be-OI 2013 **Finale** 9 maart 2013	**Invullen in HOOFDLETTERS en LEESBAAR aub** VOORNAAM (**HOOFDLETTERS**): ..... ACHTERNAAM (**HOOFDLETTERS**): ..... SCHOOL :	**Gereserveerd**

Belgische Informatica-olympiade (duur : 1u15' maximum)

Dit is de vragenlijst van de finale van de Belgische Informatica-olympiade 2013. Ze bevat zes vragen, en je krijgt **maximum 1u15'** de tijd om ze op te lossen. Naast elke vraag vind je een schatting van de benodigde tijd om ze op te lossen, en de maximale score die je kan behalen.

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, schrijf dan alleen verder op hetzelfde blad papier. Anders kunnen we je antwoord niet verbeteren.

2. Je mag alleen schrijfgerief 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 wil, vraag ze dan aan een toezichthouder.

4. Voor alle code in de opgaven werd **pseudo-code** gebruikt. Op de volgende bladzijde vind je een **beschrijving** van de pseudo-code die we hier gebruiken.

   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. Tenzij het anders vermeld staat, mag je geen voorgedefinieerde functies gebruiken, met uitzondering van `max(a,b)`, `min(a,b)` en `pow(a,b)`, waarbij deze laatste a^b berekent.

5. Voor elke meerkeuzevraag (waarbij hokjes moeten worden aangekruist) doet een fout antwoord je evenveel punten verliezen als je met een correct antwoord zou winnen. Als je de vraag niet beantwoordt dan win je noch verlies je punten. Behaal je op die manier een negatieve score voor een volledige vraag, dan wordt de score teruggezet naar nul. Om op een meerkeuzevraag te antwoorden, kruis je het hokje aan (☒) dat met jouw antwoord overeenkomt. Om een antwoord te annuleren, maak je het hokje volledig zwart (■).

6. Je mag **op geen enkel moment met iemand communiceren**, behalve met de toezichthouders of organisatoren. Vragen over de inhoud van de proef mogen enkel aan de organisatoren worden gesteld. Voor alle andere vragen (bvb voor kladpapier) kan je terecht bij de toezichthouders.

7. Je mag in principe je **plaats niet verlaten** tijdens de proef. Moet je dringend naar het toilet, meldt dit dan aan een toezichthouder. Hij of zij kan dit toelaten of weigeren, afhankelijk van de mogelijkheid om je te laten vergezellen door een van de toezichters.

Overzicht pseudo-code

Gegevens worden opgeslagen in variabelen. Je kan de waarde van een variabele veranderen met $\leftarrow$. In een variabele kunnen we gehele getallen, reële getallen of arrays opslaan (zie verder), en ook booleaanse (logische) waarden : waar/juist (**`true`**) of onwaar/fout (**`false`**). Op variabelen kan je wiskundige bewerkingen uitvoeren. Naast de klassieke operatoren $+, -, \times$ en $/$, kan je ook $\%$ gebruiken: als a en b gehele getallen zijn, dan zijn a/b en $a\%b$ respectievelijk het quotiënt en de rest van de gehele deling (staartdeling). Bijvoorbeeld, als $a = 14$ en $b = 3$, dan geldt: $a/b = 4$ en $a\%b = 2$. In het volgende stukje code krijgt de variabele $leeftijd$ de waarde 19.

```
geboortejaar ← 1993
leeftijd ← 2012 − geboortejaar
```

Als we een stuk code alleen willen uitvoeren als aan een bepaalde voorwaarde (conditie) is voldaan, gebruiken we de instructie **`if`**. We kunnen eventueel code toevoegen die uitgevoerd wordt in het andere geval, met de instructie **`else`**. Hieronder wordt getest of iemand meerderjarig is. Het resultaat wordt bewaard in de booleaanse (waar/onwaar) variabele $volwassen$.

```
if (leeftijd ≥ 18)
{
  volwassen ← true
}
else
{
  volwassen ← false
}
```

Wanneer we in één variabele tegelijkertijd meerdere waarden willen stoppen, gebruiken we een array. De afzonderlijke elementen van een array worden aangeduid met een index (die we tussen vierkante haakjes schrijven achter de naam van de array). Het eerste element van een array arr heeft index 0 en wordt genoteerd als $arr[0]$. Het volgende element heeft index 1, en het laatste heeft index $N - 1$ als de array N elementen bevat. (N noemen we de *lengte* van de array). Dus als de array arr de drie getallen 5, 9 en 12 bevat (in die volgorde) dan is $arr[0] = 5$, $arr[1] = 9$ en $arr[2] = 12$. De lengte is 3, maar de hoogst mogelijke index is slechts 2.

