**H4: CASTING, CONVERSIE EN PARSING**

Hoe veranderen we data van vorm? Maw, hoe combineren we twee datatypes?

*Int age = 4.3;*

* Dit zal een ERROR geven
* Je zal moeten converteren! Hier zijn 3 methodes voor:
* **Casting** => klassieke manier die in veel andere programmeertalen ook werkt
* **Converteren** => via de *Convert.*-bibliotheek van .NET
* **Parsing** => enkel nuttig om strings om te zetten

METHODE 1: CASTING

* Deze methode passen we toe wanneer we een variabele van een datatype willen toekennen aan een variabele van een ander datatype en er **dataverlies zal plaatsvinden!**

Definitie van casting:

* Met casting zul je een variabele van een bepaald datatype **tijdelijk** laten doorgaan als een variabele van een ander datatype, om er voor te zorgen dat de berekening vervolledigd kan worden. Een variabele casten heeft enkel een gevolg **tijdens** het uitwerken van de expressie.
* IN HET GEHEUGEN ZAL DE VARIABELE ALTIJD VAN HET ORIGINEEL TOEGEKENDE DATATYPE ZIJN!!
* Je dient enkel aan casting te doen wanneer je aan **narrowing** doet!

Definitie van narrowing:

* *Narrowing* is het omzetten van een variabele naar een ander datatype dat **een verlies van data met zich zal meebrengen**. Er zal dus data verloren gaan door het casten.

Syntax:

*int mijngetal= (int)3.5;*

Of

***double*** *kommagetal= 13.8;****int*** *kommaNietWelkom= (int)kommagetal;*

* Hiermee zeg je tegen de compiler dat de variabele van de **double** toegekend moet worden aan de variabele van het type **int**.
* Hierdoor zal er data verloren gaan bij de **double** variabele, nl alles na de komma. (**narrowing**)
* De programmeur BESEFT dat hier data verloren zal gaan en draagt alle verantwoordelijkheid voor het verlies van die data!

NARROWING (explicit casting)

* Wanneer je een variabele toekent aan een datatype waarin die variabele eigenlijk niet inpast, en dus versmalt moet worden.

*double var1;  
int var2;*

*var1 = 20.4;  
var2 = (int)var1;*

* Resultaat: **20**, alles na de komma valt weg omdat een *double*gecast is naar een *int*.
* LET OP: het datatype van var1 is ONVERANDERD, en nog steeds een **double**.
* De variabele is slechts **tijdelijk** gecast naar een *int* om in de expressie te passen (en is versmalt/narrowed)

PRAKTISCH VOORBEELD:

*int tempGisteren= 20;  
int tempVandaag= 25;*

*int tempGemiddeld= (tempGisteren + tempVandaag)/2;*

* We werken hier enkel met *int*, dus er zal een geheel antwoord teruggegeven worden: 22
* Echter, 22 is niet het antwoord wat we willen. Het resultaat is nl 22.5.
* We verkrijgen niet het gewenste antwoord want het antwoord is een **double**, en hier werken we enkel met **int.** Ook dit is *narrowing*.
* Het gemiddelde van 2 ints geeft niet altijd een geheel getal als antwoord!

*double tempGemiddeld= (tempGisteren + tempVandaag)/2;*

* Ongeacht dat de variabele als *double* toegewezen is, zal de expressie nog steeds een *int* als antwoord geven. Dus *tempGemiddeld* zal nog steeds 22 zijn, een geheel getal, maakt niet uit dat zijn datatype een *double* is.
* Oplossing: de rechterkant/expressie zal ook als een *double* beschouwd moeten worden:

*double tempGemiddeld= ((double)tempGisteren + (double)tempVandaag)/2;*

* De *int tempGisteren* en de *int tempVandaag* zijn **gecast** als *double*
* Het resultaat zal nu wel 22.5 zijn!

Andere schrijfwijzen:

*(tempGisteren + tempVandaag)/2.0;*

* deze ken je al

*((double)(tempGisteren + tempVandaag))/2;*

* volgorde van bewerkingen, eerst optelling, dan omzetting naar *double*, dan deling door int. Resultaat is double.

*((double)tempGisteren + tempVandaag)/2;*

* eerst wordt *tempGisteren* omgezet naar een double, daarna wordt double opgeteld met int, wat een double als resultaat geeft. Daarna deling door int, waardoor resultaat double is.

