BABEŞ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Math & CompSci Contest - model

Written Test in Computer Science

Important observations for candidates:

1. All arrays are indexed starting from 1.
2. Multiple answer problem statements (Part A) can have one or more correct answers. Candidates must write the answers on their contest sheet (not the sheets with the problem statements). Solutions are graded only if they identify all the correct answers, and only them.
3. Complete solutions are expected for problem statements in Part B.
   a. Solutions will be described using *pseudocode* or a *programming language* (*Pascal/C/C++).*
   b. The first criterion when evaluating solutions is algorithm ***correctness***, followed by its ***performance*** regarding *time of execution* and *required memory space*.
   c. ***Providing description and justification*** for all (sub)algorithms before their implementation is ***mandatory***. In addition, ***comments*** will be used to describe details of the technical implementation, meaning of used identifiers, data structures and so on. Failure to observe these requirements leads to a 10% penalty from the problem score.
   d. Using predefined functions or libraries is prohibited (for example: *STL*, predefined string functions).

Part A (60 points)

A.1. What does the following subalgorithm do? (6 points)

The `generate(n)` subalgorithm processes natural number ***n*** ($0 < n < 100$).

```
Subalgorithm generate(n):
    nr ← 0
    For i ← 1, 1801 do
        used_i ← false
    EndFor
    While not used_n do
        sum ← 0, used_n ← true
        While (n ≠ 0) do
            digit ← n MOD 10, n ← n DIV 10
            sum ← sum + digit * digit * digit
        EndWhile
        n ← sum, nr ← nr + 1
    EndWhile
    return nr
EndSubalgorithm
```

State the effect of this subalgorithm.

A. repeatedly calculates the sum of the cubes of ***n***'s digits until the sum is equal to ***n***, after which it returns the number of repetitions
B. calculates the sum of the cubes of ***n***'s digits and returns this sum
C. calculates the sum of the cubes of ***n***'s digits, replaces ***n*** with the obtained sum and returns it
D. calculates how many times number ***n*** was replaced with the sum of the cubes of ***n***'s digits until a previously calculated value or the number itself is obtained; this number is then returned

A.2. What values are required? (6 points)

Consider subalgorithm `process(v, k)`, where ***v*** is an array of ***k*** natural numbers ($1 \leq k \leq 1\,000$).

```
Subalgorithm process(v, k)
    i ← 1, n ← 0
    While i ≤ k and v_i ≠ 0 do
        y ← v_i, c ← 0
        While y > 0 do
            If y MOD 10 > c then
                c ← y MOD 10
            EndIf
            y ← y DIV 10
        EndWhile
        n ← n * 10 + c
        i ← i + 1
    EndWhile
    return n
EndSubalgorithm
```

State for which values of ***v*** and ***k*** the subalgorithm returns 928.

A. ***v*** = (194, 121, 782, 0) and ***k*** = 4
B. ***v*** = (928) and ***k*** = 1
C. ***v*** = (9, 2, 8, 0) and ***k*** = 4
D. ***v*** = (8, 2, 9) and ***k*** = 3

A.3. Logical Evaluation (6 points)

Let us consider array ***s*** with ***k*** boolean elements and subalgorithm `evaluation(s, k, i)`, where ***k*** and ***i*** are natural numbers ($0 \leq i \leq k \leq 100$).

```
Subalgorithm evaluation(s, k, i)
   If i ≤ k then
      If si then
         return si
      else
         return (si or evaluation(s, k, i + 1))
      EndIf
   else
      return false
   EndIf
EndSubalgorithm
```

State how many times subalgorithm `evaluation(s, k, i)` is self-called by executing the following instruction sequence:

```
s ← (false, false, false, false, false, false, true, false, false, false)
k ← 10
i ← 3
evaluation(s, k, i)
```

A. 3 times
B. the same number of times as in the following instruction sequence

```
s ← (false, false, false, false, false, false, false, true)
k ← 8
i ← 4
evaluation(s, k, i)
```

