German University in Cairo
Media Engineering and Technology
Assoc. Prof. Dr. Georg Jung

November 20, 2010

CSEN 102: Introduction to Computer Science
Winter term 2010
Midterm Exam

Bar Code

Instructions: Read carefully before proceeding.

1) Duration of the exam: 2 hours (120 minutes).
2) (Non-programmable) Calculators are allowed.
3) No books or other aids are permitted for this test.
4) This exam booklet contains 10 pages, including this one. Three extra sheets of scratch paper are attached and have to be kept attached. **Note that if one or more pages are missing, you will lose their points. Thus, you must check that your exam booklet is complete**.
5) Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem or on the four extra sheets and make an arrow indicating that. **Scratch sheets will not be graded unless an arrow on the problem page indicates that the solution extends to the scratch sheets**.
6) When you are told that time is up, stop working on the test.

Good Luck!

Don't write anything below ;-)

Exercise	1	2	3	4	5	6	$\sum$
Possible Marks	6	9	10	12	27	16	80
Final Marks							

Exercise 1 Sequential algorithms (6 Marks)

Cook in a hurry

You want to cook some pasta, but you are short of time. To be sure you can finish cooking before your next appointment, you need to know how long it takes for the water to boil.

At its highest setting, your stove needs two minutes per liter to reach the boiling point. You use a cylindric pot.

Write a program that, given the diameter of the pot and the hight of the water in it, calculates the the time needed for the water to boil.

Solution:

- Getting the right inputs: 1 Mark
- Calculating the right volume in liters: 3 Marks
- Calculating the time needed: 1 Mark
- Giving the right output: 1 Mark

```
1  get diameter, height
2
3  // calculate the volume
4  set radius to diameter / 2
5  set volume to 3.14 * radius * radius * heigt
6
7  // We assume that diameter as well as heigth are given in centimeters
8  set liters to volume / 1000
9
10 // The stove needs two minutes per liter
11 set time to liters * 2
12
13 print "Your␣stove␣needs␣"
14 print time
15 print "␣minutes␣to␣bring␣the␣water␣to␣boil."
```

Background. In the business world as well as in chemistry or engineering you often need to compare different vendors. Ideally, you programm your calculator to quickly infer the value that you are interested in from the values given by the vendors.

Exercise 2 Sequential operations (6 Marks)
Air conditioning

You shop for air conditioners. The capacity of the models you find is given in *cubic meters per minute* (*i. e.*, $[\frac{\mathrm{m}^3}{\mathrm{min}}]$). You know the *height*, *length*, and *width* of your room. Further, you want to make sure, that your room gets cool within no more than 10 minutes.

Write an algorithm that calculates the minimum capacity of an air conditioner in $[\frac{\mathrm{m}^3}{\mathrm{min}}]$ for your room.

Solution:

- Getting the right inputs: 1 Mark
- Calculating the volume: 2 Marks
- Calculating the capacity: 2 Marks
- Giving the right output: 1 Mark

```
1 get height, length, width
2
3 set volume = height * length * width
4 set minCapacity = volume / 10
5
6 print "To␣cool␣down␣your␣room␣in␣10␣minutes␣or␣less,␣"
7 print "the␣capacity␣should␣be␣at␣least␣"
8 print minCapacity
9 print "␣cubic␣meters␣per␣minute."
```

Background. In the business world as well as in chemistry or engineering you often need to compare different vendors. Ideally, you programm your calculator to quickly infer the value that you are interested in from the values given by the vendors.

Exercise 3 Conditional operations (9 Marks)
Road code

You are driving your car through Germany. To keep track of the regulations, you use a program. These are the rules:

- If you are in a **pedestrian area**, you are probably confused
- If you are in a traffic-reduced **living area**, the speed limit is 30kph
- Within **city limits**, the speed limit is 50kph
- On **federal roads**, the speed limit is 100kph
- On highways ("**Autobahn**") there is no general speed limit

You also need to check for any local speed limit.

Given an `area`, a `local-limit`, and a `speed`, please write a program that outputs either: "`Hurry up`!", "`Slow down`!", or "`You are confused`!"