*(tempGisteren + (double)tempVandaag)/2;*

* zelfde als vorige maar *tempVandaag* wordt eerst omgezet naar double.

WIDENING (implicit casting)

* casting is NIET nodig als je aan widening doet. Een kleiner datatype in een groter datatype steken werkt zonder probleem! Dit heb je trouwens al gedaan wanneer je een int berekent met een double. Dan krijg je een double als resultaat. (widening)

*int var1;  
double var2;*

*var1= 20;  
var2 =var1;*

* de *int* 20 van var1 wordt toegekend aan de *double var2*: de compiler zal er simpelweg een 20.0 van maken. Volledig compatibel!
* Er gaat hier GEEN inhoud verloren. De computer ziet zelf dat er toegekend kan worden zonder dataverlies.
* Je MAG nog steeds casten, maar het hoeft niet!

METHODE 2: CONVERSIE

* *Casting* is old skool, maar is nuttig want het is compact en werkt ook in andere C#-gerelateerde talen zoals C, C++ en Java.
* .NET (developer platform) heeft een ingebouwde conversie-methode, nadeel is dat het iets meer typ-werk vereist.
* ***Convert.***-bibliotheek

Enkele voorbeelden:

*int getal= Convert.ToInt32(5.2);*

* Converteert *double* naar *int*

*double anderGetal= Convert.ToDouble(5);*

* Converteert int naar een double

*bool isWaar= Convert.ToBoolean(1);*

* Converteert int naar bool. LET OP: elke waarde boven 0 zal in een conversie naar *bool* op **True** slaan. Enkel een 0 zal converteren naar een **False**.

*int userAge= Convert.ToInt32(“23”);*

* Converteert een string naar een int!

*int ageOther= Convert.ToInt32(anderGetal);*

* Converteert de double *anderGetal* naar de *int ageOther*

LET OP: naar een *int* converteren gebeurt met *.ToInt32()*. Converteren naar een *short* gebeurt met *.ToInt16()*.

LET OP: de conversie zal zo goed mogelijk data proberen omzetten, en zal zelf *narrowing* en *widening* toepassen waar nodig. Kijk uit bij het omzetten van een *string* wat er intern juist gebeurd!

Meer informatie over de ***Convert.***-bibliotheek vindt je hier: <https://docs.microsoft.com/en-us/dotnet/api/system.convert?redirectedfrom=MSDN&view=netcore-3.1>

METHODE 3: PARSING (vertaling: ontleden)

* Voorlopig hebben wij dit enkel nodig om *string* naar getallen om te zetten.
* Echter, **intern** zal er vrijwel altijd *Convert.To…* gebruikt worden!

Elk ingebouwd datatype in C# heeft een ***.Parse()*** methode die je kan aanroepen om **strings om te zetten naar het gewenste type**.

*int numVal= Int32.Parse(“-105”);  
Console.WriteLine(numVal);*

Gebruik parsing enkel wanneer:

1. Een string waarvan je weet dat die altijd een specifiek type zal zijn, zoals een int, dan gebruik je *Int32.Parse()* of *Int.Parse();* => gebruik dit om input van de gebruiker te parsen!
2. Wanneer je input van de gebruiker vraagt (via Console.ReadLine() en je bent niet zeker of de input een getal zal bevatten, dan gebruik je *Int32.TryParse()*.

**H2: INPUT VAN DE GEBRUIKER VERWERKEN**

* Input van de gebruiker via Console.ReadLine() **zal ALTIJD een string teruggeven!**

Als bvb de gebruiker zijn leeftijd invoert, zal dat getal in het string datatype zijn, en zullen we het moeten **converteren** naar het juiste datatype.

*int userInput= Console.ReadLine();*

* ERROR! Console.ReadLine() moet ALTIJD toegekend zijn aan het datatype *string*!

Input converteren gebeurt in 3 stappen:

1. Input lezen met Console.ReadLine();
2. Input **bewaren** in een string variabele
3. De variabele **parsen**met *.Parse()* naar het gewenste datatype

INPUT CONVERTEREN

* Gebruik *.Parse* of *Convert.To*

*Console.WriteLine(“Geef je gewicht:”);  
string inputGewicht= Console.ReadLine();  
double gewicht=* ***double.Parse(inputGewicht);***

* Wat gebeurt er precies: de string *inputGewicht* wordt **geparst** en dus **geconverteerd** naar het datatype **double** en wordt dan toegekend aan de variabele *gewicht* (double)
* ONTHOUD: “parsen” betekent grammaticaal ontleden. De string input van de gebruiker wordt ontleedt en omgezet naar de datastructuur die gelezen kan worden door de computer!

