Paavo Räisänen

C# Perusopas

www.ohjelmoimaan.net

Sisällysluettelo

1 Alkuun

1.1: Alkusanat

Miksi valita ohjelmointikieleksi C# (äännetään C Sharp. Lyhennetään joskus myös CS. Huomaa, että eri asia kuin tyylikieli CSS.)? C# on ensimmäiseksi ohjelmointikieleksi yhtä hyvä kieli, kuin joku muukin. Ei parempi, eikä huonompi. C# on lähinnä Javan kilpailija, joka on yleisinpiä ohjelmointikieliä erilaisilla korkeakoulutason ohjelmoinnin alkeiskursseilla. Myös C# :ia käytetään ohjelmoinnin alkeis- ja jatkokielenä myös korkeakoulutasolla. Edelleen perus C tai Basicit ovat helpoimpia kieliä ohjelmoinnin opetteluun. Basicin opettelulla ei vain ole myöhemmin ohjelmoinnissa suurtakaan hyötyä, poislukien tästä tosin Visual Basic. Jos olet nyt kirjoittamassa ihka ensimmäista ohjelmaa, joku QBasic on kyllä kieltämättä helppo valinta opetella ihan ohjelmoinnin ensiaskeleet. Kuitenkin, kunhan ihan ensimmäiset ohjelmat saa tehtyä, ohjelmointi sujuu ihan samoin C # :kin. Koneenläheisessä ohjelmoinnissa C yleisin kieli. C# on kehitetty Javasta ja C++ :sta. C# on ohjelmoinnin opiskeluun hyvin käytännöllinen kieli sen jälkeen kun alkukankeuksista pääsee. Ihan kuten Javalla ja C++:lla, C#:lla pystyy lähes kaikkeen, mitä tavallinen ohjelmoija tarvitsee. Lisäksi kun osaa yhtä ohjelmointikieltä, muiden opiskelu on helppoa. Ongelmat ratkaistaan pitkälti samalla tavalla, joskin eri syntaksilla, ja erilaisilla erikoistoiminnoilla. C#:lla pystyy tekemään myös Windows mobiiliohjelmia. C#:lla pystyy tekemään sekä konsoli-, Windows että nettiohjelmia. Sillä pystyy palvelinohjelmointiin ASP.NET ohjelmointina, ja se toimii tietokantojen, kuten MySQL ja MSSQL kanssa. C# ASP.NET toimii hyvin yhteen scriptikielien, kuten ActionScript (Flash) ja JavaScript (Ajax) kanssa. Erikoisuutena pelipuolella on, että Microsoftin XNA peliohjelmointi scriptikirjasto käyttää C#:ia ja samoin Unity 3D pelimoottorin nykyisin ainoa sallittu ohjelmointikieli on C#. C# on myös yksi Microsoft Silverlightin sallimista kielistä. Silverlight on Microsoftin vastine Flashille. C# on tällä hetkellä eniten käytetty .NET alustainen kieli. Muita .NET alustalla toimivia kieliä ovat mm. Visual Basic .NET, Visual C++ ja Visual J#. Visual J#:n avulla voidaan Javaa käyttää .NET alustalla. Javan tapaan C# on ns. tavukoodikieli, joka ajetaan virtuaalikoneessa, joka mahdollistaa ohjelmien toimimisen erilaisissa käyttöympäristöissä. Javan tapaan myös C# on olio-ohjelmointikieli.

Mainittakoon, että jos tässä kehun C# :ia, Java ja PHP ovat kieliä, joita eritoten internet ohjelmoinnissa ei voi ohittaa, eikä ole tarpeenkaan. Internet ohjelmointi on oma maailmansa, joka on opeteltava erikseen, vaikka esim. laskutoimitukset, tekoäly jne. toimivatkin samalla lailla. C ja C++ ovat taasen eritoten joissakin ohjelmoinnin osa-alueissa, kuten koneenläheisessä- tai käyttöjärjestelmäohjelmoinnissa lähes standardeja. Kuitenkin C# pystyy jokseenkin kaikkeen samaan. Eritoten peliohjelmointipuolella C Sharppia käytetään paljon, samoin Windows ohjelmoinnissa. Internet puolella C# käytössä on ongelma, että se toimii lähinnä Windows palvelimilla (IIS), jotka ovat kalliinpia, kuin yleisimmin käytetyt Linux palvelimet (Apache). Samoin, vaikka C# ASP.NET ohjelmoinnissa voi käyttää halpaa

(ja toimivaa) MySQL tietokantaa, siitä on vaikea saada tietoa. Yleensä ASP.NET ohjelmoinnissa käytetään Microsoftin MSSQL tietokantaa, joka on kallis.

Tämä opas pyrkii lähtemään hyvin alkeista. Sinulla on kuitenkin hyötyä, jos osaat jo hieman jotain ohjelmointikieltä. Jos olet kokeneempi ohjelmoija, ja haluat opetella uuden kielen, voit kenties sivuuttaa tämän oppaan alun hyvin nopeasti.

1.2: Miten tietokone ajattelee?

Tietokone ei ajattele täysin samalla tavalla kuin ihminen. Kuitenkin, tietokone ei ajattele täysin eri tavalla kuin ihminen. Tietokoneen ohjelmointi vaatii vain analyyttistä ajattelutapaa. Tieokone ei ymmärrä asiakokonaisuuksia, vaan sille on kaikki esiteltävä täysin yksityiskohtaisesti. Näin on erityisesti konekielessä, jolla tietokone vime kädessä ajattelee. Konekielessä esim. laskutoimitus 4*6 tapahtuu näin:
Esimerkki 1:
1: laskuri=3
2: lataa muuttujaan 6
3: silmukka: lisää muuttujaan 6
4: vähennetään laskuria yhdellä ja palataan silmukkaan laskuri kertaa

Koska rivillä kaksi alustetaan jo valmiiksi muuttujaan kuusi, laskuriin laitetaan muuttuja-1, ja näin, kun muuttujaan lisätään kolme kertaa arvo kuusi, muuttuja saa lopputulokseksi 24.

Korkeamman tason kielissä on toimintoja helpotettu tekemällä monimutkaisenpia käskyjä. Voit siis suoraan kirjoittaa muuttuja=4*6. Itse tietokone kuitenkin suorittaa ohjelman aina konekielisesti. Erilaiset tietokoneen käskyt ovat näin oikeastaa konekielisten ohjelmien aliohjelmakutsuja. Tavallaan ohjelmoija myös itse tekee ”käskyjä”, aliohjelmakutsuja ,tehdessään aliohjelmia (funktioita) ja olio-ohjelmointikielissä (kuten C#), tehdessään luokkia ja olioita.

Ihminen pystyy helposti käsittelemän suuria kokonaisuuksia, opittuaan ne ensin. Kuitenkin myös tietokonetta voi opettaa. Kun ohjelmoija tekee valmiin olion, esim. olion, joka laskee päivämäärän tarkistuksen (onko karkausvuosi, onko kuukaudet 1-12, onko päivämäärä annetussa kuukaudessa mahdollinen, jne.), hän voi käyttää tätä oliota muissakin ohjelmissaan. Näin ohjelmoija on ikäänkuin luonut oman käskyn. Hän voi nyt liittää tämän olion, ”käskyn”, myöhempiin ohjelmiin, ja käyttää sitä syöttämällä vain ”käskylle” haluamansa tiedot.

Esimerkki 2: kuinka kertoa tietokoneelle, kuinka auto rakennetaan.

1: Luodaan luokka auto.
2: Määritellään eri metodeissa esim. seuraavat toiminnot

2.1: Tee kori
2.2: Tee moottori
2.3: Tee renkaat
2.4: jne
3: Asiat on vielä pikottava pienemmiksi, jotta tietokone ymmärtää sen. Niinpä tietokoneelle on yksinkertaistettava eri osaalueiden toteuttaminen, eli:
3.1: Tehdään moottori
3.1.1: Tee itse moottori
3.1.2: Lisää lisälaitteet
3.1.2.1: Lisää laturi
3.1.2.2: Lisää sytytysjärjestelmä
3.1.2.3: Lisää jäähdytysjärjestelmä
4: Tietokone ei kuitenkaan osaa tehdä esim. laturia ilman tarkenpia neuvoja. Sille on vielä kerrottava, mitä osia laturi sisältää. Sitten on vielä kerrrottava, miten osat valmistetaan jne.
Kun olet kerran tehnyt tämän, voit periaatteessa rakentaa samarakenteisen auton antamalla vain tiedot oliolle. Eli kaikista osista tehdään muuttujia. On esim seuraavat:

Esimerkki 2:
Moottori:
1: Laturin tyyppi
2: Sytytysjärjestelmä
3: Jäähdytysjärjestelmä
4: Moottorin tyyppi

Jos on tarpeen, eli osat eivät ole noin yksinkertaisissa ”paketeissa”, vaan kuten kuten esimerkissä kaksi, on tietokoneelle syötettävä tiedot kaikkia lisäosien osia myöten. Tietokone on kuitenkin taitava toistamaan ja myös muistamaan. Kun sille kerran on kerrottu joku rakenne, se osaa vaihtuvilla tiedoilla helposti tehdä samanlaisen. Uuden auton rakentaminen ei tosin ole näin yksinkertainen. Auto toki voidaan koota tarvikeosista, kuten esimerkissä kolme, mutta jos vaihdetaan moottorin tyyppiä, joudutaan todennäköisesti vaihtamaan myös tarvikeosien tyyppi. Tietokoneelle voisikin opettaa, mitä lisäosia kukin moottori käyttää. Sensijaan siinä tulee vastaan tietokoneen rajoittuneisuus, että jos esim. laturiin vaihdetaan erilainen ruuvi, luultavasti koko laturi on suunniteltava uudelleen. Tällainen suunnittelu ei ole tietokoneelle helpoa opettaa, vaan yleensä tehtäisiin koko laturi aliohjelma uudelleen.

