be-OI 2011 **Halve Finale** 16 februari 2011	**Invullen in HOOFDLETTERS en LEESBAAR aub** VOORNAAM : ..... NAAM : ..... SCHOOL :	**Gereserveerd**

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

Dit is de vragenlijst van de halve finale van de Belgische Olympiades in de Informatica voor de **de categorie hoger onderwijs**. Ze bevat 9 vragen waarvoor je **maximum 3u** de tijd krijgt. 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 zeker verder op hetzelfde blad. Anders kunnen we je antwoord niet verbeteren.

2. Je mag alleen iets om te schrijven bij je hebben. Rekentoestel, GSM, ... zijn **verboden**.

3. Schrijf je antwoorden met blauwe of zwarte **pen of balpen**. Laat geen antwoorden staan in potlood. Als je kladbladen wilt, vraag ze dan aan een toezichthouder.

4. Je moet op de vragen 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 vanaf 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 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 hoger onderwijs

Vraag 1 – Opwarming (10 min)

Voor elk van de drie deelvragen levert een goed antwoord 1 punt op, geen antwoord geeft 0 punten en een fout antwoord -0.5 punten, dus een half strafpunt. Voor de meerkeuzevragen is slechts **één antwoord** het goede, kruis dit goede antwoord aan. Als je je antwoord wilt wijzigen, kleur dan het foutieve vakje helemaal zwart en kruis het andere vakje aan.

Q1(a) Stel n een strikt positief geheel getal en gegeven volgend algoritme:

```
count ← 0
for (i ← 0 to n − 1 step 1)
{
    for (j ← 0 to n − 1 step 1)
    {
        if (i < j) { count ← count − 1 }
        else { count ← count + 1 }
    }
}
```

Welke waarde heeft de variabele `count` na het uitvoeren van het algoritme?

Q1(a)	☐ $n(n-1)$	☐ n^2	☐ n	☐ 0

Q1(b) Gegeven volgend algoritme:

```
i ← 0
while (i ≤ 15)
{
    i ← 2 ∗ i + 1
}
```

Welke waarde heeft de variabele i na het uitvoeren van het algoritme?

Q1(b) **(een waarde)**

. . . . . .

Q1(c) Welk van volgende uitdrukkingen is equivalent met "*a en b zijn oneven*":

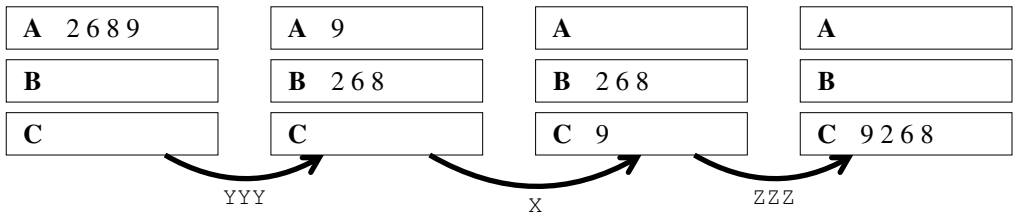
☐	`not (a % 2 = 0 or b % 2 = 0)`
☐	`a % 2 = 0 or b % 2 = 0`
☐	`a % 2 = 0 and b % 2 = 0`
☐	`not (a % 2 = 0 and b % 2 = 0)`

Vraag 2 – Kaartspel (10 min)

Je staat tegenover drie tafels die we A, B en C zullen noemen. Op tafel A bevindt zich een rij kaarten waarop telkens één cijfer staat zodat er een getal wordt gevormd. Je doel is om de kaarten naar tafel C te verplaatsen en zo het grootst mogelijke getal te maken. De **enige actie** die je mag uitvoeren is **de linkerkaart van een tafel** wegnemen en die aan de **rechterkant van een andere tafel** toevoegen. Eens op tafel C kan een kaart niet meer terug. Er zijn dus drie mogelijkheden:

- `X`: Neem de linkerkaart op A en plaats deze aan de rechterkant op C;
- `Y`: Neem de linkerkaart op A en plaats deze aan de rechterkant op B;
- `Z`: Neem de linkerkaart op B en plaats deze aan de rechterkant op C.

