The Ministry of Education and Science of the Russian Federation

Southern Federal University

A. V. Abramyan, M. E. Abramyan

PROBLEM BOOK ON PROGRAMMING

SCALAR TYPES, CONTROL STATEMENTS,
PROCEDURES AND FUNCTIONS,
ARRAYS, STRINGS, FILES,
RECURSION, DYNAMIC DATA STRUCTURES

Rostov-on-Don
2014

ББК 32.973
А16
УДК 681.3

Печатается по решению
учебно-методической комиссии
факультета математики, механики и компьютерных наук ЮФУ
от 5 мая 2014 г. (протокол № 8)

Р е ц е н з е н т :
к. ф.-м. н., доцент С. С. Михалкович

Абрамян А. В., Абрамян М. Э.

А16 Сборник задач по программированию: Скалярные типы, управляющие операторы, процедуры и функции, массивы, строки, файлы, рекурсия, динамические структуры данных (на англ. языке). — Ростов н/Д, 2014. — 158 с.

Пособие содержит 1100 учебных заданий по основным темам базового курса программирования. Формулировки заданий приводятся на английском языке, что позволяет применять пособие в курсах англоязычных учебных программ. В формулировках не используются понятия и имена, специфические для конкретного языка программирования.

Пособие предназначено для студентов математических, компьютерных и естественнонаучных специальностей.

Preface

This book of problems on programming contains 1100 learning tasks that cover almost all sections of a basic programming curriculum — beginning with scalar types and control statements to complicated data structures and recursive algorithms.

Task texts do not contain notions that are specific for some programming language. Tasks are divided on 19 task groups:

- Begin — input-output and assignment (40 tasks),
- Integer — integers (30 tasks),
- Boolean — logical expressions (40 tasks),
- If — conditional statement (30 tasks),
- Case — selection statement (20 tasks),
- For — loop with the parameter (40 tasks),
- While — loop with the condition (30 tasks),
- Proc and Func — procedures and functions (60 tasks); the Func group is a modification of the Proc group for the Python and Ruby languages,
- Series — numerical sequences (40 tasks),
- Minmax — minimums and maximums (30 tasks),
- Array — one-dimensional arrays (140 tasks),
- Matrix — two-dimensional arrays (matrices) (100 tasks),
- String — characters and strings (70 tasks),
- File — binary files (90 tasks),
- Text — text files (60 tasks),
- Param — structured data types in procedures and functions (70 tasks),
- Recur — recursion (30 tasks),
- Dynamic and ObjDyn — dynamic data structures (80 tasks),
- Tree and ObjTree — binary trees (100 tasks).

Programming Taskbook

For more efficient use of this problem book you may use it jointly with the Programming Taskbook courseware that contains all tasks included in this book and allows you to solve tasks on various programming languages such as Pascal, Visual Basic, C++, C#, Visual Basic .NET, Python, Java, Ruby.

The learning options listed below are supported by Programming Taskbook:

- display the task text and corresponding data on screen,
- display correct results for each task,
- subroutines for initial data input and results output,
- additional control of input-output operations,
- automatic correctness checking of results computed by student's program,
- storing running program data into a special log file (file of results),

- complete register of solved task as result of a series of successful test program runnings.

The important feature of the Programming Taskbook is its availability for use in different programming environments:

- Borland Delphi 7.0 and 2006, in particular, Turbo Delphi 2006 for Windows,
- Free Pascal Lazarus 1.0,
- PascalABC.NET,
- Microsoft Visual Basic 5.0 and 6.0,
- Microsoft Visual C++ 6.0,
- Microsoft Visual Studio 2003, 2005, 2008, 2010, 2012 and 2013 (C++, Visual Basic .NET, and C# languages),
- IDLE for Python 2.5, 2.6, 2.7, 3.2,
- NetBeans IDE 6.x and 7.x (Java and Ruby languages).

Programming Taskbook sufficiently facilitates educational task execution. It is due to it performs automatically standard input-output operations unlike manual coding. Its advantage is especially obvious when performing processing of arrays, strings, files, and dynamic data structures. Submitting ready input data to students, Programming Taskbook directs their efforts to the development and program implementation of the *algorithm* of the task solving; the variety of input data submitted by Programming Taskbook provides the *effective testing* of the offered algorithm.

You may obtain additional information about the Programming Taskbook on its website `http://ptaskbook.com`.

General remarks on data types and terminology

All elements of any sequence of *real numbers* are supposed to contain different values, so each sequence of real numbers contains just one minimal and maximal element. Sequences of *integers* may contain elements with equal values, so such sequences may have several elements with minimal and maximal values. Numerical arrays and files also satisfy these conditions.

If a task does not specify the maximal size of an input array then this size should be considered as 10 for one-dimensional arrays and 10×10 for two-dimensional arrays.

The “*order number*” notion is used for array elements; the first element of one-dimensional array named A has the order number 1 and is denoted as A_1. Similarly, the first element of two-dimensional array named B is denoted as $B_{1,1}$. Lines and rows of two-dimensional array are also numbered from 1. Such approach does not depend on a specific programming language and corresponds to traditional mathematical notation.

The “*procedure*” notion in task texts of Proc, Param, and Dynamic groups means not only procedures of Pascal but also subroutines of Visual Basic and functions with the void return type of C++/C#/Java/etc.

The "*nil*" notion is used for empty pointer (as in Pascal language) and the "*null*" notion is used for empty class reference (as in C# and Java languages).

There are two versions of the Dynamic and Tree groups: the first version uses *records* and *pointers* and is intended for Pascal and C++ languages, the second one (the ObjDyn and ObjTree groups) uses *objects* and is intended for C#, Visual Basic .NET, Python, Java, and Ruby languages. The corresponding types — the TNode record and the PNode pointer (for Pascal and C++), and the Node class (for PascalABC.NET, C#, VB.NET, Python, Java, Ruby) — are defined in the preamble to the Dynamic/ObjDyn and Tree/ObjTree task groups. When using Programming Taskbook you do not need to define these types because they are already defined in the Programming Taskbook modules.

1. Input-output and assignment

All input and output data are real numbers in tasks of this group.

Begin1. Given the side a of a square, find the perimeter P of the square: $P = 4\cdot a$.

Begin2. Given the side a of a square, find the area S of the square: $S = a^2$.

Begin3. The sides a and b of a rectangle are given. Find the area $S = a\cdot b$ and the perimeter $P = 2\cdot(a + b)$ of the rectangle.

Begin4. Given the diameter d of a circle, find the length L of the circle: $L = \pi\cdot d$. Use 3.14 for a value of π.

Begin5. Given the edge a of a cube, find the volume $V = a^3$ and the surface area $S = 6\cdot a^2$ of the cube.

Begin6. The edges a, b, c of a right parallelepiped are given. Find the volume $V = a\cdot b\cdot c$ and the surface area $S = 2\cdot(a\cdot b + b\cdot c + a\cdot c)$ of the right parallelepiped.

Begin7. Given the radius R of a circle, find the length L of the circumference and the area S of the circle:

$$L = 2\cdot\pi\cdot R, \qquad S = \pi\cdot R^2.$$

Use 3.14 for a value of π.

Begin8. Given two numbers a and b, find their *average*: $(a + b)/2$.

Begin9. Given two nonnegative numbers a and b, find their *geometrical mean* (a square root of their product): $(a\cdot b)^{1/2}$.

Begin10. Two nonzero numbers are given. Find the sum, the difference, the product, and the quotient of their squares.

Begin11. Two nonzero numbers are given. Find the sum, the difference, the product, and the quotient of their absolute values.

Begin12. The legs a and b of a right triangle are given. Find the hypotenuse c and the perimeter P of the triangle:

$$c = (a^2 + b^2)^{1/2}, \qquad P = a + b + c.$$

Begin13. Given the radiuses R_1 and R_2 of two concentric circles ($R_1 > R_2$), find the areas S_1 and S_2 of the circles and the area S_3 of the ring bounded by the circles:

$$S_1 = \pi\cdot(R_1)^2, \qquad S_2 = \pi\cdot(R_2)^2, \qquad S_3 = S_1 - S_2.$$

Use 3.14 for a value of π.

Begin14. Given the length L of a circumference, find the radius R and the area S of the circle. Take into account that $L = 2\cdot\pi\cdot R$, $S = \pi\cdot R^2$. Use 3.14 for a value of π.

Begin15. Given the area S of a circle, find the diameter D and the length L of the circumference. Take into account that $L = \pi\cdot D$, $S = \pi\cdot D^2/4$. Use 3.14 for a value of π.

Begin16. Two points with the coordinates x_1 and x_2 are given on the real axis. Find the distance between these points: $|x_2 - x_1|$.

Begin17. Three points A, B, C are given on the real axis. Find the length of AC, the length of BC, and the sum of these lengths.

Begin18. Three points A, B, C are given on the real axis, the point C is located between the points A and B. Find the product of the length of AC and the length of BC.

Begin19. The coordinates (x_1, y_1) and (x_2, y_2) of two opposite vertices of a rectangle are given. Sides of the rectangle are parallel to coordinate axes. Find the perimeter and the area of the rectangle.

Begin20. The coordinates (x_1, y_1) and (x_2, y_2) of two points are given. Find the distance between the points:

$$((x_2 - x_1)^2 + (y_2 - y_1)^2)^{1/2}.$$

Begin21. The coordinates (x_1, y_1), (x_2, y_2), (x_3, y_3) of the triangle vertices are given. Find the perimeter and the area of the triangle using the formula for distance between two points in the plane (see Begin20). The area of a triangle with sides a, b, c can be found by *Heron's formula*:

$$S = (p\cdot(p - a)\cdot(p - b)\cdot(p - c))^{1/2},$$

where $p = (a + b + c)/2$ is the *half-perimeter*.

Begin22. Exchange the values of two given variables A and B. Output the new values of A and B.

Begin23. Variables A, B, C are given. Change values of the variables by moving the given value of A into the variable B, the given value of B into the variable C, and the given value of C into the variable A. Output the new values of A, B, C.

Begin24. Variables A, B, C are given. Change values of the variables by moving the given value of A into the variable C, the given value of C into the variable B, and the given value of B into the variable A. Output the new values of A, B, C.

Begin25. Given an independent variable x, find the value of a function $y = 3x^6 - 6x^2 - 7$.

Begin26. Given an independent variable x, find the value of a function $y = 4(x-3)^6 - 7(x-3)^3 + 2$.

Begin27. Given a number A, compute a power A^8 using three multiplying operators for computing A^2, A^4, A^8 sequentially. Output all obtained powers of the number A.

Begin28. Given a number A, compute a power A^{15} using five multiplying operators for computing A^2, A^3, A^5, A^{10}, A^{15} sequentially. Output all obtained powers of the number A.

Begin29. The angle value α in degrees ($0 \le \alpha < 360$) is given. Convert this value into radians. Take into account that $180° = \pi$ radians. Use 3.14 for a value of π.

Begin30. The angle value α in radians ($0 \le \alpha < 2 \cdot \pi$) is given. Convert this value into degrees. Take into account that $180° = \pi$ radians. Use 3.14 for a value of π.

Begin31. A Fahrenheit temperature T is given. Convert it into a centigrade temperature. The centigrade temperature T_C and the Fahrenheit temperature T_F are connected as:

$$T_C = (T_F - 32) \cdot 5/9.$$

Begin32. A centigrade temperature T is given. Convert it into a Fahrenheit temperature. The centigrade temperature T_C and the Fahrenheit temperature T_F are connected as:

$$T_C = (T_F - 32) \cdot 5/9.$$

Begin33. X kg of sweet cost A euro. Find the cost of 1 kg and Y kg of the sweets (positive numbers X, A, Y are given).

Begin34. X kg of chocolates cost A euro and Y kg of sugar candies cost B euro (positive numbers X, A, Y, B are given). Find the cost of 1 kg of the chocolates and the cost of 1 kg of the sugar candies. Also determine how many times the chocolates are more expensive than the sugar candies.

Begin35. A boat velocity in still water is V km/h, river flow velocity is U km/h ($U < V$). The boat goes along the lake during T_1 h and then goes against stream of the river during T_2 h. Positive numbers V, U, T_1, T_2 are given. Find the distance S covered by the boat (distance = time · velocity).

Begin36. The velocity of the first car is V_1 km/h, the velocity of the second car is V_2 km/h, the initial distance between the cars is S km. Find the distance between the cars after T hours provided that the distance is increasing. The required distance is equal to a sum of the initial distance and the total distance covered by the both cars (total distance = time · total velocity).

Begin37. The velocity of the first car is V_1 km/h, the velocity of the second car is V_2 km/h, the initial distance between the cars is S km. Find the distance between the cars after T hours provided that at the start time the distance is decreasing. This distance is equal to an absolute value of a difference between the initial distance and the total distance covered by the both cars.

Begin38. Solve a linear equation $A \cdot x + B = 0$ with given coefficients A and B (A is not equal to 0).

Begin39. Solve a *quadratic equation* $A\cdot x^2 + B\cdot x + C = 0$ with given coefficients A, B, C (A and the discriminant of the equation are positive). Output the smaller equation root and then the larger one. Roots of the quadratic equation may be found by formula

$$x_{1,\,2} = (-B \pm (D)^{1/2})/(2\cdot A),$$

where $D = B^2 - 4\cdot A\cdot C$ is a *discriminant*.

Begin40. Solve a *system of linear equations*

$$A_1\cdot x + B_1\cdot y = C_1,$$
$$A_2\cdot x + B_2\cdot y = C_2$$

with given coefficients A_1, B_1, C_1, A_2, B_2, C_2 provided that the system has the only solution. Use the following formulas:

$$x = (C_1\cdot B_2 - C_2\cdot B_1)/D, \qquad y = (A_1\cdot C_2 - A_2\cdot C_1)/D,$$

where $D = A_1\cdot B_2 - A_2\cdot B_1$.

2. Integers

All input and output data are integer (i. e. whole numbers) in this group. All numbers with fixed amount of digits (for example, two-digit number, three-digit number and so on) are assumed to be positive.

Integer1. A distance L is given in centimeters. Find the amount of full meters of this distance (1 m = 1000 cm). Use the operator of integer division.

Integer2. A weight M is given in kilograms. Find the amount of full tons of this weight (1 ton = 1000 kg). Use the operator of integer division.

Integer3. A file size is given in bytes. Find the amount of full Kbytes of this size (1 K = 1024 bytes). Use the operator of integer division.

Integer4. Two positive integers A and B are given ($A > B$). Segment of length A contains the greatest possible amount of inside segments of length B (without overlaps). Find the amount of segments B placed on the segment A. Use the operator of integer division.

Integer5. Two positive integers A and B are given ($A > B$). Segment of length A contains the greatest possible amount of inside segments of length B (without overlaps). Find the length of unused part of the segment A. Use the operator of taking the remainder after integer division.

Integer6. A two-digit integer is given. Output its left digit (a tens digit) and then its right digit (a ones digit). Use the operator of integer division for obtaining the tens digit and the operator of taking remainder for obtaining the ones digit.

Integer7. A two-digit integer is given. Find the sum and the product of its digits.

Integer8. A two-digit integer is given. Output an integer obtained from the given one by exchange of its digits.

Integer9. A three-digit integer is given. Using one operator of integer division find first digit of the given integer (a hundreds digit).

Integer10. A three-digit integer is given. Output its last digit (a ones digit) and then its middle digit (a tens digit).

Integer11. A three-digit integer is given. Find the sum and the product of its digits.

Integer12. A three-digit integer is given. Output an integer obtained from the given one by reading it from right to left.

Integer13. A three-digit integer is given. Output an integer obtained from the given one by moving its left digit to the right side.

Integer14. A three-digit integer is given. Output an integer obtained from the given one by moving its right digit to the left side.

Integer15. A three-digit integer is given. Output an integer obtained from the given one by exchange a tens digit and a hundreds digit (for example, 123 will be changed to 213).

Integer16. A three-digit integer is given. Output an integer obtained from the given one by exchange a ones digit and a tens digit (for example, 123 will be changed to 132).

Integer17. An integer greater than 999 is given. Using one operator of integer division and one operator of taking the remainder find a hundreds digit of the given integer.

Integer18. An integer greater than 999 is given. Using one operator of integer division and one operator of taking the remainder find a thousands digit of the given integer.

Integer19. From the beginning of the day *N* seconds have passed (*N* is integer). Find an amount of full minutes passed from the beginning of the day.

Integer20. From the beginning of the day *N* seconds have passed (*N* is integer). Find an amount of full hours passed from the beginning of the day.

Integer21. From the beginning of the day *N* seconds have passed (*N* is integer). Find an amount of seconds passed from the beginning of the last minute.

Integer22. From the beginning of the day *N* seconds have passed (*N* is integer). Find an amount of seconds passed from the beginning of the last hour.

Integer23. From the beginning of the day *N* seconds have passed (*N* is integer). Find an amount of full minutes passed from the beginning of the last hour.

Integer24. Days of week are numbered as: 0 — Sunday, 1 — Monday, 2 — Tuesday, …, 6 — Saturday. An integer *K* in the range 1 to 365 is given. Find the number of day of week for *K*-th day of year provided that in this year January 1 was Monday.

Integer25. Days of week are numbered as: 0 — Sunday, 1 — Monday, 2 — Tuesday, …, 6 — Saturday. An integer *K* in the range 1 to 365 is given. Find the number of day of week for *K*-th day of year provided that in this year January 1 was Thursday.

Integer26. Days of week are numbered as: 1 — Monday, 2 — Tuesday, …, 6 — Saturday, 7 — Sunday. An integer *K* in the range 1 to 365 is given. Find the number of day of week for *K*-th day of year provided that in this year January 1 was Tuesday.

Integer27. Days of week are numbered as: 1 — Monday, 2 — Tuesday, …, 6 — Saturday, 7 — Sunday. An integer *K* in the range 1 to 365 is given. Find the number of day of week for *K*-th day of year provided that in this year January 1 was Saturday.

Integer28. Days of week are numbered as: 1 — Monday, 2 — Tuesday, …, 6 — Saturday, 7 — Sunday. An integer *K* in the range 1 to 365 and an integer *N* in the range 1 to 7 are given. Find the number of day of week for *K*-th day of year provided that in this year January 1 was *N*-th day of week.

Integer29. Three positive integers *A*, *B*, *C* are given. A rectangle of the size $A \times B$ contains the greatest possible amount of inside squares with the side length *C* (without overlaps). Find the amount of squares placed on the rectangle and the area of unused part of the rectangle.

Integer30. Given a year (as a positive integer), find the respective number of the century. Note that, for example, 20th century began with the year 1901.

3. Logical expressions

All tasks in this group require determining the proposition as True or False. All numbers with fixed amount of digits (for example, two-digit number, three-digit number and so on) are assumed to be positive integers.

Boolean1. Given integer *A*, verify the following proposition: “The number *A* is positive”.

Boolean2. Given integer *A*, verify the following proposition: “The number *A* is odd”.

Boolean3. Given integer *A*, verify the following proposition: “The number *A* is even”.

Boolean4. Given two integers *A* and *B*, verify the following proposition: “The inequalities $A > 2$ and $B \le 3$ both are fulfilled”.

Boolean5. Given two integers *A* and *B*, verify the following proposition: “The inequality $A \ge 0$ is fulfilled or the inequality $B < -2$ is fulfilled”.

Boolean6. Given three integers *A*, *B*, *C*, verify the following proposition: “The double inequality $A < B < C$ is fulfilled”.

Boolean7. Given three integers *A*, *B*, *C*, verify the following proposition: “The number *B* is between *A* and *C*”.

Boolean8. Given two integers *A* and *B*, verify the following proposition: “Each of the numbers *A* and *B* is odd”.

Boolean9. Given two integers *A* and *B*, verify the following proposition: “At least one of the numbers *A* and *B* is odd”.

Boolean10. Given two integers *A* and *B*, verify the following proposition: “Exactly one of the numbers *A* and *B* is odd”.

Boolean11. Given two integers *A* and *B*, verify the following proposition: “The numbers *A* and *B* have equal parity”.

Boolean12. Given three integers *A*, *B*, *C*, verify the following proposition: “Each of the numbers *A*, *B*, *C* is positive”.

Boolean13. Given three integers *A*, *B*, *C*, verify the following proposition: “At least one of the numbers *A*, *B*, *C* is positive”.

Boolean14. Given three integers *A*, *B*, *C*, verify the following proposition: “Exactly one of the numbers *A*, *B*, *C* is positive”.

Boolean15. Given three integers *A*, *B*, *C*, verify the following proposition: “Exactly two of the numbers *A*, *B*, *C* are positive”.

Boolean16. Given a positive integer, verify the following proposition: “The integer is a two-digit even number”.

Boolean17. Given a positive integer, verify the following proposition: “The integer is a three-digit odd number”.

Boolean18. Verify the following proposition: “Among three given integers there is at least one pair of equal ones”.

Boolean19. Verify the following proposition: “Among three given integers there is at least one pair of opposite ones”.

Boolean20. Given a three-digit integer, verify the following proposition: “All digits of the number are different”.

Boolean21. Given a three-digit integer, verify the following proposition: “All digits of the number are in ascending order”.

Boolean22. Given a three-digit integer, verify the following proposition: “All digits of the number are in ascending or descending order”.

Boolean23. Given a four-digit integer, verify the following proposition: “The number is read equally both from left to right and from right to left”.

Boolean24. Three real numbers *A*, *B*, *C* are given (*A* is not equal to 0). By means of a *discriminant* $D = B^2 - 4 \cdot A \cdot C$, verify the following proposition: “The quadratic equation $A \cdot x^2 + B \cdot x + C = 0$ has real roots”.

Boolean25. Given two real numbers *x*, *y*, verify the following proposition: “The point with coordinates (*x*, *y*) is in the second coordinate quarter”.

Boolean26. Given two real numbers *x*, *y*, verify the following proposition: “The point with coordinates (*x*, *y*) is in the fourth coordinate quarter”.

Boolean27. Given two real numbers *x*, *y*, verify the following proposition: “The point with coordinates (*x*, *y*) is in the second or third coordinate quarter”.

Boolean28. Given two real numbers *x*, *y*, verify the following proposition: “The point with coordinates (*x*, *y*) is in the first or third coordinate quarter”.

Boolean29. Given real numbers x, y, x_1, y_1, x_2, y_2, verify the following proposition: "The point (x, y) is inside of the rectangle whose left top vertex is (x_1, y_1), right bottom vertex is (x_2, y_2), and sides are parallel to coordinate axes".

Boolean30. Given three integers a, b, c that are the sides of a triangle, verify the following proposition: "The triangle with sides a, b, c is equilateral".

Boolean31. Given three integers a, b, c that are the sides of a triangle, verify the following proposition: "The triangle with sides a, b, c is isosceles".

Boolean32. Given three integers a, b, c that are the sides of a triangle, verify the following proposition: "The triangle with sides a, b, c is a right triangle".

Boolean33. Given three integers a, b, c, verify the following proposition: "A triangle with the sides a, b, c exists".

Boolean34. Given coordinates x, y of a chessboard square (as integers in the range 1 to 8), verify the following proposition: "The chessboard square (x, y) is white". Note that the left bottom square (1, 1) is black.

Boolean35. Given coordinates x_1, y_1, x_2, y_2 of two chessboard squares (as integers in the range 1 to 8), verify the following proposition: "Both of the given chessboard squares have the same color".