Kuten huomaat, tietokone ei ole kovin älykäs. Siltä puuttuu paljolti ihmisen suunnittelutaito ja inovatiivisuus. Se on kyllä taitava laskemaan, toistamaan ja muistamaan. Tietokonetta pyritään kehittämään älykkäämmäksi erilaisilla tekoälyohjelmilla, joilla jotain saavutuksia on saatu esim. robottitekniikassa.

1.3: Johdatus konekieleen

Tämän ymmärtäminen ei ohjelmoinnin kannalta ole välttämätöntä. Tämä kuitenkin selventää, kuinka tietokone ajattelee, ja on ikäänkuin jatkoa esimerkille yksi. Tietokone ymmärtää viime kädessä vain nollia ja ykkösiä, eli binäärilukuja, kuten 11001011. Käytännössä konekieliohjelmoijat korvaavat nämä binääriluvut symbolisella konekielellä, eli Assemblyllä. Yhtä binäärilukua vastaa jokin kirjainyhdistelmä, eli esim. add, inc, jr, djnz, jne, jotka tietokone muuttaa binääriluvuksi. On mahdollista muuttaa nämä lyhenteet taulukon avulla (tai ulkomuistista, jos on hyvä muisti) suoraan binääriluvuiksi, ja ohjelmoida siis tietokonetta suoraan nollilla ja ykkösillä. Tietokoneen prosessori ei ymmärrä kovinkaan suurta käskykantaa. Se lähinnä osaa laskea yhteen, vähentää, tallentaa, hypätä, toistaa ja lähettää ja vastaanottaa tietoja porteista (in, out). Esimerkiksi yksinkertainen ”Terve Maailma!” -ohjelma tallentaa tietokoneen muistiin sanat ”Terve Maailma!” kirjain kerrallaan peräkkäisiin muistipaikkikkoihin (ja välilyöntikoodin väliin). Sitten tulostus tapahtuu käymällä kyseinen muistikohta muistipaikka kerrallaan läpi, ja syöttämällä kirjainkoodi (binäärilukuina muistissa) numero kerrallaan näytön tulostuksen portin läpi näytölle. Tietokone osaa sitten muuttaa binääriluvun kirjaimeksi.

Lasku 4*6 tapahtuisi konekielellä seuraavasti:

lataa akkuun 6
lataa laskuriin 3
silmukka: lisää akkuun 6
palaa silmukkaan ja vähennä laskuria

Eli Z80 assemblerilla:

LD a,8
LD b,3
silmukka: ADD a,8
DJNZ silmukka

Ja tietokoneen lopulta ymmärtämässä muodossa Z80 konekielessä:

00111110
00001000
00000110
00000011
11000110
00001000
00010000
00000000

Tässä siis esimerkiksi käskyä ”LD a ”vastaa binääriluku 00111110 ja sitä seuraa numero kahdeksan binäärimuodossa,eli 00001000.

1.4: Ensimmäinen ohjelma

Oikeastaan C#:sta, kuten Javastakin hieman vaikean opittavan ensimmäiseksi tekee ensimmäisten ohjelmien kirjoittaminen. Kun ensimmäisistä ohjelmista selviää, ohjelmointi C#:lla on aivan yhtä helppoa, kuin vaikka Basicilla. Myöhemmin laajoissa isommissa ohjelmissa C#:n edistyksellisyys tekee siitä aivan eri tavalla paremman ohjelmointikielen Basiceihin verrattuna. Oikeastaan Basicit eivät sovellu kuin ohjelmoinnin opetteluun, ja kun vähän näkee vaivaa, oppii paremmin kehittyneemmillä kielillä. Basiceista tosin erikseen mainittava aiemminkin mainittu Visual Basic, joka on ihan ammattikäytössä.
Jos ohjelmoitaisiin Basicillä, ensimmäisen ohjelman teko olisi helppoa. Pitäisi vain kirjoittaa kääntäjään:

Esimerkki 4:
1: print ”Terve Maailma!”:

ja käynnistää tulkki ”run” komennolla, niin teksti näkyisi näytöllä. Mainittakoon, että toisin kuin aikoinaan pikkukoneilla Commodore 64:lla, MSX:llä jne., tietokoneiden mukana nykyisin ei tule mitään ohjelmointikieltä. Ilmaisia ohjelmointikieliä ja ohjelmointiympäristöjä on kuitenkin saatavana vaikka kuinka paljon. Käytän tässä Microsoft Visual Studio 2010 Express ohjelmaa, jonka saa ilmaiseksi ladattua osoitteesta http://www.microsoft.com/express/Downloads/ .

Ilmaisella Novellin ”Mono” kehitysympäristöllä C# ohjelmia voi kehittää ja ajaa eri käyttöjärjestelmissä. Lisätietoa löytyy Monon pääsivulta http://www.mono-project.com/Main_Page
Sitten ensimmäinen ohjelma. Käynnistä Visual Studio. Klikkaa ”New Project” . Kirjoita avautuvassa ikkunassa alas ”Name” kohtaan projektin nimeksi ”EnsimmainenOhjelma”. Klikkaa sitten kerran ”Console Application” kohtaa ja sen jälkeen alhaalta ”OK”.

Sinulle avautuu seuraava koodi.
Kuten näet, antamasi ohjelman nimi tulee namespace kohtaan. Sitäon hyvin vaikea muuttaa myöhemmin, ja säilyttää ohjelman toimivuus.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

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

Annoin projektille nimeksi ”EnsimmainenOhjelma”, joka näkyy ”namespace” kohdassa. Huomaathan: ohjelmointikielet yleensäään eivät hyväksy ääkkösiä nimissä. Ääkköset toimivat kuitenkin tulostettaessa lauseissa. Ohjelmassa on ”using” kohta. Siihen kuuluvat luokkakirjastot. Eri käskyt vaativat toimiakseen eri luokkia, jotka on sisällytettävä ohjelman alkuun.
Kirjoita Main() lohkoon aaltosulkeiden { } väliin seuraava koodi. Aaltosulkeet löytyvät näppäimistöltä AltGr 7 ja 0. // merkintä tarkoittaa kommenttia, eli sillä ohjelma ei tee mitään. Vaihtoehtoisesti kommentti voidaan sijoittaa /* ja */ merkinnän väliin. Huomaat, että Visual Studiossa on ennustava tekstin syöttö, josta ensimmäisen tai ensimmäisten kirjaimien kirjoittamisen jälkeen on helppo valita käsky (metodi).

```
//Ensimmäinen ohjelmani
Console.WriteLine("Terve Maailma!");
Console.Write("Paina jotain näppäintä jatkaaksesi...");
Console.ReadKey(true);
```

Valitse sitten Debug/Start Debugging, ja näyttöön ilmestyy ensimmäisen ohjelmasi teksti. Huomaathan, että C#:ssa käskyjen ja muuttujien nimissä isoilla ja pienillä kirjaimilla on merkityksensä. On eri asia kirjoitatko ”Luku” vai ”luku”.

Tässä vielä ensimmäinen ohjelma ja esimerkki kommentin käytöstä. Kommentteja kannattaa käyttää selventämään ohjelmaa, jotta sinun on helpompi myöhemmin ymmärtää ja muokata koodia.

1.4.1: Ensimmäinen ohjelma

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EnsimmainenOhjelma
{
    class Program
    {
        static void Main(string[] args)
        {
          /* Tässä on esimerkki ensimmäisestä ohjelmasta
                ja kommenttilauseen käytöstä */
            Console.WriteLine("Terve Maailma!");
            Console.Write("Paina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);
        }
    }
}
```

{ ja } eli aaltosulkeet ovat helpompi ymmärtää, kun tietää, mitä ne merkitsevät Pascalissa, eli { on ”Begin” ja } on ”End”. Aaltosulkeet aloittavat ja lopettavat jonkun lohkon.

Huom. Ohjelmarivit päättyvät yleensä ; merkkiin (puolipiste). Joitain poikkeuksia on, kuten for {}, while {} tai switch {} rakenteet, jotka käsitellään myöhemmin.

Jo tässä vaiheessa on hyvä vilkaista oppaan kohtaa 8: ”Visual Studion käytöstä”.

2 Numeeriset muuttujat

Muuttujat ovat sinulle varmast tuttuja matematiikasta, jossa niitä käytetään monenlaisissa yhtälöissä, esim. y=2x+4. Aivan sama periaate muuttujilla on myös ohjelmoinnissa. Numeromuuttujien lisäksi ohjelmoinnissa voi kuitenkin olla myös merkkimuuttujia. Taulukoita käyteteään paljon. Muuttujista on runsaasti erilasia tyyppejä, jotka esittelen tässä luvussa. Muuttujista on sekä C# tyyppi, että .NET tyyppi. Kummatkin toimivat C# ohjelmoinnissa, mutta jos käytät jotain muuta .NET kieltä, .NET tyypin muuttujat ovat standardi.

Käytän tässä * merkkiä kertomerkkinä, ja ^ merkkiä potenssiin korotettuna. 10^n tarkoittaa käytännössä, että lukuun lisätään n nollaa (pilkkua siirretään n numeroa oikealle) , eli 2*10^6 on 2 000 000. Jos potenssii korotuksessa on miinus merkki, pilkkua siirretään n numeroa vasemmalle, eli
2*10^-6 on 0,000 002.

Muuttujan tietotyypin voi tutkia GetType() metodilla, josta esimerkki myöhemmin kohdassa 3.1.5.

Huomaathan, että C Sharp käyttää englannikielistä merkistöä, eli muuttujien, ja metodien (käskyjen) nimissä ei voi olla ”ääkkösiä” (ä,ö,å).

C# pitää kirjaa muuttujista ja itsekin tehdyistä metodeista. Kun olet kerran määritellyt muuttujan, ja alat kirjoittamaan muuttujaa myöhemmin käyttäessäsi, ohjelma ehdottaa määrittelemiäsi muuttujia.

2.1: Kokonaislukutyypit

2.1.1: Taulukko kokonaislukutyypit

C# tyyppi	***.NET tyyppi***	***Koko tavuina***	***Arvoalue***
sbyte	Sbyte	1	-128...127 voi olla etumerkki
byte	Byte	1	0...255 huom. ei etumerkkiä
short	Int16	2	-32.768...32.767 voi olla etumerkki
ushort	UInt16	2	0...65535 huom. ei etumerkkä
int	Int32	4	-2.147.483.647...2.147.483.647 voi olla etumerkki
uint	UInt32	4	0...4.294.967.295 huom. ei etumerkkiä
long	Int64	8	noin -9*10 ^18...9*10^18 voi olla etumerkki
ulong	UInt64	8	noin 18*10^18 huom. ei etumerkkiä

Yleisimmin näistä käytetään int -tyyppiä.

Esim. 2.1:

