

be-OI 2025Finale - JUNIOR
Zaterdag 22 maart
2025

Invullen in HOOFDLETTERS en LEESBAAR aub

VOORNAAM :
NAAM :
SCHOOL :

O

Gereserveerd

Finale van de Belgische Informatica-olympiade (duur : maximum 2u)**Algemene opmerkingen (lees dit aandachtig voor je begint)**

1. Controleer of je de juiste versie van de vragen hebt gekregen (die staat hierboven in de hoofding).
 - De categorie **belofte** is voor leerlingen tot en met het 2e middelbaar,
 - de categorie **junior** is voor het 3e en 4e middelbaar,
 - de categorie **senior** is voor het 5e middelbaar en hoger.
2. Vul duidelijk je voornaam, naam en school in, **alleen op dit blad**.
3. **Jouw antwoorden** moet je invullen op de daar voor voorziene antwoordbladen. Schrijf **duidelijk leesbaar** met blauwe of zwarte **bic of pen**.
4. Gebruik een potlood en een gom wanneer in het klad werkt.
5. Je mag alleen schrijfgerief bij je hebben. Rekentoestel, mobiele telefoons, ... zijn **verboden**.
6. Je mag altijd extra kladpapier vragen aan de toezichthouder of leerkracht.
7. Wanneer je gedaan hebt, geef je deze eerste bladzijde terug (met jouw naam erop) en de pagina's met jouw antwoorden, al de rest mag je bijhouden.
8. 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.
9. Als je moet antwoorden met code, mag dat in **pseudo-code** of in eender welke **courante programmeertaal** (zoals Java, C, C++, Pascal, Python, ...). We trekken geen punten af voor syntaxfouten.

Veel succes!

De Belgische Informatica Olympiade wordt mogelijk gemaakt dankzij de steun van onze leden:

 2025 Belgische Informatica-olympiade (beOI) vzw
Dit werk is vrijgegeven onder de licentie: Creative Commons Naamsvermelding 2.0 Belgi 

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 17.

```
geboortjaar  $\leftarrow$  2008
leeftijd  $\leftarrow$  2025 - geboortjaar
```

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  $\geq$  18)
{
    prijs  $\leftarrow$  8 // Dit is een stukje commentaar
}
else
{
    prijs  $\leftarrow$  6 // Goedkoper!
}
```

Soms, als een voorwaarde onwaar is, willen we er nog een andere controleren. Daarvoor kunnen we **else if** gebruiken, wat neerkomt op het uitvoeren van een andere **if** binnen in de **else** van de eerste **if**. In het volgende voorbeeld zijn er 3 leeftijdscategorieën voor cinematickets.

```
if (leeftijd  $\geq$  18)
{
    prijs  $\leftarrow$  8 // Prijs voor een volwassene.
}
else if (leeftijd  $\geq$  6)
{
    prijs  $\leftarrow$  6 // Prijs voor een kind van 6 of ouder.
}
else
{
    prijs  $\leftarrow$  0 // Gratis voor kinderen jonger dan 6.
}
```

Wanneer we in één variabele tegelijk 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. 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
}
```

We tonen algoritmes vaak in een kader met wat extra uitleg. Na **Input**, definiëren we alle parameters (variabelen) die gegeven zijn bij het begin van het algoritme. Na **Output**, definiëren we de staat van bepaalde variabelen nadat het algoritme is uitgevoerd, en eventueel de waarde die wordt teruggegeven. Een waarde teruggeven doe je met de instructie **return**. Zodra **return** wordt uitgevoerd, stopt het algoritme en wordt de opgegeven waarde teruggegeven.

Dit voorbeeld toont hoe je de som van alle elementen van een array kan berekenen.




```
Input : arr[ ], een array van n getallen.
        n, het aantal elementen van de array.
Output : sum, de som van alle getallen in de array.

sum ← 0
for (i ← 0 to n - 1 step 1)
{
    sum ← sum + arr[i]
}
return sum
```




Opmerking: in dit laatste voorbeeld wordt de variabele i enkel gebruikt om de tel bij te houden van de **for**-lus. Er is dus geen uitleg voor nodig bij **Input** of **Output**, en de waarde ervan wordt niet teruggegeven.

Vraag 1 – Alcuin en zijn boot

De monnik Alcuin van York is de gelukkige eigenaar van een muis, een kat en een groot stuk kaas. Hij moet alles de rivier over brengen, maar zijn boot is te klein. Hij kan slechts één passagier tegelijk vervoeren: de kat, de muis of de kaas. Helaas, als de muis alleen aan de oever met de kaas wordt gelaten, zal ze hem opeten. (Als Alcuin aanwezig is aan de oever, kan hij de muis in de gaten houden.) Bovendien kan de kat niet alleen met de muis worden gelaten, anders zal hij haar opeten. Deze twee situaties worden conflicten genoemd. Er is geen conflict tussen de kat en de kaas, zij kunnen zonder toezicht samen blijven.

Alcuin wil de kat, de muis, de kaas en zichzelf van oever A naar oever B van de rivier brengen. Hij moet heen en weer gaan tussen de twee oevers, beginnend aan oever A. Hij onderzoekt verschillende scenario's. Hier is het eerste, waarin hij de richting van elke stap aangeeft en wat hij met zich meeneemt in zijn boot: de kat () , de muis () , de kaas () , of helemaal niets (niets).




