be-OI 2012 **Halve Finale** 1 februari 2012	**Invullen in HOOFDLETTERS en LEESBAAR aub** VOORNAAM : ..... NAAM : ..... SCHOOL :	**Gereserveerd**

Belgische Olympiade in de Informatica (duur : 3u maximum)

Dit is de vragenlijst van de halve finale van de Belgische Olympiade in de Informatica. Ze bevat 8 vragen, en je krijgt **maximum 3u** de tijd om ze op te lossen. Naast elke vraag vind je een indicatie van de tijd die het mogelijk kost om de vraag op te lossen. Dit 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, 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. 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 die laatste a^b berekent.

5. Voor alle opgaves geldt: 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: `a/b = 4` en `a%b = 2`

6. Een array van lengte n wordt geïndexeerd van 0 tot $n-1$. 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 herhaling.

7. Je mag **op geen enkel moment met iemand communiceren**, behalve met de toezichthouders. Je mag hen wel om bvb kladpapier vragen, maar je mag geen vragen stellen over de inhoud van de proef. Zo garanderen wij gelijke behandeling tussen deelnemers in alle regionale centra. Elke fout in de opgave moet je beschouwen als onderdeel van de proef.

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

9. Je krijgt **exact drie uur** de tijd om op de vragen te antwoorden. Een **overzicht** over pseudo-code is voorzien in bijlage, op de laatste bladzijde.

Succes !

Vragenlijst halve finale

Vraag 1 – Opwarming (5 min)

Lees de volgende code:

```
i ← 1
// A
while(i < 5)
{
  // B
  Schrijf(i)
  // C
}
```

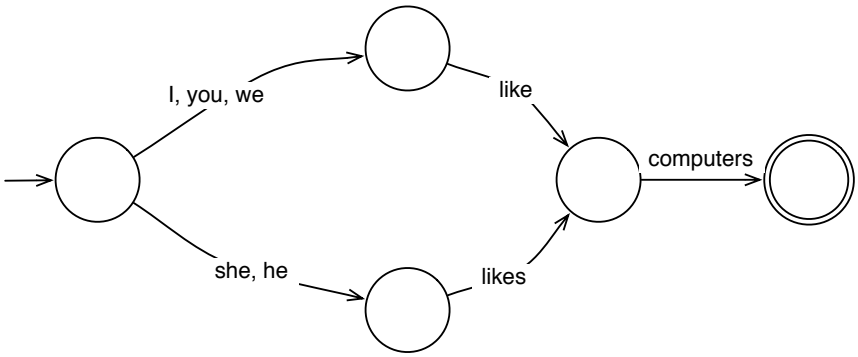
Waar moet de variabele `i` met 1 verhoogd worden zodat deze lus de opeenvolgende waarden `1 2 3 4` uitschrijft ? De functie `Schrijf(i)` schrijft de waarde van `i` op het scherm.

Op welke plaats hoort de instructie $i \leftarrow i + 1$? **Kruis het juiste vakje aan**.

Q1	☐ A	☐ B	☐ C

Vraag 2 – *Let's learn English !* (15 min)

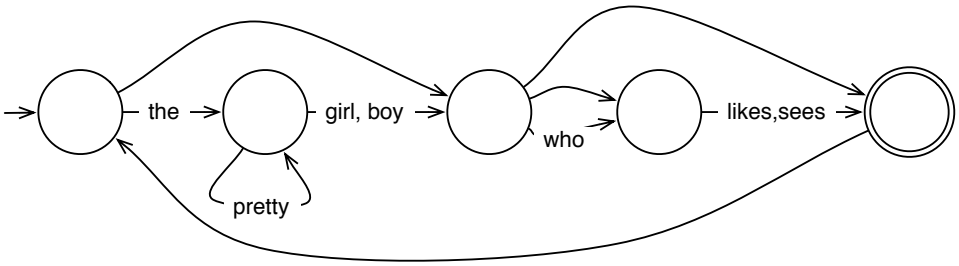
Sommige structuren uit de Engelse grammatica kunnen we beschrijven met behulp van schema's die bestaan uit cirkels en pijlen: de cirkels noemen we *toestanden* en pijlen noemen we *overgangen*. De overgangen verbinden de toestanden met elkaar, en zijn mogelijk voorzien van één of meerdere *etiketten* (gescheiden door een komma's als er meerdere zijn). Hieronder zie je een voorbeeld.