Neem bijvoorbeeld de beginsituatie `2689`. De figuur hieronder beschrijft de acties `YYYXZZZ` die het resultaat `9268` opleveren op tafel C. Als je denkt dat dit het grootst mogelijke getal is dat we op deze manier kunnen bekomen moet je `268` als antwoord geven in het bijbehorende vakje (dus de laatste drie cijfers).



Vind voor de volgende drie beginsituaties (d.w.z. deze rij kaarten ligt op tafel A, de andere tafels zijn leeg) het grootst mogelijke getal dat we kunnen bekomen en schrijf in de antwoordruimte de drie laatste cijfers van dit getal.

Q2(a) `3 5 9 7 6 2`

Q2(a) **(drie cijfers)**

. . . . . .

Q2(b) `9 1 2 8 4 5 7`

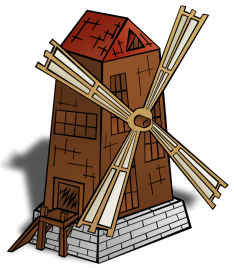
Q2(b) **(drie cijfers)**

. . . . . .

Q2(c) `8 6 4 1 9 7 5 3`

Q2(c) **(drie cijfers)**

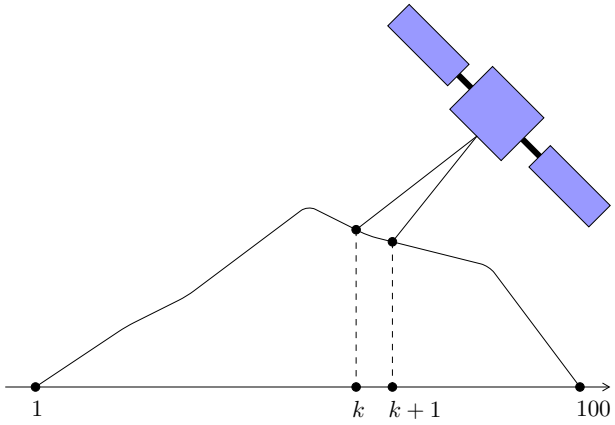
. . . . . .



Vraag 3 – Leve de wind... (15 min)

Het Departement Leefmilieu, Natuur en Energie doet een beroep op jou om de optimale plaatsing van haar nieuwe windmolenpark te bepalen. Het park moet op een heuveltop geplaatst worden, jij moet bepalen waar dat punt zich bevindt. Je weet dat de heuvel uit twee hellingen zonder platte stukken bestaat (dus van links naar rechts is de heuvel strikt stijgend tot de top wordt bereikt, om vervolgens strikt dalend te zijn, zonder dat de heuvel ooit ergens plat wordt).

Om je te helpen met deze opdracht heeft het Departement je een satelliet gegeven die in staat is om de hellingsgraad van de heuvel te bepalen op een bepaalde horizontale positie. De satelliet kan 100 verschillende punten meten langs de horizontale as. Je kan de satelliet een punt k geven (waarbij $1 \leq k < 100$) en de satelliet geeft dan als antwoord of de heuvel in het punt k hoger of lager is dan in het punt $k+1$. Als $hoogte(k) < hoogte(k+1)$ geeft de satelliet een strikt positief antwoord, in het andere geval een strikt negatief. In de situatie op de tekening hiernaast geeft de satelliet dus een strikt negatieve waarde aangezien de heuvel in het punt k hoger is dan in het punt $k+1$.



Maar let op, je moet de satelliet zeer spaarzaam gebruiken! Elke meting met de satelliet kost € 10 000 en je hebt een beperkt budget van € 150 000 gekregen van het Departement. Jouw algoritme moet er dus op toezien dat de satelliet niet meer dan 15 keer wordt gebruikt. Een programmeur van het Departement heeft een eerste versie geschreven maar hij was niet in staat om bepaalde stukken af te werken. Het volgende algoritme moet dus de horizontale positie van de top van de heuvel bepalen, aan jou om het af te werken.

```
a ← 0
b ← 100
while ([...])                                          // Q3(a)
{
    positie ← [...]                                    // Q3(b)
    if (satelliet_opvraging (positie) > 0)
    {
        a ← positie
    }
    else
    {
        b ← positie
    }
}
return a
```

Q3(a) **(een voorwaarde)**

. . . . . .