Solution:

- Getting the right inputs: 1 Mark
- Handling the special case for pedestrian areas: 1 Mark
- Setting the general limit correctly: 2 Marks
- Handling the special case for Autobahn: 2 Marks
- Handling the local limit: 1 Marks
- Giving the right output in all *specified* cases: 2 Marks

For the solution we assume that the local limit is ifinite whenever it is not lower than the general limit.

```
1  get area, local-limit, speed
2
3  if (area = pedestrian area) then   // special case: pedestrian area
4    print "You␣are␣confused"
5  else
6    if (area = living area) then     // fix general limit for each area
7      set limit to 30
8    endif
9    if (area = city limits) then
10     set limit to 50
11   endif
12   if (area = federal road) then
13     set limit to 100
14   endif
15   if (area = Autobahn) then        // special case: Autobahn (no general limit)
16     if (speed > local-limit) then
17       print "Slow␣down!"
18     else
19       if (speed < local-limit) then
20         print "Hurry␣up!"
21       endif
22     endif
23   endif
24   if (local-limit < limit) then    // is the local limit smaller than the general?
25     set limit to local-limit
```

```
26    endif
27    if (speed < limit) then
28      print "Hurry␣up!"
29    endif
30    if (speed > limit) then
31      print "Slow␣down!"
32    endif
33  endif
```

Note that the algorithm gives no output whenever the speed is exactly as required. On the Autobahn, it always tells you to speed up unless there is a local limit. However, any solution that successfully distinguishes the cases should receive full marks regardless of the actual output.

Background. Decisions like the one above generally appear in periodic, event-based, system design. An algorithm is periodically called to assess a certain situation (here the situation is a given speed, in a given area, with locally specific side-rules) for any necessary action. The example code is not unlike to what you would find in an auto-pilot event handler.

Exercise 4 Conditional operations (15 Marks)

Tea time

You want to make tea. There are four different blends: *Earl Grey*, *Assam*, *Gunpowder*, *Darjeeling*, and *English Breakfast*. Further, there are two kinds of fermentation: *Black* or *green* tea. Each blend comes as black and as green tea, except for Earl Grey (which is suffused with Bergamotte oil and hence does not taste well as green tea), which only comes in the black variety, and *Gunpowder*, which only exists as green tea.

Green tea should usually steep for 2'30" (read: two minutes and thirty seconds), unless it is Gunpowder, which can steep for any time. For black tea it depends on the blend: Earl Grey should steep for 3'30", Assam and Darjeeling for 4', and English Breakfast for 5'.

If you like your tea with sugar, add 30" to the steeping time, unless you use Darjeeling, which would become too bitter.

Write a program that, given `blend`, `fermentation`, and `sugar` (`yes` or `no`), calculates the perfect steeping time. Have the program check for corrupted input.

Solution:

- Getting the right inputs: 1 Mark
- Handling the special case for Gunpowder: 1 Mark
- Making use of a variable such as "`steep`" to avoid repeating the program for the sugar case: 1 Mark
- Handling the Green fermentation (with and without sugar, and with Gunpowder being handled): 2 Marks
- Setting the correct steep for the black blends: 4 Marks
- Using "**and**" for Assam and Darjeeling: 1 Mark
- Handling the sugar-case with correct conditional: 4 Marks
- Checking for some corrupt input (whichever): 1 Mark

```
1  get blend, fermentation, sugar
2
3  if (blend = Gunpowder) then
4    print "Steep␣for␣any␣time"
5  else
6    if (fermentation = Green) then
7      set steep to 2.5              // also accepted: 2'30'', 2.30...
8    else
9      if (blend = Earl Grey) then
10       set steep to 3.5
11     else
12       if (blend = Assam or blend = Darjeeling) then
13         set steep to 4
14       else
15         if (blend = English Breakfast) then
16           set steep to 5
17         else
18           print "Unknown␣blend!"
19         endif
20       endif
21     endif
22   endif
23   if (sugar = yes and not (blend = Darjeeling)) then
24     set steep to steep + 0.5
25   endif
26   print "Please␣steep␣for␣"
```

```
27    print steep
28    print "␣minutes."
29  endif
```

The example solution represents a compact yet readable solution. Any fully correct solution should receive full marks even if it fails to be short and concise or avoids combined conditionals. Also, avoiding some incorrect input is a Mark.

