiteiksi  
    2 = Fahrenheitit Celsiuksiksi  
    0 = Lopetus  
      
    Jos käyttäjä vastaa 1, ohjelma kutsuu CtoF( )-metodia ja vastaus 2 käynnistää FtoC()-metodin. 0 lopettaa ohjelman toiminnan, muut vastaukset jätetään huomiotta.  
    Metodeissa kysytään muunnettavat asteet ja tulostetaan vastaukset.  
    C = 5 \* ( F – 32 ) / 9   
    F = C \* 1.8 + 32
16. Ohjelmoi metodi, jolla voi tarkistaa mitatun kuumearvon järkevyyden. Kuumeen määrä annetaan parametrina. Metodi palauttaa true tai false.
17. Ohjelmoi metodi, joka palauttaa kuukauden numeroa vastaavan tekstin. Esimerkiksi 3 => maaliskuu.
18. Tee karkausvuosi-ohjelmasta versio, jossa karkausvuoden olemassaolo tsekataan metodilla Karkausvuosi(int vuosi). Metodi palauttaa arvonaan true, jos parametrina saatu vuosi on karkausvuosi.
19. Ohjelmoi metodi Kertoma(int luku), joka palauttaa parametrin kertoman.  
    esim. Viiden kertoma on 1 \* 2 \* 3 \* 4\* 5  
    merkintä 5!
20. Ohjelmoi metodi, joka tarkistaa, onko sen parametrina saama päivämäärä ollut olemassa. Parametrit ovat kolme kokonaislukua pv,kk ja vuosi.
21. Tee ohjelma, jolla saa määriteltyä kokeeseen osallistuneiden opiskelijoiden arvosanat. Yksittäisen opiskelijan arvosanan määritys tapahtuu metodilla. Lopuksi tulostetaan arvosanojen ja pisteiden keskiarvot.
22. Ohjelmoi metodi, joka vaihtaa viittausparametreina saamiensa muuttujien sisällöt keskenään.
23. Ohjelmoi metodi, joka laskee kahden luvun summan, erotuksen, tulon ja osamäärän ja palauttaa tulokset ulostuloparametrien avulla.

**Lisätehtäviä**

1. Amerikkalaiset ilmaisevat henkilön pituuden yleensä kertoen ensin henkilön pituuden jaloissa ja jäljelle jäävän pituuden tuumissa. Esim. 173 cm on amerikkalaisille 5’8”. Heittomerkki ’ on jalan lyhenne ja lainausmerkki ” on tuuman lyhenne. 173 cm on noin 68 tuumaa. Yksi tuuma on 2,54 senttimetriä ja yhdessä jalassa on 12 tuumaa. Toisin sanoen 68 tuumassa on 5 kokonaista jalkaa ja jakojäännöksesi jää 8 tuumaa. Ohjelmoi metodi, joka palauttaa argumenttina annetun cm-pituuden muodossa: jalat’tuumat” esim. 155 cm = 5’1”. Pääohjelma kysyy käyttäjältä henkilön pituutta senttimetreissä ja näyttää ”amerikkalaiseen” tyyliin muunnetun tulokset ohjelmoimaasi metodia käyttäen. Käyttäjä päättää, koska ohjelman suorittaminen lopetetaan.
2. Ohjelmoi metodi, jolla voi laskea merkkijonon numeroiden ja kirjaimien määrän. Merkkijono tulee parametrina ja numeroiden ja kirjaimien määrä palautetaan out-parametreilla. Vihje Merkin saa ’irti’ merkkijonosta: merkkijono[i], missä i on merkin järjestysnumero (0 aloittaa) ja tarkistuksen voi tehdä Char-luokan metodilla isDigit ja isLetter. Merkkijonon pituus selviää metodilla merkkijono.Length.
3. Gaussin algoritmilla voidaan laskea minä päivänä ja missä kuussa pääsiäislauantai on vuosivälillä 1900 – 2099. Päättelyketju on seuraava:  
   a = vuosi % 19 (% = jakojäännös laskutoimitus)  
   b = vuosi % 4  
   c = vuosi % 7  
   d = ( 19a + 24 ) % 30  
   e = (2b + 4c + 6d + 5) % 7  
   jos d = 29 ja e = 6 niin pääsiäissunnuntai on 19.4  
   muutoin, jos d = 28 ja e = 6 niin pääsiäissunnuntai on 18.4  
   muutoin, jos 22 + d + 3 <= 31 niin pääsiäissunnuntai on 22 + d + e päivänä maaliskuuta  
   muutoin pääsiäissunnuntai on d + e – 9 päivänä huhtikuuta  
     
