howest
hogeschool

Lussen

Programming Basics

Graduaat in het Programmeren

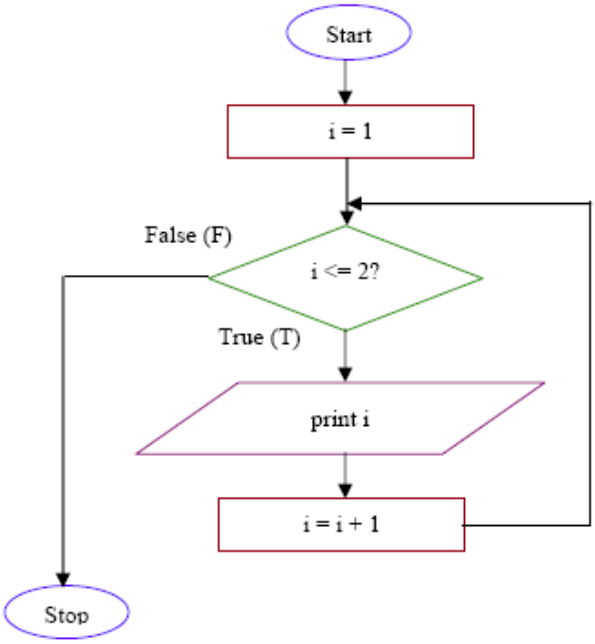
INHOUD

1 LUSSEN

- Maak een nieuwe Visual Studio solution Lussen.
- Maak in de solution Lussen een nieuwe WPF App Lussen.Wpf.

1.1 INLEIDING

Een programmalus wordt gebruikt wanneer een statement of een groep statements een aantal maal herhaald moet worden. In de afbeelding hieronder wordt dit visueel voorgesteld:



Lussen vormen een basisstructuur in om het even welke programmeertaal. Daardoor is het uitermate belangrijk om ze goed onder de knie te hebben.

Voorbeelden waarbij gebruik gemaakt wordt van lussen zijn:

- Een (variabele) boodschap moet een aantal keer op het scherm gezet worden
- Een getallenreeks wordt overlopen bij bijvoorbeeld het zoeken naar delers van een getal
- Een reeks waarden dient gesorteerd te worden
- De namen van alle bestanden in een map moeten getoond worden
- Een tekstbestand moet regel per regel worden ingelezen
- Records uit een databasetabel moeten overlopen worden
- ...

1.1 SOORTEN LUSSEN

1.1.1 WHILE

Een while-lus kan gebruikt worden om statements uit te voeren zolang een bepaalde voorwaarde waar (true) is.

```
while(booleaanse expressie)
{
       statement(s);
}
```

- De expressie moet een boolean als resultaat geven.
- De expressie moet tussen ronde haken genoteerd worden.
- Als de expressie al direct als false geëvalueerd wordt, worden de statements niet uitgevoerd.
- Als er meer dan één statement uitgevoerd moet worden, dan plaats je deze tussen accolades .

Voorbeeld: Een lus van 0 tot en met 5, waarde van de teller wordt getoond:

```
int counter = 0;
string result = "";
while(counter < 6)
{
    Console.WriteLine("Teller is nu gelijk aan " + counter);
    result += "Teller: " + counter + "\n";
    counter++;
}
MessageBox.Show(result);
```

1.1.1.1 VOOBEELD

DE COMPUTER EEN GETAL LATEN RADEN

```
Random random = new Random(); /* Deze variabele zal gebruikt kunnen worden om een
willekeurig getal te genereren */

const int MAX_NUMBER = 10000;


public MainWindow()

{

    InitializeComponent();

}


private void Guess_Click(object sender, RoutedEventArgs e)

{

    int toGuess;

    int guess = -1;

    int nrOfTries = 0;


    toGuess = int.Parse(txtToGuess.Text);


    while (guess != toGuess)

    {

        guess = random.Next(1, MAX_NUMBER + 1);

        lstGuesses.Items.Insert(0, guess); /* Met Insert voeg je iets toe op een
bepaalde plaats in een lijst */

        nrOfTries++;

    }

    lblFeedback.Content = $"Het getal {toGuess} is geraden in {nrOfTries} pogingen";

}
```