Voor het herhalen van code, kan je een **`for`**-lus gebruiken. De notatie **`for`** $(i \leftarrow a$ **`to`** b **`step`** $k)$ staat voor een lus die herhaald wordt zolang $i \leq b$, waarbij i begint met de waarde a en telkens verhoogd wordt met k aan het eind van elke stap. Het onderstaande voorbeeld berekent de som van de elementen van de array arr, veronderstellend dat de lengte ervan N is. Nadat het algoritme werd uitgevoerd, zal de som zich in de variabele sum bevinden.

```
sum ← 0
for (i ← 0 to N − 1 step 1)
{
  sum ← sum + arr[i]
}
```

En alternatief voor een herhaling is een **`while`**-lus. Deze herhaalt een blok code zolang er aan een bepaalde voorwaarde is voldaan. In het volgende voorbeeld delen we een positief geheel getal N door 2, daarna door 3, daarna door 4 ... totdat het getal nog maar uit 1 decimaal cijfer bestaat (d.w.z., kleiner wordt dan 10).

```
d ← 2
while (N ≥ 10)
{
  N ← N/d
  d ← d + 1
}
```

Vraag 1 – (5 min – 4 ptn)

Bekijk het volgende programmafragment:

```
if ( (a < b) = (b < c) ) {
   Schrijf (b)
} else if ( (a > c) = (b < c) ) {
   Schrijf (c)
} else {
   Schrijf (a)
}
```

Schrijf voor elk van de onderstaande gevallen op wat er door dit fragment wordt afgedrukt.

(a) Wanneer $a = 10$, $b = 0$ en $c = -10$.

Q1(a) [1 pt] **een getal**

0

(b) Wanneer $a = 2013$, $b = 2000$ en $c = 2012$.

Q1(b) [1 pt] **een getal**

2012

(c) Wanneer $a = 400$, $b = 400$ en $c = 400$.

Q1(c) [1 pt] **een getal**

400

(d) Wanneer a de leeftijd is van John Dundee, b de leeftijd is van Sheila Dundee (de dochter van John) en c de leeftijd is van Priscilla Dundee (de moeder van John).

Q1(d) [1 pt] **De leeftijd van welke persoon?**

De leeftijd van John Dundee

Vraag 2 – Een duidelijk programma... (10 min – 7 ptn)

Bekijk de volgende code, die als input een geheel getal x krijgt dat positief of nul is ($x \geq 0$):

```
output ← −1
while (output=−1)
{
  if (x ≤ 3)
  {
    if (x > 1)
    {
      output ← x % 2
    }
    else
    {
      output ← x
    }
  }
  else
  {
    if (x − 3 ≤ 3)
    {
      if (x % 2 = 1)
      {
        output ← x / 5
      }
      else
      {
        output ← (x % 4) / 2
      }
    }
    else
    {
       x ← x % 7
    }
  }
}
Schrijf(output)
```

Duid aan, voor elk van de volgende stellingen, of ze **juist** of **fout** zijn.

	Juist	Fout	
Q2(a) [1pt]	☐	☒	0 wordt *enkel en alleen* afgedrukt als x gelijk is aan 0, 2 of 4
Q2(b) [1pt]	☒	☐	De waarde 2 wordt nooit afgedrukt
Q2(c) [1pt]	☒	☐	Als $x = 5$, dan wordt de waarde 1 afgedrukt
Q2(d) [1pt]	☐	☒	Er bestaat een waarde voor x die ertoe leidt dat de waarde -1 wordt afgedrukt
Q2(e) [1pt]	☐	☒	Als $x = 6$, dan wordt de waarde 0 afgedrukt
Q2(f) [1pt]	☐	☒	Als de waarde van x even is, dan wordt de waarde 0 afgedrukt
Q2(g) [1pt]	☒	☐	1 wordt afgedrukt als $x\%7$ gelijk is aan 1, 3, 5 of 6

