be-OI 2014 **Finale** zaterdag 8 februari 2014	**Invullen in HOOFDLETTERS en LEESBAAR aub** VOORNAAM : ..... NAAM : ..... SCHOOL :	O **Gereserveerd**

Belgische Informatica-olympiade (duur : 1u15 maximum)

Dit is de vragenlijst van de finale van de Belgische Informatica-olympiade 2014. Ze bevat vier 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**. Jouw antwoorden moet je invullen op de daarop voorziene antwoordbladen, die zich achteraan in deze bundel papier bevinden. 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. Wanneer je gedaan hebt, geef je aan de toezichthoud(st)er(s) deze eerste bladzijde terug (met jouw naam erop), en de pagina's met jouw antwoorden. De andere pagina's mag je bijhouden.
3. Je mag alleen schrijfgerief bij je hebben. Rekentoestel, GSM, … zijn **verboden**.
4. 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.
5. Voor alle stukken 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 mag je zelf 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.
6. 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 het juiste antwoord overeenkomt. Om een antwoord te annuleren, maak je het hokje volledig zwart (■).
7. Je mag **op geen enkel moment met iemand communiceren**, behalve met de toezichthouders. Je mag hen bijvoorbeeld wel om kladpapier vragen, maar je mag geen vragen stellen over de inhoud van de proef. Elke fout in de opgave moet je beschouwen als onderdeel van de proef.
8. 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. Afhankelijk van de locatie kan het ook verboden zijn om te eten of te drinken in het lokaal.

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 allebei 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 20.

```
geboortejaar ← 1993
leeftijd ← 2013 − 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`**. Het voorbeeld hieronder test of iemand meerderjarig is, en bewaart de prijs van zijn/haar cinematicket in een variabele *prijs*. De code is bovendien voorzien van commentaar.

```
if (leeftijd ≥ 18)
{
  prijs ← 8    // Dit is een stukje commentaar
}
else
{
  prijs ← 6    // Verlaagde prijs
}
```

Als we een reeks waarden in één variabele 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. 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 van *arr* is 3, maar de hoogst mogelijke index is slechts 2.

Voor het herhalen van code, bijvoorbeeld om de elementen van een array af te lopen, 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]
}
```

Een 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 – Opwarming (15 min – 16 ptn)

We vragen je om het effect te voorspellen van verschillende stukjes code, die allemaal afhangen van de waarde van een variabele N.

Beschouw het volgende programma:

```
Input : een strikt positief geheel getal N

V ← 1

for (i ← 1 to N step 1)
{
   for (j ← 1 to N step 1)
   {
      Schrijf(V)
      V ← V + 1
   }
}
```

Q1(a) [1 pt]	**Als $N = 3$, wat is dan de reeks getallen die wordt geschreven door dit programma?**
Oplossing: 1, 2, 3, 4, 5, 6, 7, 8, 9	

Q1(b) [1 pt]	**Als $N = 16$, wat is dan het grootste getal dat wordt geschreven door dit programma?**
Oplossing: 256	

Q1(c) [2 ptn]	**Geef, in functie van N, de grootste waarde die door het programma wordt geschreven.**
Oplossing: N^2	

Beschouw nu dit nieuwe programma. Het verschil met het vorige programma is omkaderd:

```
Input : een strikt positief geheel getal N

V ← 1

for (i ← 1 to N step 1)
{
   for (j ← 1 to [i] step 1)
   {
      Schrijf(V)
      V ← V + 1
   }
}
```

Q1(d) [2 ptn]	**Als $N = 3$, wat is dan de reeks getallen die wordt geschreven door dit programma?**
Oplossing: 1, 2, 3, 4, 5, 6	

Q1(e) [3 ptn]	**Als $N = 16$, wat is dan het grootste getal dat wordt geschreven door dit programma?**
Oplossing: 136	

Bekijk tot slot dit laatste programma, met drie in elkaar geneste lussen:

```
Input : een strikt positief geheel getal N

V ← 1

for (i ← 1 to N step 1)
{
  for (j ← 1 to i step 1)
  {
    for (k ← 1 to j step 1)
    {
      Schrijf(V)
      V ← V + 1
    }
  }
}
```