   Ohjelmoi ohjelma, jolla voi selvittää edellä mainitulta vuosiväliltä pääsiäissunnuntain päivänmäärän. Ohjelma lopettaa, kun annetaan vuosiluku 0.  
   Päivämäärän laskee metodi Pääsiäinen(int vuosi,out int pv,out int kk).
4. Tee edelliseen ohjelmaan metodi Pitkäperjantai(ref int pv,ref int kk), joka pääsiäispäivän perusteella määrittää pitkänperjantain päivänmäärän.

# TAULUKOT

## Miksi taulukoita tarvitaan

Hyvin usein on tilanne, jolloin ei etukäteen voi päätellä, kuinka monta muuttujaa tarvittaisiin 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 toimintapuoleen (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 asioita ei voi ratkaista.

Ongelmaan on keksitty ratkaisuksi taulukko, joka on kokoelma samantyyppisiä 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.

Määritellään taulukko nimeltään luvut, johon voi tallettaa kokonaislukuja. Taulukon koko on kymmenen 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 indeksointiä hyväksikäyttäen.

Esimerkki 1.

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

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 hyväksi käyttäen solu kerrallaan. Ohjelmoijan on huolehdittava, ettei ohjelma viittaa sellaiseen soluun, jota ei ole olemassa. Sellaisessa tapauksessa ohjelma kaatuu.

Jos taulukosta on etsittävä tiettyä sisältöä, on taulukko käytävä läpi solu kerrallaan kunnes etsittävä löytyy tai solut loppuvat. Jos taulukko on järjestetty, etsintä voidaan suorittaa useammallakin etsintämenetelmällä (esimerkiksi katsotaan keskimmäisen solun sisältö, jos etsittävä on sitä pienempi katsotaan alkupään keskimmäinen jne.).

Esimerkki 2.

// nimen etsintä taulukosta

Console.Write("Mitä nimeä etsitään? ");

string nimi = Console.ReadLine();

int indeksi = 0;

bool löytyi = false;

for (int i = 0; i < 50; i++)

{

if (nimet[i] == nimi)

{

indeksi = i;

löytyi = true;

break;

}

}

if (löytyi)

Console.WriteLine("Nimi löytyi solusta " + indeksi);

else

Console.WriteLine("Nimeä ei löytynyt!");

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

int koko = nimet.Length;

Läpikäyntiin tarvittava for-lause voidaan kirjoittaa Length-ominaisuuteen perustuen:

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

Edelleen taulukon viimeinen indeksi on numero 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)

{

vuorossaolevan muuttujan käsittely muuttujanNimi perusteella;

}

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

Esimerkki 3.

// nimen etsintä taulukosta

Console.Write("Mitä nimeä etsitään? ");

string nimi = Console.ReadLine();

int indeksi = 0;

bool löytyi = false;

foreach (string taulukonNimi in nimet)

{

if (taulukonNimi == nimi)

{

löytyi = true;

break;

}

indeksi++; // pidetään yllä laskuria, missä kohtaa ollaan menossa

}

if (löytyi)

Console.WriteLine("Nimi löytyi solusta " + indeksi);

else

Console.WriteLine("Nimeä ei löytynyt!");

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

Esimerkki 4.

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.

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 5.