Vraag 3 – Ik wordt zeeziek van die lussen (15 min – 12 ptn)

Bekijk het volgende programmafragment:

```
for (i ← 1 to 100 step 1)
{
  for (j ← 1 to 100 step 1)
  {
    for (k ← 1 to 100 step 1)
    {
      if (i < j and j < k and i + j + k = 100)
      {
        Schrijf(i, j, k)
      }
    }
  }
}
```

Dit fragment drukt alle drietallen af waarvan de som 100 is (met enkel strikt positieve gehele getallen groter dan 0).

Het eerste drietal dat door dit fragment wordt afgedrukt, is '`1 2 97`'.

Wat is het derde drietal dat wordt afgedrukt?

Q3(a) [1 pt] **drie gehele getallen**

1 4 95

Wat is het voorlaatste drietal dat wordt afgedrukt?

Q3(b) [2 ptn] **drie gehele getallen**

31 34 35

Herschrijf nu het programma op zo'n manier dat de controles '$i < j$' en '$j < k$' worden vermeden, maar dat nog steeds dezelfde drietallen worden afgedrukt, in dezelfde volgorde.

Vul de ontbrekende delen aan in het volgende fragment. **[1 pt per antwoord]**

```
for (i ← [...] to 100 step 1)     // (c)
{
  for (j ← [...] to 100 step 1) // (d)
   {
     for (k ← [...] to 100 step 1) // (e)
      {
        if (i + j + k = 100)
         {
           Schrijf(i, j, k)
         }
      }
   }
}
```

Q3(c) **een uitdrukking**

1

Q3(d) **een uitdrukking**

$i + 1$

Q3(e) **een uitdrukking**

$j + 1$

Herschrijf tenslotte het programma zodanig dat een **`if`**-instructie niet langer nodig is en dat er bovendien slechts twee genestelde lussen zijn in plaats van drie. Dezelfde drietallen moeten nog steeds worden afgedrukt, in dezelfde volgorde.

Vul de ontbrekende delen aan in het volgende fragment.

```
for (i ← [...] to [...] step 1) // (f)
{
  for (j ← [...] to [...] step 1) // (g)
  {
    Schrijf(i, j, [...])            // (h)
  }
}
```

Q3(f) [2 ptn] **twee uitdrukkingen**

$1, 32$

Q3(g) [3 ptn] **twee uitdrukkingen**

$i + 1, (99 - i)/2$

Q3(h) [1 pt] **één uitdrukking**

$100 - i - j$

Vraag 4 – Een veerpont met valkuilen (15 min – 9 ptn)

Vier lotgenoten (een weerwolf, een mummie, een vampier en het monster van Frankenstein) willen een rivier oversteken, maar ze beschikken slechts over één bootje om van oever A naar oever B te varen. Helaas moeten ze ook rekening houden met de volgende twee beperkingen:

1. het bootje heeft maar twee plaatsen, en
2. niet alle medereizigers komen goed met elkaar overeen. De mummie kan goed overweg met de weerwolf, maar verdraagt de vampier niet, enz... We willen dus vermijden dat bepaalde personages *alleen* komen te staan met een ander die ze niet kunnen uitstaan (de aanwezigheid van een derde reiziger is genoeg om te zorgen dat ze elkaar niet in de haren vliegen). De mogelijke koppels staan samengevat in de tabel hieronder, waarbij **W** de weerwolf is, **M** is de mummie, **V** is de vampier en **F** is het monster van Frankenstein; een ☺ wil zeggen dat we ze gerust samen kunnen achterlaten, en ↯ betekent dat we ze beter niet samen alleen laten:

	M	**V**	**F**
W	☺	☺	↯
M		↯	☺
V			☺

Bijvoorbeeld, als iedereen zich op oever A bevindt, kunnen we **W** en **M** samen doen inschepen om naar oever B te gaan. Daarna kan **W** alleen terugkomen naar oever A, maar hij kan **F** niet laten opstappen om met hem terug naar oever B te gaan: ze zouden zich dan samen alleen op de boot bevinden, wat verboden is...