Background. Often, the biggest challenge in designing a nested conditional is to structure the conditional in a way that a minimum of cases has to be distinguished in each level of nesting. A full, naïve, distinction can be up to three times as long.

Exercise 5 Conditional operations (9 Marks)

Sunscreen lotion

Sunscreen lotion is labeled with a so-called *light protection factor* (LPF) that tells you how long the protection against UV radiation lasts. The intensity of the UV rays from the sun depends on your geographic altitude. In Egypt, the sun is more intense than in Germany

Depending on your skin-type, your natural resilience against UV-rays differs. The following table shows how long you can stay in direct, unfiltered, sunlight without protection before you get a sunburn.

Skin type	Egypt	Germany
Dark	60'	120'
Medium	30'	90'
Light	10'	30'

In theory, a sunscreen lotion with an LPF of n should boost your natural resilience time to n times that time. For example, if your natural resilience allows you to stay in the sun for 30', and you put on LPF 10, you should be able to stay in the sun for 300' or five hours. In reality, it's only half that time.

Write an algorithm that, given the place, skin type, and the time you would like to stay outside, returns the minimum LPF you should put on.

Solution:

- Getting the correct inputs: 1 Mark
- Handling Egypt correctly: 3 Marks
- Handling Germany correctly: 3 Marks
- Giving the correct output: 2 Marks

The algorithm assumes that the time is given in minutes.

```
get place, skinType, time
if (place = Egypt) then
  if (skinType = dark) then
    set resilience to 60
  else
    if (skinType = medium) then
      set resilience to 30
    else
      if (skinType = light) then
        set resilience to 10
      endif
    endif
  endif
else
  if (place = Germany) then
    if (skinType = dark) then
      set resilience to 120
    else
      if (skinType = medium) then
        set resilience to 90
      else
        if (skinType = light) then
          set resilience to 30
        endif
      endif
    endif
```

```
27    else
28      print "unknown␣place"
29    endif
30    set lpf to (time / resilience) * 2
31  endif
32  print "You␣should␣put␣on␣at␣least␣LPF␣"
33  print lpf
34  print "␣lotion␣to␣be␣safe␣from␣sunburn."
```

Note. This is a straightforward conditional distinction with little potential for more conciseness or clarity.

Exercise 6 Conditional operations (10 Marks)
Taking a bath

Generally, there are multiple ways to solve the same problem. Consider the following two algorithms:

(1)

```
get temperature
if (temperature > 38) then
    print "The water is too hot."
else
    if (temperature < 35) then
        print "The water is too cold."
    else
        print "The water is right."
    endif
endif
```

(2)

```
get temperature
if (temperature > 38) then
    print "The water is too hot."
endif
if (temperature < 35) then
    print "The water is too cold."
endif
if (not (temperature > 38 and temperature < 35)) then
    print "The water is right."
endif
```

Consider three different inputs: `temperature` = 15, `temperature` = 38, and `temperature` = 39

(a) What is the output of algorithms (1) and (2) for each of the three given inputs? 6 Marks

Algorithm	Input	Output?
(1)	15	
	38	
	39	
(2)	15	
	38	
	39	

(b) Are the algorithms equivalent? If yes, explain your answer, if not, fix the problem. 4 Marks

Solution:

(a) The output of algorithms (1) and (2) for each of the three given inputs (one Mark for each correct entry):

Algorithm	**Input**	**Output?**
(1)	15	"The water is too cold."
	38	"The water is right."
	39	"The water is too hot."
(2)	15	"The water is too cold.The water is right."
	38	"The water is right."
	39	"The water is too hot.The water is right."

(b) The algorithms are not equivalent. To fix, change the conditional in line 8, algorithm (2) to `if (not (temperature > 38 or temperature < 35)) then` otherwise the condition can never evaluate to `false`!

Exercise 7 Iterative operations (15 Marks)
Algae growth

Under good conditions, simple algae (such as the Cyanobacteria) can grow in numbers very quickly. A simplified model of the growth is that in each generation each single specimen divides into two new specimens.

