Introductie

Je bent nu al wat bekender met programmeren. Zo heb je al wat ervaring met web/game/XR ontwikkeling.
Voor deze extra challenges bouw je voort op die kennis, en duik je ook de Terminal in.

Opzet

Deze challenge gaat er van uit dat je in `C#` gaat programmeren.
Mocht je zelf liever een andere taal gebruiken, dan mag dat natuurlijk ook, maar dan kan het wel zijn dat je het een en ander zelf moet uitvogelen.

Als het goed is heb je inmiddels al een keertje de .NET SDK geïnstalleerd (tijdens de installatie van Unity, bijvoorbeeld). Je computer zou dus al klaar moeten zijn voor deze opdrachten. Controleer dit even door de terminal te openen (Druk op de Windows toets, voer `cmd` in, en druk op enter). Een terminal verschijnt. Voer in de terminal het volgende command uit: `dotnet`

Als alles goed geïnstalleerd is komt er een dergelijke tekst tevoorschijn:

```
Usage: dotnet [options]
Usage: dotnet [path-to-application]

Options:
  -h|--help         Display help.
  --info            Display .NET information.
  --list-sdks       Display the installed SDKs.
  --list-runtimes   Display the installed runtimes.

path-to-application:
  The path to an application .dll file to execute.
```

Zie je iets in de trand van `'dotnet' is not recognizes as an internal or external command`, dan moet je nog even `dotnet` installeren. Zie de link hier boven. Het maakt daarbij niet uit of je 8 of 9 kiest. Het kan zijn dat je hierna even je computer moet herstarten als het commando nog steeds niet herkend wordt.

Maak nu een nieuw mapje aan op je computer, bijvoorbeeld in `C:/Users/codevogel/MijnMapje` en ga daarheen door `cd "<pad naar map>"` in te voeren (dus: `cd "C:/Users/codevogel/MijnMapje"`). `cd` staat voor `change directory`.

(Tip: Je kan ook naar een relatieve folder vanaf het huidige punt gaan. Bijvoorbeeld vanuit `C:/Users/codevogel` kan ik dan `cd MijnMapje` doen. Of ik kan een mapje omhoog, met `cd ..` vanuit `C:/Users/codevogel/MijnMapje` kom ik dus weer in `C:/Users/codevogel` terrecht. Het commando `dir` laat je de files en mapjes op de huidige locatie zien. Speel hier eens mee!)

Eenmaal aangekomen in de juiste map (en wanneer je `dotnet` geinstalleerd hebt), voer je het commando `dotnet new console` uit. Dit maakt twee files voor je aan: `<naam van de map>.csproj` en `Program.cs`.

Open nu deze verse `Program.cs` in je favoriete code editor. Bijvoorbeeld Visual Studio (Code), Rider, of neovim (💪🤓🤘) .
Verwijder alle inhoud, en plak de volgende startcode:

```
class Program {
        // string[] args is een array aan argumenten die meegegeven worden
        // aan het programma.
        public static void Main(string[] args)
        {
                // het aantal argumenten
                int numArguments = args.Length;
                Console.WriteLine(numArguments);
                // print elk argument uit
                foreach (string arg in args) {
                        Console.WriteLine(arg);
                }
        }
}
```

Vergeet het bestand niet op te slaan.

Nog steeds in de terminal (*in de juiste map*!), probeer de volgende commando's uit.
Als het goed is krijg je dan te zien:

```
$ dotnet run
0

$ dotnet run abc def
2
abc
def
```

```
$ dotnet "abc def"
1
abc def
```

(N.B (Nota Bene): de lijnen met `$` geven aan dat de lijn een ingevoerd commando is. Je hoeft het dollar teken niet zelf in te voeren! `$ dotnet run` zou je dus intypen als `dotnet run`.)

Let hierboven op het verschil tussen de drie commandos.