```
//Muuttuja määritellään näin
int a;
int x,y,z;
long k;
```

Kaikki muuttujat on esiteltävä ennen käyttöä.

Kuten huomasit, samantyyppisiä muuttujia voi esitellä peräkkäin lueteltuna pilkulla erotettuna. Usein muuttujat joutuu myös alustamaan. Jos yrität käyttää muuttujaa ohjelmassa ilman, että siihen on varmasti sijoitettu arvo, kääntäjä ilmoittaa virheestä: Use of unassigned local variable 'a'. Tässä tapauksessa muuttuja a oli jätetty alustamatta.

Esim. 2.2:

```
//Muuttujat voi esittelyn yhteydessä alusta näin
int a=0;
int x=8, y=0, z;
long k=0;
//Tässä z on jätetty alustamatta
```

2.2: Reaali- eli desimaaliluvut

2.2.1: Taulukko reaali- eli desimaaliluvut

C# tyyppi	***.NET tyyppi***	***Koko tavuina***	***Arvoale***
float	Single	4	noin +/- 1,5*10^-45...+/-3,4*10^38
double	Double	8	noin +/-5*10^-324...+/-1,7*10^308
decimal	Decimal	12	noin +/-1.0*10^-28...+/-7,9*10^28 20:llä merkittävällä numerolla

Joissain tapausissa float ja double tyyppien kanssa voi sattua pyöristysvirheitä hyvin suurissa ja tarkkuutta vaativissa laskuissa. decimal tyyppi laskee 28 luvun tarkkuudella, eikä siinä voi sattua pyöristysvirheitä.

Kun käytät float tai decimal tyypin muuttujaa täytyy luvun loppuun laittaa kirjainmerkintä seuraavan esimerkin mukaisesti. Huomaa: ohjelmoinnissa käytetään desimaalipistettä, ei pilkkua.

Esim. 2.3:

```
float h=2.16f;
//tai
float a=4.12F;

decimal b=0.1612m;
//tai
decimal k=7.18M;
```

double muuttujan loppuun ei ole pakko laittaa merkkiä, mutta voidaan laittaa d tai D. Voit myös alustaa reaalilukumuuttujan eksponentaaliesityksen avulla, eli:

Esim. 2.4:

```
float a=2.438768e+4f;  //a saa arvon 24387,68 eli sama kuin 2.438768*10^4
decimal b=14.276e-3M;  //b saa arvon 0,014276  eli sama kuin 14.276*10^-3
double c=8171.2e-2;  //c saa arvon 81,712 eli sama kuin 8171.2*10^-2
```

2.3: Aritmeettiset operattorit

2.3.1: Taulukko aritmeettiset operattorit

Operaattori	***Tarkoitus***	***Esimerkki***
+	Summa (yhteenlasku)	a+b
-	Erotus (vähennyslasku)	a-b
*	Tulo (kertolasku)	a*b
/	Osamäärä (jakolasku)	a/b
%	Jakojäännös	a%b

2.4: Arvonmuunto-operaattoreita

2.4.1: Taulukko arvonmuunto-operaattoreita

Operaattori	***Kuvaus***	***Esimerkki***	***Merkitys***
+=	Yhteenlasku	x+=y	x=x+y
-=	Vähennys	x-=y	x=x-y
*=	Kerto	x*=y	x=x*y
/=	Jako	x/=y	x=x/y
%=	Jakojäännös	x%=y	x=x%y
++	Etulisäys	++x	x=x+1
--	Etuvähennyt	--x	x=x-1

Operaattori	***Kuvaus***	***Esimerkki***	***Merkitys***
++	Jälkilisäys	x++	x=x+1
--	Jälkivähennys	x--	y=y+1

Taulukossa näkyy vähennys epäselvänä, mutta siinä on miinus ja = merkit peräkkäin. Oma mielipiteeni on, että kannattaa pääsääntöisesti käyttää ”Merkitys” sarakkeessa olevia hieman pidenpiä muotoja. Ne ovat huomattavasti selkeänpiä. Nämä kannattaa kuitenkin painaa mieleen, sillä voit joutua lukemaan koodia, jossa näitä on käytetty paljonkin.

Kuitenkin ++ ja – operaattorit ovat laskureissa yleisiä, ja tavallisimmin käytettyjä vähennyt ja lisäys operaattoreita. Esim. i++ tai i-- .

3 Tiedon tulostus ja kysyminen konsoliohjelmissa

3.1: Tiedon tulostaminen

Konsoliohjelmissatieto tulostetaan Console.WriteLine() metodilla.

Esimerkki 3.1.1: Console.WriteLine(”Tulostettava teksti”);

Lauseita voi yhdistellä ja muuttujia lisätä + operaattorin avulla.

Esimerkki 3.1.2: Console.WriteLine(”Muuttujan a arvo on” + a + ”ja muuttujan b arvo on” + b);

Jos et halua rivinvaitoa lauseen jälkeen, voit käyttää myös Console.Write() metodia.

Esimerkki 3.1.3: Console.Write(”Tulostettava teksti.”);

Muuttujat voi tulostaa myös seuraavasti:

C#:ssa voi muuttujat laittaa aaltosulkeissa tulostuslauseen sekaan. Huomaa, että aaltosulkeisiin
laitetaan numero. Lauseeseen sijoitettavat muuttujat listataan pilkulla eroitettuna heittomerkien
jälkeen. Luettelon ensmmäisen muuttujan numero on 0, seuraavan 1 jne.

Esimerkki 3.1.4:

```
 int x=20, y=40;
 Console.WriteLine("X-koordinantti on {0} ja Y-
koordinantti on {1}.", x,y);
 //Lause tulostaa saman kuin:
 Console.WriteLine("X-koordinantti on " + x + " ja Y-
koordinantti on " + y + ".");
```

Avaa Visual Studio ohjelmassa konsoliohjelma (Console Application) ja anna sille nimeksi TulostamisEsimerkki. Muokkaa sitten koodi seuraavan laiseksi. Voit toki ottaa suoraan tästäkin koodin kopioi/ liitä toiminnolla.

3.1.5: Esimerkkiohjelma muuttujista, niiden tulostamisesta ja tietotyypin tulostuksesta

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace TulostamisEsimerkki
{
    class Program
    {
        static void Main(string[] args)
        {
            //Määritetään int tyyppisiä muuttujia.
            //Muuttujat voi samalla alustaa jos
            //haluaa.
            //Mikäli käyttää muuttujaa ennenkuin sihen on
            //sisällytetty mitään arvoa,
            //kääntäjä antaa virheilmoituksen.
            //Arvon voi varalta alustaa jo alussa
            //nollaksi, mikäli ei ole varma,
            //sijoitetaanko ennen käyttöä arvoa.
            int a, b = 0, c = 4;
            //Määritetään double tyyppisiä muuttujia
            double pii = 3.14, k;
            //Määritetään float tyyppisiä muuttujia
            float x, y = 5;

            //Suoritetaan laskutoimituksia. Tässä
            //tapauksessa kaikkia muuttujia ei
            //tarvinnut alustaa, koska niihin
            //lasketaan arvo ennen muuttujan muuta
            //käyttöä, kuten tulostusta.
            //Tämä ei toimisi: b=a+c, koska muuttujalla a
            //ei ole alkuarvoa.
            //Sensijaan näin toimii:
            a = b + c;
            k = pii * 2;
            x = y / 2;
            //Tulostetaan arvot
            Console.WriteLine("Muuttujan a arvo on " + a
+ ", b:n arvo on " + b + " ja c:n " + c);
            Console.WriteLine("Muuttujan pii arvo on: " +
pii + ", k:n " + k + ", x:n " + x + " ja y:n " + y);
```

```
            Console.WriteLine("Muuttujan pii tietotyyppi
on " + pii.GetType());
            Console.WriteLine("Lukujen 5 ja 2 jakojäännös
on " + 5 % 2);
            // \n merkinnällä voidaan tulostaa
            //rivinvaihtoja
            Console.Write("\nPress any key to continue .
. . ");
            Console.ReadKey(true);
        }
    }
}
```

3.2: Tiedon lukeminen ja poikkeuksien käsittely

Tiedot luetaan konsoliohjelmissa Console.ReadLine() metodilla. Tieto luetaan string tyyppisenä, joten se on muutettava numeeriseksi erillisellä Parse metodilla, mikäli halutaan käsitellä numeroa.