(a) Write a program that, given a starting number of specimens calculates how many generations it takes to reach a number of 10,000 specimens. 6 Marks

(b) Write a second program which is a modification of the first. Now you assume that, if the number of specimens exceeds 5,000, only two thirds of the algae are able to reproduce (due to the availability of food). In other words, now three specimens will produce just four new specimens in the next generation. Again, given a starting number, your algorithm must calculate the number of generations it takes to reach 10,000 specimens. 9 Marks

Solution:

(a) Simple growth:

- Getting correct inputs: 1 Mark
- Giving a correct loop initialization: 1 Mark
- Giving a correct loop condition: 1 Mark
- Giving a correct loop-body: 2 Marks
- Giving the correct output: 1 Mark

```
1  get number
2  set generations to 0
3
4  while (number < 10000) {
5    set number to number * 2
6    set generations to generations + 1
7  }
8
9  print "It␣takes␣"
10 print generations
11 print "␣generations."
```

(b) Adapted growth:

```
1  get number
2  set generations to 0
3
4  while (number < 10000) {
5    if (number < 5000) then
6      set number to number + number
7    else
8      set number to (number / 3) * 4
9    endif
10   set generations to generations + 1
11 }
12
13 print "It␣takes␣"
14 print generations
15 print "␣generations."
```

or alternatively

```
get number
set generations to 0

while (number < 5000) {
  set number to number + number
  set generations to generations + 1
}

while (number < 10000) {
  set number to (number / 3) * 4
  set generations to generations + 1
}

print "It␣takes␣"
print generations
print "␣generations."
```

Each solution should equally earn all Marks, however the second is more clever...

Exercise 8 Iterative operations (12 Marks)
LCM

Consider the following algorithm:

```
1  get A, B
2  set x to A
3  set y to B
4
5  while (not x = y) {
6      if (x < y) then
7          set x to x + A
8      else
9          set y = y + B
10     endif
11 }
12
13 print "The␣LCM␣of␣A␣and␣B␣is␣"
14 print x
```

(a) Let the input be A = 15 and B = 20. What exactly is the output of the algorithm. 3 Marks

(b) "LCM" stands for *least common multiple* (*i. e.*, the smallest number which is a multiple of both input numbers). Extend the algorithm in a way that it calculates the LCM of *three* numbers A, B, and C. 9 Marks

Solution:

(a) The output is 60. Hit or miss, all three Marks.

(b) The algorithm in (a) selects the smaller of the two multiples and adds the respective input value until the two multiples are equal. To extend the algorithm to finding the least common multiple of three input values, the smallest of three multiples needs to be found in each iteration, and the respective input added until all three are equal.

- If the algorithm indicates that the principle is understood: 3 Marks.
- Getting the right inputs and giving the right output: 2 Marks.
- Right selection of the smallest multiple: 4 Marks.

```
1  get A, B, C
2  set x to A
3  set y to B
4  set z to C
5
6  while (not (x = y and y = z)) {
7    if (x < y) then
8      if (x < z) then
9        set x to x + A
10     else
11       set z to z + C
12     endif
13   else
14     if (y < z) then
15       set y to y + B
16     else
17       set z to z + C
18     endif
19   endif
```

```
20  }
21
22  print "The␣LCM␣of␣A,␣B,␣and␣C␣is␣"
23  print x
```

Background. Finding the LCM of a list of values plays an important role in the design and analysis of periodic real-time systems such as embedded command/control systems (*e. g.*, in cars, planes, *etc.*). For a clearer example see the "Tamagochi" exercise.w

Exercise 9 Iterative operations (27 Marks)

Tamagochi

A "Tamagochi" is an electronic key-chain toy pet that was popular in the mid-nineties. It is an egg-shaped device with a display and three buttons. The display shows a little animal. The buttons are labeled *feed*, *pet*, and *clean*. Whenever the animal needs attention, the device emits a sound and the user has to press the appropriate button.

Your friend asks you to take care of his beloved Tamagochi while he is travelling. You quickly figure out, that the Tamagochi follows a simple algorithm: It demands petting every 10 minutes, feeding every 12 minutes, and cleaning every 20 minutes.

