be-OI 2012 **Finale** 14 maart 2012	**Gegevens invullen in HOOFDLETTERS en LEESBAAR aub** VOORNAAM : ..... NAAM : ..... SCHOOL :	**Gereserveerd**

Belgische Olympiade in de Informatica (duur : 1u15 maximum)

Dit is de vragenlijst van de finale van de Belgische Olympiade in de Informatica 2012. Ze bevat 6 vragen waarvoor je **maximum 1u15** 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. Op open vragen **moet** je 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 15 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

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, en 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 3 en 5 wordt gevraagd een zo efficiënt mogelijk algoritme te schrijven om de maximumscore te 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 – Opwarming (5 min)

Vervolledig dit algoritme.

```
Input: tab, een array van lengte n met gehele getallen.
Output : teruggegeven waarde: het aantal keer dat een waarde
             verschillend van 5 en 7 voorkomt in tab

count ← 0

for (i ← 0 to n − 1 step 1)
{
    if ( [...] )                                          // Q1
    {
        count ← count + 1
    }
}
return count
```

Q1 **(een expressie)**

. . . . . .

Vraag 2 – De herschrijfmachine (15 min)

John C. Onway heeft een nieuwe machine uitgevonden. De machine bestaat uit twee tapes : een invoer-tape die door de machine gelezen kan worden, en een uitvoer-tape waarop de machine kan schrijven. Op de invoer-tape schrijft John een reeks `0` en `1` symbolen. De machine werkt volgens een zeer eenvoudig principe : ze kijkt naar elke twee opeenvolgende symbolen op de invoer-tape, en schrijft voor elk paar een nieuw symbool op de uitvoer-tape, volgens deze regels :

Invoer			Uitvoer
0	0	→	0
0	1	→	1
1	0	→	1
1	1	→	0

De eerste regel geeft aan : als de machine een `0` leest, gevolgd door een andere `0` op de invoer-tape, dan zal ze een enkele `0` op de uitvoer-tape schrijven.

De machine zal zo elke twee opeenvolgende symbolen op de invoer-tape verwerken door telkens de lees- en schrijfkop één positie naar rechts te bewegen. Voor het laatste symbool op de invoer-tape veronderstelt de machine impliciet dat het volgende symbool een `0` is. Natuurlijk zorgt John ervoor dat de invoer-tape en de uitvoer-tape steeds dezelfde lengte hebben.

Q2(a) John heeft een invoer-tape met de volgende symbolen in de machine ingeladen : | 1 | 0 | 1 | 0 | 1 |

Welke symbolen zullen er op de uitvoer-tape staan nadat de machine de volledige invoer-tape verwerkt heeft ?

Q2(a) **(een reeks symbolen)**

... ...

Q2(b) John laadt de uitvoer-tape van vraag Q2(a) terug in de machine als invoer-tape, en zet een nieuwe blanco uitvoer-tape in de machine. **Welke symbolen** staan er nu op de nieuwe uitvoer-tape ?

Q2(b) **(een reeks symbolen)**

... ...

Q2(c) John herhaalt dit proces nu nog 6 keer (de uitvoer van de vorige stap wordt telkens de invoer van de volgende stap, te beginnen met als invoer het antwoord op vraag Q2(b)). **Welke symbolen** staan er na de 6^{de} iteratie op de uitvoer-tape ?

Q2(c) **(een reeks symbolen)**

... ...

Q2(c) John herhaalt dit proces nu nog eens 73 keer, te beginnen met als invoer het antwoord op vraag Q2(c). **Welke symbolen** staan er na de 73^{ste} iteratie op de uitvoer-tape ?

Q2(d) **(een reeks symbolen)**

... ...