Scenario 1

1. oever A → oever B: 
2. oever B → oever A: niets
3. oever A → oever B: 
4. oever B → oever A: niets
5. oever A → oever B: 

Om te controleren of zijn scenario geldig is, vult Alcuin een tabel in om aan te geven aan welke oever (A of B) Alcuin, de kat, de muis en de kaas zich bevinden bij elke stap. De tabel begint bij stap '0' die de beginsituatie weergeeft (voor de eerste overtocht).

De eerste twee regels van de tabel zijn al ingevuld. Je moet deze aanvullen en een kruis zetten in de laatste kolom als er een conflict is bij de betreffende stap.

Q1(a) /5 Vul de tabel van Alcuin in voor scenario 1.

	Alcuin				conflict?
stap 0 :	A	A	A	A	
stap 1 :	B	A	B	A	
stap 2 :	A	A	B	A	
stap 3 :	B	A	B	B	
stap 4 :	A	A	B	B	X
stap 5 :	B	B	B	B	

Alcuin overweegt vervolgens een nieuw scenario. Je wordt opnieuw gevraagd de tabel in te vullen.

Scenario 2

1. oever A → oever B:
2. oever B → oever A: niets
3. oever A → oever B:
4. oever B → oever A:
5. oever A → oever B:
6. oever B → oever A: niets
7. oever A → oever B:

Q1(b) /7 Vul de tabel van Alcuin in voor scenario 2.

	Alcuin				conflict?
stap 0 :	A	A	A	A	
stap 1 :	B	A	B	A	
stap 2 :	A	A	B	A	
stap 3 :	B	A	B	B	
stap 4 :	A	A	A	B	
stap 5 :	B	B	A	B	
stap 6 :	A	B	A	B	
stap 7 :	B	B	B	B	

Alcuin erft vervolgens een grotere boot, die hem in staat stelt twee passagiers naast zichzelf te vervoeren. We zeggen dat hij een boot van grootte 2 heeft (voorheen had hij een boot van grootte 1).






**Q1(c) /3 Geef een scenario in drie stappen om de kat, de muis en de kaas van oever A naar oever B te brengen met een boot van grootte 2.
Voor elke stap moet je 0, 1 of 2 passagiers noemen die door Alcuin worden vervoerd.**

1. oever A → oever B :
2. oever B → oever A : niets
3. oever A → oever B : ,

Er zijn verschillende andere oplossingen.

Stel nu dat de kat besluit van kaas te houden: er is een **nieuw conflict**, de kat en de kaas kunnen niet meer zonder toezicht samen blijven.

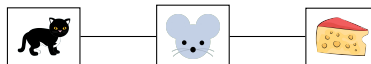
Q1(d) /3 Geef een scenario in 3 stappen met een boot van grootte 2. Je moet rekening houden met het nieuwe conflict tussen de kat en de kaas. De muis moet tijdens de eerste stap oversteken. Voor elke stap moet je 0, 1 of 2 passagiers noemen die door Alcuin worden vervoerd.

1. oever A → oever B :  , 
2. oever B → oever A : 
3. oever A → oever B :  , 

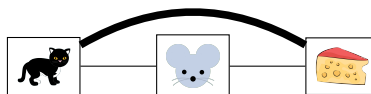
Er zijn verschillende andere oplossingen.

Met zijn ervaring besluit Alcuin het monastieke leven te verlaten en een bedrijf op te richten dat gespecialiseerd is in ingewikkelde rivierovertochten. Zijn klanten geven hem n passagiers (objecten of dieren) die de rivier moeten oversteken, evenals een diagram dat aangeeft wie in conflict is. Dit diagram wordt een 'graaf' genoemd en bestaat uit gelabelde rechthoeken, 'knopen' genoemd, één voor elke passagier. Het bevat ook verbindingen tussen de knopen, 'bogen' genoemd: we tekenen een boog tussen twee knopen als de twee bijbehorende passagiers in conflict zijn. Als twee passagiers zonder toezicht samen kunnen blijven, plaatsen we geen boog tussen de bijbehorende knopen.

Hier is de graaf van de beginsituatie toen de kat nog niet van kaas hield. Er was een conflict tussen de muis en de kat, een conflict tussen de muis en de kaas, maar geen conflict tussen de kat en de kaas.



Q1(e) /1 Wijzig deze graaf (door bogen te tekenen of door te strepen) om een conflict toe te voegen tussen de kat en de kaas.

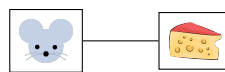


Alcuin wil nu weten hoe groot de boot moet zijn om een groep passagiers over te zetten waarvan de graaf van conflicten wordt gegeven. Natuurlijk is een boot van grootte n altijd voldoende om n passagiers te vervoeren, maar Alcuin wil de kleinste boot gebruiken.