Efficiëntere schrijfmethode:

*Console.WriteLine(“Geef je gewicht:”);  
double gewicht= double.Parse(Console.ReadLine());*

* De input van de gebruiker (die in Console.ReadLine wordt geschreven) zal ogenblikkelijk **geparst** worden en **geconverteerd** naar het datatype **double**, en daarna toegekend aan de variabele *gewicht*.

LET OP: de invoer van kommagetallen is afhankelijk van de landinstellingen van je besturingssyteem.

* In België: kommagetal met een KOMMA
* In het Engels: kommagetal met een PUNT
* In C#: kommagetal ALTIJD met een PUNT

**H4: BEREKENINGEN MET System.Math**

* De *System.Math*-bibliotheek helpt ons bij het berekenen van variabelen.

DE MATH-BIBLIOTHEEK

* Deze bibliotheek bevat methoden voor een groot aantal typische wiskundige bewerkingen
* Sinus, cosinus, vierkantswortel, macht, afronden, etc
* Gebruik deze bibliotheek om leesbare en kortere expressies te schrijven!

Vb: Je wilt de derde macht van een variabel *getal* berekenen:

*double result= getal \* getal \* getal;*

* Slechte, inefficiënte manier! Rommelige onleesbare code

*double result= Math.Pow(getal, 3);*

* 2 **parameters**:
* Grondtal: *getal*
* Exponent: *3*
* *Math.Pow* staat voor “power” “(to the nth power”)

Schrijf *Math.* In VS om een lijst te krijgen van de Math-bibliotheek.

* Een kubus-icoon voor de naam betekent dat het om **methode** gaat
* Een vierkant-icoon voor de naam betekent dat het om een **constante** gaat (zoals Pi)

METHODEN GEBRUIKEN

* Meeste methoden zijn vanzelfsprekend en werken op soortgelijke manier.
* Één of twee parameters die je tussen de haken moet meegeven
* Het resultaat van de methoden moet je altijd in een nieuwe variabele opvangen!

Als je twijfelt over de werking van een methode, doe het volgende:

1. Schrijf methode zonder parameters
2. Plaats cursor op de methode
3. Druk op F1
4. Je ziet nu de help-files van deze methode!

PI

* Kun je ook aanhalen via de Math-bibliotheek!

*double straal= 5.5;  
double omtrek=* ***Math.PI*** *\* 2 \* straal;*

**H4: AFRONDEN VAN WAARDEN**

4 manieren om “af te ronden”:

*Convert.ToInt32(value)*

* Omzetting naar int door klassieke (wiskundige) afronding!

*Int x= (int)value;*

* Omzetting naar int door AFKAPPEN tijdens casting (narrowing)

*Math.Round(value)*

* Afronden naar dichtstbijzijnde geheel getal (klassieke afronding)
* Idem aan *Convert.ToInt32*
* OPLETTEN op **Banker’s rounding**
* *Math.Round(value, cijfer na de komma)*
* 2de parameter geeft me hoeveelste cijfer na de komma de variabele moet afgerond worden.

*Math.Ceiling(value)*

* Naar boven afronden

*Math.Floor(value)*

* Naar onder afronden

BANKER’S ROUNDING

* Als je *Math.Round()* gebruikt om 4.5 en 5.5 af te ronden, bekom je 4 en 6. WAAROM?!
* Als je op de grens zit van .5, zal het gehele getal afgerond worden naar het dichtstbijzijnde EVEN GETAL!

Dit algoritme is ontstaan in de bankierswereld omdat als je bij het tellen van grote sommen geld altijd naar boven afrondt, je een steeds groter wordende hoeveelheid geld bekomt die je eigenlijk niet hebt!   
DUS wordt er altijd naar het dichtstbijzijnde even getal afgerond, waardoor je gemiddeld dichter komt bij de ware hoeveelheid geld.