Q3(b) **(een uitdrukking)**

. . . . . .

Vraag 4 – Ceci n'est pas un sudoku (Dit is geen sudoku) (25 min)

Je neef Kris is op bezoek geweest en hij bracht een spelletje mee in de vorm van een rooster waarbij elke vakje een cijfer bevat. Je moet vertrekken in de linkerbovenhoek van het rooster en uitkomen in de rechterbenedenhoek. Je kan je alleen verplaatsen naar rechts of naar beneden.

Je begint met een score van 0. Bij elke verplaatsing wordt je huidige score door 2 gedeeld (afgerond naar beneden indien nodig), vervolgens wordt de waarde van het vakje waarnaar je verspringt aan je score toegevoegd. Het doel van het spel is de rechterbenedenhoek te bereiken met de laagst mogelijke score.

Gegeven dit rooster:

0	3	9	6
1	4	4	5
8	2	5	4
1	8	5	9

De kleinst mogelijke score op dit rooster is 12, je bekomt dit door de volgende sprongen te maken: B, R, B, R, R, B (met B voor beneden en R voor rechts).

Wat is de kleinst mogelijke score voor elk van de volgende roosters:

0	2	9	6
1	1	9	1
3	2	1	2
1	2	3	3

Q4(a)

9	2	2	2	2
3	9	2	2	2
3	3	9	3	3
3	3	2	9	3
3	3	2	0	9

Q4(b)

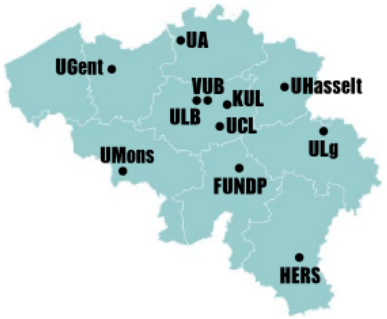
0	5	9	6	4
4	6	9	0	1
0	9	2	8	8
0	0	4	9	3
0	4	6	4	9

Q4(c)

0	3	9	6	4	2
0	3	9	6	4	2
0	6	9	0	1	1
0	9	2	9	0	1
0	0	4	4	5	6
0	4	6	8	0	9

Q4(d)

Vraag 5 – Waar organiseren we de finale? (20 min)



De halve finales van de Belgische Informatica Olympiade vinden plaatsen in meerdere regionale centra. Het comité van de Olympiades heeft beslist dat de locatie van de finale elk jaar gekozen moet worden onder de regionale centra die zich kandidaat hebben gesteld. Indien meerdere centra zich kandidaat stellen, wordt de volgende regel toegepast. De eerste keer neemt men de meest centraal gelegen kandidaat, d.w.z. diegene die gemiddeld gezien het minst aantal kilometers van de andere kandidaten gelegen is. De volgende jaren vindt de finale plaats in het regionaal centrum dat het meest centraal gelegen is, maar nog nooit eerder de finale heeft georganiseerd.

De volgende tabel geeft de afstanden weer in kilometers tussen enkele van de regionale centra. De afstand tussen Gent en Louvain-la-Neuve (LLN) bijvoorbeeld bedraagt 75 km.

	Namen	LLN	Antwerpen	Leuven	Gent	Luik
Namen	0	38	107	47	104	66
LLN	38	0	73	23	75	84
Antwerpen	107	73	0	43	53	133
Leuven	47	23	43	0	71	67
Gent	104	75	53	71	0	138
Luikt	66	84	133	67	138	0

Q5(a-b) Welke zijn de 3^e en 4^e centra die de finale mogen organiseren, als we aannemen dat de zes centra in de tabel hierboven de kandidaten zijn, en dat geen enkele van hen de finale al heeft georganiseerd?

Q5(a) : 3^e centrum

Q5(b) : 4^e centrum

Om dit probleem met behulp van informatica op te lossen, zouden we graag een algoritme hebben dat de verschillende centra rangschikt, gesorteerd op het gemiddeld aantal kilometers dat ze gelegen zijn van de andere centra. Hier vind je een versie van het algoritme - maar je moet het nog aanvullen!