(a) Sometimes, the time for feeding, petting, and cleaning comes at the same moment. You appreciate that, since this means that from that moment you have 10 minutes where you don't have to care for the thing. How often does this happen? 3 Marks

(b) Given a time-span `A` (for example the time-span between two feedings) and a time-span `B` (for example the time span between two cleanings), write an algorithm that calculates a *period*, which is the minimum time-span `P` so that

$$\underbrace{\texttt{A}+\texttt{A}+\ldots+\texttt{A}}_{n\times} \;=\; \underbrace{\texttt{B}+\texttt{B}+\ldots+\texttt{B}}_{m\times} \;=\; \texttt{P}$$

The algorithm should output the lenght of the period `P` and how often the time-span `A` (here: $n\times$) and the time-span `B` (here: $m\times$) fit into the period. 12 Marks

(c) Write a second algorithm that extends your algorithm from (b) to find the period of three time-spans `A`, `B`, and `C`. If you trace your algorithm with the three time-span values from your friends Tamagochi, what is the output? 12 Marks

Solution:

(a) The time for feeding, petting, and cleaning fall together every 60 minutes, which means six times the time-span between two feedings (10'), or five times the time-span between two pettings, or three times the time span between two cleanings.

(b) The Period is the *least common multiple* (LCM) of the time-spans A and B.

```
1  get A, B
2
3  set x to A
4  set n to 1
5
6  set y to B
7  set m to 1
8
9  while (not x = y) {
10     if (x < y) then
11         set x to x + A
12         set n to n + 1
13     else
14         set y = y + B
15         set m to m + 1
16     endif
17 }
18
19 print "The␣Period␣of␣A␣and␣B␣is␣"
20 print x
21 print ",␣which␣is␣"
22 print n
23 print "␣times␣A␣and␣"
```

```
print m
print " times B."
```

For full Marks it is *not* necessary to keep track of n and m. A solution which only gives the LCM or Period is sufficient (see LCM problem).

(c) The algorithm now has to compute the Period or the LCM of three values.

```
get A, B, C

set x to A
set n to 1
set y to B
set m to 1
set z to C
set l to 1

while (not (x = y and y = z)) {
  if (x < y) then
    if (x < z) then
      set x to x + A
      set n to n + 1
    else
      set z to z + C
      set l to l + 1
    endif
  else
    if (y < z) then
      set y to y + B
      set m to m + 1
    else
      set z to z + C
      set l to l + 1
    endif
  endif
}

print "The Period of A, B, and C is "
print x
print ", which is "
print n
print " times A, "
print m
print " times B, and "
print l
print " times C."
```

The output for the values 10, 12, and 20 is:
"The Period of A, B, and C is 60, which is 6 times A, 5 times B, and 3 times C."
Again, a solution which just outputs the period and does not keep track of n, m, and l is correct and should receive full Marks.

Exercise 10 Iterative operations (3 Marks)
Faculty!

The *faculty* on n (written $n!$) is defined as

$$n! = n \times (n-1) \times \cdots \times 1$$

Consider the following algorithm:

```
1  get n
2  set result to 0
3  set i to 1
4  while (i <= n) {
5      set result to result * i
6      set i to i + 1
7  }
8  print "The faculty of "
9  print n
10 print " is "
11 print result
```

What is the output for $n = 5$? What should be the output? Where is the error?

Solution:

The output is: "`The faculty of 5 is 0`", which is a bit off. The faculty of 5 is actually

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120.$$

Therefore the output should be "`The faculty of 5 is 120`". The error is found in the initialization, in line 2 it should say:
`set result to 1`

- 1 Mark for the correct trace of the wrong algorithm,
- 1 Mark for the correct output, and
- 1 Mark for the fix.

Exercise 11 Iterative operations (5 Marks)
Summation

The following algorithm adds the numbers from 1 to n:

```
1 get n
2 set result to 0
3 set i to 1
4 while (i <= n) {
5   set i to i + 1
6   set result to result + i
7 }
8 print result
```

In other words, it calculates

$$1 + 2 + \cdots + n$$

What is the output of this algorithm for $n = 5$? What should be the output? Correct the error.