Je ziet twee speciale toestanden: de meest linkse toestand heeft een korte binnenkomende pijl die uit het niets vertrekt. Dit is de *begintoestand*. De meest rechtse toestand is dubbel omlijnd: dat is de *eindtoestand*.

Dit soort schema's beschrijft op een compacte manier een aantal zinnen. Je gebruikt het schema als volgt: je vertrekt bij de begintoestand, en je volgt een pad van toestand naar toestand, via de overgangen, tot je bij de eindtoestand komt. Bij elke overgang kies je één van de beschikbare etiketten (als er etiketten zijn). De rij van gekozen etiketten vormt nu een zin. Merk op dat je de eindtoestand verschillende keren mag doorlopen, zolang je op het einde maar in de eindtoestand eindigt. Bijvoorbeeld: het schema hierboven geeft je (o.a.) de zin « *he like**s** computers* », maar niet « *I like**s** computers* ».

Hieronder een ander en ingewikkelder voorbeeld, dat een aantal correcte en incorrecte Engelse zinnen voorstelt.



Duid voor elk van de zinnen hieronder aan **of de zin door het schema voorgesteld kan worden** (Ja of Nee). *(1 punt per juist aangekruist vakje, -1 punt per fout aangekruist vakje, geen negatieve score op deze vraag)*

Ja	Nee	
☐	☐	« *the pretty pretty girl sees the boy who likes the pretty girl* »
☐	☐	« *the girl who likes pretty pretty boy* »
☐	☐	« »
☐	☐	« *who sees the pretty girl* »

Vraag 3 – Een beetje logica (15 min)

Q3(a) – Naamse informatici

Kruis de ontkenning aan van de volgende zin: "Alle inwoners van Namen die bruine ogen hebben worden informaticus en gaan op pensioen voor hun 50ste".

☐	Er bestaat minstens één informaticus met bruine ogen die niet in Namen woont en die voor zijn 50ste op pensioen gaat.
☐	Geen enkele Naamse informaticus met bruine ogen gaat voor zijn 50ste op pensioen.
☐	Er bestaat een inwoner van Namen met bruine ogen die geen informaticus wordt of die op pensioen gaat na zijn 50ste.
☐	Er bestaat geen inwoner van Namen met bruine ogen die informaticus wordt en op pensioen gaat na zijn 50ste.

Q3(b) – De jacht op het *goudhaantje*

Jagers organiseren een jacht op het *goudhaantje* in het bos van Tervuren. Op de laatste dag van hun drijfjacht hebben ze er nog altijd geen gevangen. Ze besluiten dan maar om er zelf 171 los te laten. Die kunnen ze uiteindelijk allemaal schieten, behalve 67. **Hoeveel** blijven er nog over?

Q3(b) **een geheel getal**

. . . . . .

Q3(c) – Het spookhuis

In een spookhuis manifesteren de geesten zich op twee manieren: met een vuil liedje, en met een kakelend gelach. Men kan hun gedrag beïnvloeden door orgel te spelen of door wierook te verbranden.

Na wat onderzoek bleek:

- men hoort het liedje niet, tenzij men orgel speelt zonder dat men gelach hoort
- als men wierook brandt, dan hoort men de lach enkel als het liedje gehoord wordt

Op dit ogenblik hoort men het liedje maar geen lach. **Kan je besluiten** dat men nu orgel speelt en geen wierook brandt?

Q3(c)

☐ **Ja** ☐ **Neen** ☐ **Dat kunnen we hier niet met zekerheid uit afleiden**

Vraag 4 – Absoluut maximum (25 min)

Het algoritme hieronder moet in een array het getal zoeken dat het verst van 0 verwijderd is. Bijvoorbeeld, voor onderstaande array tab_1, moet het algoritme -9 teruggeven.

tab_1

1	7	5	−9	0	2

```
Input  : tab, een array met gehele getallen
         n, de grootte van tab. n > 0
Output : Geef de waarde uit tab terug die het verst van nul verwijderd is. Als meerdere
         oplossingen mogelijk zijn, geef dan de eerste oplossing die je tegenkomt (van
         links naar rechts) terug.

num ← 0

for (i ← 0 to n − 1 step 1)
{
    if ( [...] )
    {
        num ← tab[i]
    }
}

return num
```

