be-OI 2011 **Finale** 30 maart 2011	**Gegevens invullen in HOOFDLETTERS en LEESBAAR, aub** VOORNAAM : ..... NAAM : ..... SCHOOL :	**Gereserveerd**

Belgische Olympiades in de Informatica (duur : 1u40 maximum)

Dit is de vragenlijst van de finale van de Belgische Olympiades 2011 voor de **categorie hoger onderwijs**. Ze bevat 6 vragen waarvoor je **maximum 1u40** de tijd krijgt. Naast elke vraag staat een indicatie van de tijd die het kan kosten om de vraag op te lossen. Dat is slechts een schatting.

Algemene opmerkingen (lees dit aandachtig voor je begint)

1. Vul je voornaam, naam en school in, **alleen op het eerste blad**. Op alle andere bladzijden mag je enkel schrijven in de daarvoor **voorziene kaders**. Als je door een fout toch buiten de antwoordkaders moet schrijven, moet je wel op hetzelfde blad papier verder schrijven. Anders kunnen we je antwoord niet verbeteren.

2. Je mag alleen iets om te schrijven bij je hebben. Rekentoestel, GSM, ... zijn **verboden**.

3. Schrijf je antwoorden met blauwe of zwarte **pen of balpen**. Laat geen antwoorden staan in potlood. Als je kladbladen wilt, vraag ze dan aan een toezichthouder.

4. Je **moet** op de vragen antwoorden in **pseudo-code** of in één van de **toegestane programmeertalen** (Java, C, C++, Pascal, Python, PHP). We trekken geen punten af voor syntaxfouten. Je mag geen voorgedefinieerde functies gebruiken, behalve diegenen die op de volgende bladzijde staan.

5. Je mag **op geen enkel moment met iemand communiceren**, behalve met de organisatoren of toezichthouders. Alle technische en inhoudelijke vragen mag je enkel stellen aan de organisatoren. Logistieke vragen mogen gesteld worden aan de toezichthouders.

6. Je mag je **plaats niet verlaten** tijdens de proef, bijvoorbeeld om naar het toilet te gaan of te roken. Afhankelijk van de plaats waar je de proef aflegt, kan het ook verboden zijn om te eten of te drinken in het lokaal.

7. Je krijgt **exact 1 uur en 40 minuten** de tijd om op de vragen te antwoorden. Een **overzicht** over pseudo-code en de afspraken en conventies die we hanteren staat op de volgende bladzijde.

Succes !

001

Vragenlijst finale hoger onderwijs

Overzicht pseudo-code

In de vragenlijst kom je opgaves tegen waarbij je een stuk code moet aanvullen. We vragen je om dat te doen met ofwel *expressies*, ofwel *instructies*. Expressies berekenen een waarde, instructies voeren een actie uit.

Expressies bezitten een waarde en worden gebruikt in berekeningen, om de waarde van een variabele te veranderen, en in condities. Sommige expressies hebben een numerieke waarde (een geheel of reëel getal), andere een booleaanse waarde (**`true`** of **`false`**). Instructies stellen acties voor die uitgevoerd moeten worden. Een algoritme is een opeenvolging van instructies.

De meest eenvoudige expressies zijn de gehele getallen zelf (zoals 3, -12...) en variabelen (zoals x, sum...). We kunnen een expressie ook samenstellen met operatoren zoals de volgende tabel toont.

operatie	**notatie**	**voorbeelden**	
gelijkheid	$=$	$3 = 2$ is **`false`**	$1 = 1$ is **`true`**
verschil	$\neq$	$1 \neq 4$ is **`true`**	$3 \neq 3$ is **`false`**
vergelijking	$>, \geq, <, \leq$	$1 > 3$ is **`false`**	$5 \leq 5$ is **`true`**
optellen en aftrekken	$+, -$	$3 + 2$ is 5	$1 - 9$ is -8
vermenigvuldigen	$\times$	2×7 is 14	-2×3 is -6
gehele deling	$/$	$10/3$ is 3	$9/3$ is 3
rest van de gehele deling	$\%$	$10\%3$ is 1	$9\%3$ is 0
logische "niet"	**`not`**	**`not`** $3 = 2$ is **`true`**	**`not`** $1 = 1$ is **`false`**
logische "en"	**`and`**	$3 = 3$ **`and`** $5 \leq 9$ is **`true`**	$1 \geq 3$ **`and`** $2 = 2$ is **`false`**
logische "of"	**`or`**	$3 = 3$ **`or`** $12 \leq 9$ is **`true`**	$1 \geq 3$ **`or`** $2 \neq 2$ is **`false`**
toegang tot het element van een array	`[]`	gegeven arr is `[1,2,3]`, dan arr`[1]` is 2	
functie oproepen die een waarde teruggeeft		`size(`*list*`)`	`min(12, 5)`