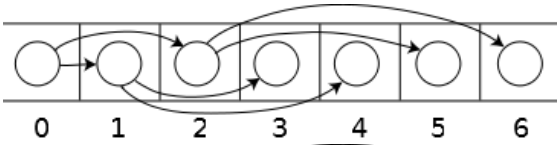
Vraag 3 – Onze gemeenschappelijke voorouders (15 min)

Een stamboom is een boom die een overzicht geeft van alle voorouders van een gegeven persoon. Die persoon zelf vormt het begin van de boom (de wortel). Net boven die persoon verdeelt de boom zich in twee takken : de linkertak vertegenwoordigt diens vader en de rechtertak diens moeder. Elk van die twee takken wordt opnieuw in twee takken onderverdeeld om de twee ouders van vader en moeder voor te stellen (dit zijn dus de grootouders van de persoon op de wortel). Elk van die takken is opnieuw in twee verdeeld, enzovoort. We kiezen ervoor om de vader steeds op de linkertak te plaatsen, en de moeder op de rechtertak.

Mijn achternicht heeft haar stamboom reeds uitvoerig bestudeerd. Zij is de wortel van haar stamboom en heeft al haar voorouders tot in de 16de eeuw er in opgenomen. Ik heb haar gevraagd me het gedeelte van haar boom te bezorgen dat ook betrekking heeft op mij, zijnde : de stamboom van onze gemeenschappelijke overgrootmoeder.

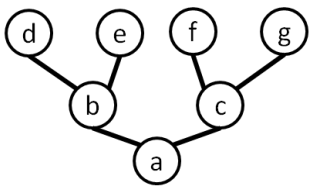
Aangezien zij haar stamboom digitaal heeft opgeslagen zouden we graag een programma hebben dat, gegeven een stamboom, een gedeelte ervan teruggeeft, vertrekkende van een bepaalde persoon in de boom. In ons bijzonder geval willen we dus het gedeelte van de boom bekomen dat zich boven onze gemeenschappelijke overgrootmoeder bevindt.

In de informatica komen we vaak *binaire* bomen tegen zoals deze. Elk element in zo'n boom wordt een *knoop* genoemd en de beginknoop wordt de *wortel* genoemd. Binaire bomen (en dus ook stambomen) kunnen we als volgt in een array opslaan. De persoon die de wortel van de boom vormt bevindt zich in de eerste cel van de array (met index 0). De ouders van de persoon in de cel met index i slaan we op in de cellen met index $2i + 1$ (vader) en $2i + 2$ (moeder). Bijvoorbeeld, wanneer ik me in de figuur hieronder op index 0 bevindt, dan bevindt mijn vader zich op index 1 en mijn moeder op index 2. Mijn grootvader langs vaderskant bevindt zich op index 3 en mijn grootmoeder langs vaderskant op index 4.



We beschouwen enkel "perfecte" bomen : indien onze stamboom h generaties bevat, dan veronderstellen we dat deze allemaal volledig zijn. Dus : voor elke generatie i met $1 \leq i \leq h$ hebben we $2^{(i-1)}$ personen van die generatie. De totale lengte van de array is dan $2^h - 1$. In de voorbeeld-array hierboven zijn er drie generaties, en dus $2^3 - 1 = 7$ personen in totaal, wat overeenkomt met de 7 cellen in de array.

Voorbeeld : de volgende boom wordt voorgesteld door de array $[a, b, c, d, e, f, g]$. De wortel van de boom is a. Als we de deelboom vertrekkende van persoon c willen bekomen, moeten we de volgende array terugkrijgen : $[c, f, g]$.



We zoeken een algoritme dat, gegeven de arrayvoorstelling van een stamboom en een persoon in die stamboom, de deelstamboom vertrekkende van die persoon teruggeeft, voorgesteld in een nieuwe array.