Evalueer de volgende uitdrukkingen. **Vink *ja*** aan als de uitdrukking het algoritme zodanig vervolledigt, dat het de juiste waarde teruggeeft voor eender welke array die als input gegeven wordt. Is dat niet het geval, **vink** dan ***nee*** aan. *(1 punt per juist aangekruist vakje, -1 punt per fout aangekruist vakje, geen negatieve score op deze vraag)*

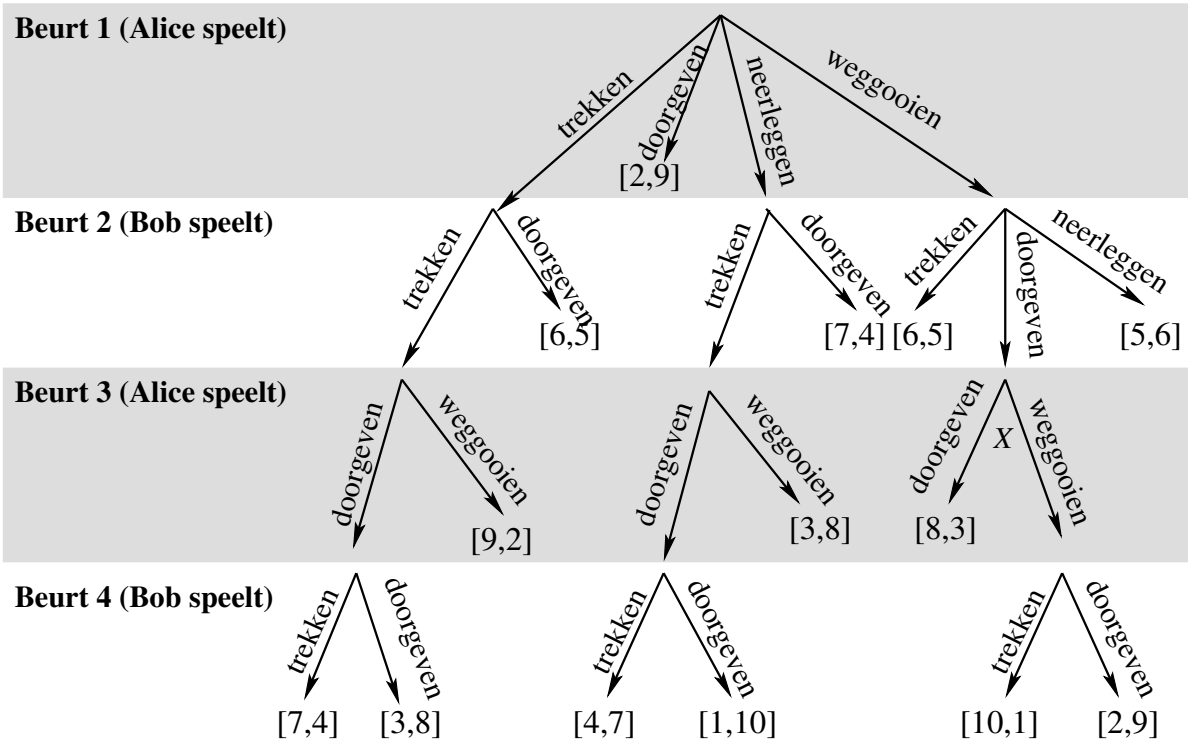
Ja	Nee	
☐	☐	$(tab[i] < 0$ **and** $-tab[i] > num)$ **or** $tab[i] > num$
☐	☐	$(num < 0$ **and** $(tab[i] < num$ **or** $tab[i] > -num))$ **or** $(num > 0$ **and** $(tab[i] > num$ **or** $tab[i] < -num))$
☐	☐	$tab[i] > 0$ **and** $tab[i] > num$
☐	☐	$(num \geq 0$ **and** $(tab[i] > num$ **or** $tab[i] < -num))$ **or** $(num \leq 0$ **and** $(tab[i] < num$ **or** $tab[i] > -num))$
☐	☐	$(tab[i] < 0$ **and** $-tab[i] > num)$ **or** $(tab[i] > 0$ **and** $tab[i] > num)$
☐	☐	$-tab[i] > num$ **or** $tab[i] > num$

Vraag 5 – Strategisch Spel (15 min)

Alice en Bob spelen een complex strategisch spel waarin zij wedijveren om een sterkere beschaving op te bouwen dan die van hun tegenstander. Een beschaving wordt opgebouwd aan de hand van een reeks door de spelers gekozen kaarten.

