Muutujad

Muutujad on mälu lingid, kus hoitakse programmi töö aeg andmeid. Algul tuleb muutuja deklareerida, mille peale operatsioonisüsteem hõivab teatud palju mälu. See tähendab seda, et sellesse mäluruumi ei kirjutata enam midagi peale selle muutuja väärtuse.

Muutujatüübid

Muutujatüübid määravad ära, mis laadi andmetega on tegu, mida mälus hoitakse ning kui palju ruumi tuleb eelnevalt reserveerida.

Integraal tüüpi muutujad (Integral type)

Integraal tüüpi muutujad on täisarvulised muutujad. Vastavalt siis kas märgiga või märgita.

Tüüp	Suurus (bittides)	Võimalikud väärtused
sbyte	8	-128 kuni 127
byte	8	0 kuni 255
short	16	-32768 kuni 32767
ushort	16	0 kuni 65535
int	32	-2147483648 kuni 2147483647
uint	32	0 kuni 4294967295
long	64	-9223372036854775808 kuni 9223372036854775807
ulong	64	0 kuni 18446744073709551615
char	16	0 kuni 65535

Näide integraal tüüpi muutuja deklareerimisest ja väärtustamisest:

```
int minuArv = 10;
```

Antud osa koodist hõivab 32 bitti mälu, ehk 4 baiti ning kirjutatakse sinna väärtus 10.

Ujukoma tüüpi muutujad (float type)

Ujukoma tüüpi muutujaid kasutatakse reaalarvude mälus hoidmiseks ja murdarvudega arvutamiseks.

Tüüp	Suurus (bittides)	Täpsus	Võimalikud väärtused
float	32	7 numbrit	1.5×10^{-45} kuni 3.4×10^{38}
double	64	15-16 numbrit	5.0×10^{-324} kuni 1.7×10^{308}
decimal	128	28-29 kümnendkohta	1.0×10^{-28} kuni 7.9×10^{28}

Näide ujukoma tüüpi muutuja deklareerimisest ja väärtustamisest:

```
float minuArv = 4.7;
```

Antud osa koodist hõivab 32 bitti mälu, ehk 4 baiti ning kirjutatakse sinna väärtus 4.7.

Tekstimuutuja (string type)

Tekst on tähemärkide ehk char tüüpi muutujate jada. Mälus hoitakse tegelikult arvu, mis vastab teatud tähemärgile. Teksti hoitakse peamiselt jutumärkide vahel.

Näide tekstimuutuja deklareerimisest ja väärtustamisest:

```
string nimi = "Tore Inimene";
```

Antud osa koodist kirjutab mällu tähemärkide jada, mis koosneb 12-st tähemärgist, sest ka tühik on tähemärk: Tore [tühik] Inimene.

Avaldised

Aritmeetiline avaldis

Aritmeetiline avaldis koostatakse arvtüüpi objektist ning aritmeetilisest tehtemärgist. Kõige tüüpilisemad tehted on liitmine, lahutamine, korrutamine ning jagamine. Arvutused sooritatakse tehete prioriteetide järgi nii nagu aritmeetikas kombeks.

Aritmeetilised operaatorid(operatsioonide tähtsuse järjekorras)	
Nimetus	**Operandi märk**
märgi muutmine	-
korrutamine	*
jagamine	/
mooduliga jagamine	%
nihutamine paremale	>>
nihutamine vasakule	<<
liitmine	+
lahutamine	-

Näited aritmeetiliste avaldistega:

```
int a = 4;
int b = 5;
int c = a + b;
```

Algul deklareeritakse muutuja nimega a, mis on int tüüpi ning siis kohe väärtustatakse see arvuga 4.
Peale seda deklareeritakse samamoodi muutuja nimega b ning väärtustatakse arvuga 5.
Viimaseks deklareeritakse muutuja nimega c ning väärtustatakse muutuja a ja b summaga. Muutuja c väärtuseks saab 9.

```
int a = 4;
int b = a + a * a;
```

Deklareeritakse muutuja nimega a ning väärtustatakse see arvuga 4.
Deklareeritakse muutuja nimega b ning väärtustatakse see 20-ga, sest 4 + 4 * 4 = 20 (4 * 4 = 16; 16 + 4 = 20).