Moniulotteinen taulukko voidaan läpikäydä helposti 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);

}

Contains-metodilla voi tutkia sisältääkö taulukko parametrina annetun tiedon.

string[] kirjaimet = { "aa", "bb", "cc" };

if (kirjaimet.Contains("aa"))

Console.WriteLine("Tieto löytyy taulukosta");

## Taulukkoharjoituksia

1. Kirjoita lauseet:
   1. Määrittele kokonaislukutaulukko, jossa on 27 solua.
   2. Määrittele boolean taulukko, jossa on neljä solua ja kaikkien solujen arvona on true.
   3. Määrittele taulukko, johon kysytään ihmisten nimiä. On varauduttava 100 nimeen.
   4. Kirjoita lauseet, joilla tulostat taulukon hölkynKölkyn sisällön. Rivi per solu.
   5. Kirjoita lauseet, joilla tulostat taulukon hölkynKölkyn ekan solun sisällön.
   6. Kirjoita lauseet, joilla tulostat taulukon hölkynKölkyn viimeisen solun sisällön.
   7. Määrittele double-tyyppinen taulukko, joka on samankokoinen kuin taulukko luvut.
2. Ohjelmoi ohjelma, johon kovakoodataan muutaman solun kokonaislukutaulukko sisältöineen.
   1. Tee metodi, jolla voi laskea taulukon solujen summan, metodi palauttaa summan.
   2. Tee metodi, joka tulostaa kaikki parilliset luvut.
   3. Tee metodi, joka etsii taulukon pienimmän luvun ja tulostaa sen.
   4. Tee metodi, joka tulostaa taulukon suurimman luvun indeksin.
   5. Tee metodi, jolla taulukosta voi etsiä parametrina annettua kokonaislukua. Metodi palauttaa true tai false riippuen etsinnän tuloksesta.
   6. Tee metodi, joka laskee ja palauttaa tiedon, kuinka monta kertaa parametrina annettu luku esiintyy taulukossa.
   7. Tee metodi, joka tutkii ja palauttaa tiedon siitä, missä solussa parametrina annettu luku esiintyy ensimmäisen kerran.
3. Ohjelmoi ohjelma, jolla voi luoda kokonaislukutaulukon, joka on käyttäjän haluaman kokoinen ja jonka solujen sisältönä on aina indeksin arvo kaksinkertaisena.
   1. Tee ohjelmaan metodi, joka laskee ja palauttaa joka toisen alkion summan.
   2. Tee metodi, joka kopioi alkuperäisen taulukon sisällön toiseen taulukkoon ja tulostaa uuden taulukon sisällön näytölle.
   3. Muokkaa b-kohdan metodia siten, että taulukon solujen sisältöä pyöräytetään ennen tulostusta. Ensimmäisen solun sisältö siirtyy toisen solun sisällön paikalle, toisen solun sisältö kolmannen paikalle jne ja viimeisen solun sisältö siirtyy ensimmäisen solun paikalle.
4. Ohjelmoi ohjelma, jonka sisältönä on seuraavat arvot: 1, 11, 101, 1001, 10001, 100001.  
   Ohjelma pyytää käyttäjältä luvun ja ohjelma tulostaa indeksin, josta löytyy lähin annettua lukua suurempi luku. Ohjelman on sisällettävä toisto-toiminto.
5. Ohjelmoi ohjelma, joka sisältää kuukausien nimet taulukossa. Ohjelma kyselee kuukausien numeroita ja tulostaa kuukauden tekstinä (0 lopettaa).
6. Ohjelmoi ohjelma, joka tulostaa lottorivin. Viilaa ohjelmaa siten, että riviin laitettavien numeroiden määrää pystyy säätämään. Edelleen viilaa ohjelmaa siten, että sillä voi samalla käynnistyksellä tulostaa halutun määrän rivejä.
7. Tee ohjelma, jossa määritellään tuotekoodit ja niitä vastaavat tuotenimet. Ohjelma sisältää metodin, joka palauttaa parametrina annettua tuotekoodia vastaavan nimen. Voit ratkaista ohjelman joko käyttäen kahta yksiulotteista taulukkoa tai yhtä kaksiulotteista taulukkoa.
8. Muodosta taulukko ArvosanaRajat, johon kyselet pistemäärät, jotka vähintään on oltava, jotta saa ykkösen, kakkosen,…, vitosen.   
   Seuraavaksi ohjelma kysyy pistemääriä ja tulostaa pistemäärää vastaavan arvosanan. Arvosan selvitetään metodin avulla. Negatiivinen pistemäärä lopettaa.
9. Lisää edelliseen ohjelmaan taulukko arvosanat, johon lasket montako nollaa, ykköstä, kakkosta jne kokeesta tuli. Ohjelma tulostaa lopuksi tilaston.
10. Lisää edelliseen ohjelmaan metodi, jolla saat tulostettua kokeen pistemääräkeskiarvona ja arvosanakeskiarvon.
11. Ohjelman taulukossa nimet on ihmisten nimiä. Ohjelmoi metodi, joilla etsit taulukosta nimen, joka on aakkosissa ensimmäisenä.