De functies die jullie mogen gebruiken zijn, naast diegenen die in de opgaves en op deze bladzijde worden gedefinieerd, ook de functies `max(`a`, `b`)`, `min(`a`, `b`)` en `pow(`a`, `b`)`. Zij berekenen respectievelijk het grootste van twee getallen, het kleinste van twee getallen, of de machtsverheffing (a^b).

De meest eenvoudige instructies staan in deze tabel samengevat :

operatie	**notatie**	**voorbeeld**
toekenning	$\leftarrow$	$x \leftarrow 20 + 11 \times 2$
teruggave	**`return`**	**`return`** 42
functie oproepen (die niets teruggeeft)	naam van de functie	`sort(`*list*`)`

Daarnaast hebben we ook *controle-instructies* waarvan we er 3 gebruiken : **`if`**–**`else`**, **`while`** en **`for`**. Die laten toe een groep andere instructies uit te voeren, afhankelijk van een conditie. De notatie **`for`** `(`$i \leftarrow a$ **`to`** b **`step`** k`) { [...] }` staat voor een lus die herhaald wordt zolang $i \leq b$, waarbij i begint vanaf a en telkens verhoogd wordt met k aan het eind van elke herhaling.

We kunnen variabelen gebruiken om een waarde in op te slaan, en *arrays* om meerdere waarden op te slaan. Een array arr van grootte n wordt geïndexeerd van 0 tot $n - 1$. De notatie arr`[`i`]` geeft toegang tot het $(i + 1)$-de element van de array. Zo is het eerste element van die array arr`[0]`. We kunnen een nieuwe array arr aanmaken van grootte n, waarvan alle elementen geïnitialiseerd zijn op 0, met de notatie $arr \leftarrow$ `newArray(`n`)`. We kunnen array arr van grootte n kopiëren naar een nieuwe array $newArr$ kopiëren met de notatie $newArr \leftarrow arr$.

Over de kost van operaties

Bij vragen 2 en 3 wordt gevraagd een zo efficiënt mogelijk algoritme te schrijven om de maximumscore te kunnen halen.

Als we de tijdscomplexiteit van een programma willen kennen (een schatting van de tijd die het zal kosten om uit te voeren), is één van de mogelijke manieren het tellen van het aantal elementaire operaties die het algoritme uitvoert. Alle basisinstructies laten we meetellen als één tijdseenheid. Het aanmaken van een array van grootte n, of het kopiëren van zo'n array, kost n tijdseenheden. Voor de **`if`**–**`else`**, **`while`** en **`for`** instructies, tellen we één tijdseenheid voor het uitrekenen van de conditie. Daarna tellen we alle instructies die worden uitgevoerd.

Vraag 1 – De langste strikt stijgende reeks (10 min)

Het algoritme hieronder berekent, gegeven een array arr van n gehele getallen, de lengte van het langste aaneengesloten deel van de array waarvan de inhoud een strikt stijgende reeks getallen is. Bijvoorbeeld, in de volgende array :

1	4	2	5	7	2	2

zien we dat er 10 deelreeksen zijn die strikt stijgend zijn. Geklasseerd volgens lengte zijn ze :

- `[]`
- `[1]`, `[4]`, `[2]`, `[5]`, `[7]`
- `[1,4]`, `[2,5]`, `[5,7]`
- `[2,5,7]`.

Voor dit voorbeeld gaat het algoritme dus de waarde 3 teruggeven (de lengte van `[2,5,7]`). We vragen je om het onderstaande algoritme te vervolledigen, zodat het correcte resultaat wordt teruggegeven op de manier die we beschreven hebben, ongeacht de lengte van de array arr.