Q1(f) [3 ptn]	**Als $N = 3$, wat is dan de reeks getallen die wordt geschreven door dit programma?**
Oplossing: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10	

Q1(g) [4 ptn]	**Als $N = 5$, wat is dan het grootste getal dat wordt geschreven door dit programma?**
Oplossing: $1 + (1 + 2) + (1 + 2 + 3) + (1 + 2 + 3 + 4) + (1 + 2 + 3 + 4 + 5) = 35$	

Vraag 2 – Sudoku (15 min – 16 ptn)

Een sudoku bestaat uit een vierkant van 9 rijen en 9 kolommen, en 9 vierkanten van 3 bij 3, omrand door zwarte lijnen. Op elke rij, op elke kolom en in elk vierkant moeten alle getallen van 1 tot en met 9 exact één keer voorkomen, niet meer of minder.

De linker afbeelding hieronder is een voorbeeld van een goed ingevulde sudoku. De rechter sudoku is niet goed ingevuld. Hier komt bijvoorbeeld het getal 6 twee keer voor in het vierkant linksboven.

2	9	5	7	4	3	8	6	1
4	3	1	8	6	5	9	2	7
8	7	6	1	9	2	5	4	3
3	8	7	4	5	9	2	1	6
6	1	2	3	8	7	4	9	5
5	4	9	2	1	6	7	3	8
7	6	3	5	2	4	1	8	9
9	2	8	6	7	1	3	5	4
1	5	4	9	3	8	6	7	2

2	9	5	7	4	3	8	6	1
4	**6**	1	8	6	5	9	2	7
8	7	**6**	1	9	2	5	4	3
3	8	7	4	5	9	2	1	6
6	1	2	3	8	7	4	9	5
5	4	9	2	1	6	7	3	8
7	6	3	5	2	4	1	8	9
9	2	8	6	7	1	3	5	4
1	5	4	9	3	8	6	7	2

Het volgende programma controleert of een sudoku goed is ingevuld. Het enige argument van de functie `sudoku` is een vierkant van 9 op 9 dat volledig is ingevuld. Alle getallen in die sudoku liggen tussen 1 en 9, inclusief. Er zijn enkele lijnen code leeg gelaten die nog moeten ingevuld worden.

Q2(a) [4 ptn]	**Geef uitdrukking** (a)
Oplossing: $b[s[j][i]-1]+1$	

Q2(b) [3 ptn]	**Geef voorwaarde** (b)
Oplossing: 1	

Q2(c) [5 ptn]	**Geef uitdrukking** (c)
Oplossing: $3 \times m + i$	

Q2(d) [4 ptn]	**Geef uitdrukking** (d)
Oplossing: $a[s[x][y]-1]+1$	

Input : s, een array van 9 op 9, met daarin gehele getallen van 1 tot en met 9.
Output : OK bevat **true** als en slechts als s een goed ingevulde sudoku bevat.

In de code gebruiken we twee extra arrays a en b (arrays van 9 gehele getallen).

```
OK ← true

for(i ← 0 to 8 step 1)
{
  for(j ← 0 to 8 step 1)
  {
    a[j] ← 0
    b[j] ← 0
  }

  for(j ← 0 to 8 step 1)
  {
    a[s[i][j] − 1] ← [ ... ]              // niet gevraagd
    b[s[j][i] − 1] ← [ ... ]              // (a)
  }

  for (j ← 0 to 8 step 1)
  {
    if(a[j] > [...] or b[j] > [...])      // (b) (je hoeft enkel de eerste voorwaarde te geven)
    {
      OK ← false
    }
  }
}

for(m ← 0 to 2 step 1)
{
  for(n ← 0 to 2 step 1)
  {
    for (i ← 0 to 8 step 1)
    {
      a[i] ← 0
    }

    for(i ← 0 to 2 step 1)
    {
      for(j ← 0 to 2 step 1)
      {
        x ← [...]                         // (c)
        y ← [...]                         // niet gevraagd

        a[s[x][y] − 1] ← [...]            // (d)
      }
    }

    for (j ← 0 to 8 step 1)
    {
      if(a[j] > [...])                    // niet gevraagd
      {
        OK ← false
      }
    }
  }
}
```