Hoeveel trajecten heb je nodig (hoeveel keer moet de boot de oversteek maken), minimaal, om iedereen van de ene oever naar de andere te brengen?

Q4(a) [1 pt] **een getal**

5

In het probleem dat we hierboven hebben beschreven, proberen we van een *beginsituatie*, waarbij alle personages (en de boot) zich op oever A bevinden, naar een *eindsituatie* te gaan, waar ze allemaal (en de boot) op oever B zijn. Om van de ene naar de andere situatie te gaan passeren we duidelijk enkele *tussensituaties*, waarbij sommige groepsleden zich op de ene oever bevinden en anderen op de andere, en de boot aangemeerd op een van beide oevers.

Deze tussensituaties moeten de tabel hierboven respecteren. De trajecten die je van één situatie naar een andere brengen moeten ook deze regels respecteren. Wanneer een traject of een situatie de beperkingen uit de tabel respecteert, noemen we dat traject of die situatie *aanvaardbaar*. Het is duidelijk dat een mogelijke oplossing voor het probleem niet door situaties of trajecten mag passeren die niet aanvaardbaar zijn.

Om de aanvaardbare tussensituaties en de mogelijke trajecten daartussen visueel voor te stellen, gaan we de volgende notatie gebruiken. De afkortingen **W**, **M**, **V** en **F** werden eerder al uitgelegd, en de boot stellen we voor met ⚓.

Een aanvaardbare tussensituatie is dan bijvoorbeeld **W M / V F ⚓**, wat wil zeggen dat **W** en **M** op oever A staan, terwijl **V** en **F** zich samen met de boot op oever B bevinden.

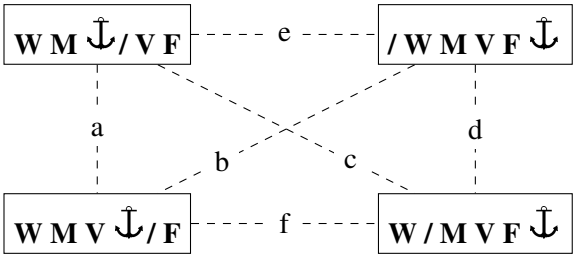
Andere aanvaardbare situaties zijn bijvoorbeeld **W M F ⚓/ V** en **M / W V F ⚓**.

De mogelijke bewegingen van de boot stellen we voor door een lijn te trekken tussen twee situaties als het mogelijk is om van de ene naar de andere situatie te gaan (en vice-versa) d.m.v. één enkel *aanvaardbaar* traject met de boot.

We trekken bijvoorbeeld de volgende lijn: **W M / V F ⚓** — **W M F ⚓/ V** omdat vanaf situatie **W M / V F ⚓**, **V** de boot kan gebruiken om van oever B naar oever A te gaan, wat de volgende situatie oplevert: **W M F ⚓/ V**. Deze twee situaties zijn aanvaardbaar, net zoals het traject ertussen.

We trekken daarentegen geen lijn tussen **W M / V F ⚓** en **M / W V F ⚓** omdat we niet van de ene naar de andere situatie kunnen gaan door middel van één enkel traject met de boot.

In de grafische voorstelling hieronder hebben we 4 aanvaardbare situaties gezet. In stippellijn staan alle mogelijke links tussen 2 situaties. (genaamd a, b,. . . , f). Slechts enkele van deze links zijn correct. Dewelke?