Boolean36. Given coordinates x_1, y_1, x_2, y_2 of two chessboard squares (as integers in the range 1 to 8), verify the following proposition: "A rook can move from one square to another during one turn".

Boolean37. Given coordinates x_1, y_1, x_2, y_2 of two chessboard squares (as integers in the range 1 to 8), verify the following proposition: "A king can move from one square to another during one turn".

Boolean38. Given coordinates x_1, y_1, x_2, y_2 of two chessboard squares (as integers in the range 1 to 8), verify the following proposition: "A bishop can move from one square to another during one turn".

Boolean39. Given coordinates x_1, y_1, x_2, y_2 of two chessboard squares (as integers in the range 1 to 8), verify the following proposition: "A queen can move from one square to another during one turn".

Boolean40. Given coordinates x_1, y_1, x_2, y_2 of two chessboard squares (as integers in the range 1 to 8), verify the following proposition: "A knight can move from one square to another during one turn".

4. Conditional statement

If1. An integer is given. If the integer is positive then increase it by 1, otherwise do not change it. Output the obtained integer.

If2. An integer is given. If the integer is positive then increase it by 1, otherwise decrease it by 2. Output the obtained integer.

If3. An integer is given. If the integer is positive then increase it by 1, if the integer is negative then decrease it by 2, if the integer equals 0 then change it to 10. Output the obtained integer.

If4. Three integers are given. Find the amount of positive integers in the input data.

If5. Three integers are given. Find the amount of positive and amount of negative integers in the input data.

If6. Given two real numbers, output the larger value of them.

If7. Given two real numbers, output the order number of the smaller of them.

If8. Given two real numbers, output the larger value and then the smaller value of them.

If9. The values of two real variables *A* and *B* are given. Redistribute the values so that *A* and *B* have the smaller and the larger value respectively. Output the new values of the variables *A* and *B*.

If10. The values of two integer variables *A* and *B* are given. If the values are not equal then assign the sum of given values to each variable, otherwise assign zero value to each variable. Output the new values of the variables *A* and *B*.

If11. The values of two integer variables *A* and *B* are given. If the values are not equal then assign the larger value to each variable, otherwise assign zero value to each variable. Output the new values of the variables *A* and *B*.

If12. Given three real numbers, output the minimal value of them.

If13. Given three real numbers, output the value between the minimum and the maximum.

If14. Given three real numbers, output the minimal value and then the maximal value.

If15. Given three real numbers, output the sum of two largest values.

If16. The values of three real variables *A*, *B*, *C* are given. If the values are in ascending order then double them, otherwise replace the value of each variable by its opposite value. Output the new values of the variables *A*, *B*, *C*.

If17. The values of three real variables *A*, *B*, *C* are given. If the values are in ascending or descending order then double them, otherwise replace the value of each variable by its opposite value. Output the new values of the variables *A*, *B*, *C*.

If18. Three integers are given. One of them differs from two other equal integers. Output the order number of the integer that differs from the others.

If19. Four integers are given. One of them differs from three other equal integers. Output the order number of the integer that differs from the others.

If20. Three points *A*, *B*, *C* on the real axis are given. Determine whether *B* or *C* is closer to *A*. Output the nearest point and its distance from *A*.

If21. Integer coordinates of a point in the coordinate plane are given. If the point coincides with the origin of coordinates then output 0, otherwise if the point lies on the x-axis or y-axis then output 1 or 2 respectively. If the point does not lie on the coordinate axes then output 3.

If22. Given coordinates of a point that does not lie on the coordinate axes, find the number of a coordinate quarter containing the point.

If23. Given integer coordinates of three vertices of a rectangle whose sides are parallel to coordinate axes, find the coordinates of the fourth vertex of the rectangle.

If24. Given a real independent variable x, find the value of a real function f defined as:

$$f(x) = \begin{cases} 2\cdot\sin(x), \text{ if } x > 0, \\ 6 - x, \text{ if } x \le 0. \end{cases}$$

If25. Given an integer independent variable x, find the value of an integer function f defined as:

$$f(x) = \begin{cases} 2\cdot x, \text{ if } x < -2 \text{ or } x > 2, \\ -3\cdot x \text{ otherwise.} \end{cases}$$

If26. Given a real independent variable x, find the value of a real function f defined as:

$$f(x) = \begin{cases} -x, \text{ if } x \le 0, \\ x^2, \text{ if } 0 < x < 2, \\ 4, \text{ if } x \ge 2. \end{cases}$$

If27. Given a real independent variable x, find the value of an integer function f defined as:

$$f(x) = \begin{cases} 0, \text{ if } x < 0, \\ 1, \text{ if } x \text{ belongs to } [0, 1), [2, 3), \ldots, \\ -1, \text{ if } x \text{ belongs to } [1, 2), [3, 4), \ldots . \end{cases}$$

If28. Given a number of year (as a positive integer), find the amount of days in the year. Note that the length of year is 365 days for an ordinary year and 366 days for a leap year. A *leap year* is every year whose number is divisible by 4, as 1980, except centenary years that are not divisible by 400 (for example, 1300 and 1900 are ordinary years, 1200 and 2000 are leap years).

If29. Given an integer, output its description string as: “negative even number”, “zero number”, “positive odd number”, etc.

If30. An integer in the range 1 to 999 is given. Output its description string as: “two-digit even number”, “three-digit odd number”, etc.

5. Selection statement

Case1. An integer in the range 1 to 7 is given. Output the name of the respective day of week: 1 — “Monday”, 2 — “Tuesday”, …, 7 — “Sunday”.

Case2. Given an integer K, output the respective examination mark: 1 — “bad”, 2 — “unsatisfactory”, 3 — “mediocre”, 4 — “good”, 5 — “excellent”. If K is not in the range 1 to 5 then output string “error”.

Case3. A number of month is given (as an integer in the range 1 to 12): 1 — January, 2 — February, etc. Output the name of the respective season: “Winter”, “Spring”, “Summer”, “Autumn”.

Case4. A number of month is given (as an integer in the range 1 to 12): 1 — January, 2 — February, etc. Output the amount of days in the month for a non-leap year.

Case5. The arithmetic operations are numbered as: 1 — addition, 2 — subtraction, 3 — multiplication, 4 — division. The order number N of an operation and two real numbers A and B are given (N is an integer in the range 1 to 4, B is not equal to 0). Perform the operation with the operands A and B and output the result.

Case6. The units of length are numbered as: 1 — decimeter, 2 — kilometer, 3 — meter, 4 — millimeter, 5 — centimeter. The order number N of a unit of length and also the length L of a segment are given (N is an integer in the range 1 to 5, L is a real number). Output the length of the segment in meters.

Case7. The units of weight are numbered as: 1 — kilogram, 2 — milligram, 3 — gram, 4 — ton, 5 — centner (= 100 kilograms). The order number N of a unit of weight and the mass M of a solid are given (N is an integer in the range 1 to 5, M is a real number). Output the mass of the solid in kilograms.

Case8. Given two integers D (day) and M (month) representing a correct date of a non-leap year, output values D and M for the previous date.

Case9. Given two integers D (day) and M (month) representing a correct date of a non-leap year, output values D and M for the next date.

Case10. A robot can move in four directions (“N” — north, “W” — west, “S” — south, “E” — east) and perform three digital instructions: 0 — “move in the former direction”, 1 — “turn left”, −1 — “turn right”. A symbol C (an initial direction of the robot) and an integer N (an instruction) are given. Output the direction of the robot (as symbol) after performing the instruction.

Case11. A locator can be focused on the directions “N” (north), “W” (west), “S” (south), “E” (east) and perform three digital instructions: 1 — “turn left”, −1 — “turn right”, 2 — “turn on 180°”). A symbol C (an initial direction of the locator) and two integers N_1 and N_2 (instructions) are given. Output the direction of the locator (as symbol) after performing the instructions.

Case12. Elements of a circle are numbered as: 1 — radius R, 2 — diameter $D = 2·R$, 3 — length $L = 2·\pi·R$ of the circumference, 4 — area $S = \pi·R^2$. The order number of one element and its value (as a real number) are given. Output values of other elements in the same order. Use 3.14 for a value of π.

Case13. Elements of a right isosceles triangle are numbered as: 1 — leg a, 2 — hypotenuse $c = a·(2)^{1/2}$, 3 — altitude h drawn onto hypotenuse ($h = c/2$), 4 —

area $S = c\cdot h/2$. The order number of one element and its value (as a real number) are given. Output values of other elements in the same order.

Case14. Elements of an equilateral triangle are numbered as: 1 — side a, 2 — radius R_1 of inscribed circle ($R_1 = a\cdot(3)^{1/2}/6$), 3 — radius R_2 of circumscribed circle ($R_2 = 2\cdot R_1$), 4 — area $S = a^2\cdot(3)^{1/2}/4$. The order number of one element and its value (as a real number) are given. Output values of other elements in the same order.

Case15. The suits of playing cards are numbered as: 1 — spades, 2 — clubs, 3 — diamonds, 4 — hearts. Card values "Jack", "Queen", "King", "Ace" are numbered as 11, 12, 13, 14 respectively. A card value N (as an integer in the range 6 to 14) and a suit M (as an integer in the range 1 to 4) are given. Output the card description as: "six of diamonds", "queen of spades", etc.

Case16. Given an age in years (as an integer in the range 20 to 69), output its alphabetic equivalent as: "twenty years", "thirty-two years", "forty-one years", etc.

Case17. Given an order number of some training task (as an integer in the range 10 to 40), output its alphabetic equivalent as: "the eighteenth task", "the twenty-third task", "the thirtieth task", etc.

Case18. Given an integer in the range 100 to 999, output its alphabetic equivalent. For example, 100 — "one hundred", 256 — "two hundred and fifty-six", 814 — "eight hundred and fourteen", 901 — "nine hundred and one".

Case19. One of the Asian calendars uses 60-years periods divided into 12-years cycles, which are associated with a color: green, red, yellow, white, black. Each year in a cycle is connected with some animal: rat, cow, tiger, hare, dragon, snake, horse, sheep, monkey, hen, dog, pig. Given some year (as positive integer), output its name provided that 1984 is "The Green Rat`s year".

Case20. Given two integers D (day) and M (month) that represent a correct date, output the zodiacal name corresponding to this date: "Aquarius" 20.1–18.2, "Pisces" 19.2–20.3, "Aries" 21.3–19.4, "Taurus" 20.4–20.5, "Gemini" 21.5–21.6, "Cancer" 22.6–22.7, "Leo" 23.7–22.8, "Virgo" 23.8–22.9, "Libra" 23.9–22.10, "Scorpio" 23.10–22.11, "Sagittarius" 23.11–21.12, "Capricorn" 22.12–19.1.

6. Loop with the parameter

For1. Given integers K and N ($N > 0$), output the number K N times.

For2. Given two integers A and B ($A < B$), output in ascending order all integers in the range A to B (including A and B). Also output the amount N of these integers.

For3. Given two integers A and B ($A < B$), output in descending order all integers in the range A to B (excluding A and B). Also output the amount N of these integers.

For4. Given the price of 1 kg of sweets (as a real number), output the cost of 1, 2, …, 10 kg of these sweets.

For5. Given the price of 1 kg of sweets (as a real number), output the cost of 0.1, 0.2, …, 1 kg of these sweets.

For6. Given the price of 1 kg of sweets (as a real number), output the cost of 1.2, 1.4, …, 2 kg of these sweets.

For7. Given two integers A and B ($A < B$), find the sum of all integers in the range A to B inclusive.

For8. Given two integers A and B ($A < B$), find the product of all integers in the range A to B inclusive.

For9. Given two integers A and B ($A < B$), find the sum of squares of all integers in the range A to B inclusive.

For10. Given an integer N (> 0), find the value of a following sum (as a real number):

$$1 + 1/2 + 1/3 + \ldots + 1/N.$$

For11. Given an integer N (> 0), find the value of a following sum (as an integer):

$$N^2 + (N + 1)^2 + (N + 2)^2 + \ldots + (2 \cdot N)^2.$$

For12. Given an integer N (> 0), find the value of a following product of N factors:

$$1.1 \cdot 1.2 \cdot 1.3 \cdot \ldots .$$

For13. Given an integer N (> 0), find the value of the following expression of N terms with alternating signs:

$$1.1 - 1.2 + 1.3 - \ldots .$$

Do not use conditional statements.

For14. Given an integer N (> 0), compute N^2 by means of the formula

$$N^2 = 1 + 3 + 5 + \ldots + (2 \cdot N - 1).$$

Output the value of the sum after addition of each term. As a result, squares of all integers in the range 1 to N will be output.

For15. Given a real number A and an integer N (> 0), find A raised to the power N (i. e., the product of N values of A):

$$A^N = A \cdot A \cdot \ldots \cdot A.$$

For16. A real number A and an integer N (> 0) are given. Using one loop-statement compute and output powers A^K for all integer exponents K in the range 1 to N.

For17. A real number A and an integer N (> 0) are given. Using one loop-statement compute the sum

$$1 + A + A^2 + A^3 + \ldots + A^N.$$

For18. A real number A and an integer N (> 0) are given. Using one loop-statement compute the expression

$$1 - A + A^2 - A^3 + \ldots + (-1)^N \cdot A^N.$$

Do not use conditional statements.

For19. Given an integer N (> 0), find the value of a following product:

$$N! = 1\cdot 2\cdot \ldots \cdot N$$

(N–*factorial*). To avoid the integer overflow, compute the product using a real variable and output the result as a real number.

For20. An integer N (> 0) is given. Using one loop-statement compute the sum

$$1! + 2! + 3! + \ldots + N!,$$

where $N!$ (N–*factorial*) is the product of all integers in the range 1 to N: $N! = 1\cdot 2\cdot \ldots \cdot N$. To avoid the integer overflow, compute the sum using real variables and output the result as a real number.

For21. An integer N (> 0) is given. Using one loop-statement compute the sum

$$1 + 1/(1!) + 1/(2!) + 1/(3!) + \ldots + 1/(N!),$$

where $N!$ (N–*factorial*) is the product of all integers in the range 1 to N: $N! = 1\cdot 2\cdot \ldots \cdot N$. The result is an approximate value of the constant $e = \exp(1)$.

For22. A real number X and an integer N (> 0) are given. Compute the expression

$$1 + X + X^2/(2!) + \ldots + X^N/(N!)$$

($N! = 1\cdot 2\cdot \ldots \cdot N$). The result is an approximate value of $\exp(X)$.

For23. A real number X and an integer N (> 0) are given. Compute the expression

$$X - X^3/(3!) + X^5/(5!) - \ldots + (-1)^N\cdot X^{2\cdot N+1}/((2\cdot N+1)!)$$

($N! = 1\cdot 2\cdot \ldots \cdot N$). The result is an approximate value of $\sin(X)$.

For24. A real number X and an integer N (> 0) are given. Compute the expression

$$1 - X^2/(2!) + X^4/(4!) - \ldots + (-1)^N\cdot X^{2\cdot N}/((2\cdot N)!)$$

($N! = 1\cdot 2\cdot \ldots \cdot N$). The result is an approximate value of $\cos(X)$.

For25. A real number X ($|X| < 1$) and an integer N (> 0) are given. Compute the expression

$$X - X^2/2 + X^3/3 - \ldots + (-1)^{N-1}\cdot X^N/N.$$

The result is an approximate value of $\ln(1 + X)$.

For26. A real number X ($|X| < 1$) and an integer N (> 0) are given. Compute the expression

$$X - X^3/3 + X^5/5 - \ldots + (-1)^N\cdot X^{2\cdot N+1}/(2\cdot N+1).$$

The result is an approximate value of $\operatorname{atan}(X)$.

For27. A real number X ($|X| < 1$) and an integer N (> 0) are given. Compute the expression

$$X + 1\cdot X^3/(2\cdot 3) + 1\cdot 3\cdot X^5/(2\cdot 4\cdot 5) + \ldots +$$
$$+ 1\cdot 3\cdot \ldots\cdot(2\cdot N-1)\cdot X^{2\cdot N+1}/(2\cdot 4\cdot \ldots\cdot(2\cdot N)\cdot(2\cdot N+1)).$$

The result is an approximate value of $\operatorname{asin}(X)$.

For28. A real number X ($|X| < 1$) and an integer N (> 0) are given. Compute the expression

$$1 + X/2 - 1\cdot X^2/(2\cdot 4) + 1\cdot 3\cdot X^3/(2\cdot 4\cdot 6) - \ldots +$$
$$+ (-1)^{N-1}\cdot 1\cdot 3\cdot \ldots\cdot(2\cdot N-3)\cdot X^N/(2\cdot 4\cdot \ldots\cdot(2\cdot N)).$$

The result is an approximate value of the square root of $1 + X$.

For29. An integer N (> 1) and two points A, B ($A < B$) on the real axis are given. The segment $[A, B]$ is divided into N sub-segments of equal length. Output the length H of each sub-segment and then output the sequence of points

$$A,\quad A + H,\quad A + 2\cdot H,\quad A + 3\cdot H,\quad \ldots,\quad B,$$

which forms a partition of the segment $[A, B]$.

For30. An integer N (> 1) and two points A, B ($A < B$) on the real axis are given. The segment $[A, B]$ is divided into N sub-segments of equal length. Output the length H of each sub-segment and then output the values of a function $F(X) = 1 - \sin(X)$ at points dividing the segment $[A, B]$:

$$F(A),\quad F(A + H),\quad F(A + 2\cdot H),\quad \ldots,\quad F(B).$$

For31. An integer N (> 0) is given. A sequence of real numbers A_K is defined as:

$$A_0 = 2,\qquad A_K = 2 + 1/A_{K-1},\quad K = 1, 2, \ldots .$$

Output terms $A_1, A_2, \ldots, A_N$ of the sequence.

For32. An integer N (> 0) is given. A sequence of real numbers A_K is defined as:

$$A_0 = 1,\qquad A_K = (A_{K-1} + 1)/K,\quad K = 1, 2, \ldots .$$

Output terms $A_1, A_2, \ldots, A_N$ of the sequence.

For33. An integer N (> 0) is given. An integer-valued sequence of the *Fibonacci numbers* F_K is defined as:

$$F_1 = 1,\qquad F_2 = 1,\qquad F_K = F_{K-2} + F_{K-1},\quad K = 3, 4, \ldots .$$

Output terms $F_1, F_2, \ldots, F_N$ of the sequence.

For34. An integer N (> 1) is given. A sequence of real numbers A_K is defined as:

$$A_1 = 1,\qquad A_2 = 2,\qquad A_K = (A_{K-2} + 2\cdot A_{K-1})/3,\quad K = 3, 4, \ldots .$$

Output terms $A_1, A_2, \ldots, A_N$ of the sequence.

For35. An integer N (> 2) is given. A sequence of integers A_K is defined as:

$$A_1 = 1,\qquad A_2 = 2,\qquad A_3 = 3,\qquad A_K = A_{K-1} + A_{K-2} - 2\cdot A_{K-3},\quad K = 4, 5, \ldots .$$

Output terms $A_1, A_2, \ldots, A_N$ of the sequence.

Nested loops

For36. Given positive integers N and K, find the sum

$$1^K + 2^K + \ldots + N^K.$$

To avoid the integer overflow, compute the sum using real variables and output the result as a real number.

For37. Given an integer N (> 0), find the sum

$$1^1 + 2^2 + \ldots + N^N.$$

To avoid the integer overflow, compute the sum using real variables and output the result as a real number.

For38. Given an integer N (> 0), find the sum

$$1^N + 2^{N-1} + \ldots + N^1.$$

To avoid the integer overflow, compute the sum using real variables and output the result as a real number.

For39. Positive integers A and B ($A < B$) are given. Output all integers in the range A to B, with an integer of a value K being output K times (for example, the number 3 must be output 3 times).

For40. Integers A and B ($A < B$) are given. Output all integers in the range A to B, with the number A being output once, the number $A + 1$ being output twice, and so on.

7. Loop with the condition

While1. Two positive real numbers A and B ($A > B$) are given. A segment of length A contains the greatest possible amount of segments of length B (without overlaps). Not using multiplication and division, find the length of unused part of the segment A.

While2. Two positive real numbers A and B ($A > B$) are given. A segment of length A contains the greatest possible amount of segments of length B (without overlaps). Not using multiplication and division, find the amount of segments B, which are placed on the segment A.

While3. Two positive integers N and K are given. Using addition and subtraction only, find a quotient of the integer division N on K and also a remainder after this division.

While4. An integer N (> 0) is given. If it equals 3 raised to some integer power then output True, otherwise output False.

While5. Given an integer N (> 0) that equals 2 raised to some integer power: $N = 2^K$, find the exponent K of the power.

While6. Given an integer N (> 0), compute the *double factorial of N*:

$$N!! = N\cdot(N-2)\cdot(N-4)\cdot\ldots,$$

where the last factor equals 2 if N is an even number, and 1 otherwise. To avoid the integer overflow, compute the double factorial using a real variable and output the result as a real number.

While7. Given an integer N (> 0), find the smallest positive integer K such that its square is greater than N: $K^2 > N$. Do not use the operation of extracting a root.

While8. Given an integer N (> 0), find the largest integer K such that its square is not greater than N: $K^2 \le N$. Do not use the operation of extracting a root.

While9. Given an integer N (> 1), find the smallest integer K such that the inequality $3^K > N$ is fulfilled.

While10. Given an integer N (> 1), find the largest integer K such that the inequality $3^K < N$ is fulfilled.

While11. An integer N (> 1) is given. Find the smallest integer K such that the sum $1 + 2 + \ldots + K$ is greater than or equal to N. Output K and the corresponding sum.

While12. An integer N (> 1) is given. Find the largest integer K such that the sum $1 + 2 + \ldots + K$ is less than or equal to N. Output K and the corresponding sum.

While13. A real number A (> 1) is given. Find the smallest integer K such that the sum $1 + 1/2 + \ldots + 1/K$ is greater than A. Output K and the corresponding sum.

While14. A real number A (> 1) is given. Find the largest integer K such that the sum $1 + 1/2 + \ldots + 1/K$ is less than A. Output K and the corresponding sum.

While15. A principal of 1000 euro is invested at a rate of P percent compounded annually. A real number P is given, $0 < P < 25$. Find, how many years K it will take for an investment to exceed 1100 euro. Output K (as an integer) and the compound amount S of the principal at the end of K years (as a real number).

While16. The skier began trainings having run 10 km. Each next day he increased the run distance by P percent from the distance of the last day. A real number P is given, $0 < P < 50$). Find, how many days K it will take for a total run to exceed 200 km. Output K (as an integer) and the total run S (as a real number).

While17. Given an integer N (> 0), output all digits of the number N starting from the right digit (a ones digit). Use the operators of integer division and taking the remainder after integer division.

While18. Given an integer N (> 0), find the amount and the sum of its digits. Use the operators of integer division and taking the remainder after integer division.

While19. An integer N (> 0) is given. Output an integer obtained from the given one by reading it from right to left. Use the operators of integer division and taking the remainder after integer division.

While20. An integer N (> 0) is given. Determine whether its decimal representation contains a digit ”2” or not, and output True or False respectively. Use the operators of integer division and taking the remainder after integer division.