Vraag 3 – Zoeken in een geordende tabel (25 min – 24 ptn)

We willen een element t opzoeken in een tweedimensionale array (een tabel) tab. Het element op de i^{e} rij en j^{e} kolom noemen we $tab[i][j]$. Zoals gewoonlijk zijn de rijen en kolommen genummerd vanaf 0. Onze tabel heeft N rijen en M kolommen en is geordend: dat wil zeggen dat de waarden op elke rij en op elke kolom in stijgende lijn gaan. Hier is een voorbeeld van een geordende tabel met $N = 2$ en $M = 3$:

1	2	2
1	3	7

Stel dat $N = M = 5$. We vragen je om voor elk van de volgende vier voorwaarden de vakjes van de geordende tabel **aan te kruisen** waar t zich **zeker niet** kan bevinden.

Q3(a) [1 pt] **In welke vakjes kan t zich niet bevinden als $t < tab[2][2]$?**

Oplossing:

	0	1	2	3	4
0					
1					
2			×	×	×
3			×	×	×
4			×	×	×

Q3(b) [1 pt] **In welke vakjes kan t zich niet bevinden als $t > tab[2][4]$?**

Oplossing:

	0	1	2	3	4
0	×	×	×	×	×
1	×	×	×	×	×
2	×	×	×	×	×
3					
4					

Q3(c) [2 ptn] **In welke vakjes kan t zich niet bevinden als $tab[0][2] < t < tab[0][3]$ *en* $tab[2][0] < t < tab[3][0]$?**

Oplossing:

	0	1	2	3	4
0	×	×	×	×	×
1	×			×	×
2	×			×	×
3	×	×	×	×	×
4	×	×	×	×	×

Q3(d) [3 ptn]	**In welke vakjes kan t zich niet bevinden als $tab[0][2] < t < tab[0][4]$ *en* $tab[1][0] < t < tab[3][0]$?**

Oplossing:

	0	1	2	3	4
0	×	×	×		×
1	×				×
2					×
3	×	×	×	×	×
4	×	×	×	×	×

Als we een element t willen opzoeken in een geordende tabel, kunnen we dat simpelweg doen door de hele tabel te doorlopen en voor elk element te verifiëren of $tab[i][j] = t$. We willen echter een efficiënter algoritme vinden, gebruik makend van het feit dat de tabel al min of meer 'gesorteerd' is, zodat we ze niet meer helemaal moeten doorlopen. Professor Yunoac komt met het volgende voorstel op de proppen:

We beginnen door de middelste rij te definiëren als rij nummer $m = \frac{N-1}{2}$. We vergelijken t met het laatste element van de middelste rij, dat is $tab[m][M-1]$. Als t groter of gelijk is aan dat element, herhalen we dit algoritme op de tabel die we bekomen als we rijen 0 tot en met m verwijderen. In het andere geval, herhalen we dit algoritme op de tabel die we bekomen als we rijen $m+1$ tot en met $N-1$ verwijderen. Wanneer de tabel nog slechts uit 1 rij bestaat zoeken we waar t zich bevindt door het te vergelijken met alle elementen in die rij.

Professor Yunoac was duidelijk niet uitgeslapen op het moment dat hij dit voorstelde. Een van zijn studentes realiseert zich onmiddellijk dat het fout is! Zij vond een tabel met slechts 3 rijen en 2 kolommen, en een waarde t, waarvoor het algoritme een fout antwoord geeft. Kan jij ook zo een tegenvoorbeeld vinden?

Q3(e) [3 ptn]	**Geef een geordende tabel met $N = 3$ en $M = 2$, en een waarde t, waarvoor het algoritme van professor Yunoac een fout antwoord geeft.**

Oplossing:

	0	1
0	1	3
1	2	4
2	3	5

$t = 2$

De studente van professor Yunoac stelt vervolgens een heel ander algoritme voor:

We beginnen bij het element linksonder in de tabel, dat is op positie $tab[N-1][0]$. Voor elke stap, vergelijken we t met de huidige positie. Als t zich op de huidige positie bevindt, stoppen we (en we printen uit dat t gevonden is). Als t groter is dan de waarde op de huidige positie, gaan we een vakje naar rechts (naar de volgende kolom). In het andere geval gaan we een vakje naar boven (naar de vorige rij). Als we daardoor buiten de tabel terecht zouden komen, stoppen we en printen we uit dat t zich niet in tab bevindt.