Vervolledig het volgende algoritme zodat het doet wat gevraagd wordt. De invoer bestaat uit een perfecte stamboom (voorgesteld in de array *tab*1 van grootte N) en de index i ($0 \leq i < N$) van de cel die de persoon bevat waarvoor we de deelboom moeten berekenen. Het resultaat moet in de array *tab*2 geplaatst worden. *tab*2 moet een geldige stamboom zijn volgens dezelfde structuur.

```
Input  :
 - tab1, een array van grootte N die een perfecte stamboom voorstelt
    (m.a.w. er bestaat h ≥ 1 zodat N = 2^h − 1)
 - 0 ≤ i < N.
 - tab2, de array waarin de deelboom van tab1 geplaatst moet worden,
    heeft reeds de juiste grootte om deze deelboom voor te stellen.
Output :
 - tab2 stelt de deelboom van tab1 voor vertrekkende van de persoon in tab1[i].

k ← 0
n ← [...]                    //Q3(a)
while (i < N)
{
    for(x ← 0 to n − 1 step 1)
    {
        [...]                //Q3(b)
        k ← k + 1
    }
    i ← [...]                //Q3(c)
    n ← [...]                //Q3(d)
}
```

Q3(a) **(een expressie)**

. . . . . .

Q3(b) **(een toekenning)**

. . . . . .

Q3(c) **(een expressie)**

. . . . . .

Q3(d) **(een expressie)**

. . . . . .

Vraag 4 – Dit algoritme maakt je rijk ! (10 min)

Je beslist om een verzameling aan te leggen van *beOI* kaarten : ruilkaarten van de 50 beste deelnemers van de Informatica Olympiade. Ze worden verkocht in pakjes van 2 tot 5 kaarten. Je kent de samenstelling van een pakje al voor de aankoop. Elk van de 50 kaarten zit in minstens één pakje, maar een kaart kan natuurlijk ook in meerdere pakjes zitten. Een pakje heeft nooit dubbels.

Je streeft naar een volledige verzameling, en dat tegen de laagst mogelijke prijs. Elk pakje kost even veel. Bovendien is de lijst van alle beschikbare pakjes kaarten, en hun inhoud, bekend. Plots gaat er een lichtje branden : het is mogelijk om hiervoor een computerprogramma te schrijven.

Je realiseert je snel dat dit een complex probleem is, en je vraagt aan een vriend om een optimale aankoopstrategie hiervoor. Hij geeft je de volgende raad :

> *Koop het pakje dat zoveel mogelijk kaarten bevat die je nog niet hebt verzameld.*
> *Herhaal dit tot je alle kaarten hebt.*

Zal deze goede raad altijd, in elk geval, tot een optimale oplossing leiden, d.w.z. de goedkoopste oplossing ?

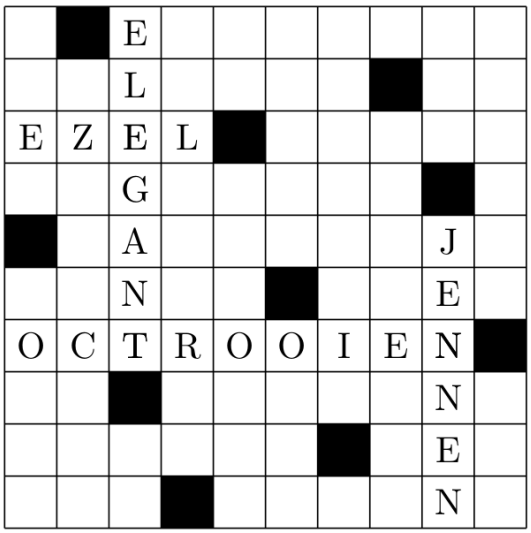
Q4	☐ Ja	☐ Nee

Als je *nee* antwoordde hierboven, geef dan een tegenvoorbeeld in de tabel hieronder. Duid aan welke kaarten in welk pakje (moeten) zitten. Je hoeft niet alle kaarten te gebruiken, en ook niet alle pakjes, maar je mag geen kolommen of rijen toevoegen.