# MERKIT JA MERKKIJONOT

## Merkit

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 (28) erilaista merkkiä. Näin ollen kansalliset erikoismerkit koodataan milloin mitenkin ja esimerkiksi ääkkösten 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 ääkköset 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 samuuta 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-tyyppiselle tiedolle on olemassa valmiina useita sen käsittelyyn sopivia metodeja.

char merkki;

Console.Write("Anna merkki! ");

merkki = char.Parse(Console.ReadLine());

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 merkkiä voidaan muokata

char uusiMerkki = char.ToLower(merkki); // merkki muutetaan pieneksi kirjaimeksi

char uusiMerkki = char.ToUpper(merkki); // merkki muutetaan isoksi kirjaimeksi

string merkkijono = merkki.ToString(); // merkki muutetaan merkkijonoksi

## 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.

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

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 samansisältöisiä");

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 irroittaa 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 hölynpölyä = "ff" + "Kiva" + "99" + " Jihuu";

string lisää = "!";

string lisääHölynpölyä = hölynpölyä+lisää;

## Harjoituksia merkeistä ja merkkijonoista

1. Tee ohjelma, joka kysyy kaksi merkkiä.
   1. Lisää ohjelmaan metodi, joka tulostaa merkit suuruusjärjestyksessä, pienin ensin.
   2. Lisää metodi, joka tutkii, onko annetut merkit kirjaimia ja jos ovat, ovatko ne vokaaleja vai konsonantteja.
2. Tee ohjelma, joka kysyy kaksi merkkijonoa.
   1. **Lisää metodi, joka tulostaa merkkijonot aakkosjärjestyksessä.**
   2. **Lisää metodi, joka tulostaa parametrina saadun merkkijonon merkki kerrallaan.**
   3. **Lisää metodi, joka tulostaa parametrina saadusta merkkijonosta ensimmäisen ja viimeisen merkin.**
   4. Lisää metodi, joka tulostaa parametrina saadun merkkijonon kolmen merkin mittaisina paloina siten, että ensimmäinen merkki siirtyy aina pykälällä eteenpäin.  
      Esim. abcdef => abc, bcd, def
   5. **Lisää metodi, joka tutkii löytyykö parametrina saadusta merkkijonosta, parametrina annettua merkkiä. Metodi palauttaa true tai false.**
   6. Lisää metodi, joka tutkii löytyykö parametrina saadusta merkkijonosta, parametrina annettua merkkijonoa. Metodi palauttaa true tai false.
   7. **Lisää metodi, joka laskee, montako tyhjä-merkkiä parametrina saadussa merkkijonossa on. Metodi tulostaa myös, mikä on tyhjien merkkien prosenttiosuus.**

Muokkaa edellistä harjoitusta siten, että teksti luetaan tiedostosta

* 1. **Lisää metodi, joka tulostaa parametrina saadun merkkijonon nurinpäin**.
  2. **Lisää metodi, joka tutkii, onko parametrina saatu merkkijono palindromi.**

1. **Tee ohjelma, joka kysyy osoitteen muodossa Postinumero Postitoimipaikka Katuosoite. Ohjelma pätkii tiedon siten, että se tulostaa:  
   Postinumero  
   Postitoimipaikka  
   Katuosoite**
2. Tee ohjelma, joka laskee annetusta tekstinpätkästä erilaisten kirjaimien määrän.  
   TEKSTI ON TÄSSÄ => T E K S I O N Ä  
    3 1 1 3 1 1 1 2
3. Tee ohjelma, joka muuttaa sukunimen nimikentän lopusta alkuun.   
   Esim. Maija Meikäläinen => Meikäläinen Maija
4. Tee ohjelma, joka osaa vaihtaa annetusta merkkijonosta skandit pois. ä =ae, ö = oe.
5. Tee ohjelma, joka tulostaa annetusta tekstinpätkästä erilaisten kirjaimien jakauman (eri kirjainten prosenttiosuudet) (ks. teht. 105)