Op het einde van het spel wordt de score van elke speler bepaald door 11 overwinningspunten te verdelen onder de spelers, aan de hand van verschillende criteria (2 punten voor militaire overmacht, 3 punten voor technologische ontwikkeling, enz.) De speler met de hoogste score wint. Een spannend spel kan bijvoorbeeld nipt gewonnen worden door Alice met 6 punten, tegenover 5 punten voor Bob (deze uitkomst van het spel wordt voorgesteld als $[6, 5]$).

Alice wordt uitgeloot om het spel te beginnen. Ze beslist om niets aan het toeval over te laten en schetst alle mogelijke uitkomsten van het spel (zie hieronder). In haar eerste beurt heeft Alice vier mogelijke acties (een kaart *trekken*, *doorgeven*, *neerleggen* of *weggooien*). Vervolgens is het de beurt aan Bob, enzovoort, tot het einde van het spel, dat hier wordt voorgesteld door de eindscore. Het aantal mogelijke acties is afhankelijk van de spelsituatie. Merk ook op dat een spel sneller kan eindigen, bijvoorbeeld in verlies voor Alice als zij in haar eerste beurt een kaart doorgeeft.



Q5(a) Als Alice weet dat de strategie van Bob erin bestaat om altijd *door te geven*, **met welke actie** moet Alice dan beginnen om haar score te maximaliseren, en **wat zal die score zijn**?

Q5(a) **(een actie en een getal)**

Vraag 5 (vervolg)

Q5(b) In een volgend spel weten Alice en Bob van elkaar dat ze dezelfde optimale strategie zullen gebruiken. Beide spelers zullen dus zo goed als mogelijk spelen en proberen allebei hun score te maximaliseren. Bijvoorbeeld, in de spelsituatie in beurt 3 gemarkeerd met een X in de figuur, zal Alice *doorgeven* kiezen (voor een score van 8). Indien ze had gekozen voor *weggooien* (in de hoop een score van 10 te behalen), zou Bob kiezen voor *doorgeven* (waardoor Alice slechts een score van 2 zou behalen). Toeval beslist opnieuw dat Alice mag beginnen. **Met welke actie** moet Alice ditmaal beginnen om haar score te maximaliseren, en **wat zal die score zijn**?

Q5(b) **(een actie en een getal)**

. . . . . .

Vraag 6 – Populatiegroei (20 min)

Bart is een wetenschapper die onderzoek doet naar de groei van populaties. Hij voert een reeks experimenten uit waarbij hij de grootte van een populatie meet op regelmatige tijdstippen. Bart wil dan, gebaseerd op deze data, de aangroei van de populatie in de toekomst voorspellen.

Bart roept de hulp in van computerwetenschapper Lisa om een algoritme te ontwerpen dat de grootte van een populatie kan berekenen op een bepaald tijdstip, gegeven de grootte van de populatie op een of meer vroegere tijdstippen. Hij geeft Lisa een reeks meetpunten, inclusief de initiële populatiegrootte. Lisa berekent hiermee de relatie tussen opeenvolgende populatiegroottes, en schrijft op basis daarvan het algoritme.

Bart heeft net enkele nieuwe algoritmen ontvangen van Lisa. Vooraleer Bart deze algoritmen gaat gebruiken om voorspellingen te maken, wil hij zeker zijn dat het algoritme de correcte resultaten geeft voor de meetpunten die hij aan Lisa heeft doorgegeven.

