

```

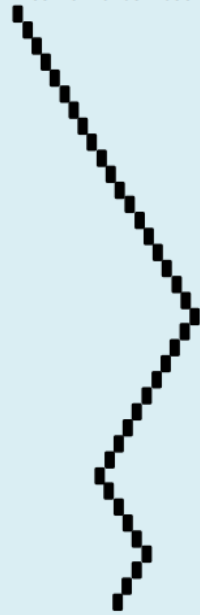
expanded, !0), n: (b[0].offsetWidth, b.addClass( "in" )); b.removeClass( "fade" ); b.parent( ".dropdown-menu" )
find( '[data-toggle="tab"]' ).attr( "aria-expanded", !0 ), e&&e() } var g = d.find( "> .active" ), h = e&&
) || !d.find( "> .fade" ).length; g.length&&h?g.one( "bsTransitionEnd", f ).emulateTransitionEnd
r d = a.fn.tab; a.fn.tab = b, a.fn.tab.Constructor = c, a.fn.tab.noConflict = function() { return a.fn.
ow" ); a( document ).on( "click.bs.tab.data-api", '[data-toggle="tab"]', e ).on( "click.bs.tab.data
strict"; function b(b) { return this.each( function() { var d = a( this ), e = d.data( "bs.affix" ), f = "of
peof b&&e[b]() ) } } var c = function(b, d) { this.options = a.extend( {}, c.DEFAULTS, d ), this.$target = a
a.proxy( this.checkPosition, this ) ).on( "click.bs.affix.data-api", a.proxy( this.checkPosition
1, this.pinnedOffset = null, this.checkPosition() ); c.VERSION = "3.3.7", c.RESET = "affix affix-top
ate = function(a, b, c, d) { var e = this.$target.scrollTop(), f = this.$element.offset(), g = this.$tar
ottom" == this.affixed ) return null != c ? ! ( e + this.unpin <= f.top ) && "bottom" : ! ( e + g <= a - d ) && "bottom
c&&e <= c ? "top" : null != d && i + j >= a - d && "bottom" }, c.prototype.getPinnedOffset = function() { if ( thi
RESET ).addClass( "affix" ); var a = this.$target.scrollTop(), b = this.$element.offset(); return
hEventLoop = function() { setTimeout( a.proxy( this.checkPosition, this ),
height(), d = this.options.offset, e = d.top, f = d.bott
f e&& ( e = d.top( this.$element )
, cse( "top"

```


C programming fundamentals: Labo**Analyse: Oefening 5.3.17****1) Opgave:**

5.3.17. Write a program that draws a zigzag line until the user enters 'n' to stop. The first portion of the line is made out of a number of blocks determined by the user. The next portion is only half that length, ... If the user wants to draw several zigzag lines, make sure they are all well positioned and drawn correctly.

```
Welcome to ZIGZAG world!  
How wide do you want your ZIGZAG line?  
Enter a number between 3 and 60: 20
```



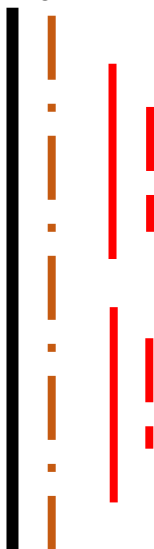
```
Do you want to draw another ZIGZAG line? (y/n) n
```





“Schrijf een programma dat een zigzaglijn tekent tot dat de gebruiker ‘n’ invoert om te stoppen. Het eerste deel van de lijn is gemaakt uit een aantal blokken bepaald door de gebruiker. Het volgende deel is slechts half die lengte,.. Als de gebruiker een paar zigzaglijnen wil tekenen zorg dan ervoor dat deze correct gepositioneerd zijn.”

In de opgave wordt er gezegd dat de eerste lijn gemaakt wordt van een aantal blokken bepaald door de gebruiker. Uiteindelijk is dit aantal, de breedte van de eerste lijn aangezien er voor elke nieuwe lijn een blokje wordt geplaatst op het einde van deze lijn dus er worden ‘n’ aantal lijnen en ‘n’ blokjes geprint voor de eerste zigzaglijn(zzlijn).