Opmerking : De functie `max(a, b)` berekent het maximum van a en b, d.w.z. geeft de grootste van de twee waarden terug.

```
Input  : • arr, een array van gehele getallen.
         • n, een positief geheel getal (n ≥ 0), de lengte van de array arr.
Output : • de lengte van de langste strikt stijgende deelreeks in arr wordt
           teruggegeven.
         • de array arr werd niet gewijzigd.

length ← [...]                                                  // Q1(a)
m ← [...]                                                       // Q1(b)

for (i ← [...] to [...] step 1)                                 // Q1(c,d)
{
    if ([...])                                                  // Q1(e)
    {
        length ← 1
    }
    else
    {
        length ← length + 1
    }
    m ← max (m, length)
}
return m
```

Q1(a) **(een expressie)**

. . . . . .

Q1(b) **(een expressie)**

. . . . . .

Q1(c) **(een expressie)**

. . . . . .

Q1(d) **(een expressie)**

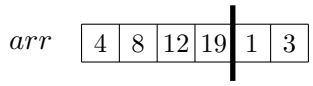
. . . . . .

Q1(e) **(een expressie)**

. . . . . .

Vraag 2 – Wat is het grootste element ? (15 min)

Gegeven een array van positieve gehele getallen (allemaal verschillend) die al oplopend gesorteerd is en die een rotatie heeft ondergaan. Een rotatie van k plaatsen wil zeggen dat alle elementen in de array k posities zijn opgeschoven naar rechts, terwijl de elementen die aan het rechtereinde uit de array komen, opnieuw aan het begin worden toegevoegd. Hier is een voorbeeld van zo'n array, met een dikke lijn links van het eerste element van de originele array. Die originele array was $[1, 3, 4, 8, 12, 19]$ en heeft hier een rotatie van 4 plaatsen ondergaan :



Schrijf een algoritme dat toelaat om het grootste element in deze reeks terug te vinden. Bovendien moet je algoritme **zo efficiënt mogelijk zijn**.

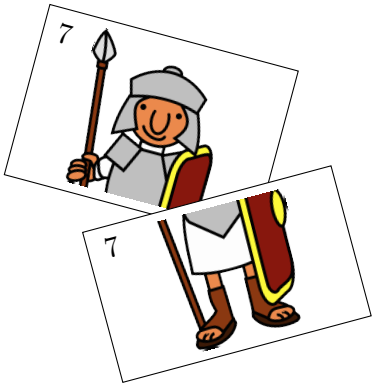
Input : • arr, een array van positieve gehele getallen (allen verschillend) waarvan de elementen oplopend gesorteerd zijn en die een rotatie van een zeker aantal plaatsen heeft ondergaan (zie opgave).
• n, een strikt positief geheel getal ($n > 0$), lengte van de array arr.
Output : • het grootste element van de array wordt teruggegeven.
• de array arr werd niet gewijzigd.

Je algoritme moet correct werken, maar we houden in de beoordeling ook rekening met :

– het aantal toegangsoperaties $arr[i]$ die worden uitgevoerd **in het slechtst mogelijk geval** ;
– de eenvoud van je algoritme. Ideaal bevat die slechts één **`while`**-lus, één **`if`**–**`else`**-structuur, enkele toekenningen aan variabelen en als laatste een instructie **`return`**.

Q2

Vraag 3 – Mijn onvolledige stickerverzameling... (15 min)



Er werd onlangs een nieuwe collectie van 84 stickers uitgebracht door PANINI© rond het thema *"Harry Potter en de Relieken van de Dood"*. De stickers zijn speciaal want ze stellen niet een volledig personage voor, maar beelden slechts de helft ervan af. 42 stickers stellen bovenkanten voor van personages, en 42 andere stickers de bijhorende onderkanten. Twee stickers (boven- en onderkant) die overeenkomen met hetzelfde personage, hebben hetzelfde nummer.

Je wilt weten hoeveel volledige personages je kan maken met je huidige verzameling stickers. Je krijgt hulp van je broer Adriaan. Je neemt zelf alle stickers van bovenkanten van personages, Adriaan neemt alle stickers van onderkanten. Je wil nu zo snel mogelijk tellen hoeveel volledige personages je kan vormen.