Vervolledig het volgende stukje pseudo-code zodat het dit algoritme implementeert. Op het einde van het algoritme moet de variabele $gevonden$ de waarde $true$ (ja) of $false$ (nee) bevatten, al naargelang t wel of niet werd gevonden in de tabel.

```
Input:
- tab: een geordende tabel (tweedimensionale array)
- N: het aantal rijen van de tabel
- M: het aantal kolommen van de tabel

i ← N-1
j ← 0
gevonden ← false

while( !gevonden and i >= 0 and [...])  // (f)
{
  if( [...] )                           // (g)
  {
    gevonden ← true
  }
  else if(t > tab[i][j])
  {
    [...]                               // (h)
  }
  else
  {
    [...]                               // (i)
  }
}
```

Q3(f) [4 ptn]	**Vervolledig de voorwaarde** `(f)`
Oplossing: $j < M$	

Q3(g) [2 ptn]	**Vervolledig de voorwaarde** `(g)`
Oplossing: $t = tab[i][j]$	

Q3(h) [2 ptn]	**Vervolledig de instructie** `(h)`
Oplossing: $j \leftarrow j + 1$	

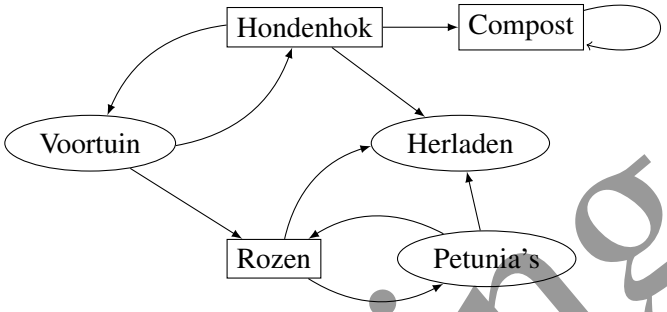
Q3(i) [2 ptn]	**Vervolledig de instructie** `(i)`
Oplossing: $i \leftarrow i - 1$	

Q3(j) [4 ptn]	**Wat is, in functie van N en M, het maximale aantal keer dat de voorwaarde** `(g)` **zal worden getest?**
Oplossing: $N + M - 1$	

Vraag 4 – De tuinrobot (20 min – 19 ptn)

De SupraMow 3000 ® is een nieuwe tuinrobot die voor jou het gazon maait en de dode bladeren opruimt. De bedenkers van de robot doen beroep op jouw kunde om de controle-software te schrijven waarmee de robot ongehinderd door de tuin moet kunnen zoemen.

De SupraMow 3000 ® heeft een kaart van de tuin in zijn geheugen, waarvan je hieronder een voorbeeld ziet. Elke ellips en elke rechthoek stellen een *plaats* voor, die ook een naam heeft. De pijlen geven aan hoe de robot zich kan verplaatsen. In het voorbeeld hieronder begint de robot zijn parcours op de plaats *voortuin* en wil terechtkomen op de plaats *herlaadpunt* om zijn batterijen op te laden:

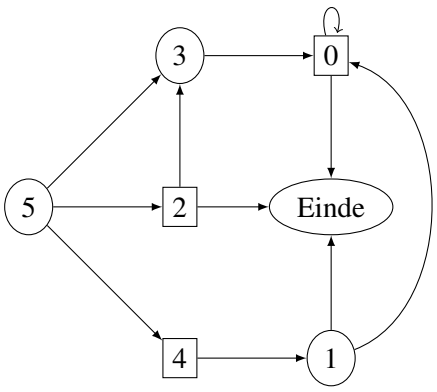


Helaas heeft de SupraMow 3000 ® geen controle over wat er gebeurt op sommige locaties, wat zich in het diagram hierboven vertaalt door de twee verschillende vormen (rechthoek en ellips) voor de locaties. De ellipsvormige locaties worden *controleerbaar* genoemd: de SupraMow 3000 ® kan van daaruit zelf beslissen waarheen hij verder gaat. Bijvoorbeeld, vanuit de voortuin kan de robot zelf beslissen of hij richting hondenhok of richting rozenperk gaat. De plaatsen in rechthoeken zijn *oncontroleerbaar*: de SupraMow 3000 ® heeft geen controle over waar hij van daaruit verder gaat. Het hondenhok bijvoorbeeld: als de hond eruit komt en de robot met zijn poot wegmept, kan de robot terug terechtkomen in de voortuin, maar ook even goed in de composthoop belanden. Bij de rozentuin is het terrein heel heuvelachtig, en kan de robot onverwachts wegrollen naar de petunia's (vanwaar hij wel zelf kan beslissen naar het herlaadpunt te gaan of terug te keren naar de rozen).

We zien dus dat, vertrekkende vanaf de *voortuin*, kiezen om naar het *hondenhok* te gaan geen goede *strategie* is, want de hond kan de robot herhaaldelijk terug naar de *voortuin* meppen. Erger nog, de hond kan de robot in de *composthoop* doen belanden, waar de robot niet uit kan ontsnappen. Daarentegen is het wel een goede keuze om eerst naar de *rozen* te gaan: of hij van daaruit nu bij de *petunia's* terechtkomt of niet, hij zal altijd het *herlaadstation* kunnen bereiken. Bijgevolg zijn sommige plaatsen *zeker*: dat zijn die plaatsen van waaruit de robot een strategie kan vinden die hem toelaat om gegarandeerd het herlaadpunt te bereiken in een eindig aantal stappen. Bijvoorbeeld, de *voortuin* is een *zekere* plaats, maar niet de *composthoop*. Uiteraard is het *herlaadpunt* zelf ook *zeker*.

Q4	**Zijn deze andere plaatsen zeker ?**		
	Zeker	Niet zeker	Plaats
Q4(a) [1 pt]	☒	☐	Rozen
Q4(b) [1 pt]	☒	☐	Petunia's
Q4(c) [1 pt]	☐	☒	Hondenhok

Bekijk de volgende kaart. Wat zijn alle *zekere* plaatsen, van waaruit de robot gegarandeerd het *einde* bereikt?



Q4(d) [4 ptn]	**Geef de lijst van de zekere plaatsen.**
Oplossing: Einde, 1, 4, 5	

Met het algoritme verderop willen we bepalen, gegeven zo'n kaart, of een plaats *zeker* is. Opnieuw moet je het vervolledigen. Het algoritme veronderstelt dat de verschillende plaatsen genummerd zijn van 0 tot $N-1$, en dat de plaats die we willen bereiken plaats $N-1$ is. Een plaats i moet dus *zeker* verklaard worden als de robot met zekerheid $N-1$ kan bereiken vanuit i. Een array `toegang` van grootte $N \times N$ bevat Booleaanse waarden om aan te duiden welke verplaatsingen de robot kan maken: elk element `toegang[`i`][`j`]` bevat **`true`** als en slechts als de robot zich kan verplaatsen van plaats i naar plaats j (er gaat een pijl van i naar j in het diagram). Tot slot is er een array `controle` van grootte N die ook Booleaanse waarden bevat, om aan te geven welke plaatsen controleerbaar zijn: plaats i is controleerbaar als en slechts als `controle[`i`]` de waarde **`true`** bevat. Hieronder staan bijvoorbeeld de arrays `toegang` en `controle` voor de vorige kaart (waarbij het einde nummer 6 heeft gekregen, en **`true`** wordt voorgesteld door **T** en **`false`** door F).