	Kaart A	Kaart B	Kaart C	Kaart D	Kaart E	Kaart F	Kaart G
Pakje 1	☐	☐	☐	☐	☐	☐	☐
Pakje 2	☐	☐	☐	☐	☐	☐	☐
Pakje 3	☐	☐	☐	☐	☐	☐	☐
Pakje 4	☐	☐	☐	☐	☐	☐	☐
Pakje 5	☐	☐	☐	☐	☐	☐	☐
Pakje 6	☐	☐	☐	☐	☐	☐	☐

Vraag 5 – Kruiswoordraadsels (15 min)

Je bent een verstokte kruiswoordpuzzelaar en daarom besluit je om software te schrijven voor *kruiswoordraadsels op de computer*. Je beslist om een kruiswoordraadsel voor te stellen door een vierkante tabel. Daarin staan ofwel letters uit het alfabet, ofwel het "lege" (blanco) teken, ofwel de waarde '`0`' die een zwart vakje voorstelt. Hier volgt een voorbeeld van een rooster, en hoe dat in het computergeheugen wordt voorgesteld.




```
crosswords[10][10] = {
    {' ','0','E',' ',' ',' ',' ',' ',' ',' '},
    {' ',' ','L',' ',' ',' ',' ','0',' ',' '},
    {'E','Z','E','L','0',' ',' ',' ',' ',' '},
    {' ',' ','G',' ',' ',' ',' ',' ','0',' '},
    {'0',' ','A',' ',' ',' ',' ',' ','J',' '},
    {' ',' ','N',' ',' ','0',' ',' ','E',' '},
    {'O','C','T','R','O','O','I','E','N','0'},
    {' ',' ','0',' ',' ',' ',' ',' ','N',' '},
    {' ',' ',' ',' ',' ',' ','0',' ','E',' '},
    {' ',' ',' ','0',' ',' ',' ',' ','N',' '}
}
```

Vak `(i,j)` van het rooster kan je uitlezen via : `crosswords[i][j]`. Bijvoorbeeld, vakje `crosswords[0][1] = '0'` en vakje `crosswords[3][2] = 'G'`.

De eerste functionaliteit die je in je programma wil inbouwen is het tellen van het aantal woorden dat je nog moet invullen in het rooster. Een woord heeft minstens twee letters. In het voorbeeld hierboven moet het algoritme als resultaat **32** geven.

Er wordt nu gevraagd om het volgende algoritme te vervolledigen.

```
Input  :
 - nb, een strikt positief geheel getal
 - crosswords, een tabel met nb rijen en nb kolommen, waarvan elk element
   één van de volgende tekens bevat:
    - een blanco teken (' '),
    - een alfabetisch teken,
    - het teken '0' dat een zwart vakje voorstelt
Output : nbwords is het aantal woorden van minstens twee tekens die nog
                  in het rooster moeten worden ingevuld

nbwords ← 0
for i ← 0 to nb − 1 step 1
{
    p ← 0
    q ← 0
    for j ← 0 to nb − 1 step 1
    {
        if ( [...] )                                    // Q5(a)
        {
            if (p > 1)
            {
                nbwords ← nbwords + 1
            }
            [...]                                       // Q5(b)
        }
        else
        {
            p ← p + 1
        }
        if ( [...] )                                    // Q5(c)
        {
            if (q > 1)
            {
                nbwords ← nbwords + 1
            }
            [...]                                       // Q5(d)
        }
        else
        {
            q ← q + 1
        }
    }
    if ( [...] )                                        // Q5(e)
    {
        nbwords ← nbwords + 1
    }
    if ( [...] )                                        // Q5(f)
    {
        nbwords ← nbwords + 1
    }
}
return nbwords
```

Q5(a) **(een voorwaarde)**

. . . . . .

Q5(b) **(een toekenning)**

. . . . . .

Q5(c) **(een voorwaarde)**

. . . . . .

Q5(d) **(een toekenning)**

. . . . . .

Q5(e) **(een voorwaarde)**

. . . . . .