Solution:

The algorithm wrongly computes

$$2 + 3 + \cdots + (n + 1).$$

Hence the output for $n = 5$ would be `20`, which is wrong. Instead it should be `15`. To correct the error, swap lines 5 and 6, so the algorithm reads:

```
1 get n
2 set result to 0
3 set i to 1
4 while (i <= n) {
5   set result to result + i
6   set i to i + 1
7 }
8 print result
```

Exercise 12 Iterations and lists (18 + 5 Marks)

LCM

Consider the following algorithm:

```
1  get A, B
2  set x to A
3  set y to B
4
5  while (not x = y) {
6      if (x < y) then
7          set x to x + A
8      else
9          set y = y + B
10     endif
11 }
12
13 print "The␣LCM␣of␣A␣and␣B␣is␣"
14 print x
```

(a) Let the input be A = 15 and B = 20. What exactly is the output of the algorithm. 3 Marks

(b) "LCM" stands for *least common multiple* (*i. e.*, the smallest number which is a multiple of both input numbers). Extend the algorithm in a way that it calculates the LCM of *three* numbers A, B, and C. 9 Marks

(c) Explain, using snippets of pseudo code if necessary, how you would extend the algorithm to calculate the LCM of a list $I_1, \ldots, I_n$ of n input numbers. To earn bonus marks, write down the complete algorithm. 6 Marks + 5 Bonus

Solution:

(a) The output is 60. Hit or miss, all three Marks.

(b) The algorithm in (a) selects the smaller of the two multiples and adds the respective input value until the two multiples are equal. To extend the algorithm to finding the least common multiple of three input values, the smallest of three multiples needs to be found in each iteration, and the respective input added until all three are equal.

- If the algorithm indicates that the principle is understood: 3 Marks.
- Getting the right inputs and giving the right output: 2 Marks.
- Right selection of the smallest multiple: 4 Marks.

```
1  get A, B, C
2  set x to A
3  set y to B
4  set z to C
5
6  while (not (x = y and y = z)) {
7    if (x < y) then
8      if (x < z) then
9        set x to x + A
10     else
11       set z to z + C
12     endif
13   else
14     if (y < z) then
15       set y to y + B
16     else
```

```
17        set z to z + C
18      endif
19    endif
20  }
21
22  print "The␣LCM␣of␣A,␣B,␣and␣C␣is␣"
23  print x
```

(c) The inputs change from the fixed number of three values to a single list with an arbitrary number of values and a given length.

First, we have to get the input list $A_1, \ldots, A_c$, and its length, which in the algorithm will be designated by c:

```
1  get c
2  get A1, ..., Ac
```

Next, we initialize the counters as a list $N_1, \ldots, N_c$, all with the value 1 (this is not necessary to find the LCM, but it keeps track of the factors):

```
1  set N1, ..., Nc to 1
```

Then we initialize a list of multiples $X_1, \ldots, X_c$, which are each set to $X_i = 1 \times A_i$:

```
1  set i = 1
2  while (i <= c) {
3    set Xi to Ai
4    set i to i + 1
5  }
```

Now we can start a loop which finds the LCM. It is found as soon as all multiples are equal. The Condition for the loop is hence:

```
1  while (not (X1 = ... = Xc)) {
```

Inside the loop we use the algorithm from the lecture to find the *minimum* in the list of multiples (in the lecture we searched for the *maximum*. The Algorithm finds the first occurrence of the minimum multiple:

```
1    set min to X1
2    set location to 1
3    set i to 2
4    while (i =< c) {
5      if (Xi < min) then
6        set min to Xi
7        set location to i
8      endif
9      set i to i + 1
10   }
11   set Xi to Xi + Ai
12   set Ni to Ni + 1
```

A solution which just describes this idea (as in "*search the smallest in the list of multiples and add the base value to it*") should already receive full Marks. The key are the lines 22 and 23:

```
1    set Xi to Xi + Ai
2    set Ni to Ni + 1
```

The smallest of the multiples was found at index i, so we add the input value A_i to it and increment the counter N_i.