Wij kunnen dit **aanpassen:**

*Math.Round(value, MidpointRounding.AwayFromZero)*

* Met het 2de parameter *MidpointRounding.AwayFromZero* zal je ALTIJD naar boven afronden!
* *MidpointRounding.* Na de punt krijg je verschillende opties om op allerlei manieren af te ronden. 3

**H4: RANDOM GETALLEN GENEREREN**

De *Random*-bibliotheek (klasse) laat je toe om willekeurige integers/reals te genereren. Je moet hiervoor 2 zaken doen:

1. **Eenmalig** een Random-generator object aanmaken
2. De *Next* methode aanroepen telkens je een nieuw willekeurig getal nodig hebt.

*Random randomGenerator= new Random();*   
*int getal= randomGenerator.Next(value, value);*

* *Random* is de bibliotheek die je aanhaalt. ZIE HET OOK ALS EEN SOORT DATATYPE!
* *randomGenerator* is de **identifier** voor je generatorobject, die mag dus elke naam hebben (zolang het aan de identifier-regels voldoet)
* *[identifier].Next* gebruik je elke keer als je de generatoroject een nieuw random getal wilt vragen.
* *Random()* => tussen deze haakjes kan je een **seed** plaatsen voor de generator

Efficiëntere aanroepmethode:

*Console.WriteLine($”Een random getal: {randomGenerator.Next()}”);*

In de *Next()* methode passen 2 parameters, gescheiden door een komma.

* Deze parameters zijn de grenzen waarbinnen het getal gegenereerd moet worden.
* Het **tweede parameter** is EXCLUSIEF het random getal. Dus als je binnen 0 en 10 een getal wilt, schrijf je *Next(0, 11)*.
* De parameters kunnen ook variabelen zijn!

*NextDouble()*

* Hiermee kun je kommagetallen genereren tussen 0.0 en 1.0 (exclusief 1.0)
* Voor grotere kommagetallen moet je vermenigvuldigen: *NextDouble() \* 10.0*
* Vermenigvuldig met het bereik dat je wenst te hebben
* Voor bvb een bereik te verkrijgen tussen 5.0 en 12.5 doe je het volgende:

*double getal= 5.0+ (myRan.NextDouble() \* 7.5);*

**H4: DEBUGGEN**

Logische fouten vs C# fouten:

* C# fouten gebeuren wanneer je een fout schrijft qua grammatica of syntax volgens de programmeertaal. Als je code foutloos is geschreven, zal de code gecompileerd worden.
* Logische fouten gebeuren wanneer je code die je geschreven hebt niet correct uitgevoerd wordt. Het wordt WEL gecompileerd, maar de fout zit ergens in de LOGICA van je code.

DEBUGGEN MET VISUAL STUDIO

**Breakpoint**:

* Deze kun je gebruiken door links in het rechtse vak voor een lijn code te klikken. Er zal dan een rood stopicoontje verschijnen.
* Wanneer je je code uitvoert, zal elke code lijn uitgevoerd worden TOT HET AAN DE LIJN KOMT MET DE BREAKPOINT. De lijn code zelf zal NIET uitgevoerd worden!!
* Wanneer de code gepauzeerd wordt, zal VS terug openspringen in **debug modus**

**Debug modus**:

* Onderaan zijn de tabs **autos** en **locals:** in deze tabs zie je de waarden van iedere variabele die zichtbaar/uitgevoerd is tot aan de breakpoint.
* Gebruik deze tabs om het resultaat van expressies te bekijken!!!
* Je kan ook over expressies hoveren om het resultaat ervan te bekijken!

DOOR JE CODE STEPPEN

Wanneer je gepauzeerd bent zie je bovenaan (naast de startknop) een hoop icoontjes:

**Continue**:

* Het programma zal verder uitgevoerd worden vanaf het breakpoint, ofwel tot het volgende breakpoint, ofwel tot het einde van het programma.

**Rode stop**:

* Hiermee stop je debug modus en eindigt het programma. Je kunt je code terug aanpassen.

**Step-over:**

* Met deze knop kun je je code LIJN PER LIJN uitvoeren.
* Zo zie je bvb de flow van je programma en kun je zien hoe variabelen evolueren.
* BELANGRIJKE KNOP!!

Debuggen is een ESSENTIËLE SKILL!

* De flow van debuggen gaat als volgt:

1. Bepaal welke output je verwacht (VOORSPEL dus eerst de output)
2. Test programma met getallen van je berekening en verwachting
3. Klopt de output?
4. Wees kritisch!