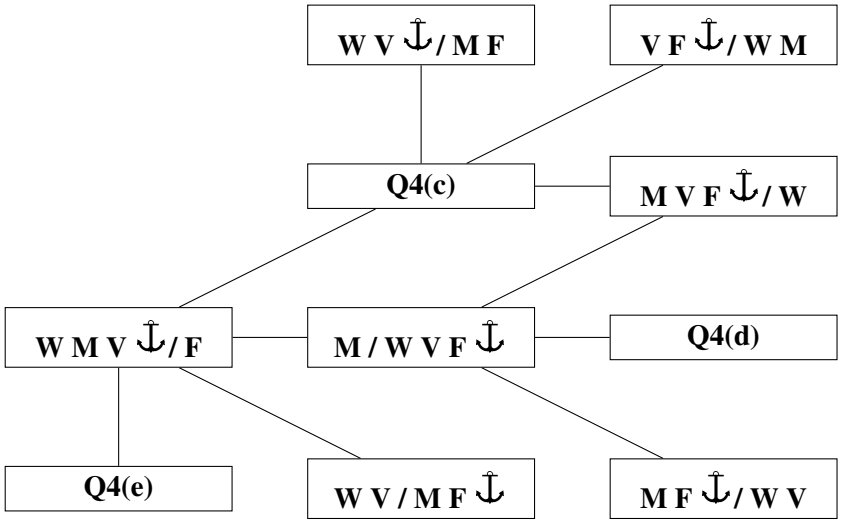


Q4(b) [2 ptn] **Nul, één of meerdere letters**

e,c

(Wordt vervolgd op de volgende pagina...)

Hieronder staat een deel van de volledige grafische voorstelling van het probleem, waarbij we enkele vakjes hebben opengelaten (**Q4(c)**, **Q4(d)**, **Q4(e)**). Wat moeten we in deze vakjes zetten, wetende dat we enkel aanvaardbare situaties en trajecten weergeven, en dat elke aanvaardbare situatie slechts éénmaal mag voorkomen ?



Q4(c) [2 ptn] **een aanvaardbare situatie**

V / L M F ⚓

Q4(d) [2ptn] **een aanvaardbare situatie**

M L ⚓ / V F

Q4(e) [2ptn] **een aanvaardbare situatie**

L M / V F ⚓

Vraag 5 – Mutuele exclusie (10 min – 11 ptn)

Op moderne computers is het mogelijk om *meerdere programma's tegelijk uit te voeren*. In de praktijk kan de computer echter maar één enkele lijn van één programma tegelijkertijd uitvoeren. Het besturingssysteem heeft de taak om de uitvoering zo te organiseren dat er eerst enkele lijnen van het ene programma worden uitgevoerd, dan enkele lijnen van een ander, enzovoort... Het wisselen tussen programma's gebeurt zo snel dat de gebruiker er niets van merkt, en zo lijkt het alsof de programma's *tegelijkertijd* functioneren.

Deze techniek verhoogt dan wel het gemak voor de gebruiker, maar ze veroorzaakt grote problemen wanneer er *globale variabelen* worden gebruikt. Dit zijn variabelen die door meerdere verschillende programma's gelezen en geschreven kunnen worden (ze worden dan ook vaak gebruikt als manier van communicatie tussen deze programma's).

Ter illustratie: beschouw de twee programma's P_1 en P_2 hieronder. Ze worden *tegelijkertijd* uitgevoerd, waarbij `i` een globale variabele is (zowel P_1 als P_2 kunnen ze lezen en schrijven), terwijl `j` en `k` enkel toegankelijk zijn voor respectievelijk P_1 en P_2. (We hebben elke lijn code hier genummerd, voor de duidelijkheid.) Elk van de programma's heeft duidelijk als doel om 1 op te tellen bij de globale variabele `i`. Veronderstel dat de waarde van `i` in het begin reeds 0 is, dan willen we dat `i` gelijk is aan `2` na de uitvoering van beide programma's.

Initialisatie: $\texttt{i} = 0$.

P_1:

```
1  j ← i
2  j ← j+1
3  i ← j
```

P_2:

```
4  k ← i
5  k ← k+1
6  i ← k
```

Bij dit voorbeeld is het probleem dat we niet weten *in welke volgorde de instructies van beide programma's uitgevoerd zullen worden*. Natuurlijk wordt instructie 1 van programma P_1 uitgevoerd voor de daaropvolgende instructie 2. Maar we kunnen echter niet voorspellen of instructie 1 van P_1 uitgevoerd zal worden *voor* of *na* instructie 4 van P_2, bijvoorbeeld.

Helaas kan deze volgorde een invloed hebben op het eindresultaat van de uitvoering van de programma's. Als de computer de instructies uitvoert in de volgorde 1, 2, 3, 4, 5, 6, dan zal de waarde van `i` gelijk zijn aan 2. Als de volgorde van uitvoering echter 1, 4, 2, 3, 5, 6 is, dan zal de uiteindelijke waarde van `i` gelijk zijn aan 1!