In dit algoritme nummeren we de N regionale centra van 0 tot $N-1$. Het algoritme gebruikt een tweedimensionale ($N \times N$) matrix, genoemd $afstanden$, zo gedefinieerd dat voor elk paar centra i ($0 \leq i < N$) en j ($0 \leq j < N$), $afstanden[i][j]$ de afstand voorstelt tussen centrum i en centrum j.

Het doel is een tabel $ranking$ ($N \times 2$) te maken, waarbij elke rij bevat: het nummer van een regionaal centrum, én het gemiddeld aantal kilometers van dit centrum tot de **andere** centra. Dit algoritme maakt gebruik van een ander algoritme $invoegen$ (niet gegeven) dat toelaat een koppel (i, m) in te voegen in de matrix $ranking$ op de positie k ($0 \leq k < N$). Na het uitvoeren van $invoegen(ranking, k, i, m)$ geldt: $ranking[k, 0]$ bevat i, $ranking[k][1]$ bevat m en alle elementen die zich bevonden in $ranking$ vanaf k voordat $invoegen$ werd uitgevoerd, zijn nu 1 positie verplaatst (hun index is met 1 verhoogd).

Input : $afstanden$, een symmetrische vierkante matrix, gevuld met positieve waarden die de afstanden tussen de verschillende regionale centra voorstellen. De matrix heeft de dimensies $N \times N$.
N, een geheel getal > 1, dat het aantal centra voorstelt.
$ranking$, een matrix van gehele getallen met dimensies $N \times 2$, geïnitialiseerd als volgt: $ranking[i][1] = -1$ voor alle i.

Output : Voor elke i, zullen $ranking[i][0]$ en $ranking[i][1]$ deze gegevens bevatten:
$ranking[i][0]$: het nummer van een regionaal centrum
$ranking[i][1]$: gemiddeld aantal km die het scheidt van de andere centra.
De elementen in de tweede kolom van $ranking$ moet gerangschikt zijn in dalende volgorde, d.w.z. voor elke rij k in $ranking$ $(0 \leq k < N-1)$ geldt: $ranking[k][1] \geq ranking[k+1][1]$.
Elk centrum mag slechts eenmaal voorkomen.
Het algoritme geeft geen waarde terug.

```
for (i ← 0 to N − 1 step 1)
{
    s ← 0
    for (j ← 0 to N − 1 step 1)
    {
        s ← [...]                                          // Q5(c)
    }
    m ← [...]                                              // Q5(d)
    k ← 0
    while ([...])                                          // Q5(e)
    {
        k ← k + 1
    }
    invoegen (ranking, k, i, m)
}
```

Vervolledig het algoritme zodat de berekeningen gebeuren zoals het hoort.

Q5(c) **(een uitdrukking)**

. . . . . .

Q5(d) **(een uitdrukking)**

. . . . . .

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

. . . . . .

Vraag 6 – Het spel van de krijtjes (10 min)


Mike Plante

Je vriend Sebastiaan heeft je onlangs een nieuw spel aangeleerd. Dertig krijtjes worden neergelegd op een tafel. De twee spelers nemen om de beurt één, twee of drie krijtjes van de tafel weg. De speler die het laatste krijtje neemt **verliest** het spel. Sebastiaan beweert dat hij altijd zal winnen als hij het spel mag beginnen. Het volgend algoritme geeft de strategie van Sebastiaan weer, telkens hij aan de beurt is.

```
Input  : remaining, een positief geheel getal ≤ 30, zijnde het aantal krijtjes dat nog
           op de tafel ligt.
         last, het aantal krijtjes dat de tegenstander tijdens de vorige beurt van de
           tafel nam, dus 1, 2 of 3  (of 0 bij de eerste beurt van het spel).
         Sebastiaan is zeker om te winnen als hij slim speelt.
Output : het aantal krijtjes dat Sebastiaan deze beurt van de tafel moet
           wegnemen: 1, 2 of 3.
```

Welke **wiskundige uitdrukking** heb je nodig (bestaande uit slechts 1 instructie) om het optimale aantal krijtjes voor Sebastiaan te berekenen ?

```
return [...]                                                    // Q6
```

Q6 **(een uitdrukking)**

. . . . . .

Als je de oplossing niet vindt, ga dan naar de volgende bladzijde voor een alternatieve vraag.