When the loop finishes, we only need to output the resulting LCM:

```
1  print "The␣LCM␣of␣A1,␣...,␣Ac␣is␣"
2  print X1
```

As "icing on the cake" we can add an output for the counter of each original list item:

```
print ", which is "
set i = 1
while (i < c) {
  print Ni
  print " times A"
  print i
  print ", and "
  set i to i + 1
}
print Nc
print " times Ac."
```

Full Algorithm. Thus, the full algorithm looks as this:

```
get c
get A1, ..., Ac
set N1, ..., Nc to 1

set i = 1
while (i <= c) {
  set Xi to Ai
  set i to i + 1
}

while (not (X1 = ... = Xc)) {
  set min to X1
  set location to 1
  set i to 2
  while (i =< c) {
    if (Xi < min) then
      set min to Xi
      set location to i
    endif
    set i to i + 1
  }
  set Xi to Xi + Ai
  set Ni to Ni + 1
}

print "The LCM of A1, ..., Ac is "
print X1

print ", which is "
set i = 1
while (i < c) {
  print Ni
  print " times A"
  print i
  print ", and "
  set i to i + 1
}
print Nc
print " times Ac."
```

The **attempt** to code the full algorithm, even if there are errors or insufficiencies, should receive full Marks.

Notes. The invariant is that in each iteration of the loop, each multiple X_i is the product of the input value A_i and the factor N_i, which means

$$X_i = N_i \times A_i$$

So in the end we found a common multiple of all input values, since with

$$X_1 = \cdots = X_c$$

we have that

$$N_1 \times A_1 = \cdots = N_c \times A_c$$

So we only need to convince ourselves that we also found the *least* (*i. e.*, smallest) common multiple.

Background. Finding the LCM for a list of values has a range of technical applications. In one of the most important areas of application, which is real-time systems, usually the list is *harmonic*, which means that each item (except for the first) is a multiple of at least one of the preceding list items (*e. g.*, 10, 20, 50).

Exercise 13 Iterations over lists (16 Marks)
Delivery date

An online shop wants to inform their customers about the prospective delivery date of a product. Given the current date and the number of days that it usually takes to deliver a specific item, calculate the delivery date.

You can assume that you have a list $M_1, \ldots, M_{12}$ which gives the number of days for each month (*i. e.*, $M_1 = 31$, $M_2 = 28$, $M_3 = 31, \ldots, M_{12} = 31$. The current date is given as `year`, `month`, and `day`.

Write an algorithm with current date and number of days for delivery as input and the delivery date as output. Do not worry about leap years.

(a) First, assume that a delivery always takes less than 28 days (*i. e.*, never longer than one month). (4 Marks)

(b) Write a second algorithm in which deliveries also can take more than 28 days. (12 Marks)

Solution:

(a) The algorithm adds the days-to-delivery to the current date's day. After that it checks the "spill-over" to the next month, and possibly the next year.

- Get the right inputs: 1 Mark
- Calculate the day: 1 Mark
- Calculate the month: 1 Mark
- Calculate the year: 1 Mark

```
1  get year, month, day, daysToDelivery
2
3  set resultDay to day + daysToDelivery
4
5  if (resultDay > Mmonth) then       // does the date reach into the next month?
6    set resultDay to resultDay - Mmonth
7    set resultMonth to month + 1
8  else
9    set resultMonth to month
10 endif
11
12 if (resultMonth > 12) then         // does the date reach into the next year?
13   set resultMonth to resultMonth - 12
14   set resultYear to year + 1
15 else
16   set resultYear to year
17 endif
18
19 print "The␣delivery␣date␣is␣"
20 print resultDay
21 print "."
22 print resultMonth
23 print "."
24 print resultYear
```

(b)
- Get the right inputs: 1 Mark
- Devising a meaningful loop to repeatedly take the "spill-over" into the next month or year: 10 Marks
- Giving a reasonable output: 1 Mark

```
1  get year, month, day, daysToDelivery
2
3  set resultDay to day + daysToDelivery
```

```
4  set resultMonth to month
5  set resultYear to year
6
7  while (resultDay > MresultMonth) {
8    set resultDay to resultDay - MresultMonth
9    set resultMonth to resultMonth + 1
10   if (resultMonth > 12) then
11     set resultMonth to 1
12     set resultYear to resultYear + 1
13   endif
14 }
15
16 print "The␣delivery␣date␣is␣"
17 print resultDay
18 print "."
19 print resultMonth
20 print "."
21 print resultYear
```

Note that the algorithm can be written without a loop, since the length of the list in question is fixed. However, this would mean a large number of error-prone conditionals.