Hij redeneert als volgt:

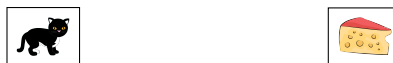
“Ik moet de passagiers in 2 groepen verdelen. Degenen in de eerste groep moeten permanent onder toezicht in de boot blijven. De anderen kunnen zonder toezicht blijven omdat er geen conflict tussen hen is. De passagiers die onder toezicht moeten blijven, moeten zo zijn dat als ik alle knopen die bij hen horen en de bogen die hen raken uit de graaf verwijder, er geen bogen meer in de graaf overblijven.”.

Bijvoorbeeld, het verwijderen van de knoop 'kat' is geen goede oplossing, want er blijft een conflict tussen de muis en de kaas.



Met andere woorden, we kunnen de kat niet alleen in de boot houden en de muis en de kaas aan de oever laten.

Anderzijds, toen de kat nog niet van kaas hield, geeft het verwijderen van de muis de volgende graaf die geen conflicten bevat:

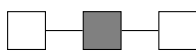


Met andere woorden, we konden (toen de kat nog niet van kaas hield) de muis alleen in de boot laden en de kat en de kaas aan de oever laten.

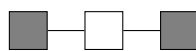
Wat Alcuin wil berekenen, wordt een *bedekking* genoemd: **een verzameling knopen zodanig dat, als we deze knopen en de bogen die ze raken uit de graaf verwijderen, er geen bogen meer overblijven.**

Hier zijn zes grafen, waarvan sommige knopen grijs zijn gekleurd:

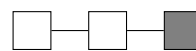
graaf 1:



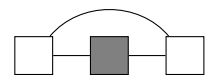
graaf 2:



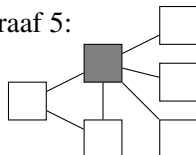
graaf 3:



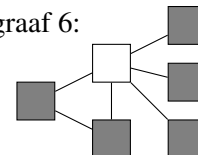
graaf 4:



graaf 5:



graaf 6:



	Ja	Nee	Geef aan of de grijze knopen een bedekking vormen in elke graaf.
Q1(f) /1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	De grijze knopen van graaf 1 vormen een bedekking.
Q1(g) /1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	De grijze knopen van graaf 2 vormen een bedekking.
Q1(h) /1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	De grijze knopen van graaf 3 vormen een bedekking.
Q1(i) /1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	De grijze knopen van graaf 4 vormen een bedekking.
Q1(j) /1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	De grijze knopen van graaf 5 vormen een bedekking.
Q1(k) /1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	De grijze knopen van graaf 6 vormen een bedekking.

Q1(l) /4	Kleur de knopen van de onderstaande graaf. De gekleurde knopen moeten een bedekking vormen met een minimaal aantal knopen.
-----------------	---



Alcuin wil een algoritme hebben dat een bedekking van een graaf berekent. Na er lang over te hebben gediscussieerd met de kat, vindt hij de volgende strategie (die niet per se erg intelligent is, maar het is het idee van de kat).

De knopen zijn genummerd van 1 tot n en worden in het algoritme weergegeven door hun nummer.

Als er een boog is tussen de knopen met nummers i en j , noteren we deze als (i, j) .

Het algoritme beschouwt achtereenvolgens alle bogen (i, j) van de graaf en 'behandelt' de uiteinden ervan, dat wil zeggen de knopen met nummers i en j .

Het behandelen van een knoop bestaat uit:

1. de knoop in de bedekking plaatsen,
2. de knoop en alle bogen waarvan hij een uiteinde is uit de graaf verwijderen.

We gaan door totdat er geen bogen meer in de graaf zijn.

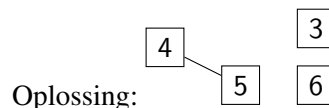
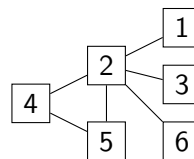
Alcuin merkt op dat er geen richting is aan de bogen, bijvoorbeeld de boog $(3, 1)$ is hetzelfde als $(1, 3)$.

Hij kan zich dus beperken tot het onderzoeken van de bogen (i, j) waar $i < j$ (om te voorkomen dat dezelfde boog twee keer wordt behandeld).

Alcuin besluit de bogen in de volgende volgorde te behandelen (als ze bestaan):

eerst de bogen $(1, 2), (1, 3), \dots, (1, n)$; dan de bogen $(2, 3), (2, 4), \dots, (2, n)$; dan de bogen $(3, 4), (3, 5), \dots, (3, n)$; en zo verder tot de laatste boog $(n - 1, n)$.

Q1(m) /3	Teken de graaf die aan het begin van het algoritme wordt verkregen na alleen de eerste boog $(1, 2)$ te hebben beschouwd en dus na het behandelen van de knopen 1 en 2 van de graaf hieronder.
-----------------	--



Q1(n) /4	Wat is de bedekking die wordt berekend na volledige uitvoering van het algoritme op de graaf van de vorige vraag? Je moet de lijst van knopen in de berekende bedekking geven.
-----------------	---

Oplissing: $\{1, 2, 4, 5\}$

Q1(o) /4 Wat is de bedekking die door het algoritme wordt berekend op de onderstaande graaf? Je moet de lijst van knopen in de berekende bedekking geven.