C. 6 times
D. never

A.4. Reunion (6 points)

Consider subalgorithm `belongs(x, a, n)`, which checks whether natural number ***x*** belongs to set ***a*** having ***n*** elements; ***a*** is an array of ***n*** elements that represents a natural numbers set ($1 \leq n \leq 200$, $1 \leq x \leq 1000$). Let us consider subalgorithms `reunion(a, n, b, m, c, p)` and `compute(a, n, b, m, c, p)`, described below, where ***a***, ***b*** and ***c*** are arrays that represent natural number sets having ***n***, ***m*** and ***p*** elements respectively ($1 \leq n \leq 200$, $1 \leq m \leq 200$, $1 \leq p \leq 400$). Input parameters are ***a***, ***n***, ***b, m*** and ***p***, and output parameters are ***c*** and ***p***.

```
1.      Subalgorithm reunion(a, n, b, m, c, p):
2.        If n = 0 then
3.          For i ← 1, m do
4.            p ← p + 1, cp ← bi
5.          EndFor
6.        else
7.          If not belongs(an, b, m) then
8.            p ← p + 1, cp ← an
9.          EndIf
10.         reunion(a, n - 1, b, m, c, p)
11.       EndIf
12.     EndSubalgorithm
```

```
1.  Subalgorithm compute(a, n, b, m, c, p):
2.    p = 0
3.    reunion(a, n, b, m, c, p)
4.  EndSubalgorithm
```

Which of the following statements are always true:

A. when set ***a*** contains a single element, calling subalgorithm `compute(a, n, b, m, c, p)` results in an infinite loop
B. when set ***a*** contains 4 elements, calling subalgorithm `compute(a, n, b, m, c, p)` results in executing the instruction on line 10 a number of 4 times
C. when set ***a*** contains 5 elements, calling subalgorithm `compute(a, n, b, m, c, p)` results in executing the instruction on line 2 of the `reunion` subalgorithm a number of 5 times
D. when sets ***a*** and ***b*** have the same elements, after the execution of subalgorithm `compute(a, n, b, m, c, p)` set ***c*** will have the same number of elements as set ***a***

A.5. Exponentiation (6 points)

Which of the following algorithms correctly computes a^b, where a and b are natural numbers ($1 \le a \le 11$, $0 \le b \le 11$).

A.
```
Subalgorithm expo(a, b):
    result ← 1
    While b > 0 do
        If b MOD 2 = 1 then
            result ← result * a
        EndIf
        b ← b DIV 2
        a ← a * a
    EndWhile
    return result
EndSubalgorithm
```

B.
```
Subalgorithm expo(a, b):
    If b ≠ 0 then
        If b MOD 2 = 1 then
            return expo(a * a, b / 2) * a
        else
            return expo(a * a, b / 2)
        EndIf
    else
        return 1
    EndIf
EndSubalgorithm
```

C.
```
Subalgorithm expo(a, b):
    result ← 0
    While b > 0 do
        result ← result * a
        b ← b - 1
    EndWhile
    return result
EndSubalgorithm
```

D.
```
Subalgorithm expo(a, b):
    If b = 0 then
        return 1
    EndIf
    return a * expo(a, b - 1)
EndSubalgorithm
```

A.6. Largest multiple (6 points)

Which of the following subalgorithms returns the largest multiple of number a, multiple that is smaller or equal with natural number b ($0 < a < 10\,000$, $0 < b < 10\,000$, $a < b$)?