Esim 3.2.1: string syote = Console.ReadLine();
int luku = int.Parse(syote);

C Sharpissa Console.ReadLine() metodilla ei voi kysymykseen laittaa tekstiä, vaan itse kysymys on kirjoitettava Console.WriteLine() metodilla. Tyyppimuutoksen voi tehdä myös suoraan kysymyslauseessa.

Esim 3.2.2: int luku;
Console.WriteLine(”Anna luku: ”);
luku=int.Parse(Console.ReadLine());

Jos nyt kuitenkin syötetään vastauksesi kirjain, tapahtuu virhe, kun yritetään muuttaa kirjainta numeroksi (int tyypiseksi). Tämän voi estää käyttämällä ”try/catch” rakennetta. Seuraavassa esimerkki.

Esim. 3.2.3: int luku;
Console.WriteLine(”Anna luku: ”);
try
{
//Yritetään tehdä tyyppimuunnos
luku=int.Parse(Console.ReadLine());
}
catch
{
/*Tämä kohta suoritetaan,
mikäli tapahtui virhe*/

```
        Console.WriteLine(”Tapahtui virhe.”);
    }
    finally
    {
        //Tämä lohko suoritetaan aina
     }
```

Poikkeuskäsittelyyn voi lisätä Exception kohdan, joka tulostaa tapahtuneen virheen. Muuta vain catch osiota näin:

Esim.3.2.4:
```
catch (Exception ex)
{
 Console.Writeline(ex.Message)
}
```

Poikkeuskäsittelyyn voi lisätä myös ”finally” kohdan, joka suoritetaan, tapahtuipa virhe, tai ei. Poikkeuskäsittely kannattaa opetella jo ohjelmoinnin alkuvaiheissa. ”finally” lohkoa ei sensijaan useinkaan käytetä.

4 Boolean-operaattorit

C#:ssa on boolean tyyppinen muuttuja, joka sisältää joko arvon ”true” (tosi) tai ”false” (epätosi). Boolean-operaattorilla suoritetaan laskutoimitus, jonka tulos on joko ”true” tai ”false”.

4.1: Vertailuoperaattorit

Operaattori	Kuvaus	Esimerkki	Tulos, jos luku on 8
==	Yhtä suuri kuin	luku==9	false
!=	Eri suuri kuin	luku!=8	false
<	Pienempi kuin	luku<9	true
<=	Pienempi tai yhtäsuuri kuin	luku<=8	true
>	Suurempi kuin	luku>7	true
>=	Suurempi tai yhtäsuuri kuin	luku>=9	false

4.2: Loogiset operaatorit

Operaattori	Kuvaus	Esimerkki	Tulos, jos luku on 8
!	Ei	!(luku==8)	false
AND &&	Ja	(luku==8) && (luku<12)	true
OR ||	Tai	(luku>12) || (luku<9)	true

Loogisissa operaattoreissa voidaan käyttä joko AND tai sitä korvaavana && ja yhtälailla OR operaattorin tilalla voidaan käyttää merkintää || (löytyy näppäimistöltä AltGr <).

4.3: Ehto-operaattori ?

Ehto-operaattorissaon kolme osaa: ehto, lauseke1 ja lauseke2 .Paluuarvo on lausekkeen yksi arvo, jos lauske on tosi, ja lausekkeen kaksi arvo, jos lauseke on epätosi.

Esimerkki 4.3.1:

```
int x=2, y=4;
int tulos = x < y ? (int)3 : (int)6;    //lopputuloksena
  //tulos=3
tulos = x >= y ? (int)3 : (int)6  //lopputuloksena
  //tulos=6
```

Vertailulauseet

5.1: if lause

if (jos) lauseella verrataan ehtoja toisiinsa. Se toimii seuraavasti.
Esim. 5.1.1:

```
if (x<10)                    //huom. Ei puolipistettä lopussa
{
   //suoritetaan, jos x<10
}
else if (x<12)               //huom. Ei puolipistettä lopussa
{
   //suoritetaan, jos x>9 mutta pienempi kuin 12
 }
else if (x<=14)
{
  //suoritetaan, jos x>11 ja x on pienenpi tai yhtäsuuri kuin 14
}
else                         //huom. Ei  puolipistettä lopussa
{
 //suoritetaan, jos mikään aiemmista tapauksista ei toteudu
}
```

Aina lauseessa ei ole ”else if” eikä ”else” osioita ollenkaan.

5.1.1: Esimerkkiohjelma karkausvuosi
Anna ohjelmalle nimeksi Karkausvuosi

Karkausvuosi lasketaan seuraavasti: vuosi on karkausvuosi, jos se on neljällä jaollinen. Kuitenkaan vuosi ei ole karkausvuosi, jos se on sadalla jaollinen. Vuosi on kuitenkin aina karkausvuosi, jos se on 400:lla jaollinen. Huomioi kuitenkin, että ajanlaskutapa on välillä muuttunut, eikä kovin vanhasta ajanlaskusta ole tarkkaa tietoa.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

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

            Console.WriteLine("Anna vuosiluku: ");
            try
            {
                int vuosi =
int.Parse(Console.ReadLine());
                if (vuosi % 4 == 0 && vuosi % 100 != 0 ||
vuosi % 400 == 0)
                    Console.WriteLine("Vuosi on
karkausvuosi.");
                else
                    Console.WriteLine("Vuosi ei ole
karkausvuosi.");

            }
            catch
            {
                Console.WriteLine("Virheellinen syöte.");
            }
            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);

        }
    }
}
```

Ohjelmassa ei tarvita erikseen sulkuja lausekkeessa ((vuosi % 4 == 0 && vuosi % 100 != 0) || vuosi % 400 == 0), vaikka ne myös toimisivat, koska AND operaatiot suoritetaan ennen OR operaatioita, eli AND operatioilla on korkeampi prioriteetti. Ohjelmassa näkyy hyvin myös % (jakojäännös) operaation käyttö. Esim. 1600 % 4 = 0, kun taasen 1998 % 4 = 2.

Huomaa myös, että if ja else lauseiden jälkeen tulevaa koodia ei ole laitettu aaltosulkeisiin. Se tosin toimisi, mutta kun lauseita seuraa vain yksi, aaltosulkeet eivät ole pakollisia.

5.2: switch lause

Switch lause on toinen C Sharpin vertailulause. Yleensä switch lauseen voi korvata if-else rakenteella, mutta usein switchlause on selkeänpi.

Switch lause toimii niin, että switch avainsanalle annetaan vertailtava arvo, esim. switch(numero). case lauseessa tarkistetaan, sopiiko ehto. default lauseeseen voi laittaa toiminnon, joka suoritetaan, jos yksikään case ehto ei toteudu. Hyvin usein default kohta jätetään pois. Kaikkien case lauseiden on päätyttävä break käskyyn, poislukien esimerkeissä 5.2.1 ja 5.2.2 näkyvät tapaukset, joissa ehtoja on peräkkäin lueteltuna ennen suoritettavaa toimintoa. default päättyy aina break käskyyn.

5.2.1: Lausekkeen rakenne on:

```
switch(numero)                    //huom. Ei  puolipistettä lopussa
{
  case ehto:                      //kaksoispiste
     //lausekkeet
     break;
  case ehto2:
     //toiset lausekkeet
     break;
  case ehto3:
  case ehto4:
  case ehto5:
    //kolmannet lauseet
    break;
 default:                          //kaksoispiste
    //suoritetaan, jos mikään aienpi ehto ei toteudu
   //jätetään useissa tpauksissa pois
}
```

5.2.2: Esimerkkiohjelma kuukausien päivien lukumäärästä
Anna ohelmalle nimeksi KuukausienPaivat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

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

```
        {
            int pvm;
            Console.WriteLine("Anna kuukauden numero:");
            try
            {
                pvm = int.Parse(Console.ReadLine());
                switch (pvm)
                {
                    case 2:
                        Console.WriteLine("Kuukaudessa on
28 päivää. Karkausvuonna 29.");
                        break;
                    case 4:
                    case 6:
                    case 9:
                    case 11:
                        Console.WriteLine("Kuukaudessa on
30 päivää");
                        break;
                    default:
                        Console.WriteLine("Kuukaudessa on
31 päivää");
                        break;
                }
            }
            catch
            {
                Console.WriteLine("Virheellinen syöte.");
            }
            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);

        }
    }
}
```

switch rakenteen olisi voinut toteuttaa if-else rakenteella seuraavasti:

```
if (pvm==4 || pvm==6 || pvm==9 || pvm==11)
                Console.WriteLine("Kuukaudessa on 30 päivää");
   else if (pvm==2)
                 Console.WriteLine("Kuukaudessa on 28 päivää. Karkausvuonna 29.");
  else
   Console.WriteLine("Kuukaudessa on 31 päivää");
```

swich rakenne on kuitenkin tässä tapauksessa selkeänpi. Tämä ohjelma ei vielä tutki sitä mahdollisuutta, että käyttäjä antaa kuukaudeksi pienemmän numeron kuin yksi, tai suuremman kuin 12.

switch lausekkeen ehto voi olla myös merkki (char) tai merkkijono (string). Tällöin vertailu tapahtuu seuraavasti:

5.2.3: Esimerkkiohjelma merkin vertaaminen
Anna ohjelmalle nimeksi Vokaali

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Vokaali
{
    class Program
    {
        static void Main(string[] args)
        {
            char merkki;
            Console.WriteLine("Anna jokin kirjain:");
            try
            {

                merkki = char.Parse(Console.ReadLine());
                switch (merkki)
                {
                    case 'a':
                    case 'e':
                    case 'i':
                    case 'o':
                    case 'u':
                    case 'y':
                        Console.WriteLine("Merkki on
vokaali.");
                        break;
                    case 'ä':
                    case 'ö':
                        Console.WriteLine("Merkki on
vokaali ja ääkkönen.");
                        break;
                    default:
                        Console.WriteLine("Merkki on
```

```
konsonantti, numero tai erikoismerkki.");
                        break;
                }
            }
            catch
            {
                Console.WriteLine("Virheellinen syöte.
Syötä vain yksi kirjain.");
            }

            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);
        }
    }
}
```

Ohjelmassa näkyy myös tyyppimuunnos char.Parse() jolla muutetaan string muuttuja char muotoon.
Myös tässä tarvitaan ”try/catch” rakennetta tarkistukseen, tai muuten tapahtuu virhe, jos käytäjä syöttää useamman merkin.