Vraag 6 (alternatief)

Als je er niet in slaagt het door Sebastiaan gebruikte algoritme te vinden, **en enkel in dat geval**, kan je proberen het algoritme te vinden dat zijn zus Fanny gebruikt. Het is even efficiënt maar een beetje langer dan dat van Sebastiaan. Je dient volgend algoritme aan te vullen met **een voorwaarde en twee wiskundige uitdrukkingen**. [1]

```
if ([...])                                                    // Q6alt(a)
{
    return [...]                                              // Q6alt(b)
}
else
{
    return [...]                                              // Q6alt(c)
}
```

Q6 alternatief (a) **(een voorwaarde)**

. . . . . .

Q6 alternatief (b) **(een uitdrukking)**

. . . . . .

Q6 alternatief (c) **(een uitdrukking)**

. . . . . .

[1] Als je antwoord op vraag **Q6** correct is, zal het antwoord op de vraag **Q6 alternatief** genegeerd worden.

Vraag 7 - Spreekt u *Zwendeltaal* ? (20 min)

Claire en Bart willen met elkaar praten zonder dat andere mensen hun gesprek kunnen verstaan. Daarom dachten ze aan **Verlan**, waarbij de lettergrepen van een woord omgekeerd worden. In Verlan zegt men "*chelou*" in plaats van louche. Deze taal bleek vrij gemakkelijk te ontcijferen, daarom besloten ze om de Zwendeltaal, uitgevonden door Zwendel, te gebruiken. In de Zwendeltaal wordt de volgorde van alle letters van een woord omgekeerd. In plaats van louche zegt men dan "*ehcuol*".

Het algoritme dat gebruikt wordt om zinnen uit de "Zwendeltaal" te vertalen en begrijpen bestaat uit twee delen. Het eerste algoritme (Q7(a)) laat toe om een deel van een array van karakters om te keren. Bijvoorbeeld, als de array met karakters `[l, e, v, e, , z, w, e, n, d, e, l, .]` is en we voeren het algoritme uit met $start = 5$ en $einde = 11$, dan wordt de array aangepast naar `[l, e, v, e, , l, e, d, n, e, w, z, .]`.

```
Input: phrase, een array van karakters
       start en end, positieve getallen (start ≤ end), indices van de eerste en laatste
         letter van een woord.
Output: de volgorde van letters beginnend bij index start en eindigend bij index end
        is omgekeerd

function invert(phrase, start, end)
{
    [...]                                                          // Q7(a)
}
```

Schrijf het volledige algoritme.

Q7(a) **(een algoritme)**

Vraag 7 (vervolg)

Het algoritme dat het algemene probleem oplost zal een zin overlopen. Elke keer dat een woord herkend wordt, doet het beroep op het voorgaande algoritme om dat woord om te keren.

```
Input: phrase, een array met karakters, bevat enkel de kleine letters uit het alfabet,
           het punt en spaties. De woorden van de zin zijn gescheiden door een enkele
           spatie. Er komen geen spaties voor het eerste en na het laatste woord
           van een zin. De zin eindigt met een punt.
       n, postief geheel getal, zijde het aantal karakters in phrase

Output: phrase is vertaald in de "Zwendeltaal". Het algoritme geeft geen waarde terug.

wordstart ← 0

for (i ← 0 to n − 1 step 1)
{
    if ([...])                                                          // Q7(b)
    {
        invert (phrase, wordstart, [...])                               // Q7(c)
        [...]                                                           // Q7(d)
    }
}
```

Welke instructies moet je invullen in het algoritme zodat het gewenste resultaat bereikt wordt?

Q7(b) **(een voorwaarde)**

.....

Q7(c) **(een uitdrukking)**

.....

Q7(d) **(een instructie)**

.....

Vraag 8 – Numerieke anagrammen (20 min)

In dit probleem beschouwen we *numerieke anagrammen*. Twee getallen zijn een *anagram* als ze gevormd kunnen worden met exact dezelfde cijfers, in dezelfde of een andere volgorde. Bijvoorbeeld, 121, 211 en 112 zijn anagrammen, maar 411, 144 en 511 zijn er geen.