Oplossing: {1, 2, 3, 5}

Om het algoritme te implementeren, gebruikt Alcuin een tabel G van grootte $n \times n$ om de bogen op te slaan. Omdat de bogen geen richting hebben, beschouwt hij alleen de bogen (i, j) met $i < j$. De tabel bevat **true** in de cel $G[i][j]$ (met $i < j$) als en alleen als er een boog (i, j) in de graaf is. De andere cellen van de tabel bevatten **false**.

Q1(p) /4 Geef de tabel G die overeenkomt met de onderstaande graaf, waarbij de cel $G[i][j]$ zich op rij i , kolom j bevindt. Schrijf een T in de cellen die **true** bevatten.

	1	2	3	4	5	6
1		T		T		
2			T	T		
3					T	T
4					T	
5						
6						

Alcuin heeft zijn algoritme op een stuk perkament geschreven, maar helaas heeft de muis sommige delen ervan opgegeten. Alcuin heeft je hulp nodig om de ontbrekende delen terug te vinden.

Het algoritme begint met twee functies:

- `InitCountEdges` telt het aantal bogen in de graaf en slaat dit aantal op in de variabele `countEdges`,
- `ProcessNode(i)` behandelt de knoop `i` (zoals hierboven beschreven).

Naast de tabel `G[][]`, gebruiken we een Booleaanse tabel `InCover[]` van grootte `n` waarvan alle elementen zijn geïnitieerd op **false**. Het algoritme geeft aan dat de knoop `i` deel uitmaakt van de bedekking door de waarde van `InCover[i]` te wijzigen naar **true**.

Q1(q) /4 Vul de in de functie `InitCountEdges` in (antwoorden zo kort mogelijk).

```
function InitCountEdges()
{
    CountEdges = 0
    for (i ← 1 to n-1 step 1)
    {
        for (j ← i+1 to n step 1)
        {
            if (G[][) CountEdges ← 
        }
    }
}
```

Q1(r) /6 Vul de in de functie `ProcessNode` in (antwoorden zo kort mogelijk).

```
function ProcessNode(i)
{
    InCover[i] ← true
    for (k ← 1 to i-1 step 1)
    {
        if (G[k][i])
        {
            G[k][i] ← false
            CountEdges ← 
        }
    }
    for (k ← i+1 to n step 1)
    {
        if (G[i][k])
        {
            G[i][k] ← false
            CountEdges ← 
        }
    }
}
```

Het algoritme wordt aangevuld met de functie `Cover`.

Deze functie ontvangt een tabel `G[] []` van grootte $n \times n$ die de bestaande bogen tussen de n knopen vertegenwoordigt. De tabel `InCover[]` (van grootte n) wordt geïntialiseerd op **false** zoals hierboven uitgelegd voordat de functie `Cover` wordt uitgevoerd.

De berekende bedekking bestaat uit alle knopen i waarvoor `InCover[i]` **true** is na uitvoering van de functie `Cover`.

Q1(s) /6 Vul de in de functie `Cover` in (antwoorden zo kort mogelijk).

```
function Cover()
{
  InitCountEdges()
  i ← 1
  j ← 2
  while (CountEdges != 0)
  {
    if (G[i][j])
    {
      ProcessNode(i)
      ProcessNode(j)
    }
    j ← j+1
    if (j = n+1){
      i ← 
      j ← 
    }
  }
}
```

Vraag 2 – Zeeslag

We willen het spel Zeeslag programmeren dat wordt gespeeld op een 10×10 rooster waarop we schepen plaatsen.

- Er zijn 10 rijen genummerd van 1 tot 10 van boven naar beneden.
- Er zijn 10 kolommen genummerd van 1 tot 10 van links naar rechts.
- Een horizontaal schip bezet opeenvolgende vakjes op dezelfde rij.
- Een verticaal schip bezet opeenvolgende vakjes in dezelfde kolom.
- We gebruiken notatie (r, c) voor de coördinaten van het vakje op het snijpunt van rij r en kolom c .

Voorbeeld :

- 3 schepen **A**, **B** en **C** zijn in grijs getekend op de afbeelding.
- **A** is horizontaal, **B** is verticaal, **C** is zowel horizontaal als verticaal.
- **B** bezet de vakjes met coördinaten $(5, 7)$, $(6, 7)$ en $(7, 7)$.

	1	2	3	4	5	6	7	8	9	10
1										
2			A	A	A	A				
3										
4										
5							B			
6							B			
7							B			
8				C						
9										
10										

We twijfelen tussen 2 systemen om de schepen in ons programma voor te stellen.

Elk systeem gebruikt 4 attributen om de positie van een schip op het rooster aan te geven.

Attributen in het systeem Oriëntatie.

- De rij r van het vakje links bovenaan het schip.
- De kolom c van het vakje links bovenaan het schip.
- De lengte L van het schip.
- De richting hor van het schip :
true als het horizontaal ligt, **false** als het verticaal ligt.

Oriëntatie-attributen van de schepen op de afbeelding.

Schip	r	c	L	hor
A	2	3	4	true
B	5	7	3	false
C	8	4	1	true of false naar keuze

Attributen in het systeem Rechthoek.