5.2.4: Esimerkkiohjelma merkijonosta switch lauseessa

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Vuodenaika
{
    class Program
    {
        static void Main(string[] args)
        {
            string kuukausi;
            Console.Write("Anna kuukauden nimi: ");
            kuukausi=Console.ReadLine();
            switch (kuukausi)
            {
                case "tammikuu":
                case "helmikuu":
                case "joulukuu":
                    Console.WriteLine("Kuukausi on
talvikuukausi.");
```

```
                    break;
                case "maaliskuu":
                case "huhtikuu":
                case "toukokuu":
                    Console.WriteLine("Kuukausi on
kevätkuukausi.");
                    break;
                case "kesäkuu":
                case "heinäkuu":
                case "elokuu":
                    Console.WriteLine("Kuukausi on
kesäkuukausi.");
                    break;
                case "syyskuu":
                case "lokakuu":
                case "marraskuu":
                    Console.WriteLine("Kuukausi on
syksykuukausi.");
                    break;
                default:
                    Console.WriteLine("Annoit kuukauden
nimen väärin.");
                    break;
            }

            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);

        }
    }
}
```

Ohjelmassa näkyy myös Console.Write() metodin käyttö, joka ei vaihda riviä automaattisesti käskyn jälkeen, kuten Console.WriteLine() metodi. Poikeuskäsittelijää (try/catch) ei tässä tapauksessa tarvita, koska string muuttujaan kelpaavat syötteenä kaikki merkit ja merkkijonot.

6 Toistolauseet

6.1: for silmukka

for silmukka toistaa ehdossa mainitun määrän silmukan sisässä olevia lausekkeita.

6.1.1: Rakenne

```
for(alkuehto,loppuehto,laskurin kasvatus)                    //huom. Ei  puolipistettä
lopussa
{
   //lausekkeet;
}
```

Esim. 6.1.2:

```
for (int i=0;i<50;i++)
{
   //suoritettavat toiminnot;
   //silmukka suoritetaan 50 kertaa ja saa arvot 0...49
}
```

Esim. 6.1.3: Silmukka voi olla myös vähenevä

```
int i;
for(i=50;i>0;i--)
  //Suoritettavat lausekkeet, silmukka suoritetaan 50 kertaa ja saa arvot 50...1
```

Huom. for silmukan jälkeen ei tarvitse laittaa aaltosulkuja, jos suoritettavia lauseita on vain yksi

for silmukan lausekkeiden suoritusjärjestys on seuraava. Suluissa suoritusvaihe numerona.

```
for(aloitusosa(1);ehto(2);kasvatus(4))
{
 lauseet(3)
}
```

Lauseet siis suoritetaan ennen kasvatusvaihetta. Alustusosa suoritetaan vain kerran, mutta muut osat toistuvasti. Ohjelman kulku siis on 1,2,3,4,2,3,4,2,3,4.

Jos for lause on:

```
for(i=0;i<2;i++)
{
  //lauseet
}
```

lauseet suoritetaan kaksi kertaa, eli i saa arvot 0 ja 1.

6.1.4: Esimerkkiohjelma for silmukasta ja satunnaislukugeneraattorista

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ForEsimerkki
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, arvottu;
            Random satluku = new Random(); //Luodaan
               //satunnaisluku olio satluku
            for (i = 1; i < 21; i++)
            {
                arvottu = satluku.Next(1,101); //arvotaan
           //luku väliltä 1-101.
           //arvottu voi saada kokonaislukuarvon 1...100
                Console.WriteLine(i + ". satunnaisluku
on: " + arvottu);
            }

            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);

        }
    }
}
```

Ohjelmassa on luotu satunnaislukuolio satluku. Olioista ei tässä vaiheessa tarvitse tietää enemmän. Mainittakoon kuitenkin, että olio luodaan new avainsanalla. Satunnaisluvun käytöstä on enemmän luvussa ”17 Satunnaisluku”.

6.1.5: Sisäkkäiset for silmukat

for silmukoita voi luoda myös sisäkkäisiksi, mikä on hyvin tavallista.
Tyyppiesimerkki on taulukko, jossa on y ja x koordinaatit. Useanpiakin sisäkkäisiä silmukoita tarvitaan ajoittain.

Esim. 6.1.5.1: Kaksiulotteinen for silmukka

```
int i,j;
for (i=0;i<10;i++)  for (j=20;j<40;j++)  //suoritettava lause;
```

Huomaa, että jos for lauseet ovat peräkkäin, ja suoritettavia lauseita vain yksi, aaltosulkeita ei tarvita.

Aaltosulkeita käytetään, jos lauseita on enemmän. Huomaa, että laskurin luku voi olla mitä vain, kuten seuraavassa esimerkissä j :n arvo 20...39.

```
int i,j;
for (i=0;i<10;i++)
{
   //suoritettavia lauseita
   for (j=20;j<40;j++)
   {
      //suoritettavia lauseita;
      //lisää lauseita
   }
}
```

6.1.5: Esimerkkiohjelma sisäkkäisistä silmukoista

Tyypillisinpiä esimerkkejä sisäkkäisestä silmukasta on kertotaulu, joka on kaksiulotteinen taulukko.
Seuraavassa kertotaulu lasku- ja tulostusohjelma.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Kertotaulu
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, j;
```

```
            for (i = 1; i < 11; i++)
            {
                for (j = 1; j < 11; j++)
                {
                    //tulostetaan kertotaulu. \t on
                    //vaakasuuntainen sarkain
              //jonka avulla rivit saadaan kohdalleen
                    //pystysarkain on \v
                    Console.Write(j * i + "\t");
                }
                //tulostetaan rivin vaihto
                Console.WriteLine();
            }
            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);
        }
    }
}
```

6.2: while silmukka

while silmukan rakenne on seuraava:

```
while(ehto)
{
  //suoritettavia lauseita
}
```

eli esim.

```
int i=0;
while(i<10)
{
  //suoritettavia lauseita
  //suoritetaan tässä tapauksessa 10 kertaa
  i++;
}
```

6.2.1: Esimerkkiohjelma kertoma while lauseella
Esim. kuuden kertoma lasketaan kertomalla 2*3*4*5*6.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Kertoma
{
    class Program
    {
        static void Main(string[] args)
        {
            int luku=-1;
            while (luku != 0)
            {
                    try
                    {
                     Console.WriteLine("Anna kokonaisluku
väliltä 1-150, jonka kertoma lasketaan.");
                     Console.Write("Syötä 0 kun haluat
lopettaa: ");
                     luku =
int.Parse(Console.ReadLine());
                     if (luku > 150)
                     {
                         Console.WriteLine("Luku on liian
suuri kertoman laskemiseen tällä ohjelmalla.\n");
                         continue;
                     }
                     if (luku < 0)
                     {
                        Console.WriteLine("Luku on
negatiivinen.\n");
                        continue;
                     }
                     double kertoma = 1;
                     int i = 1;
                     while (i <= luku)
                     {
                        kertoma = kertoma * i;
                        i++;
                     }
                     if (luku != 0)
```

```
Console.WriteLine("Luvun " + luku + " kertoma on " +
kertoma);
                        }
                        catch
                        {
                            Console.WriteLine("Virheellinen
syöte.");
                        }

                }
            }
        }
}
```

Ohjelmassa on myös continue lause, jolla voi palata silmukan alkuun, tässä tapauksessa while(luku!=0) kohtaan.
while(luku!=0) tai esim. while(merkki!=”q” || merkki!=”Q”) rakennetta käytetään silmukan toistamiseen. while(merkki!=”q”) rakenne esim. toistaa silmukkaa niin kauan, kunnes käyttäjä syöttää ”q” kirjaimen.

6.3: do-while silmukka

do-while silmukka on lopetusehtoinen. Sitä käytetään while silmukan tapaan, mutta lauseen ehto tarkistetaan vasta ilmukan lopussa. Siitä johtuen do-while silmukka toistetaan aina vähintään yhden kerran. Jos alkuehto ei toteudu, toisinaan while silmukkaa ei suoriteta lainkaan.

do-while silmukan rakenne:

```
do
{
  //suoritettavat lauseet
}
 while(ehto)
```

7 Hyppylauseet

7.1: continue

continue käskyllä voidaan hypätä esim. switch-case tai while silmukan alkuun. Tästä oli esimerkki 6.2.1 esimerkkiohjelmassa.

7.2: break

break käskyllä voi katkaista silmukan suorittamisen. Tällöin ohjelma siirtyy suorittamaan lauseita silmukan jälkeiseltä riviltä.