Help Bart door voor elk van de volgende algoritmen de populatiegrootte uit te rekenen op tijdstippen $t = 0, 1, 2, 3, 4$ en 5. De parameter $t \geq 0$ stelt het tijdstip voor waarop de populatiegrootte gemeten wordt. Voor $t = 0$ berekent het algoritme de initiële grootte van de populatie.

```
function populatie1(t)
{
    if (t = 0)
    {
        return 0
    }
    else
    {
        return populatie1(t-1) + 3
    }
}
```

Q6(a) Bereken de uitkomst t_i van de oproep `populatie1(i)`:

Q6(a)	$t_0 =$.....	$t_1 =$.....	$t_2 =$.....	$t_3 =$.....	$t_4 =$.....	$t_5 =$.....

```
function populatie2(t)
{
    if (t = 0)
    {
        return 1
    }
    else
    {
        return populatie2(t − 1) × 3
    }
}
```

Q6(b) Bereken de uitkomst t_i van de oproep `populatie2(i)`:

Q6(b)	$t_0 =$.....	$t_1 =$.....	$t_2 =$.....	$t_3 =$.....	$t_4 =$.....	$t_5 =$.....

Vraag 6 (Vervolg)

Lisa heeft net van Bart een reeks nieuwe meetpunten ontvangen en moet nu een algoritme schrijven dat overeenstemt met deze data. Hieronder vind je de reeks meetpunten van Bart en een onvolledig algoritme van Lisa. Voor onderstaand algoritme, **vink de rij van uitdrukkingen aan** die samen voor een resultaat zorgen dat overeenkomt met Barts metingen.

$t_0 = 0$	$t_1 = 1$	$t_2 = 4$	$t_3 = 9$	$t_4 = 16$	$t_5 = 25$

```
function populatie3(t)
{
    if ([...])                  // A
    {
        return [...]            // B
    }
    else
    {
        return [...]            // C
    }
}
```

Q6(c)	A	B	C
☐	`t = 0`	`0`	`population3(t-1) + t × t`
☐	`t ≤ 1`	`t`	`population3(t-1) × 2`
☐	`t = 0`	`0`	`population3(t-1) + ( 2 × t ) - 1`
☐	`t < 1`	`t`	`population3(t-1) × population3(t-1)`

Vraag 6 (Vervolg)

Hieronder vind je nog een onvolledig algoritme van Lisa. **Vul de ontbrekende delen in dit algoritme zelf aan**. Zorg er ook deze keer voor dat de resultaten van het algoritme overeenkomen met Bart's onderstaande metingen.

$t_0 = 0$	$t_1 = 1$	$t_2 = 1$	$t_3 = 2$	$t_4 = 3$	$t_5 = 5$	$t_6 = 8$

```
function populatie4(t)
{
  if (t < 2)
  {
    return [...]            // Q6(d)
  }
  else
  {
    return [...]            // Q6(e)
  }
}
```

Q6(d) **(een uitdrukking)**

. . . . . .

Q6(e) **(een uitdrukking)**

. . . . . .

Vraag 7 – Het rangeerstation (25 min)

Voor vrachttreinen is het belangrijk dat de wagons in de volgorde staan waarin ze afgeleverd moeten worden. Op die manier volstaat het immers dat de treinbestuurder op elk station de laatste wagons (met de ladingen bedoeld voor dat station) afkoppelt. Bij een verkeerde volgorde moeten er wagons in midden van de trein afgekoppeld worden, op een ander spoor gezet, enzovoort, wat veel tijd in beslag neemt.

Het beroep "wagonswapper" werd per toeval uitgevonden door een spoorman die een draaiende spoorwegbrug bediende. Zo'n brug over de rivier roteert rond een pilaar in het centrum van de rivier. Door de brug 90 graden om die as te draaien kunnen boten zowel links als rechts van de brugpilaar passeren. De eerste *wagonswapper* ontdekte dat de brug ook werkte wanneer hij er twee treinwagons op plaatste. Door de brug 180 graden te draaien wisselden de wagons van plaats ("to swap" in het Engels). De juiste wagonvolgorde kan dan verkregen worden door een reeks opeenvolgende *swaps*.

Nu bijna alle treinswappers op pensioen gaan wil de NMBS hun werk automatiseren. Als onderdeel van het te ontwikkelen computerprogramma moet er een routine geschreven worden die voor een gegeven treinsamenstelling beslist hoeveel swaps van naast elkaar geplaatste wagons er minimaal nodig zijn om de gewenste volgorde te bekomen. Jij bent aangesteld om die routine te schrijven. Je routine krijgt als invoer een geheel getal N die de lengte van de trein aanduidt en een array *wagons* van lengte N die de huidige wagonvolgorde van de trein voorstelt en een permutatie van de getallen 1 tot en met N bevat.