Schrijf een algoritme dat het aantal volledige personages telt dat je kan vormen. Het algoritme krijgt als parameters twee arrays van gehele getallen t_1 en t_2, die de kaartnummers bevatten van respectievelijk de stickers *"boven"* en de stickers *"onder"*. Bijvoorbeeld, als t_1 gelijk is aan $[1, 11, 9, 7, 7, 3, 9, 2]$ en t_2 gelijk is aan $[7, 7, 2, 8]$, moet je algoritme 3 teruggeven omdat je twee keer personage 7 en één keer personage 2 kan vormen.

Input :
- t_1, een array van gehele getallen tussen 0 en 41 (inclusief), die de nummers zijn van stickers van "bovenkanten" in je collectie.
- t_2, een array van gehele getallen tussen 0 en 41 (inclusief), die de nummers zijn van stickers van "onderkanten" in je collectie.
- m, een positief geheel getal ($m \geq 0$), de lengte van array t_1.
- n, een positief geheel getal ($n \geq 0$), de lengte van array t_2.

Output :
- het aantal volledige personages dat je kan vormen met de stickers in je collectie.
- de arrays t_1 en t_2 worden niet gewijzigd.

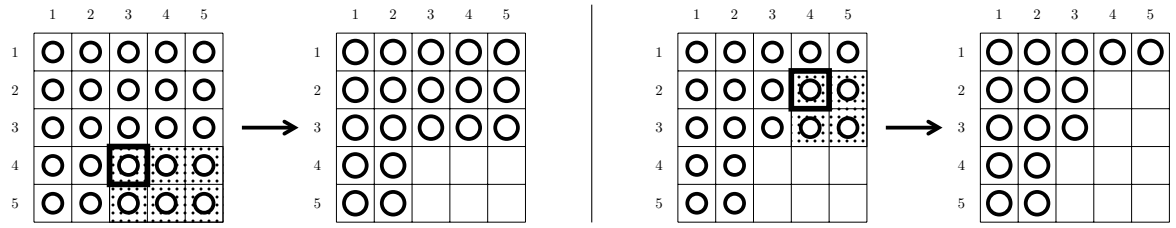
Je algoritme moet **zo efficiënt mogelijk zijn**.

Q3

Vraag 4 – Het spel Chomp (20 min)

Het spel *"Chomp"* wordt gespeeld op een spelbord met een raster van $n \times n$ vakjes waarbij $n > 2$. In het begin van het spel wordt een pion geplaatst op elk van de vakjes van het bord. Elk om beurt kiezen de spelers een pion op het bord, en nemen ze alle pionnen weg die zich in rechtsonder van die pion bevinden. De speler die de pion helemaal linksboven moet nemen, verliest het spel.

Nemen we bijvoorbeeld een bord van 5×5, dat we indexeren vertrekkend van de linkerbovenhoek. De eerste speler kiest de pion op positie $(3, 4)$. Hij zal dan alle pionnen wegnemen waarvan de verticale indices gelijk zijn aan of groter zijn dan 4, en de horizontale indices gelijk aan of groter dan 3 (figuur links). De tweede speler kan dan bijvoorbeeld $(4, 2)$ kiezen, zodat het bord belandt in de situatie hieronder afgebeeld in de figuur rechts.



Vind de strategie die je toelaat om altijd te winnen, zolang je zelf het spel mag beginnen. Deze strategie hangt niet af van de manier waarop de andere speler speelt, en ook niet van de grootte van het spelbord. Vervolledig het onderstaande algoritme dat deze strategie voorstelt, en dat je dus uitvoert telkens je aan de beurt bent.

Input : • n, een positief geheel getal ($n > 2$), de grootte van het spelbord.
• Het huidige spelbord is een configuratie waarvoor je deze zelfde strategie vanaf het begin van het spel hebt toegepast. Alle spelers respecteren de spelregels.
• $last_i$ en $last_j$, twee gehele getallen tussen 0 en $n-1$ (inclusief), respectievelijk de horizontale en verticale index van de laatste pion die je tegenspeler koos. Deze getallen zijn 0 tijdens de eerste beurt.

Output : • Een koppel waarden (i, j) (met i en j tussen 0 en $n-1$ (inclusief)) die overeenkomen met de pion die je kiest en waarmee je zeker bent dat je wint als je op dezelfde manier verder blijft spelen.

```
if (last_i = 0 and last_j = 0)
{
    return ([...], [...])                                   // Q4(a)
}
else
{
    return ([...], [...])                                   // Q4(b)
}
```

Q4(a) **(2 expressies)**

. . . . . .