Loogiline avaldis

Loogiline avaldis sisaldab loogika operaatorit. Kuid samas võib avaldis sisaldada ka aritmeetilisi avaldisi. Loogilise avaldise tulemus on tõeväärtus, ehk siis, kas avaldis on tõene või väär. Seda tüüpi avaldisi saab kasutada teatud tingimuste kontrollimiseks. Kui teatud tingimus on tõene või väär, siis vastavalt sellele käitub programm edasi.

Loogilised operaatorid	
Nimetus	**Operandi märk**
Loogiline eitus (NOT)	!
Loogiline korrutamine (AND)	&
Loogiline liitmine (OR)	|
Mitteekvivalentsus (XOR)	^
Tingimuslik korrutamine (AND)	&&
Tingimuslik liitmine (OR)	||
Võrdsus	==
Mittevõrdsus	!=
Suurem kui	>
Suurem või võrdne kui	>=
Väiksem kui	<
Väiksem või võrdne kui	<=

Näiteid		
Loogiline avaldis	**Tagastusväärtus**	**Selgitus**
!tõene	väär	tõese eitamine
tõene && väär	väär	tõene NING väär (1 * 0 = 0)
tõene || väär	tõene	tõene VÕI väär (1 + 0 = 1)

Näide koodi baasil:

```
bool olemas = true;
bool poleOlemas = false;

olemas && poleOlemas  //tagastusväärtus on
false (1 * 0 = 0)
```

Tingimuslaused

Tingimuslauseid kasutatakse programmikoodi juhtimiseks või öeldes siis, et programmi käitumise juhtimiseks. Tingimustega saame määrata mida programm peaks edasi tegema, kui mõni tingimus on täidetud või vastupidi, et tingimus ei ole täidetud.

IF...ELSE

If Else on kahe teeline tingimus. Mis tähendab, et kui tingimus on tõene, siis täidetakse üks osa koodist ning kui tingimus on väär, siis täidetakse hoopis teine osa koodist.
Üleskirjutus on vastav:

```
if(<tingimus>){
   <kood, mis täidetakse, kui tingimus on tõene>
} else {
   <kood, mis täidetakse, kui tingimus ei ole tõene>
}
```

Else osa ei ole kohustuslik. Kui välja jätta else, siis täidetakse kood ainult siis kui tingimus on tõene. Kui programm on tingimuse blokist väljunud, siis täidetakse edaspidi kõik käsud, olenemata eelnenud tingimusest.

```
int a = 5;
int b = 6;

if( a + b == 11)
{
   Console.WriteLine("Tõsi");
}
Console.WriteLine("Väljun");
```

Tulemus:

```
Tõsi
Väljun
```

Antud juhul trükitakse konsooli *Tõsi* kui ka *Väljun*, sest *Väljun* osa ei ole enam tingimusest sõltuv vaid ta täidetakse olenemata tingimusest.

Kui on vaja tingimuse peale täita ainult üks käsk, siis ei pea olema ka loogelisi sulge, mis määravad tingimusbloki. Kompilaator teab, et loogeliste sulgude puudumisel tuleb tingimusega seostada ainult üks käsk. Kõik teised käsud täidetakse tingimusest olenemata.
Järgnevalt on näide, mis on samaväärne eelmisega:

```
int a = 5;
int b = 6;
```

```
if( a + b == 11)
   Console.WriteLine("Tõsi");

Console.WriteLine("Väljun");
```

Tulemus:

```
Tõsi
Väljun
```

Else'i kasutamise näide:

```
int a = 6;
int b = 7;

if( a > b )
{
   Console.WriteLine("Tõsi");
} else {
   Console.WriteLine("Väär");
}
```

Väljund:

```
Väär
```

Programmi väljundiks on *Väär*, sest 6 ei ole seitsmest suurem ning täidetakse ainult *else* osa koodist.

SWITCH

Switch on juhtlause, mis juhib programmi käitumist just nagu If, aga antud juhul tehtakse loend sellest, mis tingimusel teatud koodi osa täidetakse. Erinevus on selles, et enam ei kirjutata tingimus sulgudesse vaid väärtus mida tahetakse kontrollida. Loogeliste sulgude vahele kirjutatakse mis väärtuse korral mingi osa täidetakse.

```
int lyliti = 1;

switch (lyliti)
{
    case 1:
        Console.WriteLine("Valitud on 1");
        break;
    case 2:
        Console.WriteLine("Valitud on 2");
        break;
    default:
        Console.WriteLine("Midagi ei olnud
valitud");
        break;
```

```
}
```

Väljund:

```
Valitud on 1
```

Antud programm otsib loendist vastet muutujale *lyliti*, mis on väärtusega 1. Esimene juhtum (case) ongi 1 ja programm hakkab seda osa koodist täitma. Selle tulemusena trükitakse konsooli *Valitud on 1*. Edasi minnes tuleb tähele panna, et on kirjutatud käsk *break;*, see tähendab seda, et enam ei otsita edasi ning väljutakse antud blokist.