A.
```
Subalgorithm f(a, b):
  c ← b
  While c MOD a = 0 do
    c ← c – 1
  EndWhile
  return c
EndSubalgorithm
```

B.
```
Subalgorithm f(a, b):
  If a < b then
    return f(2 * a, b)
  else
    If a = b then
      return a
    else
      return b
    EndIf
  EndIf
EndSubalgorithm
```

C.
```
Subalgorithm f(a, b):
  return (b DIV a) * a
EndSubalgorithm
```

D.
```
Subalgorithm f(a, b):
  If b MOD a = 0 then
    return b
  EndIf
  return f(a, b - 1)
EndSubalgorithm
```

A.7. Data types (6 points)

An integer data type represented using x bits (x a strictly positive natural number) can hold values from the following range:

A. $[0, 2^x]$
B. $[0, 2^{x-1}-1]$
C. $[-2^{x-1}, 2^{x-1}-1]$
D. $[-2^x, 2^x-1]$

A.8. Number of calls (6 points)

Consider subalgorithm `f(a, b)`:

```
Subalgorithm f(a, b):
      If a > 1 then
            return b * f(a - 1, b)
      else
            return b * f(a + 1, b)
      EndIf
EndSubalgorithm
```

How many times will subalgorithm f be self-called by executing the following instruction sequence:

```
a ← 4
b ← 3
c ← f(a, b)
```

A. 4 times
B. 3 times
C. infinite number of times
D. never

A.9. Arrays (6 points)

Let us consider all arrays of length $l \in \{1, 2, 3\}$ containing letters from the {*a*, *b*, *c*, *d*, *e*} set. How many of these arrays are sorted strictly decreasing and contain an odd number of vowels? (*a* and *e* are vowels).

A. 14
B. 7
C. 81
D. 78

A.10. Positive numbers (6 points)

Consider the `positiveNumbers(m, a, n, b)` subalgorithm.

```
void positiveNumbers(int m,int a[], int &n,
int b[]){
    n = 0;
    for(int i = 1; i <= n; i++){
      if (a[i] > 0){
          n = n + 1;
          b[n] = a[i];
      }
    }
}
```

```
procedure positiveNumbers(m:integer;
a:vector; var n:integer; var b:vector)
  begin
    n := 0;
    for i := 1 to n do
      if (a[i] > 0) then
        begin
          n := n + 1;
          b[n] := a[i];
        end;
 end;
```

What is the result of executing `positiveNumbers(k, x, p, y)` for $k = 4$, array x=(-1, 2, -3, 4), $p = -1$ and the empty array $y = ()$.

A. $p = 3$ and $y = (2, 4)$;
B. $p = 0$ and $y = (2, 4)$;
C. $p = 0$ and $y = ()$;
D. Depends on the value of k.

<u>Part B</u> (30 points)

B.1. Magic numbers (15 points)

Consider two natural numbers p and q ($2 \le p \le 10$, $2 \le q \le 10$). A natural number is *magic* if the set of digits used for writing it in base p is identical to the set of digits used for writing it in base q. For example, for $p = 9$ and $q = 7$, $(31)_{10}$ is a *magic* number because $(34)_9 = (43)_7$, and for $p = 3$ and $q = 9$, $(9)_{10}$ is a *magic* number because $(100)_3 = (10)_9$. Consider subalgorithm `baseDigits(x, b, c)` to determine the digits used for writing number x in base b (stored in array c):

```
Subalgorithm baseDigits(x, b, c):
   While x > 0 do
      c[x MOD b] ← 1
      x ← x DIV b
   EndWhile
EndSubalgorithm
```

Requirements:

a. Write a *recursive* version (without repetitive structures) for subalgorithm `baseDigits(x, b, c)` having the same signature and effect. (**5 points**)
b. Write the mathematical model for the recursive version of subalgorithm `baseDigits(x, b, c)` (developed at a). (**3 points**)
c. Write a subalgorithm that, using the `baseDigits(x, b, c)` subalgorithm, given two bases p and q determines array a of all strictly positive *magic* numbers strictly smaller than a given natural number n ($1 < n \le 10\,000$). The input parameters of this subalgorithm are p and q (the two bases) and n. Output parameters are array a and its length k. (**7 points**)

Example: if $p = 9$, $q = 7$ and $n = 500$, array a contains $k = 11$ elements: (1, 2, 3, 4, 5, 6, 31, 99, 198, 248, 297).