1. Bij de eerste wordt er geen *argument* meegegeven aan het programma: `numArguments` wordt dus `0`, en er worden geen argumenten uitgeprint.
2. Bij de tweede worden er twee argumenten meegegeven: `abc` en `def`. `numArguments` wordt dus `2`, en de argumenten worden apart geprint.
3. Vervolgens hebben we nog de derde, waar `"abc def"` het argument is. Dat is maar één argument, aangezien we het verpakken in twee string quotejes. `numArguments` is dus `1`, en het argument wordt uitgeprint als `abc def`.

Je bent nu klaar om aan de slag te gaan!

Kom je er tijdens de challenge niet helemaal uit? Probeer eerst eens even goed te googlen.
Vraag daarna vooral een mede-student of aanwezige docent om hulp!
Kom je er ook dan nog niet uit? Stuur dan (pas) een berichtje naar Kamiel de Visser (KVI) via Teams.

Optelsommen

Thomas Tijs is zijn TI rekenmachine kwijt!
Schrijf een programma dat optelsommen tussen `integers` uitrekent. Negatieve getallen zijn hierbij toegestaan.

Wanneer een gebruiker maar 1 getal invoert, een komma getal invoert, of iets wat geen getal is, geef je een foutmelding.

Hint: gebruik de `Int32.Parse` functie. Kijk ook naar de bijbehorende `FormatException`, en hoe je exceptions kan afvangen.
Hint: kijk eens naar `try ... catch` blocks om de situaties af te handelen waar een `string` niet omgezet kan worden naar een `int`.
Hint: kijk eens naar `return` om een programma vroegtijdig te stoppen.

Voorbeeld output:

```
$ dotnet run 5 7
12

$ dotnet run 13 -8
5

$ dotnet run 13
error: voer twee getallen in

$ dotnet run a 3
error: voer twee getallen in
```

Als je programma bovenstaande voorbeelden kan repliceren, dan is je programma goedgekeurd.

Letters tellen

Simon Schmidt is een gedichtje in elkaar aan het smeden. Om deze mooi uit te printen, wil hij weten hoeveel letters er in een zin zitten. Leestekens, getallen, en spaties worden niet meegeteld. Daarnaast wilt Simon ook nog eens weten hoeveel hoofdletters er in de zin zitten.

Probeer zelf de juiste functies te vinden die controleren of een karakter een letter (en/of een hoofdletter is).

Voorbeeld output:

```
$ dotnet run "Ik ben op school."
Aantal letters: 13
Waarvan hoofdletters: 1

// NB: de ! is een speciaal karakter in een string met dubbele quotes, dus
hier staan enkele quotes.
$ dotnet run 'Hallo, Wereld!'
Aantal letters: 11
Waarvan hoofdletters: 2

$ dotnet run ""
Aantal letters: 0
Waarvan hoofdletters: 0
```

```
$ dotnet run "een zin" "nog een zin"
error: voeg hoogstens 1 zin in
```

Als je programma bovenstaande voorbeelden kan repliceren, dan is je programma goedgekeurd.

Input lezen

Lukas Stroomer heeft een bestandje aangeleverd, en hij zou graag willen dat je dat bestandje inleest en uitprint.
Laat in de console zien wat er in het bestand staat.

Dit is het bestand (`input.txt`):

```
Dit is mijn bericht.

Kan je dit naar de terminal Stroomen?
Dankjewel!
```

Maak een `input.txt` aan in dezelfde map als je `Program.cs`, en zet het bericht van hierboven er in. (Op Windows kan het zijn dat je bestand niet goed wordt gevonden als je `./input.txt` gebruikt als pad. Kijk goed naar de foutmelding, en plaats het bestandje in de juiste map (bv ergens in de `build/debug` map die automatisch gegenereerd wordt als je het programma runt, of gebruik relatieve paden (`../../input.txt`) om 'omhoog' te zoeken naar de `input.txt`)

Hint: gebruik de `File.ReadAllLines` methode. Deze mag je zelf googlen!

Voorbeeld output:

```
$ dotnet run ./input.txt
Dit is mijn bericht.

Kan je dit naar de terminal Stroomen?
Dankjewel!

$ dotnet run
error: geef een bestand mee!

$ dotnet run ./een-niet-bestaand-bestand.txt
error: dit bestand bestaat niet!
```

Als je programma bovenstaande voorbeelden kan repliceren, dan is je programma goedgekeurd.