Als we nog steeds aannemen dat de beginwaarde van `i` gelijk is aan 0, wat zal dan de waarde van `i` zijn na het uitvoeren van de lijnen code in de volgorde: 4, 5, 6, 1, 2, 3 ?

Q5(a) [1 pt] **een getal**

2

Dezelfde vraag voor de volgorde: 4, 5, 1, 6, 2, 3 ?

Q5(b) [1 pt] **een getal**

1

Om deze problemen te vermijden, krijgen we de suggestie om globale variabelen `b1`, `b2` en `beurt` toe te voegen, die de programma's toelaten om zichzelf onderling te coördineren, om te vermijden dat instructies van beide programma's door elkaar worden gemengd. De variabelen `b1` en `b2` hebben in het begin de waarde 0, en de variabele `beurt` heeft in het begin de waarde 1. De instructie `doe_niets` is een instructie die geen effect heeft:

Initialisatie: `beurt` $= 1$, `b1` $=$ `b2` $= 0$, `i` $= 1$.

P_1:

```
1  b1 ← 1
2  while ([...]) // (w)
3  {
4    b1 ← 0
5    while ([...]) // (x)
6    {
7      doe_niets
8    }
9    b1 ← 1
10 }
11 j ← i
12 j ← j+1
13 i ← j
14 beurt ← 2
15 b1 ← 0
```

P_2:

```
16 b2 ← 1
17 while ([...]) // (y)
18 {
19   b2 ← 0
20   while ([...]) // (z)
21   {
22     doe_niets
23   }
24   b2 ← 1
25 }
26 k ← i
27 k ← k+1
28 i ← k
29 beurt ← 1
30 b2 ← 0
```

De voorwaarden in de **`while`** lussen (aangeduid met w, x, y et z) moeten ons toelaten om te verhinderen dat instructies worden uitgevoerd in een problematische volgorde. Stel dat deze keer, `i` in het begin de waarde 1 krijgt. Voor elk van de volgende voorstellen, vragen we je om aan te duiden of de gegeven voorwaarden ons kunnen garanderen dat **in alle gevallen, de variabele `i` na de volledige uitvoering van beide programma's de waarde 3 zal hebben**.

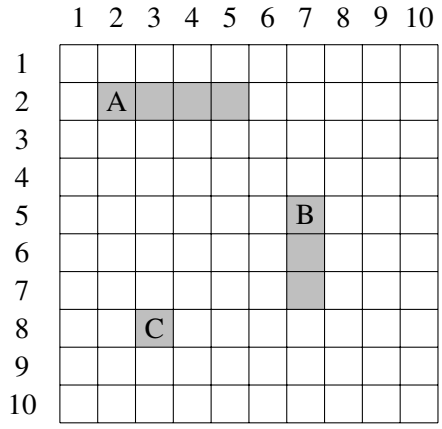
	oui	non	`(w)`	`(x)`	`(y)`	`(z)`
Q5(c) [3 ptn]	☐	☒	`b1`$= 1$	`beurt`$\neq 1$	`b2`$= 1$	`beurt`$\neq 2$.
Q5(d) [3 ptn]	☒	☐	`beurt`$\neq 1$	`b2`$= 1$	`beurt`$\neq 2$	`b1`$= 1$
Q5(e) [3 ptn]	☒	☐	`b2`$= 1$	`beurt`$\neq 1$	`b1`$= 1$	`beurt`$\neq 2$

Vraag 6 – Zeeslag (20 min – 7 + 2 ptn)

Het spelletje *Zeeslag* wordt gespeeld op een rooster van 10 bij 10 vierkante vakjes. Op dit rooster plaatst men *boten*. Een *horizontale* boot wordt voorgesteld als een aantal opeenvolgende vakjes in dezelfde rij. Een *verticale* boot bestaat uit een aantal opeenvolgende vakjes in dezelfde kolom. Een horizontale boot is dus slechts één rij hoog, en een verticale boot is slechts één kolom breed.