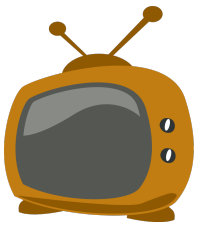
. . . . . .

Q4(b) **(2 expressies)**

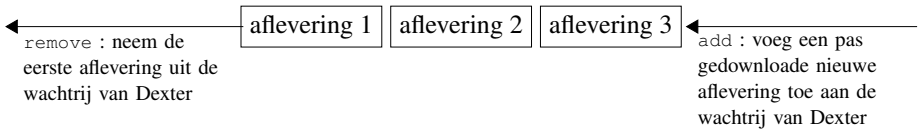
. . . . . .

. . . . . .

Vraag 5 – Wat zien we vanavond ? (20 min)



Ik download regelmatig op een legale manier de laatste afleveringen van Dexter et andere TV-reeksen. Zodra er een aflevering beschikbaar is om te downloaden, doe ik dat. Als ik een aflevering heb bekeken, verwijder ik hem van de harde schijf. Het komt wel eens voor dat ik verschillende afleveringen op mijn harde schijf heb staan omdat ik nog niet de tijd heb gehad om ze te bekijken. Ik heb dus een wachtrij van te bekijken afleveringen, en de afleveringen komen daar chronologisch in terecht. Zodra ik tijd heb, neem ik de eerste aflevering uit die wachtrij :



Situaties zoals deze kunnen we voorstellen met een gegevensstructuur die *queue* heet. Van een queue kunnen we opvragen wat de lengte is (het aantal elementen dat ze bevat), we kunnen een element toevoegen (altijd aan het eind van de queue) en we kunnen een element wegnemen (altijd van het begin van de queue). We spreken van een *FIFO*-structuur (First-In First-Out) omdat het eerste element dat wordt toegevoegd aan een queue ook het eerste is dat de queue zal verlaten.

Dit zijn de drie mogelijke operaties :

- `add (q,e)` voeg element e toe aan het eind van queue q
- `remove (q)` neem het eerste element uit de queue q en geef het terug.
- `size (q)` geef de grootte van queue q terug.

Hier is een voorbeeld waarbij een queue wordt aangemaakt, 3 elementen krijgt, en daarna volledig wordt leeggemaakt :

```
queueDexter ← een nieuwe lege queue
add (queueDexter, aflevering1)
add (queueDexter, aflevering2)
add (queueDexter, aflevering3)
while (size (queueDexter) ≠ 0)
{
    aflevering ← remove (queueDexter)
}
```

Opmerking : We spreken af dat we een queue op de volgende manier gaan voorstellen. De inhoud van de queue $queueDexter$ zullen we schrijven als $queueDexter = (aflevering1, aflevering2, aflevering3)$ waarbij $aflevering1$ het eerste element (het hoofd) van de queue is.

Nu wil graag eerst de reeksen bekijken die ik gedownload heb en die al afgesloten zijn (er komen geen nieuwe afleveringen). Elke avond twijfel ik lang over de keuze : welke reeks bekijk ik eerst ? Hier zijn bijvoorbeeld twee reeksen waartussen ik maar niet kan kiezen :

$F1 = (a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12})$ en $F2 = (b_{12}, b_{13}, b_{14})$

Ik heb nog 8 afleveringen van $F1$ te bekijken en 3 afleveringen van $F2$. Om niet meer zo te moeten twijfelen, stel ik een algoritme voor dat de keuze in mijn plaats moet maken : **we gaan de elementen van $F2$ met die van $F1$ vermengen zodat ze gelijkmatig en evenwichtig gespreid zijn, waarbij de volgorde behouden blijft, en zodat $F1$ en $F2$ op min of meer hetzelfde moment eindigen.** Het aantal afleveringen van de lijst $F1$ die bekeken worden tussen de afleveringen van $F2$ in zal altijd hetzelfde zijn, behalve op het einde waar er minder kunnen zijn.