De volgende zigzaglijnen zijn steeds half zo lang als de vorige. Dus als zzlijn(1) ‘n’ als lengte heeft dan heeft zzlijn(2) als lengte ‘n/2’.

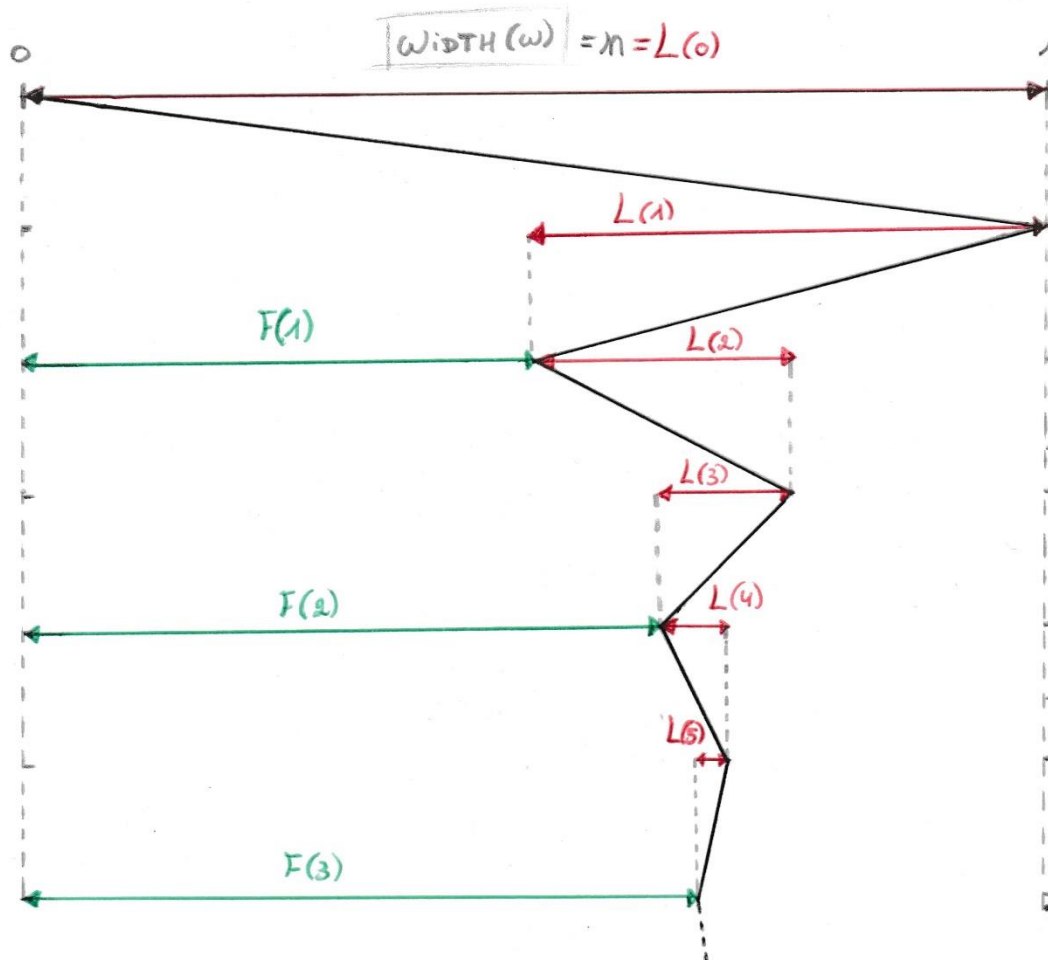
In ons C programma gaat onze opdeling er als volgt uitzien:



-  = While loop: Het doel van deze loop is om te zien of de gebruiker wild blijven lijnen tekenen.
-  = For loop: Bepaald hoeveel lijnen er maximum worden getekend.
-  = For loop: Bepaald de lengte van een lijn (hier zal onze functie L(k) worden geïmplementeerd worden).
-  = For loop: Print de nodige spaties op de huidige lijn om een blok op de juiste plaats te printen (hier zal onze functie F(v) worden geïmplementeerd).