Aan jou om een algoritme te schrijven dat bepaalt of twee getallen, a en b, voorgesteld door een array van cijfers, anagrammen zijn. Het is toegestaan om nieuwe functies te declareren, zolang je niet buiten het antwoordkader schrijft. Je kan ook nieuwe arrays van getallen declareren (geïnitialiseerd op nul) met behulp de volgende notatie:
$newarr \leftarrow$ new_array($size$).

```
Input: a, b, twee arrays met dezelfde lengte n, waarvan elk element een cijfer bevat.
Output: true als a en b anagrammen zijn, anders false
              De arrays a en b mogen aangepast worden.
```

Hulplijn

Om je op weg te helpen kan je gebruik maken van de volgende functie (niet verplicht!). Deze functie laat toe om een array van getallen te sorteren in oplopende volgorde. De functie doorloopt de array en zoekt bij elke iteratie het volgende getal dat daar geplaatst moet worden om de array te sorteren. Bijvoorbeeld, stel $t = [1, 8, 3, 5, 2]$. Na het uitvoeren van de functie $sort(t, 5)$, is $t = [1, 2, 3, 5, 8]$.

```
Input: arr, een (niet-lege) array van postieve gehele getallen.
       n > 0, de lengte van de array
Output: De elementen van de array arr zijn gesorteerd in oplopende volgorde.
        De functie geeft geen waarde terug.

function sort (arr, n)
{
    for (i ← 0 to n − 1 step 1)
    {
        max ← i
        for (j ← i + 1 to n − 1 step 1)
        {
            if (arr[j] > arr[i])
            {
                max ← j
            }
        }

        temp ← arr[i]
        arr[i] ← arr[j]
        arr[j] ← temp
    }
}
```

Q8

Bonus

Je kan een bonus verdienen voor deze vraag als je algoritme efficiënt is. Hiervoor moet je een algoritme voorstellen waarbij het aantal oproepen naar elementen van de arrays (d.w.z. het aantal keer dat $arr[index]$ gepasseerd wordt tijdens het uitvoeren) minder is dan $10 \times n$ voor alle $n > 10$. Als je gebruik maakt van bovenstaande functie $sort$, moet je de oproepen die daarin gebeuren natuurlijk meetellen.

Vraag 9 – Mijn planning voorbereiden (25 min)

We wensen een programma te maken dat ons toelaat om een agenda bij te houden. De informatie die in de agenda opgeslagen wordt bestaat uit een reeks evenementen, gesorteerd volgens tijd. Elk evenement wordt gekarakteriseerd door drie stukken informatie: het startuur, het einduur en een tekst die het evenement beschrijft. We veronderstellen dat de evenementen elkaar nooit overlappen, maar er kunnen natuurlijk wel gaten zijn tussen twee opeenvolgende evenementen. Een evenement kan ook op één "tijdstip"plaatsvinden, d.w.z. een samenvallend start- en einduur hebben.

1. **Van 10u tot 11u** : Reünie
2. **Van 12u tot 13u** : Lunch
3. **Van 13u tot 15u** : Boodschappen doen
4. **Om 17u** : De kinderen van school afhalen

Hierbij volgen evenementen 2 en 3 elkaar onmiddellijk op, en evenement 4 heeft een samenvallend start- en einduur.

Concreet kunnen we de agenda voorstellen met behulp van vier arrays `naam`, `lengte`, `interval` en `volgende`, met twee bijhorende gehele getallen `eerst` en `start`. Elk evenement zal voorgesteld worden door eenzelfde index i voor de arrays `naam`, `lengte`, `interval`. Dit gebeurt als volgt: `naam[i]` beschrijft de naam van het evenement, `interval[i]` geeft aan hoeveel uur er tussen het eind van het vorige evenement en het begin van dit evenement zit (0 voor het eerste evenement), en `lengte[i]` geeft aan hoeveel uur het evenement duurt (0 voor evenementen met samenvallend start- en einduur). Voor het evenement dat overeenkomt met index i, geeft de array `volgende` de index van het volgende evenement, of geeft -1 terug als het huidige evenement het laatste was. De variabele `eerst` geeft de index van het eerste evenement en de variabele `start` geeft het startuur van dit eerste evenement. Let op dat deze variabelen gelijk zijn aan -1 als de agenda geen enkel evenement bevat.