While21. An integer N (> 0) is given. Determine whether its decimal representation contains odd digits or not, and output True or False respectively. Use the operators of integer division and taking the remainder after integer division.

While22. An integer N (> 1) is given. If it is a *prime number*, i. e., it has not positive divisors except 1 and itself, then output True, otherwise output False.

While23. Two positive integers A and B are given. Find their *greatest common divisor* (GCD) using the *Euclidean algorithm*:

$$\mathrm{GCD}(A, B) = \mathrm{GCD}(B, A \bmod B), \quad \text{if } B \neq 0; \qquad \mathrm{GCD}(A, 0) = A,$$

where “mod” denotes the operator of taking the remainder after integer division.

While24. An integer N (> 1) is given. An integer-valued sequence of the *Fibonacci numbers* F_K is defined as:

$$F_1 = 1, \qquad F_2 = 1, \qquad F_K = F_{K-2} + F_{K-1}, \quad K = 3, 4, \ldots .$$

Determine whether N is a Fibonacci number or not, and output True or False respectively.

While25. Given an integer N (> 1), find the first Fibonacci number greater than N (see the *Fibonacci numbers* definition in While24).

While26. Given an integer N (> 1) that is a Fibonacci number: $N = F_K$, output previous and next Fibonacci numbers: F_{K-1} and F_{K+1} (see the *Fibonacci numbers* definition in While24).

While27. Given an integer N (> 1) that is a Fibonacci number: $N = F_K$, find its order number K (see the *Fibonacci numbers* definition in While24).

While28. A real number ε (> 0) is given. A sequence of real numbers A_K is defined as:

$$A_1 = 2, \qquad A_K = 2 + 1/A_{K-1}, \quad K = 2, 3, \ldots .$$

Find the first index K such that the inequality $|A_K - A_{K-1}| < \varepsilon$ is fulfilled. Output the index K and the terms A_{K-1} and A_K.

While29. A real number ε (> 0) is given. A sequence of real numbers A_K is defined as:

$$A_1 = 1, \qquad A_2 = 2, \qquad A_K = (A_{K-2} + 2 \cdot A_{K-1})/3, \quad K = 3, 4, \ldots .$$

Find the first index K such that the inequality $|A_K - A_{K-1}| < \varepsilon$ is fulfilled. Output the index K and the terms A_{K-1} and A_K.

While30. Three positive real numbers A, B, C are given. A rectangle of size $A \times B$ contains the greatest possible amount of squares with side C (without overlaps). Find the amount of squares placed on the rectangle. Do not use the operators of multiplication and division.

8. Procedures and functions

8.1. Procedures with numeric parameters

Proc1. Write a procedure PowerA3(A, B) that computes the third degree of a real number A and assigns the result to a real variable B (A is an input parameter, B is an output parameter). Using this procedure, find the third degree of five given real numbers.

Proc2. Write a procedure PowerA234(A, B, C, D) that computes the second, the third, and the fourth degrees of a real number A and assigns the results to real variables B, C, and D respectively (A is an input parameter, B, C, D are output parameters). Using this procedure, find the second, the third, and the fourth degrees of five given real numbers.

Proc3. Write a procedure Mean(X, Y, *AMean*, *GMean*) that computes the *arithmetical mean* $AMean = (X+Y)/2$ and the *geometrical mean* $GMean = (X \cdot Y)^{1/2}$ of two positive numbers X and Y (X and Y are input parameters, *AMean* and *GMean* are output parameters; all parameters are the real-valued ones). Using this procedure, find the arithmetical mean and the geometrical mean of pairs (A, B), (A, C), (A, D) provided that real numbers A, B, C, D are given.

Proc4. Write a procedure TrianglePS(a, P, S) that computes the perimeter $P = 3\cdot a$ and the area $S = a^2\cdot(3)^{1/2}/4$ of an equilateral triangle with the side a (a is an input parameter, P and S are output parameters, all parameters are the real-valued ones). Using this procedure, find the perimeters and the areas of three triangles with the given lengths of the sides.

Proc5. Write a procedure RectPS(x_1, y_1, x_2, y_2, P, S) that computes the perimeter P and the area S of a rectangle whose opposite vertices have coordinates (x_1, y_1) and (x_2, y_2) and sides are parallel to coordinate axes (x_1, y_1, x_2, y_2 are input parameters, P and S are output parameters, all parameters are the real-valued ones). Using this procedure, find the perimeters and the areas of three rectangles with the given opposite vertices.

Proc6. Write a procedure DigitCountSum(K, C, S) that finds the amount C of digits in the decimal representation of a positive integer K and also the sum S of its digits (K is an input parameter, C and S are output parameters, all parameters are the integer ones). Using this procedure, find the amount and the sum of digits for each of five given integers.

Proc7. Write a procedure InvDigits(K) that inverts the order of digits of a positive integer K (K is an input and output integer parameter). Using this procedure, invert the order of digits for each of five given integers.

Proc8. Write a procedure AddRightDigit(D, K) that adds a digit D to the right side of the decimal representation of a positive integer K (D is an input integer parameter with the value in the range 0 to 9, K is an input and output integer parameter). Having input an integer K and two one-digit numbers D_1, D_2 and using two calls of this procedure, sequentially add the given digits D_1, D_2 to the right side of the given K and output the result of each adding.

Proc9. Write a procedure AddLeftDigit(D, K) that adds a digit D to the left side of the decimal representation of a positive integer K (D is an input integer parameter with the value in the range 0 to 9, K is an input and output integer parameter). Having input an integer K and two one-digit numbers D_1, D_2 and using two calls of this procedure, sequentially add the given digits D_1, D_2 to the left side of the given K and output the result of each adding.

Proc10. Write a procedure Swap(X, Y) that exchanges the values of variables X and Y (X and Y are input and output real-valued parameters). Having input integers A, B, C, D and using three calls of this procedure, sequentially exchange the values of the pairs A and B, C and D, B and C. Output the new values of A, B, C, D.

Proc11. Write a procedure Minmax(X, Y) that exchanges, if necessary, the values of two variables X and Y so that $X \le Y$ (X and Y are input and output real-valued parameters). Using four calls of this procedure, find the minimal value and the maximal value among given real numbers A, B, C, D.

Proc12. Write a procedure SortInc3(A, B, C) that interchanges, if necessary, the values of three variables A, B, C so that $A \le B \le C$ (A, B, C are input and output real-valued parameters). Using this procedure, sort each of two given triples of real numbers (A_1, B_1, C_1) and (A_2, B_2, C_2) in ascending order.

Proc13. Write a procedure SortDec3(A, B, C) that interchanges, if necessary, the values of three variables A, B, C so that $A \geq B \geq C$ (A, B, C are input and output real-valued parameters). Using this procedure, sort each of two given triples of real numbers (A_1, B_1, C_1) and (A_2, B_2, C_2) in descending order.

Proc14. Write a procedure ShiftRight3(A, B, C) that performs a *right cyclic shift* by assigning the initial values of A, B, C to variables B, C, A respectively (A, B, C are input and output real-valued parameters). Using this procedure, perform the right cyclic shift for each of two given triples of real numbers: (A_1, B_1, C_1) and (A_2, B_2, C_2).

Proc15. Write a procedure ShiftLeft3(A, B, C) that performs a *left cyclic shift* by assigning the initial values of A, B, C to variables C, A, B respectively (A, B, C are input and output real-valued parameters). Using this procedure, perform the left cyclic shift for each of two given triples of real numbers: (A_1, B_1, C_1) and (A_2, B_2, C_2).

8.2. Functions with numeric parameters

Proc16. Write an integer function Sign(X) that returns the following value:

$$-1, \quad \text{if } X < 0; \qquad 0, \quad \text{if } X = 0; \qquad 1, \quad \text{if } X > 0$$

(X is a real-valued parameter). Using this function, evaluate an expression Sign(A) + Sign(B) for given real numbers A and B.

Proc17. Write an integer function RootCount(A, B, C) that returns the amount of roots of the quadratic equation $A\cdot x^2 + B\cdot x + C = 0$ (A, B, C are real-valued parameters, $A \neq 0$). Using this function, find the amount of roots for each of three quadratic equations with given coefficients. Note that the amount of roots is determined by the value of a *discriminant*:

$$D = B^2 - 4\cdot A\cdot C.$$

Proc18. Write a real-valued function CircleS(R) that returns the area of a circle of radius R (R is a real number). Using this function, find the areas of three circles of given radiuses. Note that the area of a circle of radius R can be found by formula $S = \pi\cdot R^2$. Use 3.14 for a value of π.

Proc19. Write a real-valued function RingS(R_1, R_2) that returns the area of a ring bounded by a concentric circles of radiuses R_1 and R_2 (R_1 and R_2 are real numbers, $R_1 > R_2$). Using this function, find the areas of three rings with given outer and inner radiuses. Note that the area of a circle of radius R can be found by formula $S = \pi\cdot R^2$. Use 3.14 for a value of π.

Proc20. Write a real-valued function TriangleP(a, h) that returns the perimeter of an isosceles triangle with given base a and altitude h (a and h are real numbers). Using this function, find the perimeters of three triangles with given bases and altitudes. Note that the leg b of an isosceles triangle can be found by the *Pythagorean theorem*:

$$b^2 = (a/2)^2 + h^2.$$

Proc21. Write an integer function SumRange(A, B) that returns a sum of all integers in the range A to B inclusively (A and B are integers). In the case of $A > B$ the function returns 0. Using this function, find a sum of all integers in the range A to B and in the range B to C provided that integers A, B, C are given.

Proc22. Write a real-valued function Calc(A, B, Op) that performs an arithmetic operation over nonzero real numbers A and B and returns the result value. An arithmetic operation is determined by integer parameter Op as follows: 1 — subtraction, 2 — multiplication, 3 — division, and addition otherwise. Having input real numbers A, B and integers N_1, N_2, N_3 and using this function, perform over given A, B three operations determined by given N_1, N_2, N_3. Output the result value of each operation.

Proc23. Write an integer function Quarter(x, y) that returns the number of a coordinate quarter containing a point with nonzero real-valued coordinates (x, y). Using this function, find the numbers of coordinate quarters containing each of three points with given nonzero coordinates.

Proc24. Write a logical function Even(K) that returns True, if an integer parameter K is an even number, and False otherwise. Using this function, find the amount of even numbers in a given sequence of 10 integers.

Proc25. Write a logical function IsSquare(K) that returns True, if an positive integer parameter K is a square of some integer, and False otherwise. Using this function, find the amount of squares in a given sequence of 10 positive integers.

Proc26. Write a logical function IsPower5(K) that returns True, if an positive integer parameter K is equal to 5 raised to some integer power, and False otherwise. Using this function, find the amount of powers of base 5 in a given sequence of 10 positive integers.

Proc27. Write a logical function IsPowerN(K, N) that returns True, if an positive integer parameter K is equal to N (> 1) raised to some integer power, and False otherwise. Having input an integer N (> 1) and a sequence of 10 positive integers and using this function, find the amount of powers of base N in the given sequence.

Proc28. Write a logical function IsPrime(N) that returns True, if an integer parameter N (> 1) is a prime number, and False otherwise. Using this function, find the amount of prime numbers in a given sequence of 10 integers greater than 1. Note that an integer (> 1) is called a *prime number* if it has not positive divisors except 1 and itself.

Proc29. Write an integer function DigitCount(K) that returns the amount of digits in the decimal representation of a positive integer K. Using this function, find the amount of digits for each of five given positive integers.

Proc30. Write an integer function DigitN(K, N) that returns the N-th digit in the decimal representation of a positive integer K provided that the digits are numbered from right to left. If the amount of digits is less than N then the function returns -1. Using this function, output sequentially 1st, 2nd, 3rd, 4th, 5th digit for each of five given positive integers K_1, K_2, …, K_5.

Proc31. Write a logical function IsPalindrome(K) that returns True, if the decimal representation of a positive parameter K is a *palindrome* (i. e., it is read equally both from left to right and from right to left), and False otherwise. Using this function, find the amount of palindromes in a given sequence of 10 positive integers.

Proc32. Write a real-valued function DegToRad(D) that converts the angle value D in degrees into the one in radians (D is a real number, $0 \le D < 360$). Note that $180° = \pi$ radians and use 3.14 for a value of π. Using this function, convert five given angles from degrees into radians.

Proc33. Write a real-valued function RadToDeg(R) that converts the angle value R in radians into the one in degrees (R is a real number, $0 \le R < 2\cdot\pi$). Note that $180° = \pi$ radians and use 3.14 for a value of π. Using this function, convert five given angles from radians into degrees.

Proc34. Write a real-valued function Fact(N) that returns a *factorial* of a positive integer N: $N! = 1\cdot2\cdot\ldots\cdot N$ (the real return type allows to avoid the integer overflow during the calculation of the factorials for large values of the parameter N). Using this function, find the factorials of five given integers.

Proc35. Write a real-valued function Fact2(N) that returns a *double factorial* $N!!$:

$$N!! = 1\cdot3\cdot5\cdot\ldots\cdot N, \quad \text{if } N \text{ is an odd number;}$$
$$N!! = 2\cdot4\cdot6\cdot\ldots\cdot N \quad \text{otherwise}$$

(N is a positive integer; the real return type allows to avoid the integer overflow during the calculation of the double factorials for large values of N). Using this function, find the double factorials of five given integers.

Proc36. Write an integer function Fib(N) that returns the value of N-th term of the sequence of the Fibonacci numbers. The *Fibonacci numbers* F_K are defined as follows:

$$F_1 = 1, \qquad F_2 = 1, \qquad F_K = F_{K-2} + F_{K-1}, \quad K = 3, 4, \ldots .$$

Using this function, find five Fibonacci numbers with given order numbers N_1, N_2, …, N_5.

8.3. Additional tasks

Proc37. Write a real-valued function Power1(A, B) that returns the power A^B calculated by the formula $A^B = \exp(B\cdot\ln(A))$ (A and B are real-valued parameters). In the case of zero-valued or negative parameter A the function returns 0. Having input real numbers P, A, B, C and using this function, find the powers A^P, B^P, C^P.

Proc38. Write a real-valued function Power2(A, N) that returns the power A^N calculated by the following formulas:

$A^0 = 1$;
$A^N = A·A·…·A$ (N factors),   if $N > 0$;
$A^N = 1/(A·A·…·A)$ ($|N|$ factors),   if $N < 0$
(A is a real-valued parameter, N is an integer parameter). Having input a real number A and integers K, L, M and using this function, find the powers A^K, A^L, A^M.

Proc39. Using the Power1 and Power2 functions (see Proc37 and Proc38), write a real-valued function Power3(A, B) that returns the power A^B calculated as follows (A and B are real-valued parameters): if B has a zero-valued fractional part then the function Power2(A, N) is called (an *integer* variable N is equal to B), otherwise the function Power1(A, B) is called. Having input real numbers P, A, B, C and using the Power3 function, find the powers A^P, B^P, C^P.

Proc40. Write a real-valued function Exp1(x, ε) (x and ε are real numbers, $ε > 0$) that returns an approximate value of the function exp(x) defined as follows:

$$\exp(x) = 1 + x + x^2/(2!) + x^3/(3!) + … + x^n/(n!) + …$$

($n! = 1·2·…·n$). Stop adding new terms to the sum when the value of the next term will be less than ε. Using this function, find the approximate values of the function exp(x) at a given point x for six given ε.

Proc41. Write a real-valued function Sin1(x, ε) (x and ε are real numbers, $ε > 0$) that returns an approximate value of the function sin(x) defined as follows:

$$\sin(x) = x − x^3/(3!) + x^5/(5!) − … + (−1)^n·x^{2·n+1}/((2·n+1)!) + … .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function sin(x) at a given point x for six given ε.

Proc42. Write a real-valued function Cos1(x, ε) (x and ε are real numbers, $ε > 0$) that returns an approximate value of the function cos(x) defined as follows:

$$\cos(x) = 1 − x^2/(2!) + x^4/(4!) − … + (−1)^n·x^{2·n}/((2·n)!) + … .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function cos(x) at a given point x for six given ε.

Proc43. Write a real-valued function Ln1(x, ε) (x and ε are real numbers, $|x| < 1$, $ε > 0$) that returns an approximate value of the function ln($1 + x$) defined as follows:

$$\ln(1 + x) = x − x^2/2 + x^3/3 − … + (−1)^n·x^{n+1}/(n+1) + … .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function ln($1 + x$) at a given point x for six given ε.

Proc44. Write a real-valued function Atan1(x, ε) (x and ε are real numbers, $|x| < 1$, $ε > 0$) that returns an approximate value of the function atan(x) defined as follows:

$$\text{atan}(x) = x − x^3/3 + x^5/5 − … + (−1)^n·x^{2·n+1}/(2·n+1) + … .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function atan(x) at a given point x for six given ε.

Proc45. Write a real-valued function Power4(x, a, ε) (x, a, ε are real numbers, $|x| < 1$, a, $\varepsilon > 0$) that returns an approximate value of the function $(1 + x)^a$ defined as:

$$(1 + x)^a = 1 + a\cdot x + a\cdot(a-1)\cdot x^2/(2!) + \ldots + a\cdot(a-1)\cdot\ldots\cdot(a-n+1)\cdot x^n/(n!) + \ldots .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function $(1 + x)^a$ at a given point x for a given exponent a and six given ε.

Proc46. Write an integer function GCD2(A, B) that returns the *greatest common divisor* (GCD) of two positive integers A and B. Use the *Euclidean algorithm*:

$$\mathrm{GCD}(A, B) = \mathrm{GCD}(B, A \bmod B), \quad \text{if } B \neq 0; \qquad \mathrm{GCD}(A, 0) = A,$$

where “mod” denotes the operator of taking the remainder after integer division. Using this function, find the greatest common divisor for each of pairs (A, B), (A, C), (A, D) provided that integers A, B, C, D are given.

Proc47. Using the GCD2 function from the task Proc46, write a procedure Frac1(a, b, p, q), that simplifies a fraction a/b to the irreducible form p/q (a and b are input integer parameters, p and q are output integer parameters). The sign of a resulting fraction p/q is assigned to its numerator, so $q > 0$. Using this procedure, find the numerator and the denominator for each of irreducible fractions $a/b + c/d$, $a/b + e/f$, $a/b + g/h$ provided that integers a, b, c, d, e, f, g, h are given.

Proc48. Taking into account that the *least common multiple* of two positive integers A and B equals $A\cdot(B/\mathrm{GCD}(A, B))$, where GCD($A$, B) is the greatest common divisor of A and B, and using the GCD2 function from the task Proc46, write an integer function LCM2(A, B) that returns the least common multiple of A and B. Using this function, find the least common multiple for each of pairs (A, B), (A, C), (A, D) provided that integers A, B, C, D are given.

Proc49. Taking into account the formula GCD(A, B, C) = GCD(GCD(A, B), C) and using the GCD2 function from the task Proc46, write an integer function GCD3(A, B, C) that returns the greatest common divisor of three positive integers A, B, C. Using this function, find the greatest common divisor for each of triples (A, B, C), (A, C, D), (B, C, D) provided that integers A, B, C, D are given.

Proc50. Write a procedure TimeToHMS(T, H, M, S) that converts a time interval T (in seconds) into the “hours H, minutes M, seconds S” format (T is an input integer parameter, H, M, S are output integer parameters). Using this procedure, find the amount of hours, minutes and seconds for each of five given time intervals T_1, T_2, …, T_5.

Proc51. Write a procedure IncTime(H, M, S, T) that increases a time interval in hours H, minutes M, seconds S on T seconds (H, M, S are input and output positive integer parameters, T is an input positive integer parameter). Having input hours *H*, minutes *M*, seconds *S* (as integers) and an integer *T* and using the IncTime procedure, increase the given time interval on *T* seconds and output new values of *H*, *M*, *S*.

Proc52. Write a logical function IsLeapYear(*Y*) that returns True if a year *Y* (a positive integer parameter) is a leap year, and False otherwise. Output the return values of this function for five given values of the parameter *Y*. Note that a year is a *leap year* if it is divisible by 4 except for years that are divisible by 100 and are not divisible by 400.

Proc53. Using the IsLeapYear function from the task Proc52, write an integer function MonthDays(*M*, *Y*) that returns the amount of days for *M*-th month of year *Y* (*M* and *Y* are integers, $1 \le M \le 12$, $Y > 0$). Output the return value of this function for a given year *Y* and each of given months M_1, M_2, M_3.

Proc54. Using the MonthDays function from the task Proc53, write a procedure PrevDate(*D*, *M*, *Y*) that changes a correct date, represented at the "day *D*, month number *M*, year *Y*" format, to a previous one (*D*, *M*, *Y* are input and output integer parameters). Apply this procedure to three given dates and output resulting previous ones.

Proc55. Using the MonthDays function from the task Proc53, write a procedure NextDate(*D*, *M*, *Y*) that changes a correct date, represented at the "day *D*, month number *M*, year *Y*" format, to a next one (*D*, *M*, *Y* are input and output integer parameters). Apply this procedure to three given dates and output resulting next ones.

Proc56. Write a real-valued function Leng(x_A, y_A, x_B, y_B) that returns the length of a segment *AB* with given coordinates of its endpoints:

$$|AB| = ((x_A - x_B)^2 + (y_A - y_B)^2)^{1/2}$$

(x_A, y_A, x_B, y_B are real-valued parameters). Using this function, find the lengths of segments *AB*, *AC*, *AD* provided that coordinates of points *A*, *B*, *C*, *D* are given.

Proc57. Using the Leng function from the task Proc56, write a real-valued function Perim(x_A, y_A, x_B, y_B, x_C, y_C) that returns the perimeter of a triangle *ABC* with given coordinates of its vertices (x_A, y_A, x_B, y_B, x_C, y_C are real-valued parameters). Using the Perim function, find the perimeters of triangles *ABC*, *ABD*, *ACD* provided that coordinates of points *A*, *B*, *C*, *D* are given.

Proc58. Using the Leng and Perim functions from the tasks Proc56 and Proc57, write a real-valued function Area(x_A, y_A, x_B, y_B, x_C, y_C) that returns the area of a triangle *ABC*:

$$S_{ABC} = (p\cdot(p-|AB|)\cdot(p-|AC|)\cdot(p-|BC|))^{1/2},$$

where *p* is the *half-perimeter*. Using the Area function, find the areas of triangles *ABC*, *ABD*, *ACD* provided that coordinates of points *A*, *B*, *C*, *D* are given.

Proc59. Using the Leng and Area functions from the tasks Proc56 and Proc58, write a real-valued function Dist(x_P, y_P, x_A, y_A, x_B, y_B) that returns the distance $D(P, AB)$ between a point P and a line AB:

$$D(P, AB) = 2\cdot S_{PAB}/|AB|,$$

where S_{PAB} is the area of the triangle PAB. Using this function, find the distance between a point P and each of lines AB, AC, BC provided that coordinates of points P, A, B, C are given.

Proc60. Using the Dist function from the task Proc59, write a procedure Alts(x_A, y_A, x_B, y_B, x_C, y_C, h_A, h_B, h_C) that evaluates the altitudes h_A, h_B, h_C drawn from the vertices A, B, C of a triangle ABC (the coordinates of vertices are input parameters, the values of altitudes are output parameters). Using this procedure, evaluate the altitudes of each of triangles ABC, ABD, ACD provided that the coordinates of points A, B, C, D are given.