Code repository

De volledige code van dit voorbeeld is te vinden op

```
git clone https://github.com/howest-gp-prb/cu-lussen-laat_de_computer_raden.git
```

1.1.2 FOR

Via een for-lus kunnen we code telkens opnieuw laten uitvoeren aan de hand van een teller tot die teller een bepaalde waarde bereikt heeft. Elke keer dat de lus doorlopen wordt, wordt de teller verhoogd met een vaste waarde.

We vragen de computer dus:

- Om te tellen van getal x tot y
- in stappen van een bepaalde grootte (meestal 1)
- en bij elke tel bepaalde code uit te voeren

In het voorbeeld hieronder laten we de computer van nul tot en met vijf tellen in stappen van één

```
string result = "";

for (int i = 0; i<= 5; i++)
{
    result += "De teller i is nu gelijk aan: " + i + Environment.NewLine;
}

MessageBox.Show(result);
```

Door het for-statement als volgt aan te passen

```
for (int i = 0; i<= 10; i += 2)
```

wordt er van nul tot en met tien geteld in stappen van twee.

Het start- en eindgetal kan ook afhankelijk zijn van een variabele of het resultaat van een functie:

```
string word = "programmeren";
string result = "Het woord " + word + " bevat de volgende letters:" +
Environment.NewLine;

for (int i = 0; i < word.Length; i++)
{
    result += i + " - " + word[i] + Environment.NewLine;
}

MessageBox.Show(result);
```

Let hier op het gebruik van < (kleiner dan) i.p.v <= (kleiner of gelijk). Als je een string van lengte 3 wil overlopen, heeft die tekens op posities 0, 1 en 2. Met woord[i] kan je dan het teken op positie i van een string opvragen. Dit is dezelfde notatie die we gebruiken bij lijsten, je kan een string namelijk beschouwen als een lijst van tekens.

1.1.3 FOREACH

Bij arrays en collections kunnen we alle elementen overlopen en bij elk element code laten uitvoeren.

In het laatste voorbeeld van de for-lus zouden we de letters van een woord kunnen opvragen via een foreach-lus. Een string is namelijk een collection van characters.

```
string word = "programmeren";
string result = "Het woord " + word + " bevat de volgende letters:" +
Environment.NewLine;

foreach (char letter in word)
{
    result += letter + Environment.NewLine;
}

MessageBox.Show(result);
```

In het volgende voorbeeld doorlopen we een array:

```
string result = "";
string[] seasons = { "lente", "zomer", "herfst", "winter" };

foreach (string season in seasons)
{
    result += season + Environment.NewLine;
}

MessageBox.Show(result);
```

Ook een list kan via een foreach-lus doorlopen worden:

```
string result = "Het dubbel van de getallen van nul tot en met tien:\n";
List<int> numbersTillTen = new List<int>();

for (int i = 0; i <= 10 ; i++)
{
    numbersTillTen.Add(i);
}

foreach (int number in numbersTillTen)
{
    result += "Het dubbel van " + number + " = " + number * 2 + Environment.NewLine;
}

MessageBox.Show(result);
```

1.1.4 DO

Bij while- en for-lussen wordt de voorwaarde bij de start van de lus getest. Als aan de voorwaarde van bij het begin niet voldaan is, worden de statements binnen de lus nooit uitgevoerd.

Een do-lus bevat de voorwaarde **na** de statements:

```
string result = "Tellen van één tot vijf:\n";
int i = 1;
do
{
    result += "De teller i is nu gelijk aan: " + i + Environment.NewLine;
    i++;
} while (i <= 5);
MessageBox.Show(result);
```

De lusstatements worden hier dus minstens één keer uitgevoerd.