In de afbeelding hieronder is A een horizontale boot, is B een verticale boot en kan C zowel een horizontale als een verticale boot voorstellen. Om een boot in het rooster aan te duiden, gebruiken we vier attributen:

- Het rijnummer r van het bovenste en meest linkse vakje van de boot.
- Het kolomnummer c van het bovenste en meest linkse vakje van de boot.
- De 'lengte' l van de boot, d.i. het aantal vakjes waaruit de boot bestaat.
- De richting hor van de boot: **`true`** als de boot horizontaal ligt, **`false`** als hij verticaal ligt.



(Rij- en kolomnummers beginnen vanaf 1.)
De boten in de afbeelding hiernaast hebben dus de volgende attributen:

boot	r	c	l	hor
A	2	2	4	**`true`**
B	5	7	3	**`false`**
C	8	3	1	**`true`**†

† **`false`** is hier ook mogelijk

Help ons om twee functies te schrijven die we nodig hebben om het spelletje *Zeeslag* te programmeren.

De eerste functie '`isGeraakt`' willen we gebruiken om na te gaan of een bepaald vakje (met rijnummer rr en kolomnummer cc) deel uitmaakt van een boot met attributen r, c, l, hor. (In spelterminologie: of je een boot zou 'raken' wanneer je 'schiet' op coördinaten rr, cc.)

Het is niet zo moeilijk om dit te programmeren door alle vakjes van de boot één voor één te overlopen, maar we willen dat je een efficiëntere versie maakt, *zonder* lussen.

Vul de ontbrekende delen aan in het volgende fragment.

```
function isGeraakt (rr, cc, r, c, l, hor)
{
    if (not hor)
    {
        return [...]        // (a)
    }
    else
    {
        return [...]        // (b)
    }
}
```

Q6(a) [1pt] **een logische uitdrukking**

$cc = c$ **and** $rr \geq r$ **and** $rr < r + l$

Q6(b) [1pt] **een logische uitdrukking**

$rr = r$ **and** $cc \geq c$ **and** $cc < c + l$

Opmerking. Een *logische* uitdrukking heeft als waarde steeds enkel **`true`** of **`false`**. Logische uitdrukkingen zijn vergelijkingen (met $<$, $>$, $\leq$, $\geq$, $=$ of $\neq$) die je eventueel kan combineren met **`and`** (en), **`or`** (of) en **`not`** (niet). Gebruik (ronde) haakjes voor de duidelijkheid. Enkele voorbeelden: "$c \leq r$ **`and`** $cc \geq rr - 5$" of "$(c \neq 10$ **`and`** $c > 5)$ **`or`** $r = rr/2$".

Belangrijk! We keuren enkel uitdrukkingen goed met **minder dan 6** vergelijkingen.

De tweede functie '`overlappen`' moet dienen om te kijken of twee boten al dan niet overlappen op het bord. De eerste boot heeft attributen $r1$, $c1$, $l1$ en $hor1$, de tweede heeft attributen $r2$, $c2$, $l2$ en $hor2$. Opnieuw is het niet zo moeilijk om dit met een lus op te lossen maar we willen ook deze keer dat je dit doet zonder lus en zonder de functie `isGeraakt` op te roepen!

Vul de twee aangeduide ontbrekende delen aan in het volgende fragment. (De tweede andere ontbrekende delen hoef je niet in te vullen.)

```
function overlappen (r1, c1, l1, hor1, r2, c2, l2, hor2)
{
   if (not hor1)
   {
        if (hor2)
        {
            return [...]       // (c)
        }
        else
        {
            return [...]       // niet invullen!
        }
   }
   else
   {
        if (hor2)
        {
            return [...]       // (d)
        }
        else
        {
            return [...]       // niet invullen!
        }
   }
}
```

Q6(c) [3+1 ptn] **een logische uitdrukking**

$c1 \geq c2$ **and** $c1 < c2 + l2$ **and** $r2 \geq r1$ **and** $r2 < r1 + l1$

Q6(d) [2+1 ptn] **een logische uitdrukking**

$r1 = r2$ **and** $c2 < c1 + l1$ **and** $c1 < c2 + l2$

We keuren opnieuw enkel uitdrukkingen goed met minder dan 6 vergelijkingen. Je krijgt telkens een **bonus** wanneer je het kortste antwoord vindt (d.w.z. met zo weinig mogelijk vergelijkingen).