9. Functions

This task group is intended for using in the Python and Ruby versions of Programming Taskbook. Tasks included in this group are similar to the tasks of the Proc group but they are formulated with taking into account Python and Ruby features connected with passing parameters and returning function values.

9.1. Functions with numeric parameters

Func1. Write an integer function Sign(X) that returns the following value:

$$-1, \quad \text{if } X < 0; \qquad 0, \quad \text{if } X = 0; \qquad 1, \quad \text{if } X > 0$$

(X is a real-valued parameter). Using this function, evaluate an expression Sign(A) + Sign(B) for given real numbers A and B.

Func2. Write an integer function RootCount(A, B, C) that returns the amount of roots of the quadratic equation $A\cdot x^2 + B\cdot x + C = 0$ (A, B, C are real-valued parameters, $A \neq 0$). Using this function, find the amount of roots for each of three quadratic equations with given coefficients. Note that the amount of roots is determined by the value of a *discriminant*:

$$D = B^2 - 4\cdot A\cdot C.$$

Func3. Write a real-valued function CircleS(R) that returns the area of a circle of radius R (R is a real number). Using this function, find the areas of three circles of given radiuses. Note that the area of a circle of radius R can be found by formula $S = \pi\cdot R^2$. Use 3.14 for a value of π.

Func4. Write a real-valued function RingS(R_1, R_2) that returns the area of a ring bounded by a concentric circles of radiuses R_1 and R_2 (R_1 and R_2 are real numbers, $R_1 > R_2$). Using this function, find the areas of three rings with given outer and inner radiuses. Note that the area of a circle of radius R can be found by formula $S = \pi\cdot R^2$. Use 3.14 for a value of π.

Func5. Write a real-valued function TriangleP(*a*, *h*) that returns the perimeter of an isosceles triangle with given base *a* and altitude *h* (*a* and *h* are real numbers). Using this function, find the perimeters of three triangles with given bases and altitudes. Note that the leg *b* of an isosceles triangle can be found by the *Pythagorean theorem*:

$$b^2 = (a/2)^2 + h^2.$$

Func6. Write an integer function SumRange(*A*, *B*) that returns a sum of all integers in the range *A* to *B* inclusively (*A* and *B* are integers). In the case of $A > B$ the function returns 0. Using this function, find a sum of all integers in the range *A* to *B* and in the range *B* to *C* provided that integers *A*, *B*, *C* are given.

Func7. Write a real-valued function Calc(*A*, *B*, *Op*) that performs an arithmetic operation over nonzero real numbers *A* and *B* and returns the result value. An arithmetic operation is determined by integer parameter *Op* as follows: 1 — subtraction, 2 — multiplication, 3 — division, and addition otherwise. Having input real numbers *A*, *B* and integers N_1, N_2, N_3 and using this function, perform over given *A*, *B* three operations determined by given N_1, N_2, N_3. Output the result value of each operation.

Func8. Write an integer function Quarter(*x*, *y*) that returns the number of a coordinate quarter containing a point with nonzero real-valued coordinates (*x*, *y*). Using this function, find the numbers of coordinate quarters containing each of three points with given nonzero coordinates.

Func9. Write a logical function Even(*K*) that returns True, if an integer parameter *K* is an even number, and False otherwise. Using this function, find the amount of even numbers in a given sequence of 10 integers.

Func10. Write a logical function IsSquare(*K*) that returns True, if an positive integer parameter *K* is a square of some integer, and False otherwise. Using this function, find the amount of squares in a given sequence of 10 positive integers.

Func11. Write a logical function IsPower5(*K*) that returns True, if an positive integer parameter *K* is equal to 5 raised to some integer power, and False otherwise. Using this function, find the amount of powers of base 5 in a given sequence of 10 positive integers.

Func12. Write a logical function IsPowerN(*K*, *N*) that returns True, if an positive integer parameter *K* is equal to *N* (> 1) raised to some integer power, and False otherwise. Having input an integer *N* (> 1) and a sequence of 10 positive integers and using this function, find the amount of powers of base *N* in the given sequence.

Func13. Write a logical function IsPrime(*N*) that returns True, if an integer parameter *N* (> 1) is a prime number, and False otherwise. Using this function, find the amount of prime numbers in a given sequence of 10 integers greater than 1. Note that an integer (> 1) is called a *prime number* if it has not positive divisors except 1 and itself.

Func14. Write an integer function DigitCount(K) that returns the amount of digits in the decimal representation of a positive integer K. Using this function, find the amount of digits for each of five given positive integers.

Func15. Write an integer function DigitN(K, N) that returns the N-th digit in the decimal representation of a positive integer K provided that the digits are numbered from right to left. If the amount of digits is less than N then the function returns −1. Using this function, output sequentially 1st, 2nd, 3rd, 4th, 5th digit for each of five given positive integers K_1, K_2, …, K_5.

Func16. Write a logical function IsPalindrome(K) that returns True, if the decimal representation of a positive parameter K is a *palindrome* (i. e., it is read equally both from left to right and from right to left), and False otherwise. Using this function, find the amount of palindromes in a given sequence of 10 positive integers.

Func17. Write a real-valued function DegToRad(D) that converts the angle value D in degrees into the one in radians (D is a real number, $0 \le D < 360$). Note that $180° = \pi$ radians and use 3.14 for a value of π. Using this function, convert five given angles from degrees into radians.

Func18. Write a real-valued function RadToDeg(R) that converts the angle value R in radians into the one in degrees (R is a real number, $0 \le R < 2{\cdot}\pi$). Note that $180° = \pi$ radians and use 3.14 for a value of π. Using this function, convert five given angles from radians into degrees.

Func19. Write a real-valued function Fact(N) that returns a *factorial* of a positive integer N: $N! = 1{\cdot}2{\cdot}\ldots{\cdot}N$ (the real return type allows to avoid the integer overflow during the calculation of the factorials for large values of the parameter N). Using this function, find the factorials of five given integers.

Func20. Write a real-valued function Fact2(N) that returns a *double factorial* $N!!$:

$$N!! = 1{\cdot}3{\cdot}5{\cdot}\ldots{\cdot}N, \quad \text{if } N \text{ is an odd number;}$$
$$N!! = 2{\cdot}4{\cdot}6{\cdot}\ldots{\cdot}N \quad \text{otherwise}$$

(N is a positive integer; the real return type allows to avoid the integer overflow during the calculation of the double factorials for large values of N). Using this function, find the double factorials of five given integers.

Func21. Write an integer function Fib(N) that returns the value of N-th term of the sequence of the Fibonacci numbers. The *Fibonacci numbers* F_K are defined as follows:

$$F_1 = 1, \qquad F_2 = 1, \qquad F_K = F_{K-2} + F_{K-1}, \quad K = 3, 4, \ldots .$$

Using this function, find five Fibonacci numbers with given order numbers N_1, N_2, …, N_5.

Func22. Write a function PowerA3(A) that returns the third degree of a real number A (A is an input parameter). Using this function, find the third degree of five given real numbers.

Func23. Write a function PowerA234(A) that computes the second, the third, and the fourth degrees of a real number A and returns these degrees as three real numbers (*A* is an input parameter). Using this function, find the second, the third, and the fourth degrees of five given real numbers.

Func24. Write a function Mean(*X*, *Y*) that computes the *arithmetical mean* $(X+Y)/2$ and the *geometrical mean* $(X\cdot Y)^{1/2}$ of two positive real numbers *X* and *Y* and returns the result as two real numbers (*X* and *Y* are input parameters). Using this function, find the arithmetical mean and the geometrical mean of pairs (*A*, *B*), (*A*, *C*), (*A*, *D*) provided that real numbers *A*, *B*, *C*, *D* are given.

Func25. Write a function TrianglePS(*a*) that computes the perimeter $P = 3\cdot a$ and the area $S = a^2\cdot(3)^{1/2}/4$ of an equilateral triangle with the side *a* and returns the result as two real numbers (*a* is an input real-valued parameter). Using this function, find the perimeters and the areas of three triangles with the given lengths of the sides.

Func26. Write a function RectPS(x_1, y_1, x_2, y_2) that computes and returns the perimeter *P* and the area *S* of a rectangle whose opposite vertices have coordinates (x_1, y_1) and (x_2, y_2) and sides are parallel to coordinate axes (x_1, y_1, x_2, y_2 are input real-valued parameters). Using this function, find the perimeters and the areas of three rectangles with the given opposite vertices.

Func27. Write a function DigitCS(*K*) that finds and returns the amount *C* of digits in the decimal representation of a positive integer *K* and also the sum *S* of its digits (*K* is an input integer parameter). Using this function, find the amount and the sum of digits for each of five given integers.

Func28. Write a function InvDigits(*K*) that inverts the order of digits of a positive integer *K* and returns the obtained integer (*K* is an input parameter). Using this function, invert the order of digits for each of five given integers.

Func29. Write a function AddRightDigit(*D*, *K*) that adds a digit *D* to the right side of the decimal representation of a positive integer *K* and returns the obtained number (*D* and *K* are input integer parameters, the value of *D* is in the range 0 to 9). Having input an integer *K* and two one-digit numbers D_1, D_2 and using two calls of this function, sequentially add the given digits D_1, D_2 to the right side of the given K and output the result of each adding.

Func30. Write a function AddLeftDigit(*D*, *K*) that adds a digit *D* to the left side of the decimal representation of a positive integer *K* and returns the obtained number (*D* and *K* are input integer parameters, the value of *D* is in the range 0 to 9). Having input an integer *K* and two one-digit numbers D_1, D_2 and using two calls of this function, sequentially add the given digits D_1, D_2 to the left side of the given K and output the result of each adding.

Func31. Write a function Swap(*X*, *I*, *J*) that exchanges the values of items X_I and X_J of a list *X* of real numbers (*I* and *J* are input integer parameters, the function returns the None value). Having input a list of four real numbers and using three calls of this function, sequentially exchange the values of the two first, two last, and two middle items of the given list. Output the new values of the list.

Func32. Write a function Minmax(X, I, J) that writes the minimal value of items X_I and X_J of a list X to the item X_I and writes the maximal value of these items to the item X_J (I and J are input integer parameters, the function returns the None value). Using four calls of this function, find the minimal value and the maximal value among four given real numbers.

Func33. Write a function SortInc3(X) that sorts the list X of three real-valued items in ascending order (the function returns the None value). Using this function, sort each of two given lists X and Y.

Func34. Write a function SortDec3(X) that sorts the list X of three real-valued items in descending order (the function returns the None value). Using this function, sort each of two given lists X and Y.

Func35. Write a function ShiftRight3(X) that performs a *right cyclic shift* of a list X of three real-valued items: the value ot each item should be assigned to the next item and the value of the last item should be assigned to the first item (the function returns the None value). Using this function, perform the right cyclic shift for each of two given lists X and Y.

Func36. Write a function ShiftLeft3(X) that performs a *left cyclic shift* of a list X of three real-valued items: the value ot each item should be assigned to the previous item and the value of the first item should be assigned to the last item (the function returns the None value). Using this function, perform the left cyclic shift for each of two given lists X and Y.

9.2. Additional tasks

Func37. Write a real-valued function Power1(A, B) that returns the power A^B calculated by the formula $A^B = \exp(B \cdot \ln(A))$ (A and B are real-valued parameters). In the case of zero-valued or negative parameter A the function returns 0. Having input real numbers P, A, B, C and using this function, find the powers A^P, B^P, C^P.

Func38. Write a real-valued function Power2(A, N) that returns the power A^N calculated by the following formulas:

$A^0 = 1$;
$A^N = A \cdot A \cdot \ldots \cdot A$ (N factors), if $N > 0$;
$A^N = 1/(A \cdot A \cdot \ldots \cdot A)$ ($|N|$ factors), if $N < 0$

(A is a real-valued parameter, N is an integer parameter). Having input a real number A and integers K, L, M and using this function, find the powers A^K, A^L, A^M.

Func39. Using the Power1 and Power2 functions (see Func37 and Func38), write a real-valued function Power3(A, B) that returns the power A^B calculated as follows (A and B are real-valued parameters): if B has a zero-valued fractional part then the function Power2(A, N) is called (an *integer* variable N is equal to B), otherwise the function Power1(A, B) is called. Having input real numbers P, A, B, C and using the Power3 function, find the powers A^P, B^P, C^P.

Func40. Write a real-valued function Exp1(x, ε) (x and ε are real numbers, $\varepsilon > 0$) that returns an approximate value of the function exp(x) defined as follows:

$$\exp(x) = 1 + x + x^2/(2!) + x^3/(3!) + \ldots + x^n/(n!) + \ldots$$

($n! = 1\cdot2\cdot\ldots\cdot n$). Stop adding new terms to the sum when the value of the next term will be less than ε. Using this function, find the approximate values of the function exp(x) at a given point x for six given ε.

Func41. Write a real-valued function Sin1(x, ε) (x and ε are real numbers, $\varepsilon > 0$) that returns an approximate value of the function sin(x) defined as follows:

$$\sin(x) = x - x^3/(3!) + x^5/(5!) - \ldots + (-1)^n\cdot x^{2\cdot n+1}/((2\cdot n+1)!) + \ldots .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function sin(x) at a given point x for six given ε.

Func42. Write a real-valued function Cos1(x, ε) (x and ε are real numbers, $\varepsilon > 0$) that returns an approximate value of the function cos(x) defined as follows:

$$\cos(x) = 1 - x^2/(2!) + x^4/(4!) - \ldots + (-1)^n\cdot x^{2\cdot n}/((2\cdot n)!) + \ldots .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function cos(x) at a given point x for six given ε.

Func43. Write a real-valued function Ln1(x, ε) (x and ε are real numbers, $|x| < 1$, $\varepsilon > 0$) that returns an approximate value of the function ln($1 + x$) defined as follows:

$$\ln(1 + x) = x - x^2/2 + x^3/3 - \ldots + (-1)^n\cdot x^{n+1}/(n+1) + \ldots .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function ln($1 + x$) at a given point x for six given ε.

Func44. Write a real-valued function Atan1(x, ε) (x and ε are real numbers, $|x| < 1$, $\varepsilon > 0$) that returns an approximate value of the function atan(x) defined as follows:

$$\mathrm{atan}(x) = x - x^3/3 + x^5/5 - \ldots + (-1)^n\cdot x^{2\cdot n+1}/(2\cdot n+1) + \ldots .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function atan(x) at a given point x for six given ε.

Func45. Write a real-valued function Power4(x, a, ε) (x, a, ε are real numbers, $|x| < 1$, $a, \varepsilon > 0$) that returns an approximate value of the function $(1 + x)^a$ defined as:

$$(1 + x)^a = 1 + a\cdot x + a\cdot(a-1)\cdot x^2/(2!) + \ldots + a\cdot(a-1)\cdot\ldots\cdot(a-n+1)\cdot x^n/(n!) + \ldots .$$

Stop adding new terms to the sum when the absolute value of the next term will be less than ε. Using this function, find the approximate values of the function $(1 + x)^a$ at a given point x for a given exponent a and six given ε.

Func46. Write an integer function GCD2(A, B) that returns the *greatest common divisor* (GCD) of two positive integers A and B. Use the *Euclidean algorithm*:

$$\mathrm{GCD}(A, B) = \mathrm{GCD}(B, A \bmod B), \quad \text{if } B \neq 0; \qquad \mathrm{GCD}(A, 0) = A,$$

where “mod” denotes the operator of taking the remainder after integer division. Using this function, find the greatest common divisor for each of pairs (*A*, *B*), (*A*, *C*), (*A*, *D*) provided that integers *A*, *B*, *C*, *D* are given.

Func47. Using the GCD2 function from the task Func46, write a procedure Frac1(*a*, *b*, *p*, *q*), that simplifies a fraction *a*/*b* to the irreducible form *p*/*q* (*a* and *b* are input integer parameters, *p* and *q* are output integer parameters). The sign of a resulting fraction *p*/*q* is assigned to its numerator, so $q > 0$. Using this procedure, find the numerator and the denominator for each of irreducible fractions $a/b + c/d$, $a/b + e/f$, $a/b + g/h$ provided that integers *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h* are given.

Func48. Taking into account that the *least common multiple* of two positive integers *A* and *B* equals *A*·(*B*/GCD(*A*, *B*)), where GCD(*A*, *B*) is the greatest common divisor of *A* and *B*, and using the GCD2 function from the task Func46, write an integer function LCM2(*A*, *B*) that returns the least common multiple of *A* and *B*. Using this function, find the least common multiple for each of pairs (*A*, *B*), (*A*, *C*), (*A*, *D*) provided that integers *A*, *B*, *C*, *D* are given.

Func49. Taking into account the formula GCD(*A*, *B*, *C*) = GCD(GCD(*A*, *B*), *C*) and using the GCD2 function from the task Func46, write an integer function GCD3(*A*, *B*, *C*) that returns the greatest common divisor of three positive integers *A*, *B*, *C*. Using this function, find the greatest common divisor for each of triples (*A*, *B*, *C*), (*A*, *C*, *D*), (*B*, *C*, *D*) provided that integers *A*, *B*, *C*, *D* are given.

Func50. Write a function TimeToHMS(*T*) that converts a time interval *T* (in seconds) into the “hours *H*, minutes *M*, seconds *S*” format and returns the values *H*, *M*, *S* (*T*, *H*, *M*, *S* are integers). Using this function, find the amount of hours, minutes and seconds for each of five given time intervals T_1, T_2, …, T_5.

Func51. Write a function IncTime(*H*, *M*, *S*, *T*) that increases a time interval in hours *H*, minutes *M*, seconds *S* on *T* seconds and returns new values of hours, minutes, and seconds (all numbers are positive integers). Having input hours *H*, minutes *M*, seconds *S* (as integers) and an integer *T* and using the IncTime function, increase the given time interval on *T* seconds and output new values of *H*, *M*, *S*.

Func52. Write a logical function IsLeapYear(*Y*) that returns True if a year *Y* (a positive integer parameter) is a leap year, and False otherwise. Output the return values of this function for five given values of the parameter *Y*. Note that a year is a *leap year* if it is divisible by 4 except for years that are divisible by 100 and are not divisible by 400.

Func53. Using the IsLeapYear function from the task Func52, write an integer function MonthDays(*M*, *Y*) that returns the amount of days for *M*-th month of year *Y* (*M* and *Y* are integers, $1 \le M \le 12$, $Y > 0$). Output the return value of this function for a given year *Y* and each of given months M_1, M_2, M_3.

Func54. Using the MonthDays function from the task Func53, write a function PrevDate(*D*, *M*, *Y*) that changes a correct date, represented at the "day *D*, month number *M*, year *Y*" format, to a previous one and returns new values of day, month, and year (all numbers are integers). Apply this function to three given dates and output resulting previous ones.

Func55. Using the MonthDays function from the task Func53, write a function NextDate(*D*, *M*, *Y*) that changes a correct date, represented at the "day *D*, month number *M*, year *Y*" format, to a next one and returns new values of day, month, and year (all numbers are integers). Apply this function to three given dates and output resulting next ones.

Func56. Write a real-valued function Leng(x_A, y_A, x_B, y_B) that returns the length of a segment *AB* with given coordinates of its endpoints:

$$|AB| = ((x_A - x_B)^2 + (y_A - y_B)^2)^{1/2}$$

(x_A, y_A, x_B, y_B are real-valued parameters). Using this function, find the lengths of segments *AB*, *AC*, *AD* provided that coordinates of points *A*, *B*, *C*, *D* are given.

Func57. Using the Leng function from the task Func56, write a real-valued function Perim(x_A, y_A, x_B, y_B, x_C, y_C) that returns the perimeter of a triangle *ABC* with given coordinates of its vertices (x_A, y_A, x_B, y_B, x_C, y_C are real-valued parameters). Using the Perim function, find the perimeters of triangles *ABC*, *ABD*, *ACD* provided that coordinates of points *A*, *B*, *C*, *D* are given.

Func58. Using the Leng and Perim functions from the tasks Func56 and Func57, write a real-valued function Area(x_A, y_A, x_B, y_B, x_C, y_C) that returns the area of a triangle *ABC*:

$$S_{ABC} = (p\cdot(p-|AB|)\cdot(p-|AC|)\cdot(p-|BC|))^{1/2},$$

where *p* is the *half-perimeter*. Using the Area function, find the areas of triangles *ABC*, *ABD*, *ACD* provided that coordinates of points *A*, *B*, *C*, *D* are given.

Func59. Using the Leng and Area functions from the tasks Func56 and Func58, write a real-valued function Dist(x_P, y_P, x_A, y_A, x_B, y_B) that returns the distance *D*(*P*, *AB*) between a point *P* and a line *AB*:

$$D(P, AB) = 2\cdot S_{PAB}/|AB|,$$

where S_{PAB} is the area of the triangle *PAB*. Using this function, find the distance between a point *P* and each of lines *AB*, *AC*, *BC* provided that coordinates of points *P*, *A*, *B*, *C* are given.

Func60. Using the Dist function from the task Func59, write a function Alts(x_A, y_A, x_B, y_B, x_C, y_C) that evaluates and returns the altitudes h_A, h_B, h_C drawn from the vertices *A*, *B*, *C* of a triangle *ABC* (the coordinates of vertices are input real-valued parameters). Using this function, evaluate the altitudes of each of triangles *ABC*, *ABD*, *ACD* provided that the coordinates of points *A*, *B*, *C*, *D* are given.

10. Numerical sequences

All input numerical sequences in tasks of this group contains one or more elements (in particular, an integer *N* is always greater than 0). In tasks Series29–Series40 the amount *K* of input sequences is assumed to be nonzero too.

Series1. Given ten real numbers, find their sum.

Series2. Given ten real numbers, find their product.

Series3. Given ten real numbers, find their average.

Series4. An integer *N* and a sequence of *N* real numbers are given. Output the sum and the product of all elements of this sequence.

Series5. An integer *N* and a sequence of *N* positive real numbers are given. Output in the same order the integer parts of all elements of this sequence (as real numbers with zero fractional part). Also output the sum of all integer parts.

Series6. An integer *N* and a sequence of *N* positive real numbers are given. Output in the same order the fractional parts of all elements of this sequence (as real numbers with zero integer part). Also output the product of all fractional parts.

Series7. An integer *N* and a sequence of *N* real numbers are given. Output in the same order the rounded values of all elements of this sequence to the nearest whole number (as integers). Also output the sum of all rounded values.

Series8. An integer *N* and a sequence of *N* integers are given. Output in the same order all even-valued elements of the sequence and also their amount *K*.

Series9. An integer *N* and a sequence of *N* integers are given. Output in the same order the order numbers of all odd-valued elements of the sequence and also their amount *K*.

Series10. An integer *N* and a sequence of *N* integers are given. Output the logical value True if the sequence contains positive-valued elements, otherwise output False.

Series11. Integers *K*, *N* and a sequence of *N* integers are given. Output the logical value False if the sequence contains elements of value less than *K*, otherwise output False.

Series12. A sequence of nonzero integers terminated by zero is given (the final zero is not an element of the sequence). Output the length of the sequence.

Series13. A sequence of nonzero integers terminated by zero is given. Output the sum of all positive-valued elements of the sequence. If the sequence does not contain the required elements then output 0.

Series14. An integer *K* and a sequence of nonzero integers terminated by zero are given (the final zero is not an element of the sequence). Output the amount of elements whose value less than *K*.