2) Schema:

Stel dat 'n', het aantal blokjes en lijnen, dus de breedte (Width) is van de eerste zigzaglijn, en dat deze breedte gelijk is aan 1 dan kunnen we zien hoe deze breedte veranderd elke keer als we een bijkomende zigzaglijn aanmaken.



3) Functies: Lengte (Length) en Compensatie (Offset)

Uit het schema kunnen we een paar waarde uitleiden en al een paar functies opstellen:

1) De lengte ofwel de breedte van elke lijn.

Stel de lengte functie 'L' in functie van 'k' met $k \in [0; +\infty[$ dan krijgen we de volgende waarde tabel:

LENGTH: $L(k)$	k
1	0
$\frac{1}{2}$	1
$\frac{1}{2}$	2
$\frac{1}{4}$	3
$\frac{1}{8}$	4
$\frac{1}{16}$	5
$\frac{1}{32}$...

$L(k)$	k
1,0000000	0
0,5000000	1
0,2500000	2
0,1250000	3
0,0625000	4
0,0312500	5
0,0156250	6
0,0078125	7
0,0039063	8
0,0019531	9
0,0009766	10
0,0004883	11
0,0002441	12
0,0001221	13
0,0000610	14
0,0000305	15
0,0000153	16

Uit deze tabel kunnen we de functie $L(k)$ uitleiden:

$$(1) \quad L(k) = \frac{1}{2^k}$$

Als we deze functie willen schalen m.b.v. onze initiële breedte 'w' dan wordt deze functie als volgt:

$$(2) \quad w \cdot L(k) = \frac{w}{2^k}$$

In het schema kunnen we zien dat er zigzaglijnen zijn die zich naar rechts begeven en andere naar links. Zoals vermeld in het begin zullen er vijf "for loops" gemaakt worden in het totaal. De vier kleinere "for loops" is waar het echt over gaat aangezien in deze loops ook de lengte van een lijn wordt berekend om te weten hoeveel er moet verschuift worden naar links of naar rechts daarom zou men de $L(k)$ functie kunnen opsplitsen in 2 functies waarvan een de lengte aanduidt van de zigzaglijnen die zich naar rechts verplaatsen en een andere functie voor de zigzaglijnen die zich naar links verplaatsen.

Hier door krijgen we de volgende functies:

$$(3) \quad L(k) = \begin{cases} L_1 = \frac{1}{2^{2k-2}} & \text{als } k \bmod(2) = 0 \\ L_2 = \frac{1}{2^{2k-1}} & \text{als } k \bmod(2) = 1 \end{cases}$$

2) De compensatie tussen de origine en de getrokken zigzaglijnen.

Stel de compensatie functie 'F' in functie van 'v' met $v \in [1; +\infty[$ dan krijgen we de volgende waarde tabel:

OFFSET:	F(v)	v
	$\frac{1}{2}$	1
$+\frac{1}{8}$ ↖	$\frac{5}{8}$	2
$+\frac{1}{32}$ ↖	$\frac{21}{32}$	3
$+\frac{1}{128}$ ↖	$\frac{85}{128}$	4
$+\frac{1}{512}$ ↖	$\frac{341}{512}$	5
$+\frac{1}{2048}$ ↖	$\frac{1365}{2048}$	6

F(v)	v
0,500000000	1
0,625000000	2
0,656250000	3
0,664062500	4
0,666015625	5
0,666503906	6
0,666625977	7
0,666656494	8
0,666664124	9
0,666666031	10
0,666666508	11
0,666666627	12
0,666666657	13
0,666666664	14
0,666666666	15
0,666666667	16

Uit deze tabel kunnen we de functie $F(v)$ uitleiden:

$$(4) \quad F(v) = \sum_{i=1}^v \frac{1}{2^{(2i-1)}}$$

Deze functie kan ook geschaald worden m.b.v. 'w'. de functie wordt dan:

$$(5) \quad w \cdot F(v) = w \cdot \sum_{i=1}^v \frac{1}{2^{(2i-1)}}$$

4) Limiet van de functies

Als we de waarde tabel bekijken van $F(v)$, kunnen we zien dat, hoe groter 'v' wordt hoe meer $F(v)$ naar een bepaalde waarde convergeert.

We kunnen ook zien dat de lengte $L(k)$ steeds kleiner wordt en dat deze ook convergeert.