In andere programmeertalen bestaat soms de do … until-lus. In C# bestaat die niet, maar uiteindelijk komt zo'n lus overeen met een do …. while zolang een statement false is. In pseudocode zou je dus kunnen zeggen dat

```
teller = 0
do
      verhoog teller met 1
until teller == 5
```

op hetzelfde neerkomt als

```
teller = 0
do
      verhoog teller met 1
while teller != 5
```

1.1.5 BREAK

Normaal gezien wordt een lus doorlopen tot alles overlopen is (einde array of collection, bereiken eindwaarde in for-lus, niet meer voldoen aan een bepaalde conditie bij do en while lus).

Het is mogelijk om de lus vroegtijdig te onderbreken als aan een bepaalde voorwaarde voldaan wordt. Hiervoor kunnen we een if-statement gebruiken. Een toepassing hiervan is het doorlopen van een reeks (array, list …) tot er een bepaalde waarde gevonden is.

Zo zoeken we in onderstaand voorbeeld in de array *namen* tot er een bepaalde naam is gevonden. Eens die gevonden is, heeft het geen zin om verder te zoeken in de lijst. We gebruiken hier een **break**.

```
int FindName(string player)
{
    int index = -1;
    foreach (string name in names)
    {
        index++;
        if (name == player)
        {
            break;
        }
    }
    return index;
}
```

Het gebruik van een **break** wordt in veel situaties echter afgeraden. Voor iemand die je code leest, kan het namelijk onduidelijk zijn in welke gevalen de lus zal stoppen. Bovenstaand voorbeeld schrijf je dan ook beter met een while lus. Zoals je ziet is de lus een stuk eenvoudiger geworden.

```
int FindName(string player)
{
    int index = 0;
    while (index < names.Length && names[index] != speler)
    {
        index++;
    }

    if (index == names.Length)
    {
        /*
            * dit betekent dat de naam niet gevonden werd in de array,
            * er is namelijk geen element op index namen.Length
        */
        index = -1;
    }

    return index;
}
```

1.1.6 CONTINUE

Met een **continue**-statement wordt de code in de lus onderbroken. Als aan de gestelde voorwaarde voldaan wordt (dat je kan controleren met een if-statement), wordt de code die volgt op de continue niet uitgevoerd. Als de eindvoorwaarde nog niet bereikt is, gaat de lus echter gewoon verder. Een continue-statement kun je dus gebruiken als je een lus nodig hebt, waarbij code uitgevoerd moet worden **behalve** in bepaalde gevallen.

Bvb: Je zou een bepaalde getallenreeks één voor één kunnen overlopen om de getallen die deelbaar zijn door drie te zoeken. Bij elk getal wordt de tekst '*x is deelbaar door drie*' getoond. Indien de modulo van een getal niet gelijk is aan nul, zou een continue-statement ervoor kunnen zorgen dat de tekst niet getoond wordt.

```
void ShowTripples()
{
    for (int i = 0; i < 100; i++)
    {
        if (i % 3 != 0)
        {
            continue;
        }
        Console.WriteLine($"{i} is deelbaar door 3");
    }
}
```

Het gebruik van een **continue** statement wordt echter ook afgeraden aangezien het de flow van je programma (die vastlegt wat het volgende statement is die wordt uitgevoerd) een stuk ingewikkelder maakt. Bovenstaande code kun je dan ook veel beter herschrijven naar het volgende stukje dat exact hetzelfde doet.

```
void ShowTripples()
{
    for (int i = 0; i < 100; i+=3)
    {
        Console.WriteLine($"{i} is deelbaar door 3");
    }
}
```

1.1.7 SCOPE VAN VARIABELEN BINNEN EN LUS

Een variabele die binnen een lus gedeclareerd wordt, is niet meer toegangkelijk buiten de lus. Buiten de lus kan de variabele niet gebruikt worden. In het volgende voorbeeld zou je bij de Console.WriteLine-statements dus telkens een foutmelding krijgen:

```
for (int i = 0; i<10; i++)
{
    string number = i.ToString();
}
Console.WriteLine(i);
Console.WriteLine(number);
```

howest.be