7.3: goto

goto käskyllä voi hypätä ohjelmassa toisen paikkaa merkityyn pointteriin.
Esim.

```
alku:
 //suoritettavia lauseita
 goto alku;
```

tai esim.

```
loop:
 //suoritettavia lauseita
 if (ehto<10) goto loop;
```

goto lauseen käyttöä pidetään erittäin huonona ohjemointitapana, ja se aiheuttaa sekavia, ja jälkeen päin vaikeasti luettavia ohjelmia. Yleensä goto lauseen käyttö kertoo huonosti rakennellusta ohjelmasta.

8 Visual Studion käytöstä

8.1: Solution Explorer

On mahdollista, että koodi häviää näkyvistä. Löydät koodin sivulla oikealla olevasta ”Solution Explorer” valikosta. Koodin saat näkyviin klikkaamalla .cs päätteistä luokan nimeä. Näissä esimerkkiohjelmissa minulla on ollut luokan nimenä oletus, eli ”Program.cs”.

8.2: Ohjelman tallennus

Koko ohjelmaprojektisi saat tallennettua ”File/ Save All” komennolla. Ohjelman nimeksi tulee oletuksena ohjelmaa luodessasi antamasi nimi. Voi valita, mihin kansioon projektin tallennat. Oletusarvoisesti projekti tallentuu kansioon ”Visual Studio 2010/ Projects”.

8.3: Ohjelman ajaminen

Ohjelma ajetaan ”Debug/ Start Debugging” kohdasta. ”Debug/ Step Into” toiminnolla voit ajaa ohjelmaa rivi kerrallaan, ja näet, miten ohjelma etenee koodissa. Tästä voi olla hyötyä virheiden etsimisessä. Klikkailessasi”Step Into” valintaa, ohjelma etenee askeltaen. Jos ajat ohjelmaa ”Step Into” toiminnolla, voi lopettaa ajamisen valitsemalla ”Debug/ Stop Debugging”.

8.4: Ohjelman julkaisu

Valitse Visual Studiosta ”project/publish”. Voit nyt tallentaa ohjelman haluamaasi kansioon, tai työpödälle. Ohjelma tekee pikakuvakkeen, ja sen voi nyt myös siirtää toiseen tietokoneeseen esim. muistitikun avulla. Ensimmäiselä ajokerralla ohjelma on asennettava, joka tapahtuu pikakuvaketta tuplaklkkaamalla, jolloin asennus käynnistyy.

9 Taulukot

Taulukot voivat olla joko yksi- tai moniulotteisia ja mitä tahansa perustietotyypeistä (esim. int, double, string, char).

9.1: Yksiulotteinen taulukko

Yksiulotteinen taulukko luodaan seuraavasti:

tyyppi [] taulukon_nimi
taulukon_nimi = new tyyppi[taulukon_koko]

new operaattorilla varataan taulukolle muistitila

eli esim. int [] koordinantti;
koordinantti = new int[20];
//taulukon voi myös määritellä samala kun varaa tilan
int [] koordinantti = new int[2};

string [] taulukon_nimi = new string[maara]
eli esim. string [] nimet= new string[8];

Taulukot voi myös alustaa esittelyn yhteydessä seuraavasti:

```
int [] arvosanat = {1,2,3,4,5};   //C# varaa automaattisesti oikean muistimäärän

string [] nimet = new string[3];
//alustetaan yksitellen
nimet[0] = ”Matti”;
nimet[1] = ”Kalle”;
nimet[2] = ”Ville”;
```

Voi tehdä myös näin:

```
string[] nimet = { "Matti", "Kalle" , "Ville" };  //C# varaa automaattisesti oikean muistimäärän

char [] vokaali = {'a','e','i','o','u','y','ä','ö'};   //C# varaa automaattisesti oikean muistimäärän
```

Huomaa, että taulukko alkaa aina alkiosta nolla, eli ”int [3] arvosanat” määrittely varaa taulukon paikat:
arvosanat[0];
arvosanat[1];
arvosanat[2];

9.2: Moniulotteinen taulukko

Kaksiulotteista taulukkoa voi verrata koordinaatistoon, jossa on korkeus ja leveys. Kolmiulotteisessa on vielä syvyys. Yksi ja kaksiulotteisia taulukoita käytetään eniten, mutta periaateessa tietokone voi muodstaa vaikka kuusiulotteisen taulukon, jonka hahmottaminen on vaikeaa. Itse olen joskus käyttänyt neliulotteista taulukkoa, mutta harvoin käytetään yli kaksi- tai kolmiulotteisia taulukkoja.
Moniulotteista taulukkoa voisi havainnollistaa monessa käytössä myös seuraavasti:

alkio[x,y,z]

jokaista x alkiota vastaa y kappaletta ominaisuuksia ja jokaista y alkiota kohti on z kappaletta lisäominaisuuksia, eli x alkiolla on y*z lisäominaisuutta.

Esimerkiksi voisi olla kysymyskaavake. Taulukon alkiossa[x] olisi kysymysten määrä. Alkiossa [y] olisi tieto, onko kysymyksessä kaksi-, kolme- vai neljä vastausvaihtoehtoa (a,b?, a,b,c?, a,b,c,d?). Kaavakkeita tasaen voisi olla kolme erilaista samaa henkilöä kohti, ja haluttaisiin tallentaa tiedot samaan taulukkoon, mutta taulkkoon pitäisi taltioida tieto, mihin kaavakkeeseen vastaus kuuluu, eli tarvittaisiin kolmas ulottuvuus z. Tämä edellyttää että kaikissa kaavakkeissa on saman verran kysymyksiä. Tällaisen ohjelman tekemisessä kannattaisi tosin harkita sisäkkäistä taulukkoa, joka esitellään kohdassa 9.3.

Tällöin luotaisiin taulukko: alkio[x][y][z] jonka muistinvaraus voisi olla vaikka seuraava:

int [,,] alkio = new int[kysymysten_maara,4,3]

eli esim. int[,,] = new int[20,4,3];

9.2.1: Esimerkkiohjelma yksi- ja moniulotteisesta taulukosta

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arvosanat
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, j, x;
            Console.Write("Kuinka monta oppilasta
kurssilla oli? ");
```

```
            try
            {
                x = int.Parse(Console.ReadLine());
                //tässä x kuvaa oppilaiden lukumäärää,
                //joka kysytään käyttäjältä
                string[] nimet = new string[x];
                int[,] arvosanat = new int[x, 3];
                //Kysytään oppilaan nimi ja arvosanat
                for (i = 0; i < x; i++)
                {
                    Console.Write("Anna oppilaan nimi:
");
                    nimet[i] = Console.ReadLine();
                    for (j = 0; j < 3; j++)
                    {
                        Console.Write("Anna kurssin " +
(j + 1) + " arvosana: ");
                        arvosanat[i, j] =
int.Parse(Console.ReadLine());
                    }
                }

                //Tulostetaan tiedot
                for (i = 0; i < x; i++)
                {
                    Console.Write("\nOppilaan " +
nimet[i] + " arvosanat:");
                    for (j = 0; j < 3; j++)
                      Console.Write("\t" + arvosanat[i,
j]);

                }
            }
            catch
            {
                Console.WriteLine("Virheellinen syöte");
            }
            Console.WriteLine("\n\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);

        }
    }
}
```

9.3: Sisäkkäinen taulukko

Sisäkkäisissä taulukoissa on taulukoita alkioina. Sisäkkäinen taulu esitellään ja luodaan seuraavasti:

tyyppi [][]...[] taulukon_nimi = new tyyppi[m][n]...[]

[] merkkien lukumäärä määrittää sisäkkäisen taulukon taulualkioiden lukumäärän. Viimeisessä [] kohdassa ei ole mitään numeroa, vaan se voi vaihdella eri ulottuvuuksilla ja määrätään erikseen.

9.3.1: Esimerkkiohjelma sisäkkäisistä taulukoista
Seuraavassa tehtävässä voisi kyseessä olla kolme eri koepaperia, joissa kaikissa on eri määrä tehtäviä. Ensimmäisesä paperissa on 4 tehtävää, toisessa 2 tehtävää ja kolmannessa 3 tehtävää.
Jokainen tehtävä on sitten arvosteltu asteikolla 1-5.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SisakkainenTaulukko
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, j;

            //määritellään sisäkkäinen taulukko
            //tässä kaksiulotteisessa sisäkkäisssä
            //talukossa
            //on kolme taulukkoa alkiona
            int[][] numerot = new int[3][];

            //määritellään sisäkkäisen taulukon
            //ensimmäinen rivi
            //joka on nelialkioinen taulukko
            numerot[0] = new int[4];

            //alustetaan taulukko
            numerot[0][0] = 4;
            numerot[0][1] = 3;
            numerot[0][2] = 3;
```

```
            numerot[0][3] = 4;

            //määritellään toinen rivi
            numerot[1] = new int[2];

            //alustetaan toinen rivi
            numerot[1][0] = 5;
            numerot[1][1] = 2;

            //määritellään kolmas rivi, joka on myös
            //taulukko
            //tässä myös toinen tapa alustaa taulukot
            //taulukossa on kolme alkiota
            numerot[2] = new int[] { 3, 3, 4 };

            for (i=0;i<3;i++) {
             for (j = 0; j < numerot[i].Length; j++) {
                 Console.Write("\t" + numerot[i][j]);
             }
                Console.WriteLine();
            }

            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);


        }
    }
}
```

Tässä ei ole vielä huomioitu, että oppilaita on todennäköisesti useampia. Kymmenelle oppilaalle sisäkkäinen taulukko pitäisi esitellä seuraavasti:

```
//määritetään kolmiulotteinen sisäkkäinen taulukko
        int[][][] numerot2= new int[3][][];

           //määritetään sisäkkäiseen taulukkoon
           //kaksiulotteinen taulukko
        numerot2[0] = new int[4][];

           //sijoitus ensimmäisen opiskelijan
           //ensimmäiseen numeroon
        numerot2[0][0][0] = 4;

           //sijoitus viidennen oppilaan kolmanteen
           //numeroon
           //muistathan: taulukko alkaa alkiosta 0
           //joten luku neljä viittaa viidenteen
           //arvoon
           //ja kaksi kolmanteen arvoon
        numerot2[0][2][4] = 3;
```

10 strig lauseista ja char tyypistä

strig lause on merkkijono, joka on myös samalla taulukko. Esim. jos on lause string auto=”Volvo” voidaan kirjaimeen ”V” viitata auto[0] ja ”l” auto[2]. Merkkijonoja voi liittää + operaattorilla, esim. uusi=”asunto” + ”vaunu” operaation jälkeen uusi on ”asuntovaunu”.

