**H5: BESLISSINGEN – INTRO**

Tot nu toe waren al onze programma’s lineair in opbouw

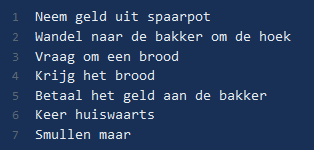
* Ze werden lijn per lijn uitgevoerd, van start tot einde (en afsluit)
* Er was geen mogelijkheid om de **program flow** van het programma aan te passen
* Maw: onze programma’s waren een simpele lijst met opdrachten die achter elkaar uitgevoerd werden.

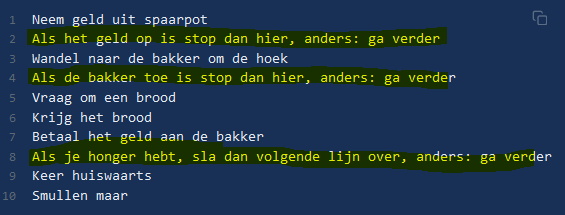
Probleem: als één van de stappen faalt in de “lijst”, STOPT het programma met werken.

Oplossing: Een algoritme dat **foutgevoeliger** én **interactief** is voor de eindgebruiker, door de mogelijkheid te bieden om **alternatieve stappen** te ondernemen **afhankelijk** van de **omstandigheden**

* **Het programma zal beslissingen maken gebaseerd op keuzes**

Vb:





**Programmaflow**:

3 basistypes:

1. ‘straight line’ (wat we eerst deden)
2. Gekozen naargelang een *gegeven voorwaarde* (if/else)
3. Herhaalt zolang een voorwaarde geldt (loops)

* Met deze 3 types kun je zowat elk programma maken dat je wilt!!!

**H5: RELATIONELE EN LOGISCHE OPERATORS**

Nieuwe soort operators (zoals +, -, …) nodig:

* Operators waarmee we kunnen testen of iets WAAR of NIET WAAR is
* C# kan dan een actie uitvoeren bij de ene, en geen actie uitvoeren bij de andere, of stap overslaan
* **Relationele operators & logische operators**

BOLEAANSE EXPRESSIES

* Een expressie/stuk code die een *bool* als resultaat zal geven.
* De relationele & logische operatoren zullen altijd een *bool* teruggeven

RELATIONELE OPERATORS

* Aan het hart van de booleaanse expressies
* Simpel en je kent ze al
* Ook vergelijkingsoperators genoemd

“>” = groter dan

“<” = kleiner dan

“==” = gelijk aan (NIET hetzelfde als “=”)

“!=” = niet gelijk aan

“<=” = kleiner of gelijk aan

“>=” = groter of gelijk aan

* Altijd 2 operanden nodig van HETZELFDE datatype
* Het resultaat is altijd ***bool***

Vb:

*Bool isKleiner = 65 > 67;   
Console.WriteLine(isKleiner):*

* *Output: false*

LOGISCHE OPERATORS

* Voor meer complexe keuzes (waar bvb 2 keuzes gemaakt moeten worden: “ga verder indien ik honger heb EN genoeg geld bij heb”)
* 3 operators: EN, OF, NIET (AND, OR, NOT) DENK AAN LOGISCHE POORTEN!!

“&&” = EN: geeft enkel *true* als beide operanden *true* zijn

“||” = OF: geeft enkel *true* als minstens één operand *true* is, of als BEIDE *true* zijn!

“!” = NIET: inverteert de waarde van de expressie (*true* wordt *false* en vice versa)

* Deze operators geven altijd een *bool* als resultaat
* Verwachten ook enkel *bool* als operanden
* && heeft VOORRANG op || volgens de volgorde van bewerkingen!

Vb:

*bool result= (4 < 6) && (“ja” == “nee”);*

* Output= false

Waarom:

(4 < 6) = true (4 is idd kleiner dan 6)

(“ja” == “nee”) = false (ja is niet gelijk aan nee)

True && false = false (beide operanden moeten *true* zijn)

*bool result= !(0==2);*

* Output= false

Waarom:

(0==2) = false (0 is niet gelijk aan 2)

!(false) = true (NOT operator zal de operand inverteren, dus *false = true*)

TEST JEZELF:

3>2 = true

4!=4 = false

4<5 && 4<3 = false

(3==3 && 2<1) || 5!=4 = true

!(4<=3) = true

True || false = true

!true && false = false

**H5: IF**

*If* of “als”-statements zijn één van de meest elementaire uitdrukkingen in elke programmeertaal

* *if*-statements laten toe om ‘vertakkingen’ te maken in de programmaflow, zodat er meerdere keuzes gemaakt kunnen worden afhankelijk van het resultaat van een expressie.