Halve pyramide tekenen

Bart van Thiel is naar Egypte geweest, en wil graag hier aan herinnerd worden. Schrijf een programma dat een halve pyramide tekent in je terminal.

- De schuine zijde van de pyramide moet links zitten, en de top is twee breed.
- De pyramide geef je aan met `#` en de lucht geef je aan met `.`
- Als de gebruiker een getal lager dan 3 of groter dan 20 invult geef je een error.
- Je mag er bij deze opdracht van uitgaan dat de gebruiker altijd precies 1x een `int` meegeeft als argument aan het programma.

```
$ dotnet run 7
......##
.....###
....####
...#####
..######
.#######
########

$ dotnet run 2
error: vul een getal tussen 3 t/m 20 in.

$ dotnet run 4
...##
..###
.####
#####

$ dotnet run 3
..##
.###
####
```

Als je programma bovenstaande voorbeelden kan repliceren, dan is je programma goedgekeurd.

Hint: `Console.Write` is net anders dan `Console.WriteLine`
Hint: Reken eerst eens met de hand het algoritme uit wat je nodig gaat hebben om te weten hoeveel puntjes en hekjes je per lijn nodig gaat hebben. Kijk goed naar de verandering lijn per lijn, van boven naar beneden...

Stappenteller

Martijn Koning is zijn kroon kwijt!
Gelukkig houdt zijn stappenteller bij waar hij is geweest. Bij een `>` liep Martijn een stap vooruit. Bij `<` liep hij een stap achteruit.
De vraag is nu... hoeveel stappen voorwaarts heeft Martijn gelopen, en hoeveel stappen achterwaarts? En waar is Martijn dan terecht gekomen?

Schrijf een programma dat uitrekent waar Martijn terecht komt gegeven een stappenteller-string.
Je mag bij deze opdracht er van uitgaan dat er altijd een juist geformateerde 'schatkaart' wordt ingevoerd. Je hoeft dus geen error handling te doen.

Voorbeeld output:

```
$ dotnet run "><>>>"
Stappen voorwaarts: 4
Stappen achterwaarts: 1
Komt uit op: +3 stappen

$ dotnet run ">>><<<<"
Stappen voorwaarts: 3
Stappen achterwaarts: 4
Komt uit op: -1 stappen
```

Martijn's stappenteller geeft het volgende aan:

```
>>>>><<><><><><<<<<>>>><<<><><<><><><><<<<<<<><><><><<<><><><><><><
<>>>>>>>>><<<><><><><><><>><<<<<<><><><><<<<<><><><><>>><><><<<><><<<<><>><><><
<<<<><<><><><><><><><><<<<<<<<<<<<<>>>><<<<<><><<><><><<<<<<<<<<>><<<><>>>>><
<><><><><><<><><><><><>>><<><><><>>><<><><<<<<<<<<><><<><><><><><<<<<><<><><><<>>
<><><<<<<<<<<><><><><><><><<<<<<><><<<>>>>>><><><><<<<<<<<<
```

Als je programma het juiste aantal stappen uitrekent, dan is je programma goedgekeurd. Je kunt navragen of je antwoord correct is bij de docent. (Of kom naar de uitwerkingenmiddag op woensdag!)

Haakjes sluiten

Perry Spee vindt al die haakjes soms erg onoverzichtelijk. Daarom schrijven we nu een programma dat voor hem controleert of alle haakjes een tegenhanger hebben.

Als je een regel code invoert moet het programma teruggeven of er een probleem is met het aantal haakjes en wat het probleem in dat geval is.

Bij deze opdracht mag je er van uitgaan dat er altijd een correct geformateerde regel code in staat. Je hoeft dus geen errors af te handelen.