Een voorbeeld van bovenstaande instructies:

`eerst`	=	1				
`start`	=	10				
`interval`	=	1	0		2	0	
`lengte`	=	1	1		0	2	
`volgende`	=	4	0		-1	3	
`naam`	=	Lunch	Reunie		Kinderen	Boodschappen	

Als de variabele `eerst` gelijk is aan 1, dan weten we dat de waarden `naam[1]`, `lengte[1]` en `interval[1]` het eerste evenement beschrijven. Dit evenement start om 10u, gegeven door variabele `start`. Het gaat hier dus om de reünie, die 1 uur duurt en eindig om 11u. De waarde van `volgende[1]` is gelijk aan 0, dus het tweede evenement wordt beschreven op index 0. Vermits de waarde van `interval[1]` gelijk is aan 1, start het volgende evenement (genaamd Lunch) om 11u + 1u = 12u. Dit evenement duurt 1u en eindigt dus om 13u. Enzovoort...

Het volgende algoritme manipuleert een agenda beschreven in arrays, zoals hierboven beschreven. Dit algoritme voegt een evenement toe waarvan de naam (parameter n) en het start- en einduur (parameters s en e) zijn voorzien. We geven ook mee waar de gegevens in de arrays opgeslagen moeten worden (welke index). Het algoritme veronderstelt dat het toe te voegen evenement niet tegenstrijdig is met enig ander evenement in de agenda. Vervolledig nu het algoritme.

```
Input: n, naam van het evenement
       s en e, start- en einduur van het evenement (s ≤ e)
       plaats, vrije index in de tabel. Hier zal het evenement opgeslagen worden.
Output: bijgewerkte agenda, het algoritme geeft geen waarde terug

naam[plaats] ← n
lengte[plaats] ← [...]                                              // Q9(a)
if ([...])                                                          // Q9(b)
{
    if (eerst ≠ −1)
    {
        interval[eerst] ← start − e
    }
    volgende[plaats] ← eerst
    interval[plaats] ← 0
    start ← plaats
    eerst ← s
}
else
{
    i ← eerst
    start_i ← start
    einde_voor_i ← 0
    while (start_i ≤ s and [...])                                   // Q9(c)
    {
        voor_i ← i
        einde_voor_i ← start_i + lengte[i]
        i ← volgende[i]
        if (i ≠ −1)
        {
            start_i ← einde_voor_i + interval[i]
        }
    }
    volgende[voor_i] ← plaats
    [...] ← i                                                       // Q9(d)
    interval[plaats] ← s − einde_voor_i
    if (i ≠ −1)
    {
        interval[i] ← [...]                                         // Q9(e)
    }
}
```

Wat moeten we toevoegen aan het algoritme om het gewenste resultaat te bereiken?

Q9(a) **(een uitdrukking)**

. . . . . .

Q9(b) **(een voorwaarde)**

. . . . . .

Q9(c) **(een voorwaarde)**

. . . . . .

Q9(d) **(een uitdrukking)**

. . . . . .

Q9(e) **(een uitdrukking)**

. . . . . .

Overzicht pseudo-code

In pseudo-code slaan we gegevens op in variabelen. Daarmee kan je dan wiskundige bewerkingen uitvoeren. Naast de vier klassieke operatoren ($+, -, \times$ et $/$), kan je ook $\%$ gebruiken (zie de eerste pagina voor de definitie hiervan). We kunnen waarde van een variabele 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 ← 1992
leeftijd ← 2011 − 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 reeks of een lijst van waarden, en we krijgen toegang tot elk ervan via hun index. Het eerste element van de reeks is het element met index 0. Het volgende element heeft index 1 ... en het laatste heeft dus index $N - 1$ als de reeks N elementen bevat (N is de lengte van de array). Bijvoorbeeld, gegeven de array tab met als inhoud de reeks getallen $[5, 9, 12]$. Dan is $tab[0]$ het eerste element van de reeks (5) en $tab[2]$ 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 voorwaarde is voldaan. In het volgende voorbeeld delen we een positief geheel getal N door 2, dan door 3, door 4 ... totdat het getal nog maar uit 1 cijfer bestaat (d.w.z. het is < 10).

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