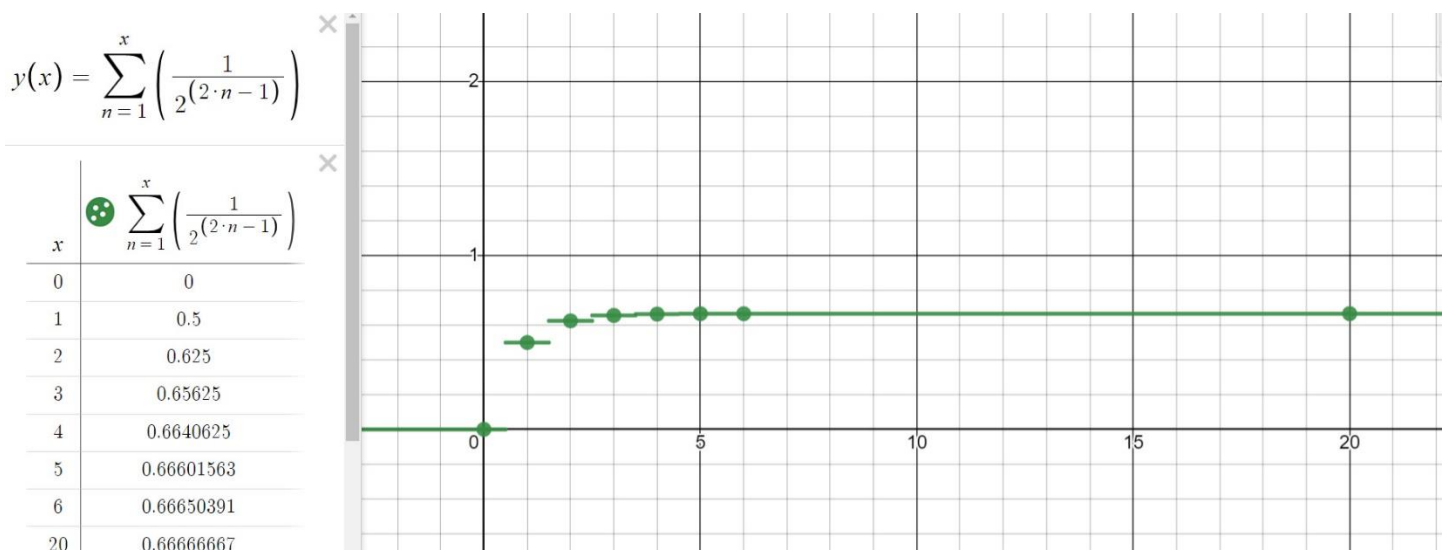
(Hoe we kunnen zien of $F(v)$ en $L(k)$ een convergente som is of een divergente som is een oefening die ik overlaat aan de lezer.)

De notatie voor de limiet te berekenen van $F(v)$ schrijven we als volgt:

$$(6) \quad \lim_{v \rightarrow \infty} F(v) = \sum_{i=1}^{\infty} \left(\frac{1}{2^{(2i-1)}} \right) \simeq \frac{2}{3}$$

We kunnen dan ook zo deze limiet schalen met 'w'