Kui ei leita ühtegi sobivat vastet siis täidetakse *default:* osa. Sedasi toimib alljärgnev kood:
Kasutaja sisestab konsooli arvu 3.

```
Console.WriteLine("Kommi tüüp: 1=Magus
2=Hapu");
Console.Write("Palun tee valik: ");
string s = Console.ReadLine();
int kommiSort = int.Parse(s);

switch (kommiSort)
{
    case 1:
        Console.WriteLine("Soovid magusat
kommi");
        break;
    case 2:
        Console.WriteLine("Soovid haput
kommi");
        break;
    default:
        Console.WriteLine("Sellist kommi
ei ole");
        break;
}
```

Väljund:

```
Kommi tüüp: 1=Magus 2=Hapu
Palun tee valik: 3
Sellist kommi ei ole
```

Tsüklid

Tsüklid lasevad programmil täita ühte koodi osa mitu korda järjest. Kui me tahame ekraanile trükkida arvud ühest kuni tuhandeni, siis ei ole mõistlik kirjutada tuhat rida koodi, kus iga rea peal väljastame ainult ühe arvu. Sellist ülesannet on kõige parem lahendada tsükliga.

WHILE tsükkel

While tsüklit täidetakse nii kaua, kui sulgude sees olev tingimus on tõene. Antud kordus teostab eelkontrolli, mis tähendab, et bloki sees olevat osa läbitakse ainult siis, kui tingimus on täidetud. Korduse üleskirjutus on järgnev:

```
while(<tingimus>)
{
   <täidetav koodi>
}
```

Näide:

```
int a = 0;

while( a < 5 )
{
   Console.WriteLine(a);
   a++;
}
```

Algul väärtustatakse muutuja a arvuga 0. Seejärel tehtakse kontroll, kas a on väiksem, kui 5. Antud näite puhul ta seda tõesti on ja täidetakse tsükli osa, kus trükitakse a väärtus konsooli ning suurendatakse a'd ühe võrra. Nüüd on a väärtus 1. Ja tsükkel algab otsast peale tingimuse kontrollimisega. Kordust täidetakse nii kaua, kuni a enam ei ole 5-st väiksem, ehk siis on võrdne 5-ga.

Väljund:

```
0
1
2
3
4
```

DO tsükkel

Do Loop on sarnane While tsükliga, aga erinevus on selles, et tingimust kontrollitakse bloki lõpus. See tähendab seda, et blokis olev kood käidakse läbi vähemalt ühe korra ning täidetakse seal olevad käsud. Üleskirjutus on järgnev:

```
do
```

```
{
   <koodi mida korrata, aga täidetakse
vähemalt üks kord>
}while( <tingimus> );
```

FOR tsükkel

For tsükkel on eelkontrolliga tsükkel, kus väärtustamised ja tingimused on võimalik märkida ühes kohas. Üleskirjuts on järgnev:

```
for( <tegevus enne tsükli alustamist>;
<tingimus>; <tegevus, mis täidetakse
tsükli lõpus>)
{
   <kood, mis täidetakse, kui tingimus on
tõene>
}
```

Näide:

```
for(int i = 0; i < 5; i++)
{
   Console.WriteLine( i );
}
```

Algul väärtustatakse muutuja i arvuga 5. Seejärel kontrollitakse, kas i on väiksem 5-st. Antud näite puhul on see nii ja mindakse edasi blokis olevat koodi täitma. Selle tulemusena trükitakse konsooli *0* ja jõutakse kohe bloki lõppu ja suurendatakse i-d ühe võrra ja kontrollitakse uuesti tingimust. Nii käib see tsükkel kuni i enam ei ole väiksem 5-st ja mindakse programmi koodiga edasi.