Syntax:

*If (booleaanse expressie)*

*{*

*//hier wordt code uitgevoerd indien de booleaanse expressie* ***true*** *is*

*}*

* ENKEL indien de booleaanse expressie als resultaat de bool *true* geeft, zal de code binnenin de accolades van het **if-blok**uitgevoerd worden.
* Als het resultaat van de booleaanse expressie de bool *false* geeft, zal de code OVERGESLAGEN worden en gaat het programma verder met de code eronder.

Vb:

*int number= 3;  
if (number < 5)  
{*

*Console.WriteLine(“A”);*

*}  
Console.WriteLine(“B”);*

* Indien *(3 < 5) = true*, zal de code in het if-blok uitgevoerd worden: dus zal er “A” geprint worden in console. Anders zou er “B” op het scherm verschijnen.
* PAS OP: de lijn *Console.WriteLine(“B”);* zal ZOWIEZO uitgevoerd worden in dit voorbeeld!
* Gebruik altijd een “blok” na een if-statement door accolades te gebruiken, ANDERS zal enkel de eerste lijn na een if-statement uitgevoerd worden indien *true*.

VEELGEMAAKTE FOUTEN

* Appelen en peren vergelijken bij booleaanse expressie (bvb een *int* en een *string*)
* Accolades vergeten: als je geen blok maakt zal enkel de eerste lijn na een if-statement uitgevoerd worden
* Puntkomma plaatsen na booleaanse expressie (bij een if): hierdoor zal de if-statement volstrekt nutteloos zijn. MAAK EEN BLOK!

COMPLEXERE IF-STATEMENTS MET LOGISCHE/RELATIONELE OPERATORS

* Gebruik de logische operators om meerdere booleaanse expressies (gemaakt met relationele operators) te combineren met elkaar en daaruit if-statements te maken!

Vb:

*If ((a > c) || (a == b))* => vergeet de haakjes tussen de gehele booleaanse expressie niet!

*If ((a <= c) && (b >= c))*

IF/ELSE

* Met een **if/else-statement** kunnen we ook zeggen **welk specifiek stuk code moet uitgevoerd worden indien een conditie *false* is**.

Vb:

*int nummer= 10;  
int nummer2= 5;*

*if (nummer > max)  
{  
 Console.WriteLine($“Nummer is groter dan {nummer2}!”);  
}  
else  
{  
 Console.WriteLine($”Nummer is NIET groter dan {nummer1}!”);  
}*

* LET OP: er moet GEEN booleaanse expressie staan naast *else*!
* *Else* wordt ENKEL uitgevoerd indien de *if*-statement *true* is!

IF/ELSE IF

* Met een **if/else if-statement** kunnen we meerdere criteria opgeven die moeten uitgevoerd worden voor een bepaald stuk code uitgevoerd worden.
* Als de originele if-statement *false* is, zal de else if-statement getest worden op *true* of *false*
* Als de else if-statement ook *false* is, zal de volgende else if-statement getest worden
* Enzovoort tot het ende, waar je nog een else-statement kan plaatsen

Vb:

*Int x= 9;  
if (x == 10);  
{  
 Console.WriteLine(“x Is 10”);  
}  
else if (x == 9)  
{  
 Console.WriteLine(“x is 9”);  
}  
else  
{  
 Console.WriteLine(“getal is kleiner of gelijk aan 10”);  
}*

LET OP:

Zorg ervoor dat de VOLGORDE van de else if tests juist en efficiënt is!

Vb:

*If (x>10)  
{  
 Console.WriteLine("Groter dan 10");  
}  
else if (x>100)  
{  
 Console.WriteLine("Groter dan 100");  
}  
else*

* Hier zal er na de eerste if-test al *true* verschijnen, en dus wordt de else if-test niet meer uitgevoerd.
* DIT IS NIET GOED! Er zal een minder accuraat resultaat verschijnen tenzij je eerst de *(x>100)* expressie zet!
* ZORG ERVOOR dat de test die het MINSTE kans heeft om te slagen EERST te laten zetten. Zo voorkom je dat de computer onnodige extra tests moet doen!!