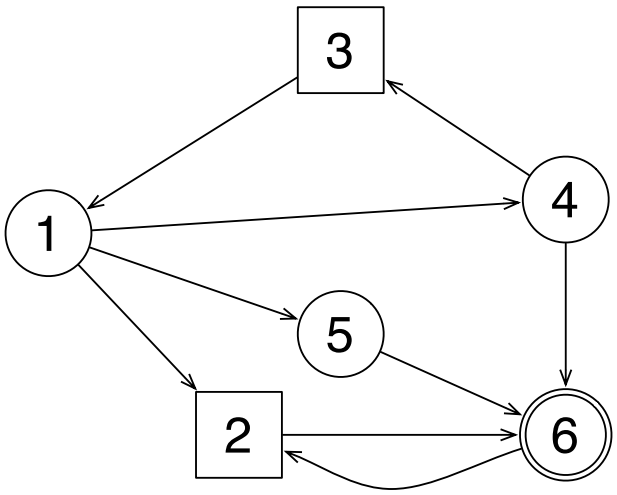
Q5(f) **(een voorwaarde)**

. . . . . .

Vraag 6 – Suske en Wiske (15 min)

Suske en Wiske spelen een zelf uitgevonden strategisch spelletje. Het spel wordt gespeeld op een bord met twee soorten vakjes : rond en vierkant. Tijdens het spel wordt een pion verplaatst over de vakjes van het bord. Tussen de vakjes bevinden zich pijlen die de geldige verplaatsingen aangeven. Het soort vakje geeft aan wie de pion mag verplaatsen : wanneer de pion op een vierkant vakje staat mag Wiske de pion verplaatsen, op een rond vakje is het Suske die mag kiezen. Beiden moeten rekening houden met de pijlen, maar het is niet verplicht dat het spel in een strikte beurtrol gespeeld wordt.

Neem bijvoorbeeld het volgende spelbord :



Het spel zou bijvoorbeeld kunnen beginnen in vakje 1, waar Suske aan de beurt is. Hij beslist om de pion te verplaatsen naar vakje 2, zodat Wiske aan de beurt is. In dat vakje heeft zij geen andere keuze dan de pion te verplaatsen naar vakje 6, waar Suske aan de beurt is. Hij verplaatst de pion naar vakje 2, enz.

Om te bepalen wie wint hebben Suske en Wiske een vakje gemarkeerd als winnend vakje voor Wiske. In het voorbeeldspelbord is dat vakje 6. Merk op dat dit zowel een rond als een vierkant vakje mag zijn. De twee vrienden hebben beslist dat Wiske wint als zij het gemarkeerde vakje bereikt. Suske wint daarentegen als hij erin slaagt het spel eindeloos te rekken zonder langs het winnende vakje te passeren. In het voorbeeld hierboven is het Wiske die wint aangezien vakje 6 bereikt wordt.

Stel dat het spel start op vakje 1 (de pion staat op vakje 1). **Naar welk vakje** moet Suske de pion verplaatsen tijdens zijn eerste beurt, en elke latere beurt waarin de pion vakje 1 bereikt, om zeker te winnen ? Als je denkt dat er geen oplossing bestaat, antwoord dan 0.

Q6(a) **(nummer van een vakje)**

. . . . . .

Als het spel daarentegen zou beginnen op een ander vakje, dan krijgt Wiske misschien de mogelijkheid om een zet te maken waardoor ze zeker wint. Beantwoord voor elk van de vakjes de volgende vraag : is het mogelijk om het spel op dat vakje te beginnen zodanig dat Wiske zeker kan winnen, als zij volgens een optimale strategie speelt, voor gelijk welke zet van Suske ? (1 punt per correct aangevinkt vakje, -1 punt per fout aangevinkt vakje, geen negatieve score mogelijk op deze deelvraag).

Q6(b)	**1**	**2**	**3**	**4**	**5**	**6**
Ja	☐	☐	☐	☐	☐	☐
Neen	☐	☐	☐	☐	☐	☐