Voorbeeld output:

```
$ dotnet run "int i = 0;"
Geen problemen gevonden.

$ dotnet run "Debug.Log('Hello World');"
Geen problemen gevonden.

$ dotnet run "if ( i < 5;"
Er mist een sluitend haakje.

$ dotnet run "while) true(;"
Een haakje sluit te vroeg.

$ dotnet run "(()))"
Een haakje sluit te vroeg.
```

Als je programma bovenstaande voorbeelden kan repliceren, dan is je programma goedgekeurd.

Rekeningen

Rubin de Bruin heeft een bruin café. Hij ziet veel klanten door de deuren komen. Graag komt hij er achter wie het meest geld uit heeft gegeven. Nu benaderd Rubin jou met een bestand van geanonimiseerde klantendata en de vraag of jij kan bepalen welke klant het meest heeft uitgegeven.

Per klant staan de prijzen van alle producten onder elkaar opgeschreven met een witregel tussen 2 klanten. Het doel van deze opdracht is om de klant met het hoogste totaalbedrag aan boodschappen te vinden.

Hier is een voorbeeld van een bestand met klantendata:

```
5.00

1.00
2.50
3.50

4.00
4.50
```

NB: Er is steeds een witregel om de rekeningen van verschillende klanten te scheiden.

Klant 1 heeft dus 5 euro betaald. Klant 2 heeft 7 euro betaald. En klant 3 heeft 8.50 betaald. Klant 3 is in dit geval dus de grote uitgever.

Hier volgt het echte klanten bestand:

```
9.61
40.46
50.95
33.48

44.92
81.91
57.14
91.52
46.29
84.80
31.35
29.62
81.04

94.99
72.60
46.86
81.63
70.95
5.49
37.24
81.80
29.75
```

6.13
57.90
16.35
72.00
89.70

31.22
25.55
33.08

24.90
1.73
14.18
10.68
40.42
26.30
96.43
8.62
66.12
62.76

8.69
20.34
20.05
5.94
77.94
32.99
93.35
8.23
31.89
56.54
5.64
48.56

34.04
87.31
19.17

22.47

60.05
32.09
91.60
42.23
54.66
62.38
46.10
41.00
22.70
79.70

84.23
27.72
1.93
91.78
33.14
31.93
28.84

90.52
55.85
9.99
81.29
8.48
81.20
44.77
71.37

92.76
66.98
6.23
99.42
43.45
41.81
8.25

31.00
56.94
4.28
67.79

```
44.54
39.19
9.96
21.68
22.05
48.14
4.23

11.63
62.53
60.58
94.15
30.68
19.00
62.32

79.63
54.13
40.57
56.39

6.69
19.28
72.82
81.09

50.77
49.74
88.28
17.53
52.21
74.78
95.46
64.69
3.99
79.13
99.78

6.48
20.13
```

34.16
20.16
45.93
50.07
91.84
12.36
18.46
77.95
91.21

50.46
58.39
22.39
52.27
22.64
86.92
95.65
56.33

67.23
7.58
7.50
65.41
89.25
23.67

91.26
59.80
65.71
20.96
48.30
69.20
26.57
54.78
25.69
46.40

91.88
12.54
57.13

```
14.37
67.00

10.04
18.03
64.51
73.97
46.98
95.37
63.07
24.43
```

Als je programma de juiste klant aanwijst als de grote gever dan voldoet je programma aan de eisen. Bonuspunten als je ook aan kan wijzen hoeveel die klant dan heeft betaald!
Vraag bij de docent na of je de juiste klant hebt gevonden. (Of kom langs bij het uitwerkingen moment op woensdag.)

Hint: Denk terug aan de 'bestand inlezen' opdracht.
Hint: We hebben al eens `Int32.Parse` gebruikt. Nu moeten we `floats` parsen...
Hint: Als een regel leeg is, kan je er van uitgaan dat we van klant wisselen.

Morse

Kamiel de Visser heeft je een geheim berichtje gestuurd door op zijn hengel te tikken. Schrijf een programma dat Morse code omzet naar tekst, en het ontsleutelde bericht vervolgens uitprint.