NESTING

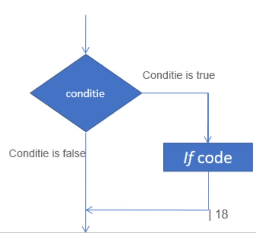
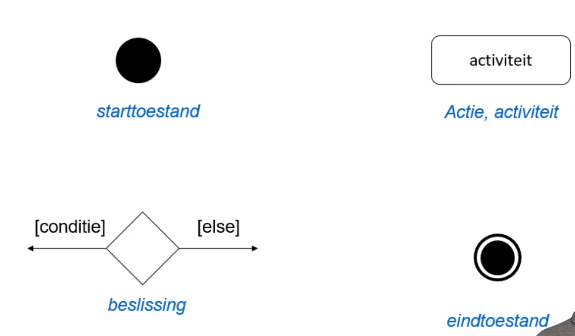
Binnen elk if/else if/else blok kunnen we ook if/else if/else blocks steken!

* Door middel van **nesting** kun je een duidelijk overzicht houden van je programmaflow en welke blocks binnenin andere blocks zitten

ADVIES:

if-structuren coderen is NIET GEMAKKELIJK (ook al lijkt het zo). Vanaf je veel code moet schrijven kan het verwarrend worden. Teken in het begin **flowcharts** om een overzicht te hebben!

UML ACTIVITY DIAGRAM



* Een gestandaardiseerde manier om een diagram/flowchart te maken van de code die je wilt schrijven.
* Als je zo’n diagram ziet, zou je in essentie de code moeten kunnen schrijven die uiteengezet is in de flowchart

**“Break down your conditions”**

* Probeer niet alle condities in één if-statement te zetten
* Splits condities op in aparte if-statements onder elkaar

Vb:

*if ((lives < 10) && (gunpower >100))  
 Console.WriteLine(“Je sterft”);*

Doe dit liever:

*if (lives <10)  
 if (gunpower > 100)  
 Console.WriteLine(“Je sterft”);*

* Dit is leesbaarder en geeft beter overzicht van wat er gebeurt!

**H5: SCOPE VAN VARIABELEN**

De **scope** van een variabele is de zichtbaarheid waarin de variabele **herkend** wordt door de computer.

* De **omliggende accolades** bevatten de scope van een variabele!

Probleem:

* Een variabele is enkel bereikbaar BINNEN de accolades waar het gemaakt werd, vanaf het punt dat het gemaakt werd.

Dus als je een variabele aanmaakt binnen een ingenestelde block code, kan je die variabele erna NIET gebruiken BUITEN de ingenestelde block code.

MAAR je kan wel een variabele gebruiken die gemaakt werd VOOR de ingenestelde code wordt aangemaakt/gebruikt.

Vb:

*if( something == true)  
{*

*int getal = 0 ; //Start scope getal*

*getal = 6;  
} // einde scope getal*

*Console.WriteLine(getal); // FOUT! getal niet in deze scope*

* **DIT GAAT NIET**

*{*

*int getal = 0 ;*

*if( something == true)*

*{*

*getal = 6;*

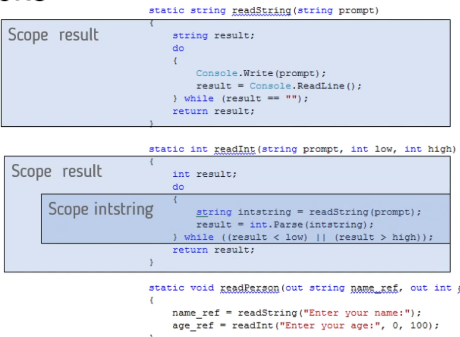
*}*

*Console.WriteLine(getal);*

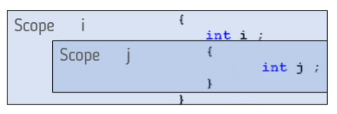
* **DIT GAAT WEL**

VARIABELEN MET ZELFDE NAAM

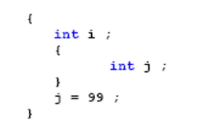
* Je kunt binnen een scope geen variabele maken met dezelfde naam als een variabele BUITEN de scope, snapte?
* Je kunt WEL een variabele met dezelfde naam maken als de vorige variabele met dezelfde naam in een afgesloten block code zit! (dus met gesloten accolades)



* Visueel voorbeeld van **scopes**
* Alle code binnen het block “Scope result” kan de *string result* aanhalen om te gebruiken!
* Alles BUITEN het block “Scope result” kan de *string result* NIET aanhalen!