- De rij $r1$ van het vakje links bovenaan het schip.
- De kolom $c1$ van het vakje links bovenaan het schip.
- De rij $r2$ van het vakje rechts onderaan het schip.
- De kolom $c2$ van het vakje rechts onderaan het schip.

Rechthoek-attributen van de schepen op de afbeelding.

Schip	r1	c1	r2	c2
A	2	3	2	6
B	5	7	7	7
C	8	4	8	4

We gaan enkele functies schrijven in de 2 systemen om te beslissen welke we in ons programma zullen gebruiken.

Conversies tussen de 2 systemen

Laten we eerst oefenen om van het ene systeem naar het andere over te gaan.

Oriëntatie → Rechthoek

Een schip wordt beschreven door zijn 4 attributen (r, c, L, hor) in het systeem **Oriëntatie**.

Geef zijn attributen $(r1, c1, r2, c2)$ in het systeem **Rechthoek**.

Q2(a) /1	$(4, 5, 2, \text{True}) \rightarrow (4, 5, 4, 6)$
----------	---

Q2(b) /1	$(3, 6, 3, \text{False}) \rightarrow (3, 6, 5, 6)$
----------	--

Q2(c) /1	$(7, 5, 1, \text{True}) \rightarrow (7, 5, 7, 5)$
----------	---

Q2(d) /1	$(1, 9, 5, \text{False}) \rightarrow (1, 9, 5, 9)$
----------	--

Rechthoek → **Oriëntatie**

Een schip wordt beschreven door zijn 4 attributen $(r1, c1, r2, c2)$ in het systeem **Rechthoek**.

Geef zijn attributen (r, c, L, hor) in het systeem **Oriëntatie**.

Q2(e) /1 $(2, 7, 5, 7) \rightarrow (2, 7, 4, \text{False})$

Q2(f) /1 $(4, 3, 4, 3) \rightarrow (4, 3, 1, \text{True})$

Q2(g) /1 $(8, 3, 8, 6) \rightarrow (8, 3, 4, \text{True})$

Q2(h) /1 $(1, 4, 1, 10) \rightarrow (1, 4, 7, \text{True})$

Laten we functies maken om deze conversies te automatiseren.

Functie O2R

De functie **O2R** converteert attributen van het systeem **Oriëntatie** naar equivalente attributen van het systeem **Rechthoek**.

Bijvoorbeeld, gebruikmakend van het schip **A** op de afbeelding, geeft **O2R** $(2, 3, 4, \text{true})$ het resultaat $(2, 3, 2, 6)$ terug.

Q2(i) /5 Vul de in de functie **O2R** aan.

```
function O2R(r, c, L, hor)
{
  if (hor)
    { return (r, c, r, c+L-1) }
  else
    { return (r, c, r+L-1, c) }
}
```

Functie R2O

De functie **R2O** converteert attributen van het systeem **Rechthoek** naar equivalente attributen van het systeem **Oriëntatie**.

Gebruik een logische uitdrukking (zie volgende pagina) om de waarde van **hor** te berekenen.

Bijvoorbeeld, gebruikmakend van het schip **A** op de afbeelding, geeft **R2O** $(2, 3, 2, 6)$ het resultaat $(2, 3, 4, \text{true})$ terug.

Q2(j) /5 Vul de in de functie **R2O** aan.

```
function R2O(r1, c1, r2, c2)
{
  L ← (r2-r1) + (c2-c1) + 1

  hor ← (r1==r2)

  return (r1, c1, L, hor)
}
```

Geldigheidstests

Het programma moet controleren of attributen een schip beschrijven dat we op het 10x10-rooster kunnen plaatsen. Dit gebeurt door logische uitdrukkingen te evalueren, waarvan het resultaat **true** of **false** is.

Dit type uitdrukking gebruikt vergelijkingsoperatoren (`==`, `!=`, `>`, `>=`, `<`, `<=`) en logische operatoren (**and**, **or**, **not**) zoals getoond in de onderstaande voorbeelden.

Logische uitdrukking	Waarde
<code>c==4</code>	true als <code>c</code> gelijk is aan 4, en anders false .
<code>r!=5</code>	true als <code>r</code> verschillend is van 5.
<code>r1>3</code>	true als <code>r1</code> groter is dan 3.
<code>c2>=c1</code>	true als <code>c2</code> groter of gelijk is aan <code>c1</code> .
<code>r+L<9</code>	true als <code>r+L</code> kleiner is dan 9.
<code>c2<=c1+2</code>	true als <code>c2</code> kleiner of gelijk is aan <code>c1+2</code> .
<code>(c<2) and (r>3)</code>	true als beide ongelijkheden waar zijn.
<code>(c1==3) or (r1==2)</code>	true als minstens een van de twee gelijkheden waar is.
<code>not (hor)</code>	true als <code>hor</code> gelijk is aan false (en false als <code>hor</code> gelijk is aan true).

Funcie Rok

In het systeem **Rechthoek** moeten de vakjes $(r1, c1)$ en $(r2, c2)$ in dezelfde rij of kolom liggen, moeten alle coördinaten tussen 1 en 10 liggen en moet het eerste vakje boven of links van het tweede vakje liggen.