B.2. Chocolate tasting (15 points)

A marketing company is advertising a new brand of chocolate and intends to distribute chocolate samples to n ($10 \leq n \leq 10\,000\,000$) children who are standing in a circle. The employees know that distributing samples to each child would cost too much money. Therefore, they decide to distribute a sample to each k^{th} ($0 < k < n$) child. They count the children standing in the circle, and hand out a sample to each k^{th} child. When the count reaches the last child, it continues with the first one, and so on. All children are counted, even if they have already received chocolate. Counting stops when a chocolate sample should be given to a child who already received one.

Requirements:

a. State the property that must be observed by the k value associated with those children who receive chocolate. Justify your answer. (**3 points**)
b. Explain (using natural and mathematical language) what number of children receive chocolate? Justify your answer. (**2 points**)
c. Write a subalgorithm that determines the number of children (***nr***) who do not receive a chocolate sample. Input parameters are natural numbers n and k, and the output parameter is natural number ***nr***. (**10 points**)

Example 1: if $n = 12$ and $k = 9$, then ***nr*** = 8 (first, second, fourth, fifth, seventh, eighth, tenth and eleventh child do not receive chocolate).
Example 2: if $n = 15$ and $k = 7$, then ***nr*** = 0 (all children receive chocolate samples).

Note:

1. All subjects are mandatory.
2. Solutions must be written on the exam sheets in detailed form (drafts are not graded).
3. Default 10 points.
4. Working time is 3 hours.

GRADING & SOLUTIONS

DEFAULT ..... **10 points**

Part A ..... **60 points**

A. 1. What does the following subalgorithm do? Answer: D ..... 6 points
A. 2. What values are required? Answer: A, C ..... 6 points
A. 3. Logical Evaluation. Answer: B ..... 6 points
A. 4. Reunion. Answer: B, D ..... 6 points
A. 5. Exponentiation. Answer: A, B, D ..... 6 points
A. 6. Largest multiple. Answer: C, D ..... 6 points
A. 7. Data types. Answer: B, C ..... 6 points
A. 8. Number of calls. Answer: C ..... 6 points
A. 9. Arrays. Answer: A ..... 6 points
A. 10. Positive Numbers. Answer: C ..... 6 points

Part B ..... **30 points**

B. 1. Magic Numbers ..... **15 points**

B.1.a. recursive version for subalgorithm `baseDigits(x, b, c)` ..... **5 points**

– correct signature ..... 1 punct
– condition to stop recursion ..... 1 punct
– self-call (logic, parameters) ..... 2 points
– returned values ..... 1 punct

```
Subalgorithm baseDigits(x, b, c):
   If x > 0 then
      c[x MOD b] ← 1
      baseDigits(x DIV b, b, c)
   EndIf
EndSubalgorithm
```

B.1.b. mathematical model for `baseDigits(x, b, c)` ..... **3 points**

$$baseDigits(x,b,c) = \begin{cases} -, & if\ x = 0 \\ baseDigits(x\ DIV\ b, b, c'), where\ c'_{x\ MOD\ b} = 1, & else \end{cases}$$

B.1.c. subalgorithm for building array a ..... **7 points**

– verify the *magic number* property
  - V1: based on the equality of the digit set's characteristic arrays for the representation in bases ***p*** and ***q*** ..... 5 points

```
// we build the apparition array for the digits in base p for number x
// determine x's digits in base q
//     if the current digit does not appear in the base p representation then x
//is not a magic number
//     else increment the value corresponding to the digit in the array
// if the digit array contains a 1 for certain digits, they appear in the base
// p representation but not in the base q one, so the number is not magic
bool magicNr(int x, int p, int q){
        //check if x is a magic number for bases p and q
        int digits[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        baseDigits(x, p, cifre);
        while (x != 0){     //determine x's digits in base q
                int uc = x % q;
                if (digits[uc] == 0)  //if the current digit (in base q) is
                                      //not used in base p
                        return false;
                digits[uc]++;
                x = x / q;
        }
        for (int i = 0; i < 10; i++){
                if (digits[i] == 1)   //if digit I is used in base p but
                                      //not in base q
                        return false;
        }
        return true;
}
```