Series15. An integer *K* and a sequence of nonzero integers terminated by zero are given (the final zero is not an element of the sequence). Output the order number of the first element whose value greater than *K*. If the sequence does not contain the required elements then output 0.

Series16. An integer *K* and a sequence of nonzero integers terminated by zero are given (the final zero is not an element of the sequence). Output the order number of the last element whose value greater than *K*. If the sequence does not contain the required elements then output 0.

Series17. A real number *B*, an integer *N* and a sequence of *N* real numbers are given. The values of elements of the sequence are in ascending order. Output the number *B* jointly with the elements of the sequence so that all output numbers were in ascending order.

Series18. An integer *N* and a sequence of *N* integers are given. The values of elements of the sequence are in ascending order, the sequence may contain equal elements. Output in the same order all distinct elements of the sequence.

Series19. An integer *N* (> 1) and a sequence of *N* integers are given. Output the elements that are less than their left-side neighbor. Also output the amount *K* of such elements.

Series20. An integer *N* (> 1) and a sequence of *N* integers are given. Output the elements that are less than their right-side neighbor. Also output the amount *K* of such elements.

Series21. An integer *N* (> 1) and a sequence of N real numbers are given. Output the logical value True if the values of elements are in ascending order, otherwise output False.

Series22. An integer *N* (> 1) and a sequence of *N* real numbers are given. Output 0 if the values of elements are in descending order, otherwise output the order number of the first element that breaks the required order.

Series23. An integer *N* (> 2) and a sequence of *N* real numbers are given. A sequence is called a *sawtooth* one if each inner element of the sequence is either greater or less than both of its neighbors (that is, each inner element is a *tooth*). Output 0 if the given sequence is a sawtooth one, otherwise output the order number of the first element that is not a tooth.

Series24. An integer *N* and a sequence of *N* integers are given. The sequence contains at least two zero-valued elements. Output the sum of the values of elements placed between two last zero-valued elements. If two last zero-valued elements are placed side by side then output 0.

Series25. An integer *N* and a sequence of *N* integers are given. The sequence contains at least two zero-valued elements. Output the sum of the values of elements placed between the first and the last zero-valued elements. If the first and the last zero-valued elements are placed side by side then output 0.

Nested loops

Series26. Positive integers *K*, *N* and a sequence of *N* real numbers A_1, A_2, …, A_N are given. For each element of the sequence find its value raised to the power of *K*:

$$(A_1)^K, (A_2)^K, \ldots, (A_N)^K.$$

Series27. An integer N and a sequence of N real numbers $A_1, A_2, \ldots, A_N$ are given. Output the following numbers:

$$A_1, (A_2)^2, \ldots, (A_{N-1})^{N-1}, (A_N)^N.$$

Series28. An integer N and a sequence of N real numbers $A_1, A_2, \ldots, A_N$ are given. Output the following numbers:

$$(A_1)^N, (A_2)^{N-1}, \ldots, (A_{N-1})^2, A_N.$$

Series29. Integers K, N and K sequences of integers are given. Each given sequence contains N elements. Find the total sum of the values of elements of all given sequences.

Series30. Integers K, N and K sequences of integers are given. Each given sequence contains N elements. Find the sum of the values of all elements for each given sequence.

Series31. Integers K, N and K sequences of integers are given. Each given sequence contains N elements. Find the amount of the sequences containing an element with the value 2.

Series32. Integers K, N and K sequences of integers are given. Each given sequence contains N elements. Output the order number of the first element with the value 2 for each given sequence (if a sequence does not contain elements with the required value then output 0).

Series33. Integers K, N and K sequences of integers are given. Each given sequence contains N elements. Output the order number of the last element with the value 2 for each given sequence (if a sequence does not contain elements with the required value then output 0).

Series34. Integers K, N and K sequences of integers are given. Each given sequence contains N elements. Process each sequence as follows: output the sum of the values of all its elements if the sequence contains an element with the value 2, otherwise output 0.

Series35. An integer K and K sequences of nonzero integers are given. Each given sequence is terminated by zero, which is not an element of the sequence. Output the length of each given sequence. Also output the total length of all given sequences.

Series36. An integer K and K sequences of nonzero integers are given. Each given sequence contains at least two elements and is terminated by zero, which is not an element of the sequence. Output the amount of the sequences whose element values are in ascending order.

Series37. An integer K and K sequences of nonzero integers are given. Each given sequence contains at least two elements and is terminated by zero, which is not an element of the sequence. Output the amount of sequences whose element values are in ascending or in descending order.

Series38. An integer K and K sequences of nonzero integers are given. Each given sequence contains at least two elements and is terminated by zero, which is not an element of the sequence. Process each sequence as follows: output 1 or −1 if its element values are in ascending or in descending order respectively, otherwise output 0.

Series39. An integer K and K sequences of nonzero integers are given. Each given sequence contains at least three elements and is terminated by zero, which is not an element of the sequence. Output the amount of the sawtooth sequences (see the definition of a *sawtooth sequence* in Series23).

Series40. An integer K and K sequences of nonzero integers are given. Each given sequence contains at least three elements and is terminated by zero, which is not an element of the sequence. Process each sequence as follows: output its length if the sequence is a sawtooth one (see Series23), otherwise output the order number of its first element that is not a tooth.

11. Minimums and maximums

All tasks of this section should be solved by means of *one-pass algorithms*, which require *one loop* for obtaining result. One-pass algorithms do not need to store all input data simultaneously, so they may be developed without using arrays.

All input sequences in tasks of this group contain one or more elements (in particular, an integer N is always greater than 0).

Minmax1. An integer N and a sequence of N real numbers are given. Find the minimal element and the maximal element of the sequence (that is, elements with the minimal value and the maximal value respectively).

Minmax2. An integer N and a sequence of N rectangles are given. Each rectangle is defined by a pair of its sides (a, b). Find the rectangle with the minimal area and output the value of its area.

Minmax3. An integer N and a sequence of N rectangles are given. Each rectangle is defined by a pair of its sides (a, b). Find the rectangle with the maximal perimeter and output the value of its perimeter.

Minmax4. An integer N and a sequence of N real numbers are given. Find the order number of the minimal element of the sequence.

Minmax5. An integer N and a sequence of N pairs of real numbers (m, v) are given. Each pair of given numbers contains the weight m and the volume v of a detail that is made of some homogeneous material. Output the order number of a detail that is made of the material with maximal density. Also output the corresponding density. Note that the density P can be found by formula

$$P = m/v.$$

Minmax6. An integer N and a sequence of N integers are given. Find the order numbers of the first minimal element and the last maximal element of the sequence.

Minmax7. An integer *N* and a sequence of *N* integers are given. Find the order numbers of the first maximal element and the last minimal element of the sequence.

Minmax8. An integer *N* and a sequence of *N* integers are given. Find the order numbers of the first and the last minimal elements of the sequence.

Minmax9. An integer *N* and a sequence of *N* integers are given. Find the order numbers of the first and the last maximal elements of the sequence.

Minmax10. An integer *N* and a sequence of *N* integers are given. Find the order number of the first *extremal* (that is, minimal or maximal) element of the sequence.

Minmax11. An integer *N* and a sequence of *N* integers are given. Find the order number of the last *extremal* (that is, minimal or maximal) element of the sequence.

Minmax12. An integer *N* and a sequence of *N* real numbers are given. Output the minimal positive number contained in the sequence. If the sequence does not contain positive numbers then output 0.

Minmax13. An integer *N* and a sequence of *N* integers are given. Output the order number of the first maximal odd number contained in the sequence. If the sequence does not contain odd numbers then output 0.

Minmax14. A positive real number *B* and a sequence of 10 real numbers are given. Find the minimum among elements that are greater than *B* and output its value and its order number. If the sequence does not contain elements greater than *B* then output 0 twice (the first zero as a real number, the second zero as an integer).

Minmax15. Two real numbers *B*, *C* ($0 < B < C$) and a sequence of 10 real numbers are given. Find the maximum among elements that are in the interval (*B*, *C*) and output its value and its order number. If the sequence does not contain elements in the interval (*B*, *C*) then output 0 twice (the first zero as a real number, the second zero as an integer).

Minmax16. An integer *N* and a sequence of *N* integers are given. Find the amount of the elements that are located before the first minimal element.

Minmax17. An integer *N* and a sequence of *N* integers are given. Find the amount of the elements that are located after the last maximal element.

Minmax18. An integer *N* and a sequence of *N* integers are given. Find the amount of the elements that are located between the first and the last maximal element. If the sequence contains the unique maximal element then output 0.

Minmax19. An integer *N* and a sequence of *N* integers are given. Find the amount of the minimal elements of the sequence.

Minmax20. An integer *N* and a sequence of *N* integers are given. Find the total amount of all *extremal* (that is, minimal or maximal) elements of the sequence.

Minmax21. An integer *N* (> 2) and a sequence of *N* real numbers are given. The elements of the sequence represent some experimental data. Find the average of the experimental data provided that the minimal and maximal values must be ignored.

Minmax22. An integer *N* (> 2) and a sequence of *N* real numbers are given. Find two smallest elements of the sequence and output their values in ascending order.

Minmax23. An integer *N* (> 3) and a sequence of *N* real numbers are given. Find three greatest elements of the sequence and output their values in descending order.

Minmax24. An integer *N* (> 1) and a sequence of *N* real numbers are given. Find the maximal sum of two adjacent elements of the sequence.

Minmax25. An integer *N* (> 1) and a sequence of *N* real numbers are given. Find two adjacent elements that have the minimal product of their values and output their order numbers in ascending order.

Minmax26. An integer *N* and a sequence of *N* integers are given. Output the maximal amount of successive elements whose values are even numbers. If the sequence does not contain even numbers then output 0.

Minmax27. An integer *N* and a sequence of *N* integers are given. The sequence contains elements of values 0 and 1 only. Find the longest subsequence of the successive elements with equal values, and output the order number of its initial element and the amount of its elements. If there are several such subsequences then output the order number of the first one.

Minmax28. An integer *N* and a sequence of *N* integers are given. The sequence contains elements of values 0 and 1 only. Find the longest subsequence of the successive elements of value 1, and output the order number of its initial element and the amount of its elements. If there are several such subsequences then output the order number of the first one. If the sequence does not contain elements of value 1 then output 0 twice.

Minmax29. An integer *N* and a sequence of *N* integers are given. Find the maximal amount of the successive elements with the minimal value.

Minmax30. An integer *N* and a sequence of *N* integers are given. Find the minimal amount of the successive elements with the maximal value.

12. One-dimensional arrays

The condition “An array of *N* integers (or real numbers) is given” means that the *actual size* of the array (an integer *N*) and all its elements are given. The size of any array is assumed to be in the range 2 to 10, if the task does not specify it explicitly. The order number of the first element of array is assumed to be equal to 1.

If a task connected with array creation or array changing does not specify output data then the resulting array elements are supposed to be output in ascending order of their indices.

12.1. Array creation

The size of a resulting array is assumed to be not greater than 10 in all tasks connected with the array creation.

Array1. Given an integer N (> 0), create and output an array of N integers that are the first positive odd numbers: 1, 3, 5, … .

Array2. Given an integer N (> 0), create and output an array of N integers that are the first positive integer powers of 2: 2, 4, 8, 16, … .

Array3. An integer N (> 1), the first term A and the common difference D of an *arithmetic sequence* are given (A and D are real numbers). Create and output an array of N real numbers that are the initial terms of this sequence:

$$A,\quad A + D,\quad A + 2\cdot D,\quad A + 3\cdot D,\quad \ldots .$$

Array4. An integer N (> 1), the first term A and the common ratio R of a *geometric sequence* are given (A and D are real numbers). Create and output an array of N real numbers that are the initial terms of this sequence:

$$A,\quad A\cdot R,\quad A\cdot R^2,\quad A\cdot R^3,\quad \ldots .$$

Array5. Given an integer N (> 2), create and output an array of N integers that are the initial terms of a sequence $\{F_K\}$ of the *Fibonacci numbers*:

$$F_1 = 1,\qquad F_2 = 1,\qquad F_K = F_{K-2} + F_{K-1},\quad K = 3, 4, \ldots .$$

Array6. Given three integers N (> 2), A, B, create and output an array of N integers such that the first element is equal to A, the second one is equal to B, and each subsequent element is equal to the sum of all previous ones.

12.2. Output of array elements

Array7. Given an array of N real numbers, output its elements in inverse order.

Array8. Given an array of N integers, output all odd numbers contained in the array in ascending order of their indices. Also output the amount K of odd numbers contained in the array.

Array9. Given an array of N integers, output all even numbers contained in the array in descending order of their indices. Also output the amount K of even numbers contained in the array.

Array10. Given an array of N integers, output all even numbers contained in the array in ascending order of their indices and then output all odd numbers contained in the array in descending order of their indices.

Array11. An array A of N real numbers and an integer K ($1 \le K \le N$) are given. Output array elements with order numbers that are multiples of K: A_K, $A_{2\cdot K}$, $A_{3\cdot K}$, … . Do not use conditional statements.

Array12. An array A of N real numbers is given (N is an even number). Output array elements with even order numbers in ascending order of indices: A_2, A_4, A_6, …, A_N. Do not use conditional statements.

Array13. An array A of N real numbers is given (N is an odd number). Output array elements with odd order numbers in descending order of indices: A_N, A_{N-2}, A_{N-4}, …, A_1. Do not use conditional statements.

Array14. An array A of N real numbers is given. Output array elements with even order numbers (in ascending order of indices) and then output array elements with odd order numbers (in ascending order of indices too):

$$A_2,\quad A_4,\quad A_6,\quad \ldots,\quad A_1,\quad A_3,\quad A_5,\quad \ldots\ .$$

Do not use conditional statements.

Array15. An array A of N real numbers is given. Output array elements with odd order numbers in ascending order of indices and then output array elements with even order numbers in descending order of indices:

$$A_1,\quad A_3,\quad A_5,\quad \ldots,\quad A_6,\quad A_4,\quad A_2.$$

Do not use conditional statements.

Array16. An array A of N real numbers is given. Output array elements as follows:

$$A_1,\quad A_N,\quad A_2,\quad A_{N-1},\quad A_3,\quad A_{N-2},\quad \ldots\ .$$

Array17. An array A of N real numbers is given. Output array elements as follows:

$$A_1,\quad A_2,\quad A_N,\quad A_{N-1},\quad A_3,\quad A_4,\quad A_{N-2},\quad A_{N-3},\quad \ldots\ .$$

12.3. Analysis of array elements

Some tasks of this section may be solved without storing all input data simultaneously, so they do not need arrays for solving. However, using arrays usually leads to more simple and obvious algorithms.

Array18. An array A of 10 nonzero integers is given. Output the value of the first element A_K that satisfies the following inequality: $A_K < A_{10}$. If the array does not contain such elements then output 0.

Array19. An array A of 10 integers is given. Output the order number of the last element A_K that satisfies the following double inequality: $A_1 < A_K < A_{10}$. If the array does not contain such elements then output 0.

Array20. An array of N real numbers and two integers K and L ($1 \le K \le L \le N$) are given. Find the sum of array elements with the order numbers in the range K to L inclusively.

Array21. An array of N real numbers and two integers K and L ($1 \le K \le L \le N$) are given. Find the average of array elements with the order numbers in the range K to L inclusively.

Array22. An array of N real numbers and two integers K and L ($1 < K \le L \le N$) are given. Find the sum of all array elements except ones with the order numbers in the range K to L inclusively.

Array23. An array of N real numbers and two integers K and L ($1 < K \le L \le N$) are given. Find the average of all array elements except ones with the order numbers in the range K to L inclusively.

Array24. An array of N distinct integers is given. If the array elements represent an *arithmetic sequence* (see Array3) then output its common difference, otherwise output 0.

Array25. An array of N nonzero integers is given. If the array elements represent a *geometric sequence* (see Array4) then output its common ratio, otherwise output 0.

Array26. An array of N integers is given. If odd and even numbers are alternated in the array then output 0, otherwise output the order number of the first element that breaks the above condition.

Array27. An array of N nonzero integers is given. If positive and negative numbers are alternated in the array then output 0, otherwise output the order number of the first element that breaks the above condition.

Array28. Given an array A of N real numbers, find the minimum among elements with even order numbers: A_2, A_4, A_6, … .

Array29. Given an array A of N real numbers, find the maximum among elements with odd order numbers: A_1, A_3, A_5, … .

Array30. An array of N real numbers is given. Find the order numbers of array elements that are greater than their right neighbor. Output these order numbers in ascending order and also output the amount of such elements.

Array31. An array of N real numbers is given. Find the order numbers of array elements that are greater than their left neighbor. Output these order numbers in descending order and also output the amount of such elements.

Array32. Given an array of N real numbers, find the order number of its first local minimum (an array element is called the *local minimum* if it is smaller than its neighbors).

Array33. Given an array of N real numbers, find the order number of its last local maximum (an array element is called the *local maximum* if it is greater than its neighbors).

Array34. Given an array of N real numbers, find the maximum among its local minimums (see the *local minimum* definition in Array32).

Array35. Given an array of N real numbers, find the minimum among its local maximums (see the *local maximum* definition in Array33).

Array36. Given an array of N real numbers, find the maximum among array elements that are neither local minimum nor local maximum (see the definitions of *local minimum* and *local maximum* in Array32 and Array33 respectively). If the array does not contain such elements then output 0 (as a real number).

Array37. Given an array of N real numbers, find the amount of its *parts of ascending* (that is, parts that contain elements whose values are in ascending order).

Array38. Given an array of N real numbers, find the amount of its *parts of descending* (that is, parts that contain elements whose values are in descending order).

Array39. Given an array of N real numbers, find the amount of its *parts of monotonicity* (that is, parts that contain elements whose values are in ascending or in descending order).

Array40. A real number R and an array A of N real numbers are given. Find the array element that is the *nearest* to the number R (i. e., an element A_K such that the value $|A_K - R|$ is minimal).

Array41. Given an array of N real numbers, find two adjacent elements with the maximal sum of values and output these elements in ascending order of its indices.

Array42. A real number R and an array of N real numbers are given. Find two adjacent elements whose sum of values is the nearest to the number R and output these elements in ascending order of its indices (see the definition of two nearest numbers in Array40).

Array43. Given an array of N integers whose values are in ascending or in descending order, find the amount of its elements with distinct values.

Array44. An array of N integers is given, the array contains two elements with equal values. Find these elements and output their order numbers in ascending order.

Array45. Given an array of N real numbers, find two nearest array elements (i. e., two different elements A_K and A_L such that the value $|A_K - A_L|$ is minimal) and output their order numbers in ascending order.

Array46. A real number R and an array of N real numbers are given. Find two different elements such that the sum of their values is the nearest to the number R, and output these elements in ascending order of indices (see the definition of two nearest numbers in Array40).

Array47. Given an array of N integers, find the amount of its elements with distinct values.

Array48. Given an array of N integers, find the maximal amount of its elements with equal values.

Array49. An array of N integers is given. If the array is a *permutation* (i. e., if the array contains all integers in the range 1 to N) then output 0, otherwise output the order number of the first inadmissible element.

Array50. An array A of N integers is given, the array is a permutation (see the *permutation* definition in Array49). Find the amount of *inversions* in this permutation (i. e., the amount of pairs of elements A_I and A_J such that $I < J$ and $A_I > A_J$).

12.4. Work with several one-dimensional arrays

Array51. Two arrays A and B of N real numbers are given. Exchange the contents of the given arrays and output elements of the changed array A and then output elements of the changed array B.

Array52. Given an array A of N real numbers, create a new array B of the same size with elements defined as:

$$B_K = \begin{cases} 2\cdot A_K, & \text{if } A_K < 5, \\ A_K/2 & \text{otherwise.} \end{cases}$$

Array53. Two arrays A and B of N real numbers are given. Create a new array C of the same size; each element C_K must be equal to the largest of elements of A and B with the same index K.

Array54. An array A of N integers is given. Write elements of A whose values are even numbers to a new array B (in the same order) and output the size of array B and all its elements.

Array55. An array A of N (≤ 15) integers is given. Write elements of A with odd order numbers (1, 3, …) to a new array B and output the size of array B and all its elements. Do not use conditional statements.

Array56. An array A of N (≤ 15) integers is given. Write elements of A with order numbers that are multiples of 3 (3, 6, …) to a new array B and output the size of array B and all its elements. Do not use conditional statements.

Array57. An array A of N integers is given. Write elements of A with even order numbers to a new array B and then write elements of A with odd order numbers to the same array B, so the array B will contain the following elements:

$$A_2,\quad A_4,\quad A_6,\quad \ldots,\quad A_1,\quad A_3,\quad A_5,\quad \ldots .$$

Do not use conditional statements.

Array58. An array A of N real numbers is given. Create a new array B of the same size; each array element B_K must be equal to the sum of elements of A with the order numbers in the range 1 to K.

Array59. An array A of N real numbers is given. Create a new array B of the same size; each array element B_K must be equal to the average of elements of A with the order numbers in the range 1 to K.

Array60. An array A of N real numbers is given. Create a new array B of the same size; each array element B_K must be equal to the sum of elements of A with the order numbers in the range K to N.

Array61. An array A of N real numbers is given. Create a new array B of the same size; each array element B_K must be equal to the average of elements of A with the order numbers in the range K to N.

Array62. Given an array A of N real numbers, create two new arrays B and C and write all positive elements of A to the array B and all negative elements of A to the array C (in the same order). Output the size of the array B and all its elements and then output the size of the array C and all its elements.

Array63. Two arrays A and B of 5 real numbers are given. Values of elements of each array are in ascending order. Write all elements of A and B to a new array C (of size 10) so that values of all elements of C were in ascending order.

Array64. Three arrays A, B, C of N_A, N_B, N_C integers are given. Values of elements of each array are in descending order. Write all elements of A, B, C to a new array D (of size $N_A + N_B + N_C$) so that values of all elements of D were in descending order.

12.5. Array changing

All tasks of this subsection should be solved without using additional arrays.

Array65. An array A of N real numbers and an integer K ($1 \le K \le N$) are given. Increase the value of each element of A by the initial value of A_K.

Array66. An array of N integers is given. Increase all even numbers contained in the array by the initial value of the first even number in the array. If the array does not contain even numbers then do not change it.

Array67. An array of N integers is given. Increase all odd numbers contained in the array by the initial value of the last odd number in the array. If the array does not contain odd numbers then do not change it.

Array68. Given an array of N real numbers, exchange values of its minimal and maximal element.

Array69. Given an array of N real numbers (N is an even number), exchange values of its first and second element, its third and fourth element, and so on.

Array70. Given an array of N real numbers (N is an even number), exchange values of the first half and the second half of its elements.

Array71. Given an array of N real numbers, change the order of its elements to inverse one.

Array72. An array A of N real numbers and two integers K and L ($1 \le K < L \le N$) are given. Change the order of the array elements between A_K and A_L (including these elements) to inverse one.

Array73. An array A of N real numbers and two integers K and L ($1 \le K < L \le N$) are given. Change the order of the array elements between A_K and A_L (not including these elements) to inverse one.

Array74. An array of N real numbers is given. Assign zero value to the array elements between the minimal element and the maximal element (not including these elements).

Array75. An array of N real numbers is given. Change the order of the array elements between the minimal element and the maximal element (including these elements) to inverse one.