Vul de functie `Rok` aan die **true** teruggeeft als de attributen $(r1, c1, r2, c2)$ aan al deze voorwaarden voldoen.

Omdat de uitdrukking erg lang is, wordt deze in meerdere stappen geëvalueerd met behulp van de logische variabele `ok`.

Q2(k) /5 Vul de in de functie `Rok` aan.

```
function Rok(r1, c1, r2, c2)
{
  ok ← (  ) or (  )
  ok ← ok and (1<=r1 and  and  and  )
  ok ← ok and (1<=c1 and  and  and  )
  return ok
}
```

Funcie Ook

In het systeem **Oriëntatie** moet de lengte groter of gelijk zijn aan 1.

Vul de functie `Ook` aan die **true** teruggeeft als het schip (r, c, L, hor) op het 10x10-rooster kan worden geplaatst.

Q2(l) /5 Vul de in de functie `Ook` aan.

```
function Ook(r, c, L, hor)
{
  ok ← (  and 1<=r and 1<=c )
  if (hor)
  { return (ok and  and  ) }
  else
  { return (ok and  and  ) }
}
```

Schip geraakt

Het programma moet controleren of een schip geraakt is wanneer we op een vakje schieten waarvan we de coördinaten geven.

Dit is het geval als het schip het beoogde vakje bezet.

Probeer in de volgende vragen de kortste en eenvoudigste antwoorden te vinden met een minimum aan vergelijkingen en logische bewerkingen.

Functie hitR (systeem Rechthoek)

De functie `hitR(rr, cc, r1, c1, r2, c2)` geeft **true** terug als het schip met attributen `(r1, c1, r2, c2)` het vakje met coördinaten `(rr, cc)` bezet en geeft **false** terug als dit niet het geval is.

Met de situatie beschreven op de afbeelding:

- `hitR(2, 5, 2, 3, 2, 6)` geeft **true** terug omdat het schip **A** het vakje met coördinaten `(2, 5)` bezet.
- `hitR(7, 6, 5, 7, 7, 7)` geeft **false** terug omdat het schip **B** het vakje met coördinaten `(7, 6)` niet bezet.

Q2(m) /6	Vul de <input type="text"/> in de functie <code>hitR</code> aan (antwoord zo kort mogelijk).
----------	--

```
function hitR(rr, cc, r1, c1, r2, c2)
{
    return (r1<=rr) and (rr<=r2) and (c1<=cc) and (cc<=c2)
}
```

Functie hitO (systeem Oriëntatie)

De functie `hitO(rr, cc, r, c, L, hor)` geeft **true** terug als het schip met attributen `(r, c, L, hor)` het vakje met coördinaten `(rr, cc)` bezet en geeft **false** terug als dit niet het geval is.

Met de situatie beschreven op de afbeelding:

- `hitO(2, 5, 2, 3, 4, true)` geeft **true** terug omdat het schip **A** het vakje met coördinaten `(2, 5)` bezet.
- `hitO(7, 6, 5, 7, 3, false)` geeft **false** terug omdat het schip **B** het vakje met coördinaten `(7, 6)` niet bezet.

Q2(n) /6	Vul de <input type="text"/> in de functie <code>hitO</code> aan (antwoord zo kort mogelijk).
----------	--

```
function hitO(rr, cc, r, c, L, hor)
{
    if (hor)
        { return (rr==r) and (c<=cc) and (cc<c+L) }
    else
        { return (cc==c) and (r<=rr) and (rr<r+L) }
}
```

Botsing

Twee schepen kunnen niet hetzelfde vakje op het rooster bezetten, anders is er een botsing.

Functie collisionR (systeem Rechthoek)

De functie `collisionR` geeft **true** terug als twee schepen in botsing zijn en **false** als dit niet het geval is.

De attributen van de schepen zijn $(r1A, c1A, r2A, c2A)$ en $(r1B, c1B, r2B, c2B)$.

Gebruik zo kort mogelijke logische uitdrukkingen. Het is desondanks erg lang en de uitdrukking wordt in meerdere stappen geëvalueerd met behulp van de logische variabelen `rbool` en `cbool`.

Q2(o) /7 Vul de _____ in de functie `collisionR` aan (antwoord zo kort mogelijk).

```
function collisionR(r1A, c1A, r2A, c2A, r1B, c1B, r2B, c2B)
{
  rbool ← (r1A<=r1B and r1B<=r2A) or (r1B<=r1A and r1A<=r2B)

  cbool ← (c1A<=c1B and c1B<=c2A) or (c1B<=c1A and c1A<=c2B)
  return rbool and cbool
}
```

Functie collisionO (systeem Oriëntatie)

De functie `collisionO` geeft **true** terug als twee schepen in botsing zijn en **false** als dit niet het geval is.

De attributen van de schepen zijn $(rA, cA, LA, horA)$ en $(rB, cB, LB, horB)$.

Om nog langere logische uitdrukkingen te vermijden, gebruiken we de functie `hitO` die eerder is gedefinieerd.