Een mogelijke instantie van *wagons* is bijvoorbeeld $[3, 2, 1]$. De wagons moeten herordend worden door te swappen zodat wagon 1 eerst komt te staan (d.w.z. *wagons*$[0] = 1$), wagon 2 als tweede komt te staan (d.w.z. *wagons*$[1] = 2$), enzovoort, met wagon N het laatst. Indien we dus beginnen met *wagons* $= [3, 2, 1]$ moeten we de array $[1, 2, 3]$ bekomen. We zouden dit kunnen doen door eerst *wagons*$[0]$ met *wagons*$[1]$ te swappen (dan bekomen we $[2, 3, 1]$); vervolgens *wagons*$[1]$ met *wagons*$[2]$ te swappen (dan bekomen we $[2, 1, 3]$) en ten slotte opnieuw *wagons*$[0]$ met *wagons*$[1]$ te swappen (dan bekomen we $[1, 2, 3]$)). In dit voorbeeld hebben we dus 3 swaps gebruikt om de juiste treinvolgorde te bekomen, en dat was voor de gegeven beginsituatie ook het minimaal aantal. Bovendien heb je om 3 wagons te ordenen nooit meer dan 3 swaps nodig (dat kan je gemakkelijk verifiëren). Dat betekent dat $[3, 2, 1]$ het *ergst mogelijke geval* is als je 3 wagons moet ordenen.

Beschouw eerst de volgende instanties van N en *wagons*. Geef voor elke instantie op **hoeveel swaps er minimaal nodig zijn** om een geordende trein te bekomen.

Q7(a)	$N = 4$	`wagons`$= [2, 4, 3, 1]$	.....
Q7(b)	$N = 4$	`wagons`$= [4, 3, 2, 1]$	.....
Q7(c)	$N = 5$	`wagons`$= [5, 4, 3, 2, 1]$	.....
Q7(d)	$N = 5$	`wagons`$= [4, 5, 1, 3, 2]$	.....

Kan je op basis van deze voorbeelden afleiden **hoeveel swaps je minimaal nodig hebt** *in het ergst mogelijke geval* om een trein van lengte N te ordenen?

Q7(e)	☐ N	☐ $2N$	☐ $\sqrt{N}$	☐ $\frac{N \times (N-1)}{2}$	☐ N^3

Vraag 7 (vervolg)

De volgende functie zet de wagons in de juiste volgorde en telt hierbij het aantal uitgevoerde swaps. Aan jou om deze af te werken!

```
Input:  wagons, een array met de wagonnummers: positieve gehele getallen
                van 0 tot en met N − 1, allemaal verschillend.
        N, De lengte van de array wagons
Output: het minimum aantal swaps dat nodig is om wagons te ordenen.

function tel_swaps(wagons, N)
{
    aantalSwaps ← 0
    for (k ← 0 to (N − 2) step 1)
    {
      for (i ← 0 to [...] step [...])        // Q7(f), Q7(g)
      {
        if (wagons[i] > wagons[i + 1])
        {
          swap(i, i + 1)                     //wissel de getallen wagons[i] en wagons[i + 1] om
          [...]                              //Q7(h)
        }
      }
    }
    return aantalSwaps
}
```

Q7(f) **(een uitdrukking)**

. . . . . .

Q7(g) **(een uitdrukking)**

. . . . . .

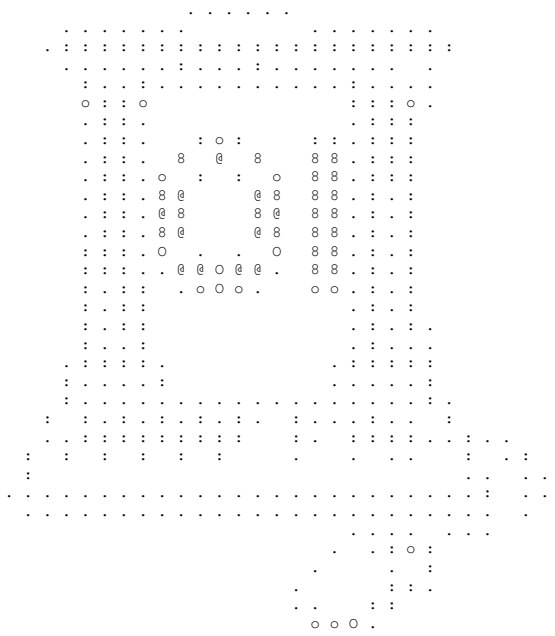
Q7(h) **(een instructie)**

. . . . . .