* Scope j zit BINNENIN Scope i en dus kan er geen *int i;* aangemaakt worden in Scope j



* Dit geeft een error want *j* is NIET gedefinieerd als een variabele!
* Waarom: omdat *int j* BUITEN de scope valt van de lijn *j = 99;*
* DUS: *j* moet terug geïdentificeerd worden als een *int j*

“Use of unassigned local variable” oplossing:

* Geef een variabele **ALTIJD EEN STARTWAARDE! NOOIT LEEGLATEN!**

**H5: SWITCH**

Een **switch-statement** is een program-flow element die je gebruikt om een if/else if/else-constructie **eenvoudiger** te schrijven.

Syntax:

*switch (value)  
{  
 case constant:  
 statements  
 break;  
 case constant:  
 statements  
 break;  
 default:  
 statements  
 break,  
}*

*value* = de variabele die je gebruikt om te testen op de booleaanse *true* of *false* resultaat.

*case* = het **keyword** dat je gebruikt, gevolgd door de WAARDE van de variabele die in *value* staat

* Merk op dat de waarden van *case* **constanten** moeten zijn (uiteraard van hetzelfde datatype als de variabele die in *(value)* staat: DEZE VOER JE ZELF IN! Mogen ook strings zijn!!

*statements* = hier mag je eender welke code zetten. Die zal dan onder de *case* vallen.

*break;* = de code die je schrijft voor de *case* moet afgesloten worden met *break;*.

*Default:*  = de case die uitgevoerd worden als er geen case gevonden wordt met een gelijke constante   
 als de waarde van *value*

* Tijdens de uitvoer zal de waarde van de variabele in *(value)* vergeleken worden met elke *constant* van *case*, van boven naar onder.
* Als een waarde van de variabele overeenkomt met één van de constanten, dan zal DIE case uitgevoerd worden.
* Als er geen gelijkheid gevonden wordt, zal de *default* case uitgevoerd worden.

FALLTHROUGH

* Als je wilt dat dezelfde code wordt uitgevoerd bij 2 of meer cases, kun je dat volgens de fallthrough cases schrijven:

*case 1:  
 statements  
 break;  
case 2:  
case 3:  
 statements  
 break;*

* In dit geval zullen de waarden “2” en “3” dezelfde statements/code volgen.

**H5: ENUM**

BESTAANSREDEN

Stel, je schrijft een programma dat afhankelijk van de dag vd week, een ander stuk code moet uitvoeren. Via beslissingen zou je dit op 2 **foutgevoelige** manieren kunnen schrijven:

1. Met een *int* die getal 1 tot 7 mogen bevatten
2. Met een *string* die de naam van de dag bevat

SLECHTE OPLOSSING 1:

* Met *int*

*if (dagKeuze ==1)  
{  
 Console.WriteLine(“We doen de maandag dingen”);  
}  
else if (dagkeuze ==2)  
{  
 Console.WriteLine(“We doen de dinsdag dingen”);  
}  
else if //enzovoort…*

NADELEN:

* Wat als *dagKeuze* een niet-geldige waarde geeft? (alles buiten 1-7 of whatever)
* De code is niet erg leesbaar, en kan verwarrend zijn.

SLECHTE OPLOSSING 2:

* Met *string*

*if (dagKeuze==”maandag”)  
{  
 Console.WriteLine(“We doen de maandag dingen”);  
}  
else if  
{  
 Console.WriteLine(“We doen de dinsdag dingen”);  
}  
else if //enzovoort…*

NADELEN:

* Zéér foutgevoelig voor schrijffouten bij input!

GOEDE OPLOSSING

* **Enumeraties!**

ENUMERATIES: HET BESTE VAN BEIDE WERELDEN

Enumeraties zijn een C# syntax die twee problemen oplost:

1. Leesbaardere code
2. Minder foutgevoelige code (dus minder potentiële bugs)
3. VS kan je helpen om de code sneller te schrijven (met IntelliSense en auto-complete)

***enum***

* Nieuw keyword
* Dit keyword geeft aan dat we een **nieuw datatype aanmaken**!
* Nadat we een nieuw datatype hebben gemaakt en **gedefinieerd**, kunnen we een **variabele** **van dit nieuwe datatype aanmaken**!
* Deze variabele zal ENKEL waarden mogen bevatten die **gedefinieerd** zijn in het nieuwe datatype.
* IntelliSense en VS zullen de gedefinieerde waarden herkennen en kunnen auto-completen!