Q2(p) /7 Vul de _____ in de functie `collisionO` aan met behulp van de functie `hitO`.

```
function collisionO(rA, cA, LA, horA, rB, cB, LB, horB)
{
  if (horA==horB)
  { return hitO(rB, cB, rA, cA, LA, horA) or hitO(rA, cA, rB, cB, LB, horB) }
  else
  {
    if (horA)
    { return hitO(rA, cB, rA, cA, LA, horA) and hitO(rA, cB, rB, cB, LB, horB) }
    else
    { return hitO(rB, cA, rA, cA, LA, horA) and hitO(rB, cA, rB, cB, LB, horB) }
  }
}
```


Risico op botsing

De schepen mogen niet te dicht bij elkaar liggen.

Meer bepaald mogen twee vakjes bezet door twee verschillende schepen geen zijde of hoek gemeen hebben.

Bijvoorbeeld, alle schepen in de onderstaande afbeelding liggen te dicht bij een ander schip.

A en **B** raken elkaar, **C** en **D** raken elkaar, **E** en **F** raken elkaar met een hoek.

	1	2	3	4	5	6	7	8	9	10
1								C		
2		A	A	A	A			C		
3			B					C		
4			B					C	D	
5									D	
6										
7										
8		E	E	E	E					
9						F	F	F		
10										

Functie riskR (systeem Rechthoek)

De functie `riskR` geeft **true** terug als twee schepen te dicht bij elkaar liggen en **false** als dit niet het geval is.

De attributen van de schepen zijn $(r1A, c1A, r2A, c2A)$ en $(r1B, c1B, r2B, c2B)$.

Het eenvoudigst is om de functie `collisionR` te gebruiken die eerder is gedefinieerd.

Q2(q) /6 Vul de in de functie `riskR` aan met behulp van de functie `collisionR`.

```
function riskR(r1A, c1A, r2A, c2A, r1B, c1B, r2B, c2B)
{
  return collisionR(, r1B-1, c1B-1, r2B+1, c2B+1)
}
```

Functie riskO (systeem Oriëntatie)

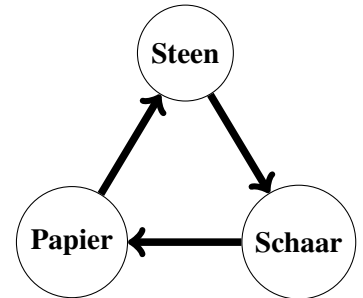
Deze functie is moeilijker te schrijven.

We geven op en besluiten het systeem Rechthoek in ons programma te gebruiken.

Vraag 3 – Steen-Schaar-Papier

In een beroemd spel bootsen de 2 spelers tegelijkertijd een wapen na met hun hand: **Steen** of **Schaar** of **Papier**.

Het resultaat van het gevecht wordt bepaald door de cirkelvormige grafiek hiernaast: **Steen** verslaat **Schaar** die **Papier** verslaat die **Steen** verslaat. Er is een gelijkspel als de spelers hetzelfde wapen kiezen.



Een speelgoedfabrikant wil kleine robots op de markt brengen die tegen elkaar kunnen spelen in **Steen-Schaar-Papier**.

Twee robots die dicht bij elkaar worden geplaatst, communiceren draadloos. Een speler drukt op een knop van zijn robot om een wedstrijd voor te stellen, de andere speler drukt op een knop van zijn robot om de wedstrijd te accepteren. Tijdens een wedstrijd spelen de robots 10 rondes van **Steen-Schaar-Papier**, elke overwinning levert 1 punt op voor de winnende robot.

De rondes en scores kunnen worden gevolgd op de schermen van de robots.

Degene die de meeste rondes wint, wint de wedstrijd.

Er kan een gelijkspel zijn als beide robots hetzelfde aantal rondes winnen.

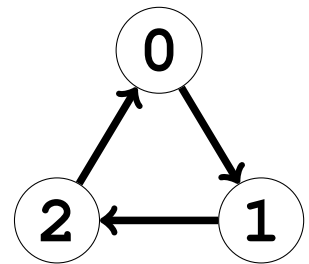
De robots spelen niet willekeurig: elk volgt een **strategie** bepaald door een programma.

De fabrikant wil een groot aantal robots aanbieden in verschillende kleuren, vormen en vooral verschillende strategieën.

Coderingen

In de programma's wordt **Steen** gecodeerd als 0, **Schaar** als 1 en **Papier** als 2. Dus 0 verslaat 1 die 2 verslaat die 0 verslaat en de grafiek van het spel wordt zoals hiernaast.

In het vervolg zullen we zeggen dat een robot de zet 0, 1 of 2 speelt.



Strategieën

De strategie van een robot wordt bepaald door 2 elementen.

- De variabele `init` die de zet bevat die gespeeld moet worden tijdens de eerste ronde.
- De tabel `strat`, met 3 rijen en 3 kolommen, die wordt gebruikt om de zetten van de andere rondes te bepalen. Als de robot `r` heeft gespeeld en zijn tegenstander `s`, dan speelt hij `strat[r][s]` in de volgende ronde.

Een voorbeeld van een tabel `strat` wordt rechts gegeven.

De rijnummers en kolomnummers beginnen bij 0 en worden in grijs weergegeven.