Hier volgt het geheime bericht:
`- --- . -./- ---/. -./- --- --/- .- -- -- ./- --- -- .- - . -./.- - . -.`

Schrijf een programma dat dit bericht omzet naar tekst.

Hier staat een tabel met morse tekens die gebruikt worden in dit bericht:

`T : -`
`M : --` A : .-
`N : -.`
`O : ---`
`E : .`

Elk woord wordt onderbroken met een `/`.
Elke letter wordt onderbroken met een spatie ( ).

Je mag er van uitgaan dat er altijd morse wordt ingevoerd, middels deze afgesproken conventies, en dat het bericht geen andere letters dan hierboven aangegeven bevat.

Voorbeeld output:

```
$ dotnet run "- --- . -."
toen

$ dotnet run "-. .- - - ./-- .- -."
natte man
```

Als je programma het geheime bericht van Kamiel kan ontcijferen, dan is je programma goedgekeurd. Je kan bij de docent navragen of je de code gekraakt hebt. Of kom naar het uitwerkingenmoment op de woensdag.

Dichtstbij

Rik Langeveld is in een lang veld aan het klussen en heeft nog een paar houten palen liggen. Hij wil die in de grond slaan en er vogelhuisjes op plaatsen. Het liefst wil hij ze op dezelfde hoogte hebben, maar zijn zaag is erg bot. Gelukkig doet zijn meetlint het wel.

Het is aan jou om een programma te schrijven dat een aantal getallen aan input inleest en de twee getallen teruggeeft die het dichtst bij elkaar liggen, zodat Rik weet welke palen het meeste op elkaar aansluiten.

Voorbeeld output:

```
$ dotnet run 11 83 64 103 99
99 en 103 liggen het dichtst bij elkaar

$ dotnet run 88 6 222 88 74 47
88 en 88 liggen het dichtst bij elkaar
```

Je programma is goedgekeurd als het aan bovenstaande output voldoet.

Extra: Priemgetal

Elmer de Gruijl is knikkers aan het ruijlen. Het liefst ruilt hij alleen in priemgetallen. Schrijf een programma dat voor hem uitrekent wanneer een getal een priemgetal is.

Voorbeeld output:

```
$ dotnet run 2
Ja, dit is een priemgetal

$ dotnet run 8
Nee, dit is geen priemgetal

$ dotnet run 6079
Ja, dit is een priemgetal
```

Je programma is goedgekeurd als het aan bovenstaande output voldoet.

N.B. Een priemgetal is een geheel getal dat alleen door 1 en zichzelf gedeeld kan worden zonder dat het antwoord een breuk vormt. (8 is géén priemgetal omdat 8 / 4 = 2 en 2 is een geheel getal)

Extra: Optellen met romeinse cijfers

Jan van Os heeft een urn in zijn osso gevonden. Hij wil graag weten uit welk jaar deze urn komt. We schrijven een programma dat een optelsom met romeinse cijfers uitrekent, en het antwoord in decimalen teruggeeft.

We behandelen alleen de volgende romeinse cijfers:

```
I = 1
V = 5
X = 10
L = 50
C = 100
```

Regels voor getallen die uit meerdere karakters bestaan:

- Als een kleiner getal vóór een groter getal staat gaat dat getal van het grotere getal af : `IV = 4, XC = 90`
- Als een kleiner getal achter een groter (of gelijk) getal staat, wordt dat getal bij het grotere getal opgeteld: `XII = 12, XLV = 155`

Voorbeeld output:

```
$ dotnet run II IV I
Dat is gelijk aan 2 + 4 + 1
```

```
En dat is 7

$ dotnet run L X VIII
Dat is gelijk aan 50 + 10 + 8
En dat is 68

$ dotnet run XCII VIII
Dat is gelijk aan 92 + 8
En dat is 100

$ dotnet run II 4
error: Een van de getallen is geen ondersteund romeins cijfer.

$ dotnet run
error: Voer minstens 1 cijfer in.

% dotnet run M III
error: Een van de getallen is geen ondersteund romeins cijfer.
```

Als je programma bovenstaande voorbeelden kan repliceren, dan is je programma goedgekeurd.