$$(7) \quad w \cdot \lim_{v \rightarrow \infty} F(v) \simeq \frac{2w}{3}$$



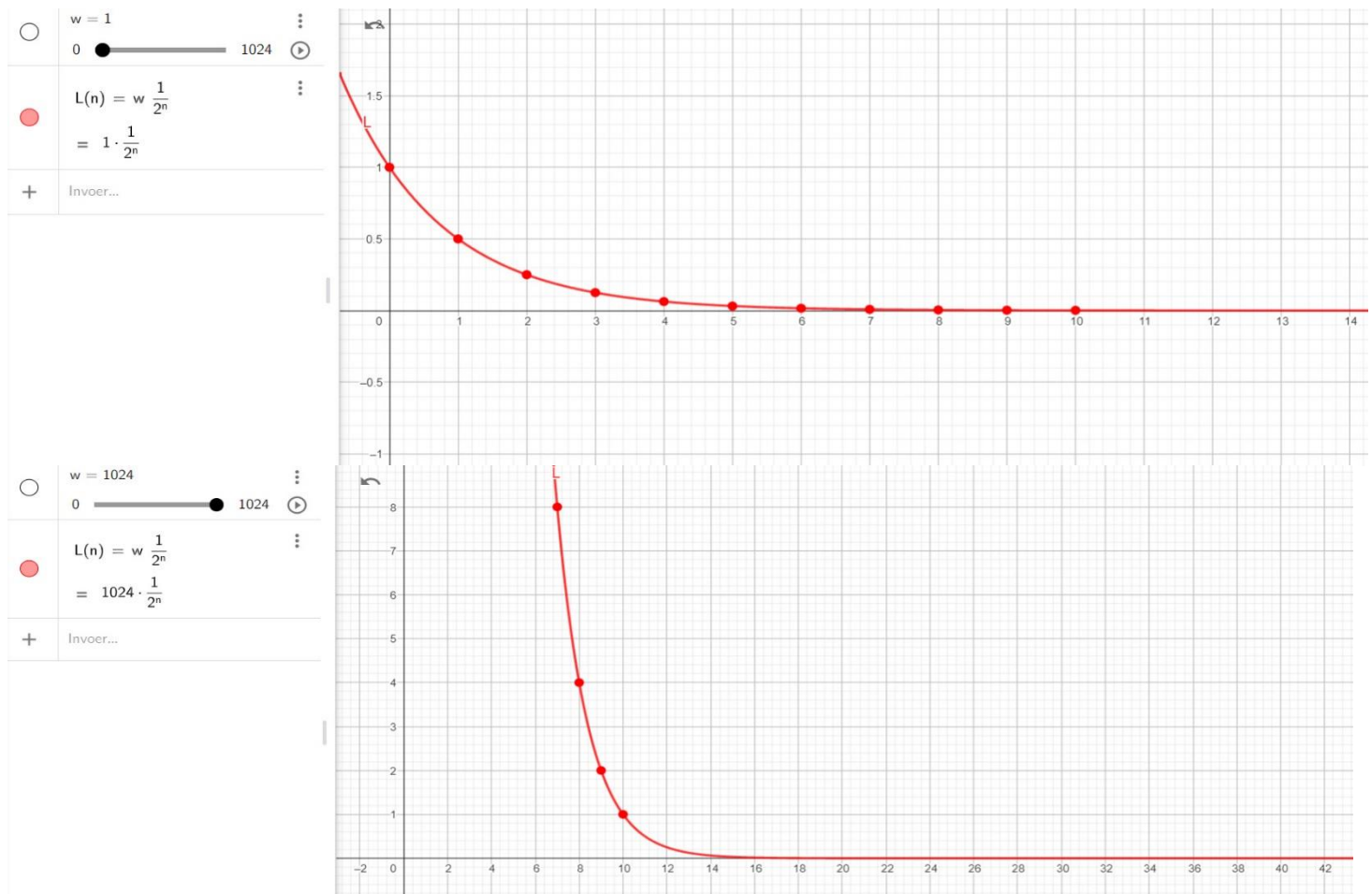
Hier kunnen we zien dat de limiet van $L(k)$ effectief $2/3$ benadert wanneer $w = 1$.

Indien $w \neq 1$ zal de limiet $w \cdot (2/3)$ benaderen.

We doen het zelfde voor $L(k)$:

$$(8) \quad \lim_{k \rightarrow \infty} L(k) = \frac{1}{2^\infty} = 0$$

De limiet schalen op basis van 'w' heeft niet veel nut want de lengte van de zigzaglijnen zal altijd convergeren naar 0. Hoe groter 'w' wordt hoe meer zigzaglijnen er wel kunnen getekend worden maar uiteindelijk zal de lengte zo klein zijn dat het onmogelijk gaat zijn voor ons programma om de zigzaglijnen te tekenen.



Hier kunnen we zien dat als $w = 1$ onze lengte al na +/- 6 zigzaglijnen 0 bereikt. Als $w = 1024$ zien we dat na +/- 14 iteraties we ons 0 punt bereiken. Alles dat op de grafiek onder de y-waarde $y = 1$ ligt zal niet kunnen worden afgeprint door ons C programma omdat 1 blokje al de breedte heeft van $w = 1$.