Vraag 8 – ASCII art (25 min)

ASCII-Art is een grafische representatie van een afbeelding waarbij pixels of groepen van pixels vervangen zijn door tekens (letters, cijfers of leestekens). De moeilijkheid van ASCII-Art zit in de keuze van het juiste teken om een gegeven pixelgroep te vervangen. Er bestaan heel ingewikkelde technieken om die keuze automatisch te maken.

In deze vraag stellen we een eenvoudige techniek voor: we gebruiken verschillende tekens om verschillende grijswaarden voor te stellen. Als voorbeeld zie je hieronder het logo van de Informatica Olympiade (oorspronkelijk een afbeelding van 120×160 pixels) omgezet naar een matrix van 30×40 gevuld met karakters. Om de omzetting te doen is een array van 7 tekens gegeven (die zie je naast de ASCII-tekening). Bij de omzetting zijn groepen van 16 pixels (4 x 4) omgezet door eerst hun gemiddelde grijswaarde (een getal van 0 (wit) tot 255 (zwart)) te bepalen en die dan om te zetten naar een teken.

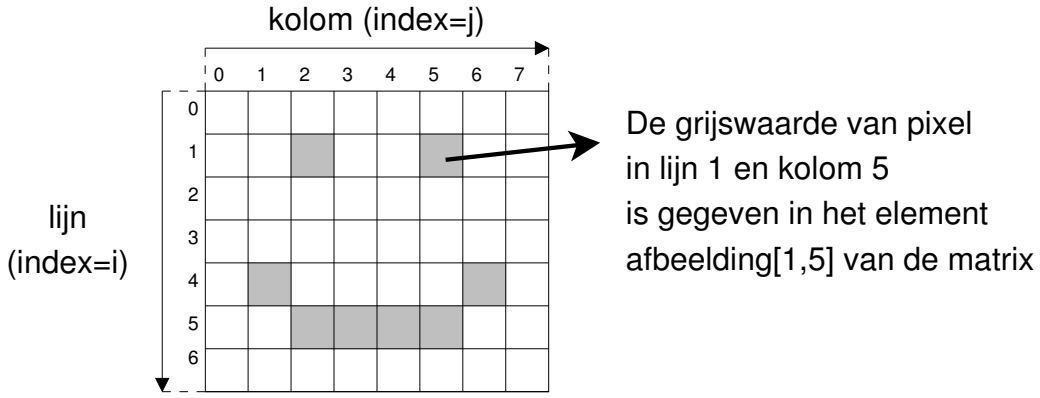


Grijswaarde van de originele afbeelding	Vervangend ASCII teken
$0 - 36$	(blanco)
$37 - 73$	.
$74 - 109$	:
$110 - 146$	o
$147 - 182$	O
$183 - 219$	8
$220 - 255$	@

Verderop vind je een algoritme om een afbeelding in grijswaarden naar *ASCII-Art* om te zetten. Een aantal delen van het algoritme zijn weggelaten: aan jou om het **algoritme te vervolledigen**.

Het algoritme krijgt als input: de originele afbeelding die gegeven is als een matrix van gehele getallen, een array met de tekens die gebruikt worden voor de omzetting, en een matrix waarin het resultaat (de tekens van de *ASCII-Art*-representatie) moet opgeslagen worden.

De originele afbeelding krijg je als een matrix `afbeelding` van grootte $m \times n$, waarbij m het aantal lijnen is, en n het aantal kolommen. Elk element `afbeelding[i,j]` van die matrix is een geheel getal in het interval $[0, 255]$ dat de grijswaarde voorstelt van de overeenkomstige pixel op lijn i ($0 \leq i < m$) en kolom j ($0 \leq j < n$) . De figuur hieronder, bijvoorbeeld, toont hoe een afbeelding van 8 pixels breed en 7 pixels hoog door een matrix van 7×8 voorgesteld wordt.