Array76. Given an array of N real numbers, assign zero value to all its local maximums (an array element is called the *local maximum* if it is greater than its neighbors).

Array77. Given an array of N real numbers, replace each local minimum with its square (an array element is called the *local minimum* if it is smaller than its neighbors).

Array78. Given an array of N real numbers, replace each array element with the average of this element and its neighbors.

Array79. Given an array of N real numbers, perform the *right shift* of array elements on one position by assigning initial values of A_1, A_2, …, A_{N-1} to elements A_2, A_3, …, A_N respectively (an initial value of the last element will be lost). Assign zero value to the first element of the changed array.

Array80. Given an array of N real numbers, perform the *left shift* of array elements on one position by assigning initial values of A_N, A_{N-1}, …, A_2 to elements A_{N-1}, A_{N-2}, …, A_1 respectively (an initial value of the first element will be lost). Assign zero value to the last element of the changed array.

Array81. Given an array of N real numbers and an integer K ($1 \le K < N$), perform the *right shift* of array elements on K positions by assigning initial values of A_1, A_2, …, A_{N-K} to elements A_{K+1}, A_{K+2}, …, A_N respectively (an initial value of the last K elements will be lost). Assign zero value to the first K elements of the changed array.

Array82. Given an array of N real numbers and an integer K ($1 \le K < N$), perform the *left shift* of array elements on K positions by assigning initial values of A_N, A_{N-1}, …, A_{K+1} to elements A_{N-K}, A_{N-K-1}, …, A_1 respectively (an initial value of the first K elements will be lost). Assign zero value to the last K elements of the changed array.

Array83. Given an array of N real numbers, perform the *right cyclic shift* of array elements on one position by assigning initial values of A_1, A_2, …, A_{N-1}, A_N to elements A_2, A_3, …, A_N, A_1 respectively.

Array84. Given an array of N real numbers, perform the *left cyclic shift* of array elements on one position by assigning initial values of A_N, A_{N-1}, …, A_2, A_1 to elements A_{N-1}, A_{N-2}, …, A_1, A_N respectively.

Array85. Given an array of N real numbers and an integer K ($1 \le K \le 4$, $K < N$), perform the *right cyclic shift* of array elements on K positions by assigning initial values of A_1, A_2, …, A_N to elements A_{K+1}, A_{K+2}, …, A_K respectively. An additional array of 4 elements may be used for performing the required shift.

Array86. Given an array of N real numbers and an integer K ($1 \le K \le 4$, $K < N$), perform the *left cyclic shift* of array elements on K positions by assigning initial values of A_N, A_{N-1}, …, A_1 to elements A_{N-K}, A_{N-K-1}, …, A_{N-K+1} respectively. An additional array of 4 elements may be used for performing the required shift.

Array87. An array of N real numbers is given. The values of all array elements, except the first one, are in ascending order. Arrange all array elements in ascending order by moving the first element to a new position.

Array88. An array of N real numbers is given. The values of all array elements, except the last one, are in ascending order. Arrange all array elements in ascending order by moving the last element to a new position.

Array89. An array of N real numbers is given. The values of all array elements, except one element, are in descending order. Arrange all array elements in descending order by moving the element, that violates ordering, to a new position.

Array90. An array of N real numbers and an integer K ($1 \le K \le N$) are given. Remove an element with the order number K from the array.

Array91. An array of N real numbers and two integers K and L ($1 \le K < L \le N$) are given. Remove elements with the order numbers in the range K to L inclusively from the array and output the size of the changed array and all its elements.

Array92. Given an array of N integers, remove all odd numbers from the array and output the size of the changed array and all its elements.

Array93. Given an array of N (> 2) integers, remove all elements with even order numbers (2, 4, …) from the array. Do not use conditional statements.

Array94. Given an array of N (> 2) integers, remove all elements with odd order numbers (1, 3, …) from the array. Do not use conditional statements.

Array95. Given an array of N integers, remove all successive equal elements (except for their first occurrence) from the array.

Array96. Given an array of N integers, remove all equal elements (except for their first occurrence) from the array.

Array97. Given an array of N integers, remove all equal elements (except for their last occurrence) from the array.

Array98. Given an array of N integers, remove all elements whose values appear less than three times in the array. Output the size of the changed array and all its elements.

Array99. Given an array of N integers, remove all elements whose values appear more than twice in the array. Output the size of the changed array and all its elements.

Array100. Given an array of N integers, remove all elements whose values appear exactly twice in the array. Output the size of the changed array and all its elements.

Array101. An array of N real numbers and an integer K ($1 \le K \le N$) are given. Insert a new element with zero value before an element with the order number K.

Array102. An array of N real numbers and an integer K ($1 \le K \le N$) are given. Insert a new element with zero value after an element with the order number K.

Array103. An array of N real numbers is given. Insert new elements with zero value before an element with the minimal value and after an element with the maximal value.

Array104. An array of N real numbers and two integers K and M ($1 \le K \le N$, $1 \le M \le 10$) are given. Insert M new elements with zero value before an element with the order number K.

Array105. An array of N real numbers and two integers K and M ($1 \le K \le N$, $1 \le M \le 10$) are given. Insert M new elements with zero value after an element with the order number K.

Array106. Given an array of N real numbers, double occurrences of elements with even order numbers (2, 4, …). Do not use conditional statements in loops.

Array107. Given an array of N real numbers, triple occurrences of elements with odd order numbers (1, 3, …). Do not use conditional statements in loops.

Array108. Given an array of N real numbers, insert an element with zero value before each element with positive value.

Array109. Given an array of N real numbers, insert an element with zero value after each element with negative value.

Array110. Given an array of N integers, double occurrences of all elements whose values are even numbers.

Array111. Given an array of N integers, triple occurrences of all elements whose values are odd numbers.

Array112. An array A of N (≤ 6) real numbers is given. Sort the array in ascending order by using the *exchange sort method* (the *bubble sorting*): compare each pair of two adjacent elements (A_1 and A_2, A_2 and A_3, and so on) and exchange their values in case the left element is greater than the right one; repeat $N - 1$ times such array pass. Output all array elements after each pass for checking results. Note that the amount of analyzed pairs of the array elements can be reduced by 1 after each pass.

Array113. An array A of N (≤ 6) real numbers is given. Sort the array in ascending order by using the *selection sort method*: find the greatest element in the array and exchange the values of the greatest element and the last element (with the order number N); repeat $N - 1$ times this process, reducing the amount of analyzed elements by 1 after each array pass. Output all array elements after each pass for checking results.

Array114. An array A of N (≤ 6) real numbers is given. Sort the array in ascending order by using the *insertion sort method*: compare an element A_2 with the first element A_1 and exchange their values, if necessary, so that these elements were in ascending order; then move an element A_3 to the left (sorted) part of the array, so that three elements were in ascending order; repeat this process for other array elements. Output all array elements after processing of each element (from A_2 to A_N). Note that it is convenient to assign the array element being processed to an additional array element A_0 (the *barrier element*).

Array115. An array A of N real numbers is given. Without changing the array A, output the order numbers that correspond to array elements in ascending order of their values. For solving the task create an additional *index array* I, which contains order numbers in the range 1 to N, and use the bubble sorting (see Array112) as follows: compare elements of array A (with the order numbers I_1 and I_2, I_2 and I_3, and so on) and exchange, if necessary, values of corresponding elements of index array *I*. After repeating $N - 1$ times of such array pass the required sequence of order numbers will be contained in the array *I*.

12.6. Series of equal numbers

Array116. An array *A* of *N* integers is given. A group of successive array elements with equal values is called a *series of equal numbers*, the amount of its elements is called a *length of series* (a length of series may be equal to 1), the value of its elements is called a *value of series*. Create and output two new integer-valued arrays *B* and *C* containing respectively lengths and values of all series of equal numbers of the array *A*.

Array117. Given an array of *N* integers, insert an element with zero value before each series of equal numbers of the array (see the *series* definition in Array116).

Array118. Given an array of *N* integers, insert an element with zero value after each series of equal numbers of the array (see the *series* definition in Array116).

Array119. Given an array of *N* integers, increase each series of equal numbers of the array by one element (see the *series* definition in Array116).

Array120. An array of *N* integers is given. The array contains at least one series of two or more equal numbers. Decrease each series of equal numbers of the array by one element (see the *series* definition in Array116).

Array121. An integer *K* (> 0) and an array of *N* integers are given. Double the length of the *K*-th series of equal numbers of the array (see the *series* definition in Array116). If the array contains less than *K* series then do not change it.

Array122. An integer *K* (> 1) and an array of *N* integers are given. Remove the *K*-th series of equal numbers from the array (see the *series* definition in Array116). If the array contains less than *K* series then do not change it.

Array123. An integer *K* (> 1) and an array of *N* integers are given. Exchange the first and the *K*-th series of equal numbers of the array (see the *series* definition in Array116). If the array contains less than *K* series then do not change it.

Array124. An integer *K* (> 1) and an array of *N* integers are given. Exchange the last and the *K*-th series of equal numbers of the array (see the *series* definition in Array116). If the array contains less than *K* series then do not change it.

Array125. An integer *L* (> 1) and an array of *N* integers are given. Replace each series, whose length is less than *L*, by one element with zero value (see the *series* definition in Array116).

Array126. An integer *L* (> 0) and an array of *N* integers are given. Replace each series, whose length is equal to *L*, by one element with zero value (see the *series* definition in Array116).

Array127. An integer L (> 0) and an array of N integers are given. Replace each series, whose length is greater than L, by one element with zero value (see the *series* definition in Array116).

Array128. Given an array of N integers, increase its first series of the maximal length by one element (see the *series* definition in Array116).

Array129. Given an array of N integers, increase its last series of the maximal length by one element (see the *series* definition in Array116).

Array130. Given an array of N integers, increase all its series of the maximal length by one element (see the *series* definition in Array116).

12.7. Sets of points

Each set of points must be stored either in two arrays of numbers (the first array stores x-coordinates, the second one stores y-coordinates) or in one array of *records* with two fields x and y.

Array131. A set A of N points in the coordinate plane and a point B are given (all points are determined by their coordinates x, y). Find the point of A that is the nearest to the point B. The *distance* R between two points with the coordinates (x_1, y_1) and (x_2, y_2) may be found by formula:

$$R = ((x_2 - x_1)^2 + (y_2 - y_1)^2)^{1/2}.$$

Array132. A set A of N points in the coordinate plane is given (points are determined by their coordinates x, y). Find the point of A that lies in the second coordinate quarter and is the farthest from the origin. If the set A does not contain such points then output the origin (0, 0).

Array133. A set A of N points in the coordinate plane is given (points are determined by their coordinates x, y). Find the point of A that lies in the first or the third coordinate quarter and is the nearest to the origin. If the set A does not contain such points then output the origin (0, 0).

Array134. A set A of N points in the coordinate plane is given (points are determined by their coordinates x, y). Find two points of A with the maximal distance between them and output these points (in ascending order of their indices in the set A) and the value of the distance.

Array135. Two sets A and B of N_1 and N_2 points respectively are given (points are determined by their coordinates x, y). Find the point of A and the point of B with the minimal distance between them. Output the value of the distance and then the point of A and the point of B.

Array136. A set A of N points in the coordinate plane is given ($N > 2$, points are determined by their coordinates x, y). Find the point of A such that the sum of distances between this point and other points of A is minimal and output this point and the corresponding sum.

Array137. A set A of N points in the coordinate plane is given ($N > 2$, points are determined by their coordinates x, y). Find the maximal perimeter of a triangle with vertices belonging to A. Output this perimeter and the corresponding vertices (in ascending order of their indices in the set A).

Array138. A set A of N points in the coordinate plane is given ($N > 2$, points are determined by their coordinates x, y). Find the minimal perimeter of a triangle with vertices belonging to A. Output this perimeter and the corresponding vertices (in ascending order of their indices in the set A).

Array139. A set A of N points with integer-valued coordinates x, y is given. The *order in the plane* is defined as follows:

$$(x_1, y_1) < (x_2, y_2), \quad \text{if either } x_1 < x_2 \text{ or } x_1 = x_2 \text{ and } y_1 < y_2.$$

Using this order definition, rearrange points of A in ascending order.

Array140. A set A of N points with integer-valued coordinates x, y is given. The *order in the plane* is defined as follows:

$$(x_1, y_1) < (x_2, y_2), \quad \text{if either } x_1 + y_1 < x_2 + y_2 \text{ or } x_1 + y_1 = x_2 + y_2 \text{ and } x_1 < x_2.$$

Using this order definition, rearrange points of A in descending order.

13. Two-dimensional arrays (matrices)

The condition “An $M \times N$ matrix of integers (or real numbers) is given” means that the actual size of a two-dimensional array and all its elements are given (M is the number of rows and N is the number of columns, the matrix has $M{\cdot}N$ elements). The amount of rows (columns) of any matrix is assumed to be in the range 2 to 10 if the task does not specify it explicitly. The order number of the first row (column) of matrix is assumed to be equal to 1, the element $A_{I,J}$ is assumed to be in the I-th row and J-th column ($I = 1, \ldots, M$, $J = 1, \ldots, N$). An input/output of matrix elements must be performed *in the order of rows*, that is, in ascending order of their indices with the second index (the *column index*) changing faster than the first one (the *row index*).

A matrix having the size $M \times M$ is called a *square matrix of order M*; the actual size of square matrix is defined by *one* integer M.

If a task connected with matrix creation or matrix changing does not specify output data then the resulting matrix elements are supposed to be output (by rows).

13.1. Matrix creation

The size of a resulting matrix is assumed to be not greater than 10×10 in all tasks connected with the matrix creation.

Matrix1. Given two positive integers M and N, create and output an $M \times N$ matrix of integers such that all its elements of the I-th row are equal to $10{\cdot}I$ ($I = 1, \ldots, M$).

Matrix2. Given two positive integers M and N, create and output an $M \times N$ matrix of integers such that all its elements of the J-th column are equal to $5{\cdot}J$ ($J = 1, \ldots, N$).

Matrix3. Two positive integers M, N and a sequence of M real numbers are given. Create and output an $M \times N$ matrix of real numbers such that each of its columns contains all numbers from the given sequence (in the same order).

Matrix4. Two positive integers M, N and a sequence of M real numbers are given. Create and output an $M \times N$ matrix of real numbers such that each of its rows contains all numbers from the given sequence (in the same order).

Matrix5. Two positive integers M and N, a real number D, and a sequence of M real numbers are given. Create and output an $M \times N$ matrix of real numbers such that its first column contains all numbers from the given sequence (in the same order), and elements of each next column are equal to the sum of the corresponding element of the previous column and the number D (so each row of the matrix will be an *arithmetic sequence* with the common difference D).

Matrix6. Two positive integers M and N, a real number D, and a sequence of M real numbers are given. Create and output an $M \times N$ matrix of real numbers such that its first row contains all numbers from the given sequence (in the same order), and elements of each next row are equal to the sum of the corresponding element of the previous row and the number R (so each column of the matrix will be a *geometric sequence* with the common ratio R).

13.2. Output of matrix elements

Matrix7. An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le M$). Output elements of the matrix row with the order number K.

Matrix8. An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le M$). Output elements of the matrix column with the order number K.

Matrix9. An $M \times N$ matrix of real numbers is given. Output elements of its rows with even order numbers (2, 4, …). An output of matrix elements must be performed in the order of rows. Do not use conditional statements.

Matrix10. An $M \times N$ matrix of real numbers is given. Output elements of its columns with odd order numbers (1, 3, …). An output of matrix elements must be performed in the order of columns. Do not use conditional statements.

Matrix11. An $M \times N$ matrix of real numbers is given. Output elements of the matrix in the following order: the first row from left to right, the second row from right to left, the third row from left to right, the fourth row from right to left, and so on.

Matrix12. An $M \times N$ matrix of real numbers is given. Output elements of the matrix in the following order: the first column from up to down, the second column from down to up, the third column from up to down, the fourth column from down to up, and so on.

Matrix13. A real-valued square matrix A of order M is given. Starting with the element $A_{1,1}$, output its elements as follows: all elements of the first row, all elements of the M-th column except the element $A_{1,M}$ (which is already output), all remaining elements of the second row, all remaining elements of the

(M−1)-th column, and so on; the element $A_{M,1}$ must be output in the end. All rows must be output from left to right, all columns must be output from up to down.

Matrix14. A real-valued square matrix A of order M is given. Starting with the element $A_{1,1}$, output its elements as follows: all elements of the first column, all elements of the M-th row except the element $A_{M,1}$ (which is already output), all remaining elements of the second column, all remaining elements of the (M−1)-th row, and so on; the element $A_{1,M}$ must be output in the end. All rows must be output from left to right, all columns must be output from up to down.

Matrix15. A real-valued square matrix A of order M is given (M is an odd number). Starting with the element $A_{1,1}$ and moving clockwise, output all matrix elements *in the spiral order*: the first row from left to right, the last column from up to down, the last row from right to left, the first column from down to up, all remaining elements of the second row (from left to right), and so on; the central element of the matrix must be output in the end.

Matrix16. A real-valued square matrix A of order M is given (M is an odd number). Starting with the element $A_{1,1}$ and moving counter-clockwise, output all matrix elements *in the spiral order*: the first column from up to down, the last row from left to right, the last column from down to up, the first row from right to left, all remaining elements of the second column (from up to down), and so on; the central element of the matrix must be output in the end.

13.3. Analysis of matrix elements

Matrix17. An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le M$). Find the sum and the product of elements of the matrix row with the order number K.

Matrix18. An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le N$). Find the sum and the product of elements of the matrix column with the order number K.

Matrix19. An $M \times N$ matrix of real numbers is given. Find the sum of elements for each matrix row.

Matrix20. An $M \times N$ matrix of real numbers is given. Find the product of elements for each matrix column.

Matrix21. An $M \times N$ matrix of real numbers is given. Find the average of elements for each matrix row with odd order number (1, 3, …). Do not use conditional statements.

Matrix22. An $M \times N$ matrix of real numbers is given. Find the sum of elements for each matrix column with even order number (2, 4, …). Do not use conditional statements.

Matrix23. An $M \times N$ matrix of real numbers is given. Find the minimal element for each matrix row.

Matrix24. An $M \times N$ matrix of real numbers is given. Find the maximal element for each matrix column.

Matrix25. An $M \times N$ matrix of real numbers is given. Find the order number of the matrix row with the maximal sum of elements. Output this order number and the maximal sum value.

Matrix26. An $M \times N$ matrix of real numbers is given. Find the order number of the matrix column with the minimal product of elements. Output this order number and the minimal product value.

Matrix27. An $M \times N$ matrix of real numbers is given. Find the maximal value among the minimal elements of matrix rows.

Matrix28. An $M \times N$ matrix of real numbers is given. Find the minimal value among the maximal elements of matrix columns.

Matrix29. An $M \times N$ matrix of real numbers is given. For each matrix row find the amount of elements that are smaller than the average of all elements of this row.

Matrix30. An $M \times N$ matrix of real numbers is given. For each matrix column find the amount of elements that are greater than the average of all elements of this column.

Matrix31. An $M \times N$ matrix of real numbers is given. Find the order numbers of row and column for an element whose value is the closest to the average of all matrix elements.

Matrix32. An $M \times N$ matrix of integers is given. Find the order number of the first matrix row that contains the equal amount of positive and negative elements (zero elements are not considered). If the matrix does not contain the required rows then output 0.

Matrix33. An $M \times N$ matrix of integers is given. Find the order number of the last matrix column that contains the equal amount of positive and negative elements (zero elements are not considered). If the matrix does not contain the required columns then output 0.

Matrix34. An $M \times N$ matrix of integers is given. Find the order number of the last matrix row that contains even numbers only. If the matrix does not contain the required rows then output 0.

Matrix35. An $M \times N$ matrix of integers is given. Find the order number of the first matrix column that contains odd numbers only. If the matrix does not contain the required columns then output 0.

Matrix36. An $M \times N$ matrix of integers is given, values of its elements are in the range 0 to 100. A matrix row is called the *similar* with the other row if these rows contain the same set of numbers. For example, rows (1, 3, 3, 2) and (2, 2, 1, 3) contain the same set {1, 2, 3} and therefore they are the similar rows. Find the amount of matrix rows that are the similar with the first row.

Matrix37. An $M \times N$ matrix of integers is given, values of its elements are in the range 0 to 100. A matrix column is called the *similar* with the other column if these columns contain the same set of numbers. For example, columns (1, 3, 3, 2) and (2, 2, 1, 3) contain the same set {1, 2, 3} and therefore they are the similar columns. Find the amount of matrix columns that are the similar with the last column.

Matrix38. An $M \times N$ matrix of integers is given. Find the amount of its rows that contain no elements with equal values.

Matrix39. An $M \times N$ matrix of integers is given. Find the amount of its columns that contain no elements with equal values.

Matrix40. An $M \times N$ matrix of integers is given. Find the order number of the last row that contains the maximal amount of elements with equal values.

Matrix41. An $M \times N$ matrix of integers is given. Find the order number of the first column that contains the maximal amount of elements with equal values.

Matrix42. An $M \times N$ matrix of real numbers is given. Find the amount of its rows whose values of elements are sorted in ascending order.

Matrix43. An $M \times N$ matrix of real numbers is given. Find the amount of its columns whose values of elements are sorted in descending order.

Matrix44. An $M \times N$ matrix of real numbers is given. Find minimal element among elements of all matrix rows whose values of elements are sorted in ascending or descending order. If the matrix does not contain the required rows then output 0 (as a real number).

Matrix45. An $M \times N$ matrix of real numbers is given. Find maximal element among elements of all matrix columns whose values of elements are sorted in ascending or descending order. If the matrix does not contain the required columns then output 0 (as a real number).

Matrix46. An $M \times N$ matrix of integers is given. Find the matrix element that is the maximum in its row and the minimum in its column. If the matrix does not contain such elements then output 0.

13.4. Matrix changing

Matrix47. An $M \times N$ matrix of real numbers and two integers K_1, K_2 are given ($1 \le K_1 < K_2 \le M$). Exchange matrix rows with the order numbers K_1 and K_2.

Matrix48. An $M \times N$ matrix of real numbers and two integers K_1, K_2 are given ($1 \le K_1 < K_2 \le N$). Exchange matrix columns with the order numbers K_1 and K_2.

Matrix49. An $M \times N$ matrix of real numbers is given. For each matrix row exchange values of its minimal and maximal element.

Matrix50. An $M \times N$ matrix of real numbers is given. For each matrix column exchange values of its minimal and maximal element.

Matrix51. An $M \times N$ matrix of real numbers is given. Exchange matrix rows containing the minimal and the maximal element of the matrix.

Matrix52. An $M \times N$ matrix of real numbers is given. Exchange matrix columns containing the minimal and the maximal element of the matrix.

Matrix53. An $M \times N$ matrix of real numbers is given. Exchange the column with the order number 1 and the last column that contains positive numbers only. If the matrix does not contain the required columns then do not change it.

Matrix54. An $M \times N$ matrix of real numbers is given. Exchange the column with the order number N and the first column that contains negative numbers only. If the matrix does not contain the required columns then do not change it.

Matrix55. An $M \times N$ matrix of real numbers is given (M is an even number). Exchange the upper and lower half of the matrix.