Väljund:

```
0
1
2
3
4
```

FOREACH tsükkel

Foreach teebki sellise tsükli nagu selle otsetõlge ütleb. Mingi massiivi kõik elemendid käidakse ükshaaval läbi ning tehtakse nendega siis mingi soovitud toiming. Üleskirjutus on järgnev:

```
foreach (<objekti tüüp> <ajutine nimetus>
in <massiivi nimi>)
```

```
{
   <toiming antud ajutise objektiga>
}
```

Näide:

```
string[] nimed = {"Mari", "Kalle",
"Toomas", "Sille" };

foreach( string nimi in nimed )
{
   Console.WriteLine("{0} on tore
inimene.", nimi);
}
```

Esmalt loodakse string tüüpi massiiv, millesse lisatakse neli nime. Edasi toimub *foreach* tsükkel. Sulgudesse on kirjutatud objekti tüüp, mis on sama massiivi elemetide tüübiga ehk *string*. Selle järel on kirjutatud elemendi nimi mida hakatakse tsüklis kasutama ja siis tuleb massiivi nimi, mille elemendid kõik ükshaaval läbi käidakse. Esimena antakse muutujale *nimi* väärtus *Mari* (string nimi = "Mari"). Toimub konsooli kirjutamine, kus kirjutatakse muutuja *nimi* väärtus ja selle järele *on tore inimene*. Kui jõutakse tsükli lõppu, siis mindakse algusesse tagasi ja võetakse massiivist järgmine väärtus ("Kalle") ja tehtakse sama asi uuesti läbi. Tsükkel kestab kuni on kõik elemendid läbi käidud.

Väljund:

```
Mari on tore inimene.
Kalle on tore inimene.
Toomas on tore inimene.
Sille on tore inimene.
```

Massiivid

Massiive kasutatakse rohkemate andmete mälus hoidmiseks. Näiteks kui ma tahame mälus hoida ainult ühte arvu, siis piisab ühest muutujast ning selle väärtustamisest. Aga kui on vaja mälus hoida suuremas koguses andmeid, siis on mugavam neid käsitleda ja töödelda, kui nad on massiivis. Massiivid on 0 indekseeritud, mis tähendab, et massiivielementide adresseerimine algab nullist.

Massiiv deklareeritakse kantsulgude ([]) abil, need kirjutatakse kohe muutujatüübi järele.
Ilma kindla suuruseta massiivi deklareerimine:

```
string[] nimed;
```

Kahemõõtmelise massiivi deklareerimine käib komaga. Koma paigutatakse kantsulgude sisse:

```
int[,] arvud;
```

Võimalik on ka massiivi panna teise massiivi:

```
int[][] teisedArvud;
```

Kuna C#-s on massiivid tegelikult objektid, siis tuleb need massiivid tegelikult luua:

```
//Ühemõõtmelise massiivi loomine, mille
suurus on neli elementi.
string[] nimed = new string[4];

//Kahemõõtmelise massiivi loomine, mille
suurus on kuus korda viis.
int[,] arvud = new int[6, 5];

//Massiivide massiivi loomine, millel on
kolm elementi, aga sisaldavad teisi
massiive, kus sees võib olla palju rohkem
elemente.
int[][] teisedArvud = new int[3][];
```

Ühemõõtmelise massiivi kasutamise näide, kus iga element kohe algul väärtustatakse:

```
int[] lotoNumbrid = new int[5]{ 10, 6, 22,
49, 12 };

Console.WriteLine(lotoNumbrid[4]);
Console.WriteLine(lotoNumbrid[2]);
```

Loodakse massiiv, mille suurus on viis elementi ja algväärustatakse need elemendid siis vastavate arvudega. Edasi toimub konsooli trükkimine, kus trükitakse välja *lotoNumbrid* viies element ning seejärel kolmas element. Algul võib tunduda, et midagi on valesti, aga tegelikult on kõik õige, sest adresseerimine algab nullist, mis tähendab, et esimese elemendi välja trükkimiseks peaks olema kantsulgudes 0.

Väljund:

```
12
22
```

Kahemõõtmelise massiivi andmetele ligi pääsemiseks tuleb käituda sama moodi nagu ühemõõtmelise massiivi puhul, aga erinevus seisneb adresseerimises. Nüüd tuleb massiivi elemendile ligipääsemiseks kasutada kahte indekseerijat, teisiti öeldes adresseerijat.

```
int yksArv = arvud[2,3];
```

Peale sellist käsklust on muutuja *yksArv* väärtus sama, mis oli *arvud* elemendil aadressil 2, 3.

Koostas Mikk Pärast