Met deze tabel en afhankelijk van de zetten van de vorige ronde, moet de robot spelen:

- `strat[1][2]=0` als de robot 1 heeft gespeeld en zijn tegenstander 2.
- `strat[2][0]=2` als de robot 2 heeft gespeeld en zijn tegenstander 0.
- 1 als er een gelijkspel was omdat `strat[0][0]=strat[1][1]=strat[2][2]=1`.

	0	1	2
0	1	2	0
1	2	1	0
2	2	2	1

Strategieën coderen

In de volgende vragen houden we ons niet bezig met de variabele `init`.

Een strategie wordt beschreven door een tekst die uitlegt wat de robot moet spelen afhankelijk van wat hij en zijn tegenstander in de vorige ronde hebben gespeeld.

Je moet de tabel `strat` coderen die overeenkomt met de strategie die in de tekst wordt beschreven.

Je kunt de roosters op de volgende pagina als klad gebruiken.

Vergeet niet om je oplossingen **over te schrijven op de antwoordbladen**.

Q3(a) /4 Vul de tabel `strat` in volgens de strategie die in deze zin wordt beschreven.
“Speel de vorige zet van de tegenstander.”

	0	1	2
0	0	1	2
1	0	1	2
2	0	1	2

Q3(b) /4 Vul de tabel `strat` in volgens de strategie die in deze zin wordt beschreven.
“Speel de zet die wint van de vorige zet van de tegenstander.”

	0	1	2
0	2	0	1
1	2	0	1
2	2	0	1

Q3(c) /4 Vul de tabel `strat` in volgens de strategie die in deze zin wordt beschreven.
“Als er een gelijkspel was, speel de vorige zet van de robot; anders speel de zet van de vorige winnaar.”

	0	1	2
0	0	0	2
1	0	1	1
2	2	1	2

Q3(d) /4 Vul de tabel `strat` in volgens de strategie die in deze zin wordt beschreven.
“Als er een gelijkspel was, speel de zet die wint van de vorige zet; anders speel de vorige zet van de robot opnieuw.”

	0	1	2
0	2	0	0
1	1	0	1
2	2	2	1



In het vervolg gebruiken we de onderstaande notaties.

- r : de zet die de robot in de vorige ronde heeft gespeeld.
- s : de zet die de tegenstander in de vorige ronde heeft gespeeld.
- w : de zet die de winnaar van de vorige ronde heeft gespeeld (of door beide spelers in geval van gelijkspel).
- x : de zet die de verliezer van de vorige ronde heeft gespeeld (of door beide spelers in geval van gelijkspel).
- $W(c)$: de zet die wint van de zet c (bijvoorbeeld $W(0) = 2$ omdat 2 wint van 0).
- $X(c)$: de zet die verliest van de zet c (bijvoorbeeld $X(0) = 1$ omdat 0 wint van 1).

Q3(e) /4 Vul de tabel **st rat** in volgens de strategie die in deze zin wordt beschreven.
 “Als er een gelijkspel was, speel $X(r)$; anders speel de zet die niet is gespeeld.”

	0	1	2
0	1	2	1
1	2	2	0
2	1	0	0

Q3(f) /4 Vul de tabel **st rat** in volgens de strategie die in deze zin wordt beschreven.
 “Als $s=0$, speel r ; als $s=1$ en er was geen gelijkspel, speel w ; als $s=2$ en er was geen gelijkspel, speel x ; in andere gevallen, speel $W(r)$.”

	0	1	2
0	0	0	0
1	1	0	2
2	2	1	1



Een strategie verslaan

In de volgende vragen wordt een strategie A volledig gegeven met `initA` en de tabel `stratA`.

Vind een strategie B die met een score van 10-0 wint in een wedstrijd tegen strategie A.

Je moet de **eerste zet** `initB` en een **minimum aan waarden in de tabel** `stratB` geven.

Laat de vakjes van `stratB` leeg die nooit gebruikt zullen worden door je strategie in een wedstrijd tegen strategie A.

De rijnummers en kolomnummers worden niet meer genoteerd, voeg ze toe als je dat nodig acht.

Q3(g) /4 Geef een strategie B die met 10-0 wint tegen strategie A.
Geef alleen de noodzakelijke waarden in `stratB`.

`initA=0, stratA=`

2	0	0
1	0	1
2	2	1

\rightarrow `initB=2`, `stratB=`

2		

Q3(h) /4 Geef een strategie B die met 10-0 wint tegen strategie A.
Geef alleen de noodzakelijke waarden in `stratB`.

`initA=1, stratA=`

2	0	1
0	1	2
1	2	0

\rightarrow `initB=0`, `stratB=`

	2	
0		

Q3(i) /4 Geef een strategie B die met 10-0 wint tegen strategie A.
Geef alleen de noodzakelijke waarden in `stratB`.

`initA=2, stratA=`

2	0	1
2	0	1
2	0	1

\rightarrow `initB=1`, `stratB=`

	1	
		2
0		

Q3(j) /4 Geef een strategie B die met 10-0 wint tegen strategie A.
Geef alleen de noodzakelijke waarden in `stratB`.

`initA=1, stratA=`

1	1	2
0	1	0
0	2	2

\rightarrow `initB=0`, `stratB=`

	2	
		1
1		

Vergeet niet om je oplossingen **over te schrijven op de antwoordbladen**.