Matrix56. An $M \times N$ matrix of real numbers is given (N is an even number). Exchange the left and right half of the matrix.

Matrix57. An $M \times N$ matrix of real numbers is given (M and N are even numbers). Exchange the upper left and lower right quarter of the matrix.

Matrix58. An $M \times N$ matrix of real numbers is given (M and N are even numbers). Exchange the upper right and lower left quarter of the matrix.

Matrix59. An $M \times N$ matrix of real numbers is given. Reflect its elements about the horizontal axis of symmetry of the matrix (that is, exchange matrix rows with the order numbers *1* and M, 2 and $M - 1$, and so on).

Matrix60. An $M \times N$ matrix of real numbers is given. Reflect its elements about the vertical axis of symmetry of the matrix (that is, exchange matrix columns with the order numbers *1* and N, 2 and $N - 1$, and so on).

Matrix61. An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le M$). Remove the matrix row with the order number K.

Matrix62. An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le N$). Remove the matrix column with the order number K.

Matrix63. An $M \times N$ matrix of real numbers is given. Remove the matrix row that contains the minimal matrix element.

Matrix64. An $M \times N$ matrix of real numbers is given. Remove the matrix column that contains the maximal matrix element.

Matrix65. An $M \times N$ matrix of real numbers is given. Remove its first column that contains positive numbers only. If the matrix does not contain the required columns then do not change it.

Matrix66. An $M \times N$ matrix of real numbers is given. Remove its last column that contains negative numbers only. If the matrix does not contain the required columns then do not change it.

Matrix67. An $M \times N$ matrix of real numbers is given. The matrix contains both positive and negative numbers. Remove all matrix columns that contain positive numbers only. If the matrix does not contain the required columns then do not change it.

Matrix68. An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le M$). Insert a new row of elements with zero value before the matrix row with the order number K.

Matrix69. An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le N$). Insert a new column of elements with the value 1 after the matrix column with the order number K.

Matrix70. An $M \times N$ matrix of real numbers is given. Double the occurrence of the matrix row that contains the maximal matrix element.

Matrix71. An $M \times N$ matrix of real numbers is given. Double the occurrence of the matrix column that contains the minimal matrix element.

Matrix72. An $M \times N$ matrix of real numbers is given. Insert a new column of elements with the value 1 before the first matrix column that contains positive numbers only. If the matrix does not contain the required columns then do not change it.

Matrix73. An $M \times N$ matrix of real numbers is given. Insert a new column of elements with zero value after the last matrix column that contains negative numbers only. If the matrix does not contain the required columns then do not change it.

Matrix74. An $M \times N$ matrix of real numbers is given. A matrix element is called the *local minimum* if it is smaller than all its neighbors. Replace all local minimums of the matrix by zero values. An additional matrix may be used for performing the required replacement.

Matrix75. An $M \times N$ matrix of real numbers is given. A matrix element is called the *local maximum* if it is greater than all its neighbors. Replace values of all local maximums of the matrix by opposite values. An additional matrix may be used for performing the required replacement.

Matrix76. An $M \times N$ matrix of real numbers is given. Rearrange the matrix rows so that values of their first elements were in ascending order.

Matrix77. An $M \times N$ matrix of real numbers is given. Rearrange the matrix columns so that values of their last elements were in descending order.

Matrix78. An $M \times N$ matrix of real numbers is given. Rearrange the matrix rows so that minimal values of their elements were in descending order.

Matrix79. An $M \times N$ matrix of real numbers is given. Rearrange the matrix columns so that maximal values of their elements were in ascending order.

13.5. Diagonals of a square matrix

Matrix80. A real-valued square matrix A of order M is given. Find the sum of elements of its *main diagonal*:

$$A_{1,1},\quad A_{2,2},\quad A_{3,3},\quad \ldots,\quad A_{M,M}.$$

Matrix81. A real-valued square matrix A of order M is given. Find the average of elements of its *secondary diagonal*:

$$A_{1,M},\quad A_{2,M-1},\quad A_{3,M-2},\quad \ldots,\quad A_{M,1}.$$

Matrix82. A real-valued square matrix A of order M is given. Find the sum of elements of each matrix diagonal that is parallel to the main diagonal. Begin with the single-element diagonal $A_{1,M}$.

Matrix83. A real-valued square matrix A of order M is given. Find the sum of elements of each matrix diagonal that is parallel to the secondary diagonal. Begin with the single-element diagonal $A_{1,1}$.

Matrix84. A real-valued square matrix A of order M is given. Find the average of elements of each matrix diagonal that is parallel to the main diagonal. Begin with the single-element diagonal $A_{1,M}$.

Matrix85. A real-valued square matrix A of order M is given. Find the average of elements of each matrix diagonal that is parallel to the secondary diagonal. Begin with the single-element diagonal $A_{1,1}$.

Matrix86. A real-valued square matrix A of order M is given. Find the minimal value of elements of each matrix diagonal that is parallel to the main diagonal. Begin with the single-element diagonal $A_{1,M}$.

Matrix87. A real-valued square matrix A of order M is given. Find the maximal value of elements of each matrix diagonal that is parallel to the secondary diagonal. Begin with the single-element diagonal $A_{1,1}$.

Matrix88. A real-valued square matrix of order M is given. Assign zero value to the matrix elements that lie below the main diagonal. Do not use conditional statements.

Matrix89. A real-valued square matrix of order M is given. Assign zero value to the matrix elements that lie above the secondary diagonal. Do not use conditional statements.

Matrix90. A real-valued square matrix of order M is given. Assign zero value to the matrix elements that lie on the secondary diagonal or below it. Do not use conditional statements.

Matrix91. A real-valued square matrix of order M is given. Assign zero value to the matrix elements that lie on the main diagonal or above it. Do not use conditional statements.

Matrix92. A real-valued square matrix of order M is given. Assign zero value to the matrix elements that lie above the main diagonal and above the secondary diagonal simultaneously. Do not use conditional statements.

Matrix93. A real-valued square matrix of order M is given. Assign zero value to the matrix elements that lie above the main diagonal and below the secondary diagonal simultaneously. Do not use conditional statements.

Matrix94. A real-valued square matrix of order M is given. Assign zero value to the matrix elements that lie below the main diagonal (or on it) and above the secondary diagonal (or on it) simultaneously. Do not use conditional statements.

Matrix95. A real-valued square matrix of order M is given. Assign zero value to the matrix elements that lie below the main diagonal (or on it) and below the secondary diagonal (or on it) simultaneously. Do not use conditional statements.

Matrix96. A real-valued square matrix A of order M is given. Reflect its elements about the main diagonal (that is, exchange values of matrix elements $A_{1,2}$ and $A_{2,1}$, $A_{1,3}$ and $A_{3,1}$, and so on). Do not use an additional matrix.

Matrix97. A real-valued square matrix A of order M is given. Reflect its elements about the secondary diagonal (that is, exchange values of matrix elements $A_{1,1}$ and $A_{M,M}$, $A_{1,2}$ and $A_{M-1,M}$, and so on). Do not use an additional matrix.

Matrix98. A real-valued square matrix A of order M is given. Rotate its elements by 180° (that is, exchange values of matrix elements $A_{1,1}$ and $A_{M,M}$, $A_{1,2}$ and $A_{M,M-1}$, and so on). Do not use an additional matrix.

Matrix99. A real-valued square matrix A of order M is given. Rotate its elements counter-clockwise by 90° (that is, assign an initial value of $A_{1,1}$ to $A_{M,1}$, an initial value of $A_{M,1}$ to $A_{M,M}$, and so on). Do not use an additional matrix.

Matrix100. A real-valued square matrix A of order M is given. Rotate its elements clockwise by 90° (that is, assign an initial value of $A_{1,1}$ to $A_{1,M}$, an initial value of $A_{1,M}$ to $A_{M,M}$, and so on). Do not use an additional matrix.

14. Characters and strings

14.1. Basic operations

All input strings are assumed to contain no *overlapping* occurrences of a required substring in the tasks connected with search and replacement. Furthermore, substring removing (in String32 and String35) or substring replacement (in String38) do not lead to *new* occurrences of this substring in the given string.

String1. Given a character C, output its numeric value in the character set.

String2. Given an integer N ($32 \le N \le 126$), output a character with the numeric value N in the character set.

String3. Given a character C, output two characters: the first character precedes C in the character set, the second one follows C in the character set.

String4. Given an integer N ($1 \le N \le 26$), output N first *capital* (that is, uppercase) letters of the English alphabet (“A”, “B”, “C”, and so on).

String5. Given an integer N ($1 \le N \le 26$), output N last *small* (that is, lowercase) letters of the English alphabet in inverse order (“z”, “y”, “x”, and so on).

String6. A character C representing a digit or a letter of the Latin alphabet is given. If C is a digit then output the string “digit”, if C is a capital letter then output the string “capital”, otherwise output the string “small”.

String7. Given a nonempty string, output numeric values of its first and last character in the character set.

String8. Given an integer N (> 0) and a character C, output a string that is of length N and contains characters C.

String9. Given an even integer N (> 0) and two characters C_1, C_2, output a string that is of length N, begins with C_1, and contains alternating characters C_1 and C_2.

String10. Given a string, output a new string that contains the given string characters in inverse order.

String11. Given a nonempty string, output a new string that contains the given string characters separated by a blank character.

String12. Given a nonempty string and an integer N (> 0), output a new string that contains the given string characters separated by N characters "*".

String13. Given a string, find the amount of digits in the string.

String14. Given a string, find the amount of Latin capital letters in the string.

String15. Given a string, find the amount of Latin letters in the string.

String16. Given a string, convert all Latin capital letters of the string to lowercase.

String17. Given a string, convert all Latin small letters of the string to uppercase.

String18. Given a string, convert all Latin capital letters of the string to lowercase and all Latin small letters of the string to uppercase.

String19. A string is given. If the string represents an integer then output 1, if the string represents a real number (with nonzero fractional part) then output 2, otherwise output 0. A fractional part of a real number is preceded by the *decimal point* ".".

String20. Given a positive integer, output all digit characters in the decimal representation of the integer (from left to right).

String21. Given a positive integer, output all digit characters in the decimal representation of the integer (from right to left).

String22. Given a string that represents a positive integer, output the sum of digits of this integer.

String23. Given a string that represents an arithmetic expression of the form "<digit>±< digit>±…±<digit>" with operators "+" and "−" only (for example, "4+7−2−8"), output the value of given expression as an integer.

String24. Given a string with the binary representation of a positive integer, output a new string with the decimal representation of this integer.

String25. Given a string with the decimal representation of a positive integer, output a new string with the binary representation of this integer.

String26. An integer N (> 0) and a string S are given. Transform the string S to a new string of length N as follows: if the length of S is greater than N then remove its first characters, if the length of S is less than N then add characters "." to the beginning of S.

String27. Two positive integers N_1, N_2 and two strings S_1, S_2 are given. Output new string that contains N_1 first characters of the string S_1 and N_2 last characters of the string S_2 (in that order).

String28. Given a character *C* and a string *S*, double each occurrence of the character *C* in the string *S*.

String29. Given a character *C* and two strings *S*, S_0, insert the string S_0 into the string *S* before each occurrence of the character *C*.

String30. Given a character *C* and two strings *S*, S_0, insert the string S_0 into the string *S* after each occurrence of the character *C*.

String31. Two strings *S*, S_0 are given. If the string S_0 is a substring of *S* then output True, otherwise output False.

String32. Two strings *S*, S_0 are given. Find the amount of occurrences of S_0 in the string *S*.

String33. Two strings *S*, S_0 are given. Remove the first occurrence of S_0 from the string *S*. If the string *S* does not contain required substrings then do not change it.

String34. Two strings *S*, S_0 are given. Remove the last occurrence of S_0 from the string *S*. If the string *S* does not contain required substrings then do not change it.

String35. Two strings *S*, S_0 are given. Remove all occurrences of S_0 from the string *S*. If the string *S* does not contain required substrings then do not change it.

String36. Three strings *S*, S_1, S_2 are given. Replace the first occurrence of S_1 in the string *S* by the string S_2.

String37. Three strings *S*, S_1, S_2 are given. Replace the last occurrence of S_1 in the string *S* by the string S_2.

String38. Three strings *S*, S_1, S_2 are given. Replace all occurrences of S_1 in the string *S* by the string S_2.

String39. A string with at least one blank character is given. Output the substring of *S* that contains all characters between the first and the second blank character. If the string *S* contains only one blank character then output an empty string.

String40. A string with at least one blank character is given. Output the substring of *S* that contains all characters between the first and the last blank character. If the string *S* contains only one blank character then output an empty string.

14.2. Word processing

In this subgroup of tasks all input strings are assumed to be nonempty and without leading and trailing blank characters.

String41. A string that contains English words separated by one or more blank characters is given. Find the amount of words in the string.

String42. A string that contains English words separated by one or more blank characters is given. All string letters are in uppercase. Find the amount of words whose first letter is coincides with the last one.

String43. A string that contains English words separated by one or more blank characters is given. All string letters are in uppercase. Find the amount of words containing at least one letter “E”.

String44. A string that contains English words separated by one or more blank characters is given. All string letters are in uppercase. Find the amount of words containing exactly three letters “E”.

String45. A string that contains English words separated by one or more blank characters is given. Find the length of the shortest word.

String46. A string that contains English words separated by one or more blank characters is given. Find the length of the longest word.

String47. A string that contains English words separated by one or more blank characters is given. Output a new string that contains the given words (in the same order) separated by one character “.”.

String48. A string that contains English words separated by one or more blank characters is given. All string letters are in uppercase. Process each word as follows: replace all next occurrences of its first letter by the character “.” (for example, the word “MINIMUM” must be transformed into “MINI.U.”). Do not change blank characters in the string.

String49. A string that contains English words separated by one or more blank characters is given. All string letters are in uppercase. Process each word as follows: replace all previous occurrences of its last letter by the character “.” (for example, the word “MINIMUM” must be transformed into “.INI.UM”). Do not change blank characters in the string.

String50. A string that contains English words separated by one or more blank characters is given. Output a new string that contains the given words in inverse order. The words must be separated by one blank character.

String51. A string that contains English words separated by one or more blank characters is given. All string letters are in uppercase. Output a new string that contains the given words in alphabetic order. The words must be separated by one blank character.

String52. A string with an English sentence is given. Convert the first letter of each word to uppercase. A *word* is defined as a character sequence that does not contain blank characters and is bounded by blank characters or the string beginning/end. If the first word character is not a letter then do not change this word.

String53. A string with an English sentence is given. Find the amount of punctuation marks in the string.

String54. A string with an English sentence is given. Find the amount of vowels (“a”, “i”, “e”, “o”, “u”) in the string.

String55. A string with an English sentence is given. Output the longest word in the string. If there are several words of the maximal length then output the first one. A *word* is defined as a character sequence that does not contain blank characters, punctuation marks and is bounded by blank characters, punctuation marks or the string beginning/end.

String56. A string with an English sentence is given. Output the shortest word in the string. If there are several words of the maximal length then output the last one. A *word* is defined as a character sequence that does not contain blank characters, punctuation marks and is bounded by blank characters, punctuation marks or the string beginning/end.

String57. A string with an English sentence is given. Remove all superfluous blank characters in the string, so that its words were separated by exactly one blank character.

14.3. Additional tasks

String58. A string that contains a *fully qualified path name* (that is, the drive and directory parts, the file name and extension) is given. Extract the file name (without the path and extension) from the string.

String59. A string that contains a *fully qualified path name* (that is, the drive and directory parts, the file name and extension) is given. Extract the extension (without the preceding dot character) from the string.

String60. A string that contains a *fully qualified path name* (that is, the drive and directory parts, the file name and extension) is given. Extract the first directory name (without backslashes “\”) from the string. If the file with the given name is located in the root directory then output a backslash.

String61. A string that contains a *fully qualified path name* (that is, the drive and directory parts, the file name and extension) is given. Extract the last directory name (without backslashes “\”) from the string. If the file with the given name is located in the root directory then output a backslash.

String62. A string with an English sentence is given. Encrypt the string using the right cyclic shift of any letter by one position of the English alphabet (for instance, the letter “A” is encoded by the letter “B”, “a” is encoded by “b”, “B” is encoded by “C”, “z” is encoded by “a”, and so on). Do not change blank characters and punctuation marks.

String63. A string with an English sentence and an integer K ($0 < K < 10$) are given. Encrypt the string using the right cyclic shift of any letter by K positions of the English alphabet (for instance, if K = 2 then the letter “A” is encoded by the letter “C”, “a” is encoded by “c”, “B” is encoded by “D”, “z” is encoded by “b”, and so on). Do not change blank characters and punctuation marks.

String64. A string with an encrypted English sentence and an integer K ($0 < K < 10$) are given. The string is encrypted by means of the right cyclic shift of any letter by K positions of the English alphabet (see String63). Decrypt the given string.

String65. A string with an encrypted English sentence and its decrypted first character C are given (the character C is always an English letter). The string is encrypted by means of the right cyclic shift of any letter by K positions of the English alphabet (see String63). Find the number K and decrypt the given string.

String66. A string with an English sentence is given. Encrypt the string by moving all characters that are at the string positions with even numbers (2, 4, …) to the beginning of the string (in the same order) and moving all characters that are at the string positions with odd numbers (1, 3, …) to the end of the string (in inverse order). For instance, the string “Program” must be encrypted to “rgamroP”.

String67. A string with an encrypted English sentence is given (the method of encryption is described in String66). Decrypt the string.

String68. A string that contains digits and Latin small letters is given. If letters of the string are in alphabetic order then output 0, otherwise output the order number of the first string character that is a letter and breaks the required order.

String69. A string that contains Latin letters and parentheses “(“, “)” is given. If parentheses are in correct order (that is, each closing parenthesis ”)” corresponds to an opening one “(“) then output 0. If the string contains illegal parentheses “)” then output the order number of the first string character that is an illegal “)”. If the amount of closing parentheses is less than the amount of opening ones then output −1.

String70. A string that contains Latin letters and brackets of three types (parentheses ”()”, square brackets ”[]”, braces “{}”) is given. If brackets are in correct order (that is, each closing bracket corresponds to an opening one of the same type) then output 0. If the string contains illegal closing brackets then output the order number of the first string character that is an illegal closing bracket. If the amount of closing brackets is less than the amount of opening ones then output −1.

15. Binary files

The *binary file* is a file that contains elements of the same type in the binary format.

In Pascal, such files are called *typed files* and should be declared as `file of <element type>`.

In Visual Basic, such files are called *random access* files and should be declared with the `Random` attribute.

In C++, binary files should be opened in the `ios_base::binary` mode; the read and write operations for binary files are implemented by the `read` and `write` methods with the `((char *)&x, sizeof(x))` parameters, where the `x` variable has a type that matches the type of file elements.

In .NET languages, binary files should be processed with using the FileStream, BinaryReader, and BinaryWriter classes.

In Python and Ruby, binary files should be opened with the `b` attribute of the file access mode, for example, `"rb"` provides for opening existing binary file in read mode, `"wb"` provides for creation binary file and opening this file in write mode, `"r+b"` — provides for opening existing binary file in read and write mode. Use the unpack and pack methods while processing files with data of numerical types (in Python, these methods are defined in the struct module).

In Java, binary files should be processed with using the RandomAccessFile class.

The condition “A file of integers (real numbers, characters, strings) is given” means that the *file name* is given (as a string input data) and the corresponding file exists and is located in the working directory. In some tasks it is necessary to check the file existence; this task condition is pointed out explicitly. File components must be read using standard input procedures (or functions) of the programming language being used.

If the task requires to create a new file then the new file name is included in the input data set; this name is usually the last element of input data. Standard output procedures (or functions) of the programming language must be used for writing data to files.

The *size of a typed file* always means the amount of *file components* of the corresponding type. File components are assumed to be numbered beginning with 1.

The minimal size of any input file is assumed to be 2 (that is, any file contains at least two components), if the task does not specify it explicitly. The maximal file size is not fixed, so auxiliary *arrays* should not be used to store all file components but one may use temporary *files*.

15.1. Basic operations

File1. A string S is given. If the string S is an admissible name of file then create an empty file called S and output True. If a file with the name S can not be created then output False.

File2. A file name and an integer N (> 1) are given. Create a file of integers with the given name and write N first positive even numbers (2, 4, …) to this file.

File3. A file name and two real numbers A, D are given. Create a file of real numbers with the given name and write 10 first terms of an *arithmetic sequence* with the first term A and the common difference D (A, $A + D$, $A + 2\cdot D$, …, $A + 9\cdot D$) to this file.

File4. Four file names are given. Find the amount of files with the given names that are located in the working directory.

File5. A name of file of integers is given. Find the amount of the file components. If a file with the given name does not exist then output -1.

File6. An integer *K* and a file of nonnegative integers are given. Output the file component with the order number *K* (file components are numbered beginning with 1). If the file does not contain a component with the required number then output -1.

File7. Given a file of integers that contains at least four components, output its components with the order numbers 1, 2, $N-1$, N, where N is the amount of the file components.

File8. Two names of files of real numbers are given. It is known that the first file exists and is nonempty whereas the second file is absent in the working directory. Create the absent file and write the first and the last components of the existing file (in that order) to the file that has been created.

File9. Two names of files of real numbers are given. It is known that one of these files exists and is nonempty whereas the other file is absent in the working directory. Create the absent file and write the last and the first components of the existing file (in that order) to the file that has been created.

File10. A file of integers is given. Create a new file that contains all components of the given file in inverse order.

File11. A file of real numbers is given. Create two new files; the first resulting file must contain the given file components with odd order numbers (1, 3, …), the second resulting file must contain the given file components with even order numbers (2, 4, …).

File12. A file of integers is given. Create two new files; the first resulting file must contain the given file components whose values are even numbers, the second resulting file must contain the given file components whose values are odd numbers. If the given file does not contain odd or even numbers then the corresponding resulting file must be empty.

File13. A file of integers is given. Create two new files; the first resulting file must contain the given file components (in inverse order) whose values are positive numbers, the second resulting file must contain the given file components (in inverse order too) whose values are negative numbers. If the given file does not contain positive or negative numbers then the corresponding resulting file must be empty.

File14. A file of real numbers is given. Find the average of its components.

File15. A file of real numbers is given. Find the sum of its components with even order numbers.

File16. A file of integers is given. Find the amount of series of equal numbers that are contained in the file (a *series of equal numbers* is defined as a set of successive file components with equal values). For instance, if the file contains components with values 1, 5, 5, 5, 4, 4, 5 then the number 4 must be output.

File17. A file of integers is given. Create a new file of integers that contains lengths of all series of equal numbers from the given file (a *series of equal numbers* is defined as a set of successive file components with equal values; the amount of these components is called the *length of series*). For instance, if the given file contains components with values 1, 5, 5, 5, 4, 4, 5 then the resulting file must contain the following integers: 1, 3, 2, 1.

File18. A file of real numbers is given. Find its first local minimum (a file component is called the *local minimum* if it is smaller than its neighbors).

File19. A file of real numbers is given. Find its last local maximum (a file component is called the *local maximum* if it is greater than its neighbors).

File20. A file of real numbers is given. Find the total amount of its *local extremums*, that is, its local maximums and local minimums (see the *local minimum* and *local maximum* definitions in File18 and File19 respectively).