Isot ja pienet kirjaimet ovat eri merkkejä. Jos kirjoitat ”Q” se on ihan eri asia kuin ”q”. Usein on erilaisissa tarkistuksissa käytännöllistä muuttaa ensin kirjaimet isoiksi, joka tapahtuu metodilla ToUpper() eli esim. lauseIsoilla = lause.ToUpper() . Merkkijonon voi vastaavasti muuttaa pieniksi kirjaimiksi ToLower() metodilla sana=sana.ToLower() . Merkkijonon pituutta voi tutkia Lenght käskyllä,
eli esim. pituus = sanat.Length;

char tyypillä määritellään yksittäinen merkki. char vie muistissa yhden tavun.

```
Esimerkki:  char merkki = 'k';
             //char taulukko
            char [] kirjain = { 'a', 'b',' c',' d' };
            //tai alustamattomana esim.
            char [] merkit = new char[20];
            // .NET tyyppi on Char eli isolla kirjoitettuna

           //char muuttujallevoidaan myös antaa unicode koodi. Tässä esimerkissä n
kirjaimen
            //koodi, joka on 006E
           char merkki = '\u006E';
```

Hyvä lista unicode merkeistä löytyy Wikipediasta osoitteesta:
http://en.wikipedia.org/wiki/List_of_Unicode_characters

Unicode merkistössä on paljon muutakin kuin vain kirjaimia, joten kannattaa tutustua.

Huomaathan, että string tyypissä käytetään lainausmerkkejä ” ja char tyypissä heittomerkkiä ' .

string lauseen voi paloitella sanoiksi Split() metodilla.

```
string lause = ”C# ohjelmointi on hauskaa.”;
string[] sanat = lause.Split(); //paloittelee string taulukkoon sanat
nyt esim. sanat[0]=”C#” ja sanat[1]=”ohjelmointi”
```

10.1: Esimerkkiohjelma merkkijonojen käsittelystä: siansaksa
Ohjelma muuttaa tekstiä seuraavasti siansaksaksi:

-jos sana alkaa vokaalilla lisätään sanan loppuun way
-jos sana alkaa konsonantilla sirretään sanan ensimmäinen kirjain sanan loppuun, ja lisätään
loppuun ay
-jos alkukirjain on numero tai erikoismerkki, ohjelma ei tee siinä mitään

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Siansaksa
{
    class Program
    {
        static void Main(string[] args)
        {
            string lause="";
            string lause2;
            string[] sanat;
            char[] kon = { 'b', 'c', 'd', 'f', 'g', 'h',
'j', 'k', 'l', 'm', 'n', 'p', 'q', 'r', 's', 't', 'v',
'w', 'x', 'z' };
            while (lause != "quit") //toistetaan niin
//kauan, että käyttäjä syöttää quit
            {
                int i, j, laskuri = 0, tyyppi=0;
                string lopLause = "";
                Console.WriteLine("\nAnna
englanninkielinen lause, joka muutetaan
siansaksaksi.\nAnna quit lopettaaksesi:\n");
                lause=Console.ReadLine();
                sanat = lause.Split(); //paloitellaan
//lause sanoiksi sanat taulukkoon
                laskuri = sanat.Length; //tutkitaan
//laskuri muuttujaan sanat[] taulukon sanojen lukumäärä
                lause2 = lause.ToLower(); //tallennetaan
//lause2 muuttujaan lause pienillä kirjaimilla
                string[] sanat2 = lause2.Split();
//paloitellaan myös lause2 sanoiksi

                for (i = 0; i < laskuri; i++)
                {
                   tyyppi = 0;
                   string lopSanat = "";
```

```
                    //tehdään kaikki tarkistukset pieniksi
                    //kirjaimiksi muutettuna
                    //näin toiminto on sama, alkoipa sana
                    //isoilla tai pienillä kirjaimilla
                    string sana2 = sanat2[i];
                    char merkki = sana2[0];
                    switch (merkki) //tarkistetaan, onko
//alkukirjain vokaali, jolloin tyyppi=1
                    {
                        case 'a':
                        case 'e':
                        case 'i':
                        case 'o':
                        case 'u':
                        case 'y': tyyppi = 1;
                                  break;
                    }
                    if (tyyppi != 1) //jos alkukirjain ei
//ole vokaali, tarkistetaan onko konsonantti
                    {
                //jos alkukirjain on konsonantti, tyyppi=2
                        for (j = 0; j < 20; j++) if
(merkki == kon[j]) tyyppi = 2;
                    }

                    /*jos alkukirjain vokaali, lisätään
loppuun "way"
                      huomioi: tarkistukset tehdään
pieniksi kirjaimisi muutetulla muuttujalla,
                      mutta sijoitettaessa lopulliseen
muotoon, käytetään alkuperäistä lausetta
                      näin lopullisessa lauseessa säilyvät
isot ja pienet alkukirjaimet
                      lopSanat on muuttuja, johon
sijoitetaan lopullinen sana
                    */
                    if (tyyppi == 1) lopSanat = sanat[i] +
"way ";

                    if (tyyppi == 2)
                    {
                        //jos alkukirjain on konsonantti,
//lisätään loppuun sanan alkukirjain ja "ay"
                        string lopSanat2 = sanat[i] +
```

```
sanat[i][0] + "ay ";

            //tutkitaan sanan pituus, ja lisätään kolme
            //(ay+loppukirjain=3)
            //poistetaan alkukirjain siirtämällä merkkeja
            //yksi pykälä vasemmalle
            //eli merkki[2] paikalle tulee merkki[3]
                        for (j = 0; j < sanat[i].Length+3;
j++) lopSanat = lopSanat + lopSanat2[j + 1];
                    }

                    //jos alkukirjain on jotain muuta kuin
                    //kirjain
                    //eli numero tai erikoismerkki, ei
                    //tehdä mitään muutoksia
                    if (tyyppi != 1 && tyyppi != 2)
lopSanat = sanat[i];

                    //lopLause on lopullinen lause, johon
                    //sanat peräjälkeen liitetään
                    lopLause = lopLause + lopSanat;

                }
                //tulostetaan lopullinen lause
                Console.WriteLine(lopLause);
            }
   } }}
```

11 Lueteltu tyyppi enum ja vakio const

Luetellun tyypin enum avulla luodaan yhteen kuuluvista vakioista lista. Tyypin oletusarvo on int, mutta sillä voi kuitenkin käyttää myös muita kokonaislukuja. Oletusarvoisesti arvot alkavat luvusta nolla, mutta sille voidaan myös antaa aloitusarvo, esim. yksi. Jos annetaan aloitusarvoksi yksi, seuraava vakio saa arvokseen kaksi, seuraava kolme jne. Kaikille vakioille voidaan myös määritellä erikseen arvo. Huomaa, että enum määritellään class osiossa. Jos yrität sijoittaa enum määritteen main() lohkoon, tapahtuu virhe.

const määreellä luodaan vakio. Vakion arvoa ei voi ohjelman aikana muuttaa. Vakio voi olla mikä tahansa lukutyyppi, merkki tai merkkijono, esim. ”Kyllä”.

11.1: Esimerkkitehtävä enum lauseesta ja const vakiosta

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EnumJaConstHarjoitus
{
    class Program
    {
        //huomaa, puolipistettä lopussa ei tarvita,
        //vaikkei se olekaan virhe
        enum vuodenaika { kevat = 1, kesa, syksy, talvi }
        enum mitat { mm = 1, cm = 10, m = 1000, km =
1000000 }

        static void Main(string[] args)
        {
            const double pii = 3.14; //määritetään pii
                        //vakioksi ja annetaan arvo
            int r;
            double ympAla;
            try
            {
                Console.Write("Anna ympyrän säde
senttimetreinä: ");
                r = int.Parse(Console.ReadLine());
                ympAla = pii * r * r;
                double ala = ympAla * (int)mitat.cm;
                 //tehtävä tyyppimuunnos
                Console.WriteLine("Ympyrän ala
```

```
millimetreinä antamallasi säteellä on {0}.", ala);
            }
            catch
            {
                Console.WriteLine("Virheellinen syöte.");
            }
            //Tässäkin tehtävä tyyppimuunnos
            Console.WriteLine("\nSyksyn numero on
keväästä laskettuna " + (int)vuodenaika.syksy);

            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);

        }
    }
}
```

12 foreach() toistolause

foreach toistolauseella käydään taulukoiden ja kokoelmien arvot yksi kerrallaan läpi. foreach lauseessa käytetään apumuuttujaa, joka saa yhden taulukon arvon vuorollaan arvoksi. Huomaa, että apumuuttuja on määriteltävä foreach lauseessa.

foreach lauseen syntaksi on:

```
foreach(tyyppi apumuuttuja in taulukon_nimi)
{
  //suoritettavia lauseita
}
```

12.1: Esimerkkiohjelma foreach lauseesta

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ForeachHarjoitus
{
    class Program
    {
        static void Main(string[] args)
        {
            string lause;
            string[] sanat;
            do
            {
                int laskuri = 0;
                Console.WriteLine("Anna paloiteltava
lause. Anna quit, kun haluat lopettaa.");
                lause = Console.ReadLine();
                Console.WriteLine();
                sanat = lause.Split(); //paloitellaan
                                       //lause sanoiksi
                foreach (string sana in sanat) //käydään
                                       //taulukko sanat läpi
                {
                    Console.WriteLine(sana);
                }
                Console.WriteLine("\nSanoja oli " +
sanat.Length + " kappaletta.\n");
            }
```

```
            while (lause != "quit");
        }
    }
}
```

Edellisessä esimerkssä käytiin paloiteltiin lause string taulukoksi,ja käytiin läpi. foreach lauseella voidaan yhtä hyvin käydä läpi char tai numeerisia talukoita.