De ASCII-tekens voor de omzetting van de grijswaarden zijn gegeven in een array `tekens` met lengte k. De tekens zijn geordend in de array van kleinste grijswaarde (wit) naar grootste grijswaarde (zwart). Je moet overweg kunnen met arrays `tekens` van verschillende lengte (en inhoud). en De implementatie van het algoritme moet zich dus aanpassen aan de gegeven array `tekens`. Indien deze array bijvoorbeeld $k = 4$ karakters bevat, moet het karakter met index 0 (`tekens[0]`) gebruikt worden om de grijswaarden van 0 tot en met 63 voor te stellen; het karakter met index 1 om de grijswaarden van 64 tot en met 127 voor te stellen; enzovoort.

Het resultaat wordt geplaatst in een matrix `ascii` met grootte $p \times q$, zodat m een geheel veelvoud is van p en n een geheel veelvoud is van q. Elk element `ascii[k,l]` moet uiteindelijk een teken zijn dat de gemiddelde grijswaarde voorstelt van een rechthoekje van pixels in de originele afbeelding. De afspraken gemaakt voor de indexering van de kolommen en lijnen van `afbeelding` gelden ook voor `ascii`.

```
Input  : afbeelding, een niet-lege matrix van grootte m × n
                     gevuld met gehele getallen in het interval [0,255].
         tekens, een array van grootte k > 0 gevuld met tekens.
         ascii, een niet-lege matrix van grootte p × q met tekens,
                waarbij m en n gehele veelvouden zijn van respectievelijk p en q.

Output : De elementen van de matrix ascii hebben een waarde gekregen.
         Het algoritme geeft zelf geen waarde terug.

function zetom(afbeelding, m, n, tekens, k, ascii, p, q)
{
  for (i ← 0 to p − 1 step 1)
  {
    for (j ← 0 to q − 1 step 1)
    {
      s ← 0
      for (a ← 0 to [...] step 1)                          // Q8(a)
      {
        for (b ← 0 to [...] step 1)                        // Q8(b)
        {
          s ← s + afbeelding[ [...] , [...] ]              // Q8(c), Q8(d)
        }
      }
      ascii[i,j] ← tekens[ [...] ]                         // Q8(e)
    }
  }
}
```

Q8(a) **(een uitdrukking)**

. . . . . .

Q8(b) **(een uitdrukking)**

. . . . . .

Q8(c) **(een uitdrukking)**

. . . . . .

Q8(d) **(een uitdrukking)**

. . . . . .

Q8(e) **(een uitdrukking)**

. . . . . .

Overzicht pseudo-code

Gegevens worden opgeslagen in variabelen. Daarmee kan je dan wiskundige bewerkingen uitvoeren. Naast de vier klassieke operatoren ($+$, $-$, $\times$ en $/$), kan je ook $\%$ gebruiken (zie de eerste pagina voor de definitie hiervan). Een variabele heeft een naam die niet verandert. We kunnen waarde van een variabele wel veranderen met $\leftarrow$. In een variabele kunnen we gehele getallen, reële getallen of arrays opslaan, en ook booleaanse waarden : waar/juist (**`true`**) of onwaar/fout (**`false`**). Hieronder 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 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. Het resultaat wordt bewaard in de booleaanse (waar of onwaar) variabele $volwassen$.

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

Om in één klap meerdere elementen te manipuleren, gebruiken we een array. Een array is een opeenvolging of een lijst van waarden, en we krijgen toegang tot elk ervan via hun index. Het eerste element van de array is het element met index 0. Het volgende element heeft index $1 \ldots$ en het laatste heeft dus index $N - 1$ als de array N elementen bevat (N is de lengte van de array). Bijvoorbeeld, gegeven de array arr met als inhoud de getallen $[5, 9, 12]$. Dan is $arr[0]$ het eerste element van de array: 5, en $arr[2]$ het laatste element: 12.

Voor het herhalen van code gebruiken we een **`for`**-lus (definitie op de eerste pagina). 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, bevindt de som zich in de variabele sum.

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

We kunnen ook herhalen d.m.v. een **`while`**-lus, die code herhaalt zolang er aan een bepaalde voorwaarde is voldaan. In het volgende voorbeeld delen we een positief geheel getal N door 2, het resultaat daarvan door 3, het resultaat daarvan door $4 \ldots$ totdat het getal nog maar uit 1 decimaal cijfer bestaat (d.w.z. het is < 10).

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