In C# zitten al veel enum-datatypes ingebouwd.

* Bvb *ConsoleColor.* => deze kan enkel een pre-gedefinieerde waarde krijgen (nl een kleur)

ZELF ENUM MAKEN

* Gebeurt in 2 stappen:

1. Nieuwe datatype en zijn mogelijke waarden **definiëren**
2. Variabelen van het nieuwe datatype aanmaken en gebruiken in je code

STAP 1: Nieuwe datatype definiëren

* **LET OP!**  Dit moet gebeuren binnen het code block van ***class Program****,* VOOR de *main* methoden.

Syntax:

*enum naam{waarde1, waarde2, waarde3, etc}*

Vb:

*enum Weekdagen{Maandag,Dinsdag,Woensdag,Donderdag,Vrijdag,Zaterdag,Zondag};*

*static void Main(string[] args  
{  
}*

* Voila, nu heb je het nieuwe datatype *Weekdagen* gemaakt, die kan je nu gebruiken als datatype voor variabelen!
* LET OP: individuele enum-waarden moeten steeds met een hoofdletter beginnen!

STAP 2: Variabelen van het nieuwe datatype aanmaken en gebruiken

Syntax:

* Dezelfde als elke variabele!
* Denk aan de manier waarop *ConsoleColor.Kleur* gebruikt wordt!

*Weekdagen dagKeuze;  
Weekdagen andereKeuze:*

*Weekdagen dagKeuze = Weekdagen.Donderdag;*

* Je moet altijd het enum-datatype voor de waarde zetten WANT meerdere enum-datatypes mogen **dezelfde waarden** hebben! Dus dit voorkomt verwarring bij de computer.

ENUMS EN BESLISSINGEN WERKEN GRAAG SAMEN:

Voorbeelden:

*If (dagKeuze == Weekdagen.Woensdag)*

Of

*switch(dagKeuze)  
{  
 case Weekdagen.Maandag:  
 Console.WriteLine(“It’s Monday!”);  
 break;  
 case Weekdagen.Dinsdag:  
 //etc  
}*

Visual Studio tip:

Typ “switch” en druk 2x op TAB

* Er zal een prefab code block verschijnen voor je *switch* structuur
* Als je dan een variabele intypt voor je enum-datatype en dan klikt ERNAAST zal je switch volledig vanzelf volgeschreven worden!
* Als je een *Weekday* datatype gebruik, zul je automatisch suggesties krijgen voor alle mogelijke waarden van je enum-datatype!

CONVERSIE VAN EN NAAR ENUM VARIABELEN

* Intern worden enum-variabelen als ***ints*** bewaard.
* Tenzij anders gespecifieerd, zal de eerste waarde 0 zijn, en elke andere waarde die erop volgt telt op!

Dus je kunt enum-variabelen **casten** met *ints*!

*int keuze = 3;  
Weekdagen dagKeuze = (Weekdagen)keuze;*

* *keuze* is een *int* met waarde 3
* *Weekdagen* heeft een waarde die opgeslagen is als 3, namelijk *Weekdagen.Donderdag*
* DUS: *dagKeuze* zal de waarde *Weekdagen.Donderdag* krijgen

*Weekdagen dagKeuze= Weekdagen.Dinsdag;  
int extraDag= (int)dagKeuze + 1;  
Weekdagen nieuweDag= (Weekdagen)extraDag;*

* De variabele *dagKeuze* (met waarde *Weekdagen.Dinsdag)* wordt gecast met een *int* zodat er een *int 1* kan opgeteld worden.
* Hierdoor wordt de waarde van *nieuweDag* de *Weekdagen.Dinsdag* + 1
* DUS: *Weekdagen.Woensdag*

ANDERE INTERNE WAARDE TOEKENNEN

* Standaard worden de enum-waarden intern genummerd met een *int* vanaf 0
* Je kunt echter manueel een waarde nummeren met een zelfgekozen nummer

Vb:

*enum Weekdagen {Maandag=1, Dinsdag, etc}*

* Nu zullen de waarden vanaf 1 genummerd worden

*enum Weekdagen {Maandag=1, Dinsdag, etc, Zaterdag=50, Zondag=60}*

* Vanaf de eerste waarde wordt er van 1 genummerd, tot aan de volgende waarde met een manuele nummering! (namelijk Zaterdag, daarna Zondag)

**Enums zijn handig om menu’s te maken!**

* Met *enum* kun je een specifiek aantal waarden maken dat je nodig hebt voor een menu.