```
Esim. double [] a={1.6,2.1,4,3};
      foreach (double apu in a)
      {
      //suoritettavat lauseet
      }

tai: char [] kirjain={'a','k','s'};
     foreach (char apukirjain in kirjain)
     {
      //suoritettavat lauseet
     }
```

13 Ohjausmerkit

Ohjausmerkit, jotka toimivat tulostuslauseissa ja -kentissä. Huomaathan, että kun kenoviiva \ on itsessään merkki ohjauksesta, niin jos sinun on tulostettava kenoviiva, joudut kirjoittamaan sen tuplana, eli \\ . Samoin, kun lainausmerkki ” ja heittomerkki ' sisältävät ohjaustietoa, niiden tulostamiseen tarvitset eteen kenoviivan, joka kumoaa ohjaustarkoituksen.

13.1: Taulukko ohjausmerkit

Ohjausmerkki	***Kuvaus***
\'	Heittomerkki
\”	Lainausmerkki
\\	Kenoviiva
\0	Null-merkki (ei ole sama kuin C Sharpin null arvo)
\a	Varoitus (Alert)
\b	Askelpalautus
\f	Sivun kelus (Form feed)
\n	Rivin vaihto
\r	Vaunun palautus (Carriage return)
\t	Vaakasuorainen sarkain
\v	Pystysarkain

14 Päivämäärän ja ajan ottaminen tietokoneen kellosta

Seuraava ohjelma tulostaa päivämäärän ja kellonajan tietokoneen kellosta otettuna. Joskus voi olla tarpeen käyttää myös millisekunteja, joiden tulostamisesta on myös esimerkissä.

14.1: Esimerkkiohjelma aikamäärästä

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Aikamaara
{
    class Program
    {
        static void Main(string[] args)
        {
            int vuosi, kuukausi, paiva, tunti, minuutti,
sekuntti, millisekuntti;
            System.DateTime aika = System.DateTime.Now;
            vuosi = aika.Year;
            kuukausi = aika.Month;
            paiva = aika.Day;
            tunti = aika.Hour;
            minuutti = aika.Minute;
            sekuntti = aika.Second;
            millisekuntti = aika.Millisecond;
            Console.WriteLine("Päivämäärä on " + paiva +
"." + kuukausi + "." + vuosi);
            Console.WriteLine("Kellonaika on " + tunti +
"." + minuutti + "." + sekuntti);
            Console.WriteLine("Millisekunnit ovat: " +
millisekuntti);

   //päivämäärän ja kellonajan voi tulostaa myös näin
   //silloin mitään yllä olevia määritelmiä ei tarvita
   //vaan tämä toimii itsenäisenä
            Console.WriteLine("\nNyt on " +
DateTime.Now.ToString());
```

```
            Console.Write("\nPaina jotain näppäintä jatkaaksesi...");
                Console.ReadKey(true);
            }
        }
    }
```

15 Matemaattisia operaatioita

C#:ssa on Math luokka, jossa on paljon hyödyllisiä matemaattisia toimintoja, kuten potenssiin korotus, neliöjuuri ja trigonometriset funktiot. Siellä on myös piin ja luonnollisen luvun e arvot.

15.1: Esimerkkiohjelma matemaattisista operaatioista

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MatematiikkaToimintoja
{
    class Program
    {
        static void Main(string[] args)
        {
            //piin arvon tulostus
            double pii=Math.PI;
            Console.WriteLine("Piin arvo on " + pii);

            //luonollisen luvun e tulostus
            double e=Math.E;
            Console.WriteLine("Luonollisen luvun e arvo
on " + e);

            //neliöjuuri lasketaan seuraavasti
            double nelJuuri=Math.Sqrt(12);
            Console.WriteLine("Luvun 12 neliöjuuri on " +
nelJuuri);

            //potenssiin korotus tapahtuu seuraavasti
            double a=3, b=4, r=14;
            double pot = Math.Pow(a, b);
            Console.WriteLine("3^4 (3 potenssiin 4) on "
+ pot);
            Console.WriteLine("Ympyrän, jonka säde on 14
(kaava pii*r^2), pinta-ala on " + Math.PI * Math.Pow(r,
2));

            //trignometriset funktiot
            //luodaan vakio n, jolla desimaaliluvun saa
            //radiaaneiksi
```

```
            //C# käyttää trigonometrisissa laskuissa
            //radiaaneja
            const double n = Math.PI / 180;

            //sini
            double sin = Math.Sin(n*30);
            Console.WriteLine("Luvun 30 sini on " + sin);

            //cosini
            double cos = Math.Cos(n*40);
            Console.WriteLine("Luvun 40 cosini on " +
cos);

            //tangentti
            double tan=Math.Tan(n*80);
            Console.WriteLine("Luvun 80 tangentti on " +
tan);

            //arkusfuktiot
            //arkus sini eli sin -1
            double sinM = Math.Asin(0.8)/n;
            Console.WriteLine("Kulma sinin arvolle 0,8 on
" + sinM);

            //arkus cos eli cos -1
            double cosM = Math.Acos(0.25)/n;
            Console.WriteLine("Kulma cosinin arvolle 0,25
on " + cosM);


            //arkus tan eli tan -1
            double tanM = Math.Atan(8)/n;
            Console.WriteLine("Kulma tangentin arvolle 8
on " + tanM);

            //luonnollinen logaritmi
            double ln = Math.Log(3);
            Console.WriteLine("Luonnollinen logaritmi
luvulle 3 on " +ln);

            //kantaluku kantainen logaritmi
            //syntaksi Math.Log(kantaluku, numerus)
            double log = Math.Log(3,6);
            Console.WriteLine("Logaritmi 3 luvulle 6 on "
+ log);
```

```
            //kymmen kantainen logaritmi
            //syntaksi Math.Log10(numerus)
            double x = Math.Log10(8);
            Console.WriteLine("Logaritmi 10 luvulle 8 on
" + x);

            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);
        }
    }
}
```

16 bool tyyppi

bool tyypin muuttuja saa arvon joko true (tosi) tai false (epätosi).

Esimerkki:

bool n, m;
n = true;
m=false;

.NET muoto on isolla kirjaimella kirjoitettuna Boolean

bool tyypin koko muistissa on yksi tavu.

Esimerkki 16.1: bool tyyppi ja ehto operaattori ?

```
bool vastaus;
int x=2, y=4;
vastaus = (x <= y) ? true : false; //vastaus saa arvon
true
```

17 Satunnaisluku

Satunnaisluku luodaan Random luokasta new avainsanalla, ja sitä käytetään Next() metodilla. Alla oleva esimerkki valaisee asiaa. Next() metodille anntetaan kaksi lukua, joiden välistä sanuttaisluvut arvotaan. Lukujen on oltava kokonaislukuja, mutta miinusmerkkiset kelpaavat. Ensimmäisen luvun on oltava pienempi kuin toisen. Ohjelma ottaa ensimmäisen arvotun luvun käyttöön, ja arpoo siitä toiseen asti, eli suurenpaa lukua ei enään oteta mukaan. Next(1,4) arpoo lukuja 1,2,3;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Satunnaisluku
{
    class Program
    {
        static void Main(string[] args)
        {
            int i;
            Random satluku = new Random(); //Luodaan
             //satunnaisluku olio satluku
             //Next käskyllä arvonaan satunnaisluku annetulta
             //väliltä
            for (i=0;i<40;i++) //yhden rivin for lausekkeessa
                               //ei tarvita {} merkkejä
                Console.WriteLine(+satluku.Next(0,7));
                 //tulostaa 40 satunnaislukua
                //Huomaa, ensimmäinen luku otetaan mukaan,
                //muttei toista
                //Eli tätä arpoo luluka 0,1,2,3,4,5,6

            Console.Write("\nPaina jotain näppäintä
jatkaaksesi...");
            Console.ReadKey(true);
        }
    }
}
```

18 Hyödyllisiä linkkejä

Microsoftin .NET ohjelmointisivut: http://msdn.microsoft.com/en-us/library .
Microsoftin .NET keskustelufoorumi: http://social.msdn.microsoft.com/Forums/en-US/categories .

Visual Studio 2010 latausosoite: http://www.microsoft.com/express/Downloads/

Novellin Mono kehitysympäristö: http://www.mono-project.com/Main_Page

Omat sivuni: www.netti-lakka.com

Ohjelmointisivuni: www.ohjelmoimaan.net

19 Lähteet

-Ammattiopisto koulutus: OSAO
-Programming C#: Liberty, Jesse
-C#-ohjelmointi: Moghadampour, Ghodrat
-Microsoftin C# sivut: http://msdn.microsoft.com/en-us/library
-MSX-assembler ja -konekieli: Ian Sinclair