Als we op deze manier $F2$ in $F1$ vermengen, wordt $F1 : (a_3, a_4, a_5, \underline{b_{12}}, a_6, a_7, a_8, \underline{b_{13}}, a_9, a_{10}, a_{11}, \underline{b_{14}}, a_{12})$ en de queue $F2$ is dan leeggemaakt. Ik kijk dan elke keer 3 afleveringen van $F1$ vooraleer ik een aflevering van $F2$ bekijk, behalve op het eind. Het algoritme behoudt uiteraard de volgorde van de afleveringen zoals ze in het begin in beide queues zaten.

Voordat we het algemene algoritme bekijken, definiëren we eerst een algoritme dat toelaat om de n eerste elementen van een queue te verplaatsen naar het eind van diezelfde queue.

Input : • Q een niet lege queue met minstens n elementen
• n, positief geheel getal.
Output : • De n eerste getallen van de queue Q zijn verplaatst naar het eind van de queue met behoud van de volgorde.

```
function move (Q, n)
{
    [...]                                                    // Q5(a)
}
```

Q5(a) **(2 tot 5 instructies)**

Nu moet je het algemene algoritme vervolledigen dat de elementen van $F2$ mengt in $F1$. Daarvoor kan je de functie `move` gebruiken die je net hebt gedefinieerd.

```
Input  : • F1 en F2, twee niet lege queues zodat F2 minder groot is dan F1.
Output : • De elementen van F2 zijn gelijkmatig vermengd in F1.
           Het algoritme geeft geen waarde terug.

[...]                                                          // Q5(b)
while ([...])                                                  // Q5(c)
{
    [...]                                                      // Q5(d)
}
[...]                                                          // Q5(e)
```

Q5(b) **(een instructie)**

. . . . . .

Q5(c) **(een expressie)**

. . . . . .

Q5(d) **(twee instructies)**

. . . . . .

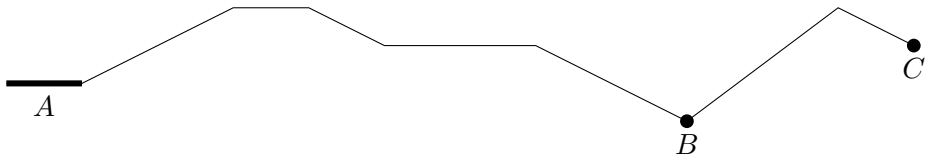
. . . . . .

Q5(e) **(een instructie)**

. . . . . .

Vraag 6 – Hoeveel dalen ? (20 min)

Je vertrekt op expeditie met je neef Simon, in een bergachtig gebied. Een vreemde vraag komt plots in jullie op : hoeveel *dalen* zijn er op jullie parcours ? De volgende figuur stelt het verloop van de hoogte voor op het hele traject dat jullie willen afleggen.



Op dit traject tellen we 3 **dalen**. We definiëren een *dal* als een plaats (een punt of een verzameling punten op gelijke hoogte) van waaruit je niet anders kan dan eerst te stijgen als je een ander punt langs het traject wil bereiken (in beide richtingen). Bijvoorbeeld, in bovenstaand schema, zijn alle vet aangeduide punten en lijnen *dalen* (aangeduid met de letters A, B en C). In het extreme geval, bevat een volledig plat traject dus één enkel dal.

Je vriend José stelt een algoritme voor dat toelaat om het aantal dalen te tellen. Hij stelt voor om de hoogtes voor te stellen als een niet-lege array van gehele getallen. Dan bevat bijvoorbeeld de array `[4]` één dal. De arrays `[3,8]`, `[2,2]` en `[12,5]` bevatten er elk één. De array `[3,1,8]` bevat er ook één. De array `[1,1,1,5,5,9,9,9,8,6,12,17,17,4,4,1,-12]` bevat er 3. Vervolledig het algoritme dat hij voorstelt, zodat het correct berekent wat er gevraagd wordt.

```
Input  : • arr, een niet-lege array van gehele getallen.
         • n, een strikt positief geheel getal (n > 0), de lengte van array arr.
Output : • Het aantal gevonden "dalen" wordt teruggegeven.
         • De array arr is niet gewijzigd.

nb ← 1
inDal ← true

for (i ← 1 to n − 1 step 1)
{
    if ([...])                                                    // Q6(a)
    {
        nb ← nb + 1
    }
    inDal ← [...]                                                 // Q6(b)
}
return nb
```

Q6(a) **(een expressie)**

. . . . . .

Q6(b) **(een expressie)**

. . . . . .