5) Recuratieve functies van de exponentiele

Het grote probleem waar we nu nog met zitten is hoe we dit zouden kunnen implementeren in ons C programma aangezien de functies die we hier voorstellen exponentiele functies zouden we de "exp()" functie kunnen gebruiken maar dat zou teveel rekenwerk vragen telkens als we dit in de "for loops" zouden moeten berekenen daarom gaan we de exponentiele functie omzetten in een recursieve functie zodat we de waarde kunnen aanpassen als we door ons C programma gaan.

Als we $L(k)$ bekijken in (3)

$$L(k) = \begin{cases} L_1 = \frac{1}{2^{2k-2}} & \text{als } k \bmod(2) = 0 \\ L_2 = \frac{1}{2^{2k-1}} & \text{als } k \bmod(2) = 1 \end{cases}$$

Dan zien we twee exponentiele functies namelijk:

$$2^{2k-1} \quad \& \quad 2^{2k-2}$$

Ook in $F(v)$ hebben we een exponentiele functie in de vorm van

$$2^{2k-1}$$

Door een paar waarde tabellen van deze functies op te stellen kunnen we volgende conclusies trekken:

$$2^{2k} \Leftrightarrow f(k) = f(k-1) \cdot 4 \\ \text{met } f(0) = 1$$

$$2^{2k-1} \Leftrightarrow f(k) = f(k-1) \cdot 4 \\ \text{met } f(0) = \frac{1}{2}$$

$$2^{mk} \Leftrightarrow f(k) = f(k-1) \cdot 2^m \\ \text{met } f(0) = 1$$

$$(9) \quad 2^{mk-R} \Leftrightarrow f(k) = f(k-1) \cdot 2^m \\ \text{met } f(0) = \frac{1}{2^R}$$

Verder gaan we hier niet op in maar nogmaals het is zeker een goede oefening om de recursieve te introduceren voor mensen die nog weinig van dit begrip hebben gezien.

m.b.v. formule (9) kunnen we in beide functies de exponentiele vervangen en zo implementeren in ons C programma.

6) Detail over de implementering. Hoe?

Zo als het in het begin schema werd getoond gaan we in totaal 5 “for loops” hebben waarvan een grote die alleen maar gebruikt wordt om het aantal iteraties te veranderen. In deze grote “for loop” hebben we vier kleinere “for loops” waar het echte werk gebeurt:

```
for (int x = 1; x < 14; x++)
{
    for (int t = 0; t < (width / (k / 4)); t++)
    {
        for (int n = 0; n < t + offset; n++)
        {
            printf(" ");
        }
        printf("\xDB\n");
    }

    for (int m = (width / (k / 4)); m > (width / (k / 2)); m--)
    {
        for (int s = 0; s < m + offset; s++)
        {
            printf(" ");
        }
        printf("\xDB\n");
    }

    offset = offset + (width / (k / 2));
    k *= 4;
}
```

Dit is de grote “for loop” die het aantal iteraties aangeeft

Dit is een “for loop” die de lengte aangeeft van een zigzaglijn m.b.v. de recursieve vorm van L_1 . De initiële waarde van deze functie is $\frac{1}{4}$ en de initiële waarde van ‘k’ is 2. Na de recursieve vorm gevonden te hebben schalen we dit nog met de breedte ‘w’.

De “offset” is de $F(v)$ functie die elke nieuwe iteratie van ‘x’ wordt aangepast

Dit is een “for loop” die de lengte aangeeft van een zigzaglijn m.b.v. de recursieve vorm van L_2 . De initiële waarde van deze functie is $\frac{1}{2}$ en de initiële waarde van ‘k’ is 2. Na de recursieve vorm gevonden te hebben schalen we dit nog met de breedte ‘w’.

De “offset” is de $F(v)$ functie die elke nieuwe iteratie van ‘x’ wordt aangepast

De “offset” is de $F(v)$ functie die hard op L_2 functie lijkt behalve dat telkens als we de volgende waarde gaan berekenen sommen we dit bij de oude waarde bij op. Bij het einde van de grote “for loop” passen we ook nog ‘k’ aan door de waarde met x 4 te verhogen.