- V2: other correct algorithm version with lower performance.....maximum 3 points

– build array ***a*** .....2 points

```
void magicNrArray(int p, int q, int n, int &k, int sir[]){
        k = 0;
        for (int i = 1; i < n; i++){
                if (magicNr(i, p, q))
                        sir[k++] = i;
        }
}
```

B. 2. Chocolate tasting..... 15 points

B.2.a. We consider counting the children in a circle as a linear count of several smaller arrays, each with ***n*** children. We obtain a larger array, of ***p*** children (***p*** is a multiple of ***n***). We finish the count when the n^{th} child in the smaller array receives chocolate (as such, the next child to receive it is the k^{th} child in the next array). For example, for ***n*** = 12 children and ***k*** = 9, we create several arrays of ***n*** children:

Child index	1	2	3	4	5	6	7	8	9	10	11	12
With/without chocolate	0	0	0	0	0	0	0	0	0	0	0	0

Index longer array	1	2	3	4	5	6	7	8	**9**	10	11	12
Child index	1	2	3	4	5	6	7	8	**9**	10	11	12
With/without chocolate	0	0	0	0	0	0	0	0	**1**	0	0	0

Index longer array	13	14	15	16	17	**18**	19	20	21	22	23	24
Child index	1	2	3	4	5	**6**	7	8	9	10	11	12
With/without chocolate	0	0	0	0	0	**1**	0	0	1	0	0	0

Index longer array	25	26	**27**	28	29	30	31	32	33	34	35	**36**
Child index	1	2	**3**	4	5	6	7	8	9	10	11	**12**
With/without chocolate	0	0	**1**	0	0	1	0	0	1	0	0	**1**

Here we stop handing out chocolate because the next one should be given to a child that already received chocolate.

Index longer array	37	38	39	40	41	42	43	44	**45**	46	47	48
Child index	1	2	3	4	5	6	7	8	**9**	10	11	12
With/without chocolate	0	0	1	0	0	1	0	0	**1**	0	0	1

Children who receive chocolate are those whose index in the longer array is a multiple of ***k***. ..... **2 points**

B.2.b. The number of children who receive chocolate: ***p*** must also be a multiple of ***k***. So, ***p*** = lcm(***n***, ***k***). Of the ***p*** children, exactly ***p*** / ***k*** children received chocolate, so the number of those without is ***n*** - ***p*** / ***k*** = ***n*** - lcm(***n***, ***k***) / ***k*** = ***n*** - (***n*** * ***k*** / lcm(***n***, ***k***)) / ***k*** = ***n*** - ***n*** / lcm(***n***, ***k***)..... **3 points**

B.2.c. Develop the subalgorithm ..... **10 points**

- V1: correctly determine the value for ***nr*** (formula is ***nr*** = ***n*** – ***n***/*lcm*(***n***, ***k***)) .....10 points

```
//calculate and return the lcm of natural numbers a and b
int lcm(int a, int b){
        if ((a == b) && (a == 0))
                return 1;
        if (a * b == 0)
                return a + b;
        while (b != 0){
                int c = b;
                b = a % b;
                a = c;
        }       //while
        return a;
}

int chocolateTasting(int n, int k){
        return n - n / lcm(n, k);
}
```

- V2: correctly determine ***nr*** (simulation, circular list) ..... 7 points

```
int chocolate(int n, int k){
        const int MAXLEN = 10000000;
        bool chocolate[MAXLEN];
        for (int i = 1; i <= n; i++)
                chocolate[i] = false;
        int i = k;
        int nrChildrenWithoutChocolate = n;
        while (!chocolate[i]){ //circular list simulation
                chocolate[i] = true;
                nrChildrenWithoutChocolate--;
                i += k;
                if (i > n){
                        i -= n;
                }
        }
        return nrChildrenWithoutChocolate;
}
```