Exercise 14 Iterations over lists (16 Marks)
Library fees

A library needs a program to calculate late-fees. The usage is as follows:

The librarian enters the checkout date and the return date of the book. If the book has been checked out for one week or less, it is free of charge. For each additional week, the charge is 2.00 £E. After six weeks, the price is fixed at 10 £E.

You can assume that you have a list $M_1, \ldots, M_{12}$ which gives the number of days of each month (*i. e.*, $M_1 = 31$, $M_2 = 28$, $M_3 = 31$, $\ldots$, $M_{12} = 31$. The dates are given as `checkout-day`, `checkout-month`, `return-day`, and `return-month`. Further, assume that if the return date is before the checkout date, that they in fact refer to consecutive years (*i. e.*, the checkout was the year before). Do not worry about leap years or inputs that do not represent valid dates.

Hint. Calculate the "day of the year" for each input-date (*e. g.*, February 6 is the 37th day of the year). Since you do not have to consider leap years, a full year has 365 days.

Example.

Input: `checkout-day` = 20, `checkout-month` = 12, `return-day` = 6, and `return-month` = 1.

Output: `Late fees are 4 EGP`

Solution:

- Getting the right inputs: 1 Mark
- successfully calculating the day-of-the-year for a date: 5 Marks
- successfully calculating the difference, including the change of years: 5 Marks
- Successfully calculating the weeks and the charge: 4 Marks
- Giving the right output: 1 Mark

```
1  get checkout-day, checkout-month, return-day, return-month
2
3  set day-one to checkout-day
4  set i to 1
5  while (i < checkout-month) {
6    set day-one to day-one + Mi
7    set i to i + 1
8  }
9
10 set day-two to return-day
11 set i to 1
12 while (i < return-month) {
13   set day-two to day-two + Mi
14   set i to i + 1
15 }
16
17 set days to day-two - day-one
18
19 if (days < 0) then
20   set days to days + 365
21 endif
22
23 set weeks to INT(days / 7)
```

```
24
25 if (weeks >= 6) then
26   print "The␣charge␣is␣EGP␣10"
27 else
28   set charge to weeks * 2
29   print "The␣charge␣is␣"
30   print charge
31 endif
```

Lines 3-8 and 10-15 calculate the day of the year for checkout and return respectively. If the return date is earlier than the checkout, you are to assume consecutive years, so add 365 to the (negative) difference in that case. After six weeks the charge is fixed, so this is a special case, otherwise calculate the actual fees.

Exercise 15 Simple search (9 Marks)
Vineyard

You are given a list $W_1, \ldots, W_n$ of wines, each with a price per bottle stored in a list $P_1, \ldots, P_n$. If you need the price for a bottle of the sort W_i, look up P_i.

write an algorithm that, given the two lists and a name of a wine outputs the price per bottle.

Solution:

Solution not needed yet.

Exercise 16 List iteration (21 Marks)
Diet Calculator

You are on a high calories diet, because you really need to gain some weight. You have compiled a list $F_1, \ldots, F_n$ of n different food items that you like. Further, you have a second list $C_1, \ldots, C_n$, which lists the calories for each food item (*i. e.*, C_i contains the calories for food item F_i).

(a) First, you create a simple menue. The menue consist of three food items (`starter`, `main-dish`, and `desert`). Write a program that looks up the three food items in the list F and sums up the corresponding calories. Output "You'll be hungry" if your menue contains less than 2,000 Kcal or "Finally enough to eat" if your menue contains more than 2,000 Kcal. (9 Marks)

(b) Now you realize that not every meal has exactly three courses. There might be an apetizer, a cheese platter, ice cream... Write a second algorithm just like the first, only instead of three variables `starter`, `main-dish`, and `desert`, the input of the menu will be in form of a list $M_1, \ldots, M_m$. (12 Marks)

Solution:

No solution needed yet.