`toegang`:

	0	1	2	3	4	5	6
0	**T**	F	F	F	F	F	**T**
1	**T**	F	F	F	F	F	**T**
2	F	F	F	**T**	F	F	**T**
3	**T**	F	F	F	F	F	F
4	F	**T**	F	F	F	F	F
5	F	F	**T**	**T**	**T**	F	F
6	F	F	F	F	F	F	F

`controle`:

0	1	2	3	4	5	6
F	**T**	F	**T**	F	**T**	**T**

Q4(e) [2 ptn]	**Vervolledig uitdrukking** `(e)`
Oplossing: $N-1$	

Q4(f) [4 ptn]	**Vervolledig uitdrukking** `(f)`
Oplossing: `zeker[`j`]=`**`false and`** `toegang[`i`][`j`]=`**`true`** of **`not`** `zeker[`j`]` **`and`** `toegang[`i`][`j`]	

Q4(g) [3 ptn]	**Vervolledig instructie** `(g)`
Oplossing: `flag`←**`false`**	

Q4(h) [3 ptn]	**Vervolledig uitdrukking** `(h)`
Oplossing: `zeker[`i`]` of `zeker[`i`]` = **`true`**	

Input : de arrays *toegang* en *controle* zoals hierboven beschreven
Output : een array *zeker* van lengte N zodat zeker[i] de waarde **true** bevat als en slechts als plaats i zeker is, en anders **false** bevat.

Alle elementen van de array *zeker* zijn in het begin geïnitialiseerd op **false**.

```
zeker[...] ← true                    // (e)
gedaan ← false

while (not gedaan)                   // gedaan is gelijk aan false
{
  gedaan ← true
  for (i ← 0 to N − 1 step 1)
  {
    if (not zeker[i])
    {
      if (controle[i])               // i is een controleerbare plaats.
      {
        for (j ← 0 to N − 1 step 1)
        {
          if (zeker[j] and toegang[i][j])
          {
            zeker[i] ← true
          }
        }
      }
      else                           // i is geen controleerbare plaats.
      {
        flag ← true

        for (j ← 0 to N − 1 step 1)
        {
          if ([...])                 // (f)
          {
            [...]                    // (g)
          }
        }
        zeker[i] ← flag
      }

      if ([...])                     // (h)
      {
        gedaan ← false
      }

    }
  }
}
```

Vul hier uw antwoorden in !

Q1(a)	**Een reeks getallen**	/1
Q1(b)	**Een getal**	/1
Q1(c)	**Een uitdrukking in functie van N**	/2
Q1(d)	**Een reeks getallen**	/2
Q1(e)	**Een getal**	/3
Q1(f)	**Een reeks getallen**	/3
Q1(g)	**Een getal**	/4
Q2(a)	**Een uitdrukking**	/4
Q2(b)	**Een voorwaarde**	/3
Q2(c)	**Een uitdrukking**	/5
Q2(d)	**Een uitdrukking**	/4
Q3(a)	**Kruis de vakjes aan waar $t < tab[2][2]$ zich niet kan bevinden.**	/ 1

	0	1	2	3	4
0					
1					
2					
3					
4					

Q3(b) **Kruis de vakjes aan waar $t > tab[2][4]$ zich niet kan bevinden.** / 1

	0	1	2	3	4
0					
1					
2					
3					
4					

Q3(c) **Kruis de vakjes aan waar t zich niet kan bevinden als** $tab[0][2] < t < tab[0][3]$ ***en*** $tab[2][0] < t < tab[3][0]$**.** / 2

	0	1	2	3	4
0					
1					
2					
3					
4					

Q3(d) **Kruis de vakjes aan waar t zich niet kan bevinden als** $tab[0][2] < t < tab[0][4]$ ***en*** $tab[1][0] < t < tab[3][0]$**.** / 3

	0	1	2	3	4
0					
1					
2					
3					
4					

Q3(e) **Een tabel (3 rijen en 2 kolommen) en een waarde t.** / 3

	0	1
0		
1		
2		

$t =$ .....

Q3(f) **Een uitdrukking** /4

.....

Q3(g) **Een uitdrukking**	/2
Q3(h) **Een instructie**	/2
Q3(i) **Een instructie**	/2
Q3(j) **Een uitdrukking**	/4

	Zeker	Niet zeker	Plaats	
Q4(a) /1	☐	☐	Rozen	/3
Q4(b) /1	☐	☐	Petunia's	
Q4(c) /1	☐	☐	Hondenhok	

Q4(d) **Een lijst van plaatsen**	/4
Q4(e) **Een uitdrukking**	/2
Q4(f) **Een uitdrukking**	/4
Q4(g) **Een instructie**	/3
Q4(h) **Een uitdrukking**	/3