File21. A file of real numbers is given. Create a file of integers that contains order numbers of all local maximums of the given file in ascending order (see the *local maximum* definition in File19).

File22. A file of real numbers is given. Create a file of integers that contains order numbers of all local extremums of the given file in descending order (see the *local extremum* definition in File20).

File23. A file of real numbers is given. Create a file of integers that contains lengths of all sets of successive file components whose values are in descending order. For instance, if the given file contains components with values 1.7, 4.5, 3.4, 2.2, 8.5, 1.2 then the resulting file must contain the following integers: 3, 2.

File24. A file of real numbers is given. Create a file of integers that contains lengths of all sets of successive file components whose values are in ascending or descending order. For instance, if the given file contains components with values 1.7, 4.5, 3.4, 2.2, 8.5, 1.2 then the resulting file must contain the following integers: 2, 3, 2, 2.

File25. Given a file of real numbers, replace the values of all file components with their squares.

File26. Given a file of real numbers, exchange values of its minimal and maximal component.

File27. Given a file of integers with components A_1, A_2, …, A_N (N is the amount of the file components), change the order of the file components as follows:

$$A_1,\quad A_N,\quad A_2,\quad A_{N-1},\quad A_3,\quad \ldots\,.$$

File28. Given a file of real numbers, replace each file component (except for the first and last one) with the average of this component and its neighbors.

File29. Given a file of integers that contains more than 50 components, reduce the size of file to 50 components by means of removing last components.

File30. Given a file of integers that contains an even number of components, remove the second half of the file components.

File31. Given a file of integers that contains more than 50 components, reduce the size of file to 50 components by means of removing first components.

File32. Given a file of integers that contains an even number of components, remove the first half of the file components.

File33. Given a file of integers, remove file components with even order numbers (2, 4, …).

File34. Given a file of integers, remove file components with negative values.

File35. Given a file of integers that contains less than 50 components, increase the size of file up to 50 components by means of inserting zero components to the beginning of the file.

File36. Given a file of integers, write all its components in the same order to the end of this file (as a result, the size of file will be doubled).

File37. Given a file of integers, write all its components in inverse order to the end of this file (as a result, the size of file will be doubled).

File38. Given a file of integers, double occurrences of file components with odd order numbers.

File39. Given a file of integers, double occurrences of file components whose values are in the range 5 to 10.

File40. Given a file of integers, replace each file component, whose order number is an even number, with two zero components.

File41. Given a file of integers, replace each file component, whose value is a positive number, with three zero components.

15.2. Untyped files processing

File42. Given two untyped files, exchange their contents.

File43. Given an untyped file and a string S, create the file copy called S.

File44. Given three untyped files of different size, replace the contents of the longest file with the contents of the shortest one.

File45. Given three untyped files of different size, replace the contents of the shortest file with the contents of the longest one.

File46. A string S_0, an integer N (≤ 4), and N files called $S_1, \ldots, S_N$ are given. Components of all given files have the identical type. Combine the contents of the given files (in the same order) in a new file called S_0.

File47. Given two files whose components have the identical type, add the initial contents of the second file to the end of the first file and the initial contents of the first file to the end of the second file.

15.3. Work with several numeric files. Archival files

File48. Three files of integers called S_A, S_B, S_C and a string S_D are given. All given files are of the same size. Create a new file called S_D; this file must contain triples of components of the given files as follows:

$$A_1,\quad B_1,\quad C_1,\quad A_2,\quad B_2,\quad C_2,\quad \ldots .$$

File49. Four files of integers called S_A, S_B, S_C, S_D and a string S_E are given. The given files are of different size. Create a new file called S_E; this file must contain quadruples of components of the given files as follows:

$$A_1,\quad B_1,\quad C_1,\quad D_1,\quad A_2,\quad B_2,\quad C_2,\quad D_2,\quad \ldots .$$

Do not write “superfluous” components of longer files to the resulting file.

File50. Two files of real numbers called S_1, S_2 and a string S_3 are given. Values of components of each given file are in ascending order. Create a new file called S_3; this file must contain all components of the given files in ascending order of their values.

File51. Three files of real numbers called S_1, S_2, S_3 and a string S_4 are given. Values of components of each given file are in descending order. Create a new file called S_4; this file must contain all components of the given files in descending order of their values.

File52. A string S_0, an integer N (≤ 4), and N files of integers called $S_1, \ldots, S_N$ are given. Create a new file called S_0; this file (an *archival file*) must contain data of all given files in the following format: the number N (the first component of the archival file), the amounts of components of each given file (the next N components of the archival file), values of all components of the file S_1 in the same order, values of all components of the file S_2 in the same order, and so on.

File53. A string S, an integer N (> 0) and an *archival file* are given. The archival file contains data of several files; the format of archival file is described in File52. Restore the data of the N-th file from the archival file and save it in the file called S. If the amount of files, which are contained in the archival file, is less than N then the resulting file S must be empty.

File54. A string S and an *archival file* are given. The archival file contains data of several (no more than six) files; the format of archival file is described in File52. Find the average of all component values for each file, which is contained in the archival file. Save all obtained averages (as a real numbers) in a new file called S.

File55. A string S_0, an integer N (≤ 4), and N files of integers called $S_1, \ldots, S_N$ are given. Create a new file called S_0; this file (an *archival file*) must contain data of all given files in the following format: the amount of components of the file S_1 and values of all its components in the same order; the amount of components of the file S_2 and values of all its components in the same order; …; the amount of components of the file S_N and values of all its components in the same order.

File56. A string S, an integer N (> 0) and an *archival file* are given. The archival file contains data of several files; the format of archival file is described in File55. Restore the data of the N-th file from the archival file and save it in the file called S. If the amount of files, which are contained in the archival file, is less than N then the resulting file S must be empty.

File57. Two strings S_1, S_2 and an *archival file* are given. The archival file contains data of several files; the format of archival file is described in File55. Create two new files called S_1, S_2; the first resulting file must contain first components of all files, which are contained in the archival file, the second resulting file must contain last components of all files, which are contained in the archival file.

15.4. Files of characters and files of strings

A *file of characters* is a binary file whose components are characters. A *file of strings* (or a *string binary file*) is a binary file whose components are strings of characters. Unlike text files, files of strings can be opened in read/write access mode (that is, allows both reading and writing), and their components can be accessed both sequentially and randomly (by moving the current file position to a specified component).

In Borland Delphi and Lazarus, variables for string binary files should be declared as `file of ShortString`.

In Borland Delphi, Lazarus, and PascalABC.NET, variables for string binary files should be declared as `file of ShortString`.

In Visual Basic, the components of a file of characters are assumed to have the `String * 1` type and the components of a string binary file are assumed to have the `String * 80` type in any task of Programming Taskbook.

In C++, the components of a string binary file are assumed to have the `char[80]` type in any task of Programming Taskbook.

In .NET languages, Python, Java, and Ruby, string binary files are assumed to contain left-aligned strings of length 80 in any task of Programming Taskbook (therefore, you may need to use a method for removing right blank characters from the string after reading it from a string binary file and a method for padding string by blank characters before writing it to a string binary file).

In Java, you should use variables of type `byte` and `byte[]` to read/write characters and strings from/to binary files. Use appropriate constructor of the String class to convert bytes to string, use the getBytes method of the String class to convert string to bytes.

File58. Given a file of characters containing at least one blank character, remove its components that are located after the first blank character (including this blank character).

File59. Given a file of characters containing at least one blank character, remove its components that are located after the last blank character (including this blank character).

File60. Given a file of characters containing at least one blank character, remove its components that are located before the first blank character (including this blank character).

File61. Given a file of characters containing at least one blank character, remove its components that are located before the last blank character (including this blank character).

File62. Given a file of characters, rearrange file components in ascending order of their numeric values in the character set.

File63. An integer K (> 0) and a file of strings are given. Create two new files; components of the first resulting file are strings that contain first K characters of each string of the given file; components of the second resulting file are K-th characters of each string of the given file. If the length of some string of the given file is less than K then the entire string and a blank character must be written to the first and second resulting file respectively.

File64. Given a file of strings, write its components of the minimal length to a new file (in the same order).

File65. Given a file of strings, write its components of the maximal length to a new file (in inverse order).

File66. Given a file of strings, write all its components to a new file in *lexicographic order* (that is, in ascending order of the numeric values of their characters).

File67. A file of strings is given. The file contains dates in the “day/month/year” format, the “day” and “month” fields contain two digits, the “year” field contains four digits (for example, “16/04/2001”). Create two new files and write integer values of days and months for each date from the given file to the first and second resulting file respectively (in the same order).

File68. A file of strings is given; the file contains dates in the “day/month/year” format (see File67). Create two new files and write integer values of months and years for each date from the given file to the first and second resulting file respectively (in inverse order).

File69. A file of strings is given; the file contains dates in the “day/month/year” format (see File67). Write its components that corresponds to summer dates to a new file (in the same order). If the given file does not contain required dates then the resulting file must be empty.

File70. A file of strings is given; the file contains dates in the “day/month/year” format (see File67). Write its components that corresponds to winter dates to a new file (in the same order). If the given file does not contain required dates then the resulting file must be empty.

File71. A file of strings is given; the file contains dates in the “day/month/year” format (see File67). Find its component that represents the earliest spring date. If the given file does not contain spring dates then output an empty string.

File72. A file of strings is given; the file contains dates in the “day/month/year” format (see File67). Find its component that represents the latest autumn date. If the given file does not contain autumn dates then output an empty string.

File73. A file of strings is given; the file contains dates in the “day/month/year” format (see File67). Write all its components to a new file in descending order of their date values.

15.5. Files containing matrices

An $M \times N$ *matrix* is a rectangular table of numbers consisting of M horizontal rows and N vertical columns. As a rule, two-dimensional array are used to store matrix elements (see tasks of the Matrix group). But matrix elements can be stored in binary files of real numbers too. The files of such a “matrix” structure are considered in this subsection. As well as in other tasks of the File group, additional *arrays* should not be used to store all file components.

The tasks of this subsection use some notions of matrix theory. Let us recall the definitions of these notions.

If A is an $M \times N$ matrix then the matrix B of order $N \times M$ formed from A by interchanging its rows with its columns is called the *transpose* of A:

$$B_{I,J} = A_{J,I}, \qquad I = 1, \ldots, N, \quad J = 1, \ldots, M.$$

Let A be an $M \times N$ matrix and B be an $N \times P$ matrix. An $M \times P$ matrix C is called the *product* of A and B (and is denoted by $A{\cdot}B$) if its elements satisfy the following relations:

$$C_{I,J} = A_{I,1}{\cdot}B_{1,J} + A_{I,2}{\cdot}B_{2,J} + \ldots + A_{1,N}{\cdot}B_{N,J}, \qquad I = 1, \ldots, M, \quad J = 1, \ldots, P.$$

A square matrix A is called an *upper triangular matrix* if all its elements below the main diagonal are equal to zero (see the *main diagonal* definition in the task Matrix80):

$$A_{I,J} = 0, \qquad I > J.$$

A square matrix A is called a *lower triangular matrix* if all its elements above the main diagonal are equal to zero:

$$A_{I,J} = 0, \qquad I < J.$$

A square matrix A is called a *tridiagonal matrix* if all its elements that are off the main diagonal and two adjacent diagonals are equal to zero:

$$A_{I,J} = 0, \qquad |I - J| > 1.$$

File74. Two integers I, J and a file of real numbers are given. This file contains elements of square matrix (by rows). Output the value of the matrix element in the I-th row and J-th column (rows and columns are numbered beginning with 1). If the given matrix does not contain the required row or column then output 0 (as a real number).

File75. A file of real numbers is given. This file contains elements of square matrix (by rows). Create a new file that contains the *transpose* of the given matrix.

File76. Two files of real numbers called S_A and S_B are given. These files contain elements of square matrix A and B (by rows). Create a new file called S_C that contains elements of the *matrix product* $A \cdot B$. If matrices A and B cannot be multiplied then the resulting file S_C must be empty.

File77. Two integers I, J and a file of real numbers are given. The first component of the file contains the amount of matrix columns, other file components contain matrix elements (by rows). Output the value of the matrix element in the I-th row and J-th column (rows and columns are numbered beginning with 1). If the given matrix does not contain the required row or column then output 0 (as a real number).

File78. A file of real numbers is given. The first component of the file contains the amount of matrix columns, other file components contain matrix elements (by rows). Create a new file that contains the *transpose* of the given matrix. The resulting file must have the same structure as the given file.

File79. Two files of real numbers called S_A and S_B are given. The first components of the files S_A and S_B contain the amount of columns of matrices A and B respectively, other file components contain elements of matrices A and B (by rows). Create a new file called S_C that contains the *matrix product* $A \cdot B$ (the resulting file must have the same structure as the given files). If matrices A and B cannot be multiplied then the resulting file must be empty.

File80. A file of real numbers is given. This file contains elements of an *upper triangular matrix* (by rows). Create a new file that contains elements of nonzero part of the given matrix (by rows).

File81. A file of real numbers is given. This file contains elements of a *lower triangular matrix* (by rows). Create a new file that contains elements of nonzero part of the given matrix (by rows).

File82. A file of real numbers is given. This file contains elements of a *tridiagonal* matrix (by rows). Create a new file that contains elements of nonzero part of the given matrix (by rows).

File83. Two integers I, J and a file of real numbers are given. This file contains elements of nonzero part of an *upper triangular matrix* (by rows). Output the matrix order and the value of the matrix element in the I-th row and J-th column (rows and columns are numbered beginning with 1). If the required element lies in zero part of the matrix then output 0 (as a real number). If the given matrix does not contain the required row or column then output -1 (as a real number).

File84. Two integers I, J and a file of real numbers are given. This file contains elements of nonzero part of a *lower triangular matrix* (by rows). Output the matrix order and the value of the matrix element in the I-th row and J-th column (rows and columns are numbered beginning with 1). If the required element lies in zero part of the matrix then output 0 (as a real number). If the given matrix does not contain the required row or column then output -1 (as a real number).

File85. Two integers *I*, *J* and a file of real numbers are given. This file contains elements of nonzero part of a *tridiagonal matrix* (by rows). Output the matrix order and the value of the matrix element in the *I*-th row and *J*-th column (rows and columns are numbered beginning with 1). If the required element lies in zero part of the matrix then output 0 (as a real number). If the given matrix does not contain the required row or column then output −1 (as a real number).

File86. A file of real numbers is given. This file contains elements of nonzero part of an *upper triangular matrix* (by rows). Create a new file that contains all elements of the given matrix (by rows).

File87. A file of real numbers is given. This file contains elements of nonzero part of a *lower triangular matrix* (by rows). Create a new file that contains all elements of the given matrix (by rows).

File88. A file of real numbers is given. This file contains elements of nonzero part of a *tridiagonal matrix* (by rows). Create a new file that contains all elements of the given matrix (by rows).

File89. Two files of real numbers called S_A and S_B are given. These files contain nonzero parts of *upper triangular* matrices *A* and *B* (by rows). Create a new file called S_C that contains nonzero part of the *matrix product* $A·B$ (by rows). If matrices *A* and *B* cannot be multiplied then the resulting file must be empty.

File90. Two files of real numbers called S_A and S_B are given. These files contain nonzero parts of *lower triangular* matrices *A* and *B* (by rows). Create a new file called S_C that contains nonzero part of the *matrix product* $A·B$ (by rows). If matrices *A* and *B* cannot be multiplied then the resulting file must be empty.

16. Text files

The condition “A text file is given” means that the *file name* is given (as a string input data) and the corresponding file exists and is located in the working directory. File components (as a rule, lines of characters) must be read using standard input procedures (or functions) of a programming language being used.

If the task requires to create a new file then the new file name is included in the input data set; this name is usually the last element of input data. Standard output procedures (or functions) of the programming language must be used for writing data to files.

The size of any input file is not fixed, so additional *arrays* should not be used to store all file components but one may use temporary *files*.

Binary files may also be used in the tasks of “Text files” group; see the beginning of the section “Binary files” for additional rules connected with files of this kind.

16.1. Basic operations

Text1. Given a file name and two positive integers *N* and *K*, create a new text file with this name. The file must contain *N* lines, each line consists of *K* characters ”*”.

Text2. Given a file name and an integer *N* ($0 < N < 27$), create a new text file with this name. The file must contain *N* lines, the first line consisting of one Latin small letter “a”, the second one consisting of two letters “ab”, and so on; the last line must consist of *N* initial small letters of the English alphabet.

Text3. Given a file name and an integer *N* ($0 < N < 27$), create a new text file with this name. The file must contain *N* lines of length *N*. The *K*-th line ($K = 1, \ldots, N$) must begin with *K* initial capital letters of the English alphabet and must end with the characters “*”. For instance, if *N* equals 4 then lines of the resulting file must be as follows: “A***”, “AB**”, “ABC*”, “ABCD”.

Text4. Given a text file, output the amount of its characters and lines. EOLN (*end-of-line*) and EOF (*end-of-file*) markers must not be counted.

Text5. Given a string *S* and a text file, add the string *S* to the end of the file.

Text6. Given two text files, add the contents of the second file to the end of the first one.

Text7. Given a string *S* and a text file, add the string *S* to the beginning of the file.

Text8. Given two text files, add the contents of the second file to the beginning of the first one.

Text9. Given an integer *K* and a text file, insert an empty line before the file line with the order number *K*. If the file does not contain a line with the required number then do not change it.

Text10. Given an integer *K* and a text file, insert an empty line after the file line with the order number *K*. If the file does not contain a line with the required number then do not change it.

Text11. Given a text file, double occurrences of all empty lines of the file.

Text12. Given a string *S* and a text file, replace all empty lines of the file with the contents of the string *S*.

Text13. Given a nonempty text file, remove its first line.

Text14. Given a nonempty text file, remove its last line.

Text15. Given an integer *K* and a text file, remove the file line with the order number *K*. If the file does not contain a line with the required number then do not change it.

Text16. Given a text file, remove all empty lines from the file.

Text17. Given two text files, add each line of the second file to the end of the corresponding line of the first one. If the second file is shorter (in lines) than the first one then do not change the remaining lines of the first file.

Text18. Given an integer *K* and a text file, remove *K* leading characters from each line of the file. If the length of some line is less than *K* then remove all characters from the line.

Text19. Given a text file, replace all its Latin capital letters with the corresponding small letters, and all Latin small letters with the capital ones.

Text20. Given a text file, replace its successive blank characters with the single blank character.

Text21. Given a text file that contains more than three lines, remove its last three lines.

Text22. Given an integer *K* ($0 < K < 10$) and a text file that contains more than *K* lines, remove its last *K* lines.

Text23. An integer *K* ($0 < K < 10$) and a text file that contains more than *K* lines are given. Create a new file that contains *K* last lines of the given file.

16.2. Text analysis and formatting

Text24. A text file is given. Find the amount of paragraphs of the given text provided that paragraphs are separated by one or more empty lines.

Text25. An integer *K* and a text file are given. Remove a paragraph with the order number *K* from the given file provided that paragraphs are separated by one or more empty lines. Empty lines must not be removed. If the file does not contain a paragraph with the required number then do not change it.

Text26. A text file is given. Find the amount of paragraphs of the given text provided that the first line of each paragraph is indented by five blank characters. Empty lines must not be counted.

Text27. An integer *K* and a text file are given. Remove a paragraph with the order number *K* from the given file provided that the first line of each paragraph is indented by five blank characters. Empty lines must not be counted and removed. If the file does not contain a paragraph with the required number then do not change it.

Text28. A text file is given. The file does not contain empty lines, the first line of each paragraph is indented by five blank characters. Insert an empty line between adjacent paragraphs (do not insert empty lines to the beginning and end of the file).

Text29. A text file is given. Output its first word of the maximal length. A *word* is defined as a character sequence that does not contain blank characters and is bounded by blank characters or the line beginning/end.

Text30. A text file is given. Output its last word of the minimal length. A *word* is defined as a character sequence that does not contain blank characters and is bounded by blank characters or the line beginning/end.

Text31. An integer *K* and a text file are given. Create a new binary file of strings that contains all words of length *K* from the given file. A *word* is defined as a character sequence that does not contain blank characters, punctuation marks and is bounded by blank characters, punctuation marks or the line beginning/end. If the given file does not contain words of length *K* then the resulting file must be empty.

Text32. A char *C* that is a Latin capital letter and a text file are given. Create a new binary file of strings that contains words of the given text with the first character *C* (of uppercase or lowercase). A *word* is defined as a character sequence that does not contain blank characters, punctuation marks and is bounded by blank characters, punctuation marks or the line beginning/end. If the given file does not contain the required words then the resulting file must be empty.

Text33. A char *C* that is a Latin small letter and a text file are given. Create a new binary file of strings that contains words of the given text with at least one character *C* (of uppercase or lowercase). A *word* is defined as a character sequence that does not contain blank characters, punctuation marks and is bounded by blank characters, punctuation marks or the line beginning/end. If the given file does not contain the required words then the resulting file must be empty.

Text34. A text file whose lines are left-aligned is given. Make the given text right-aligned by means of adding leading blank characters to all nonempty lines. The width of text must be equal to 50 characters.

Text35. A text file whose lines are left-aligned is given. Make the given text centered by means of adding leading blank characters to all nonempty lines. The width of text must be equal to 50 characters. If the length of line is an odd number then add one blank character to the beginning of this line before centering.

Text36. A text file whose lines are right-aligned is given. Make the given text centered by means of removing a half of leading blank characters from all nonempty lines. If the amount of leading blank characters of line is an odd number then remove one blank character from the beginning of this line before centering.

Text37. A text file whose lines are left-aligned is given. Paragraphs of the given text are separated by one empty line. Make the given text *full-aligned* (that is, left-aligned and right-aligned simultaneously) by means of inserting additional blank characters between words in all nonempty lines except the last line of each paragraph. Spaces between words in each line must be processed from right to left; the width of text must be equal to 50 characters.

Text38. An integer *K* (> 25) and a text file, whose lines are left-aligned, are given. Paragraphs of the given text are separated by one empty line. Format the given text so that all its lines consist of no more than *K* characters. The resulting text must contain the same paragraphs and be left-aligned. Blank characters at the end of lines must be removed. Save the formatted text in a new text file.

Text39. An integer K (> 25) and a text file, whose lines are left-aligned, are given. The file does not contain empty lines, the first line of each paragraph is indented by five blank characters. Format the given text so that all its lines consist of no more than K characters. The resulting text must contain the same paragraphs and be left-aligned. Blank characters at the end of lines must be removed. Save the formatted text in a new text file.

16.3. Text files with numeric data

The decimal separator in string representation of any real number with nonzero fractional part is assumed to be the *decimal point* "." in all tasks of this subsection.

Text40. Two files of integers are given; the files contain the equal amount of components. Create a new text file that contains string representations of components of the given binary files. These representations must be arranged in two columns, each of 30 characters width (the first and second columns contain components of the first and second file respectively). Each line of the resulting text must start and end with "|" separator (the numeric value 124), all string representations of integers are right-aligned with respect to column boundary.

Text41. Three files of integers are given; the files contain the equal amount of components. Create a new text file that contains string representations of components of the given binary files. These representations must be arranged in three columns, each of 20 characters width (the first, second, and third columns contain components of the corresponding file). Each line of the resulting text must start and end with "|" separator (the numeric value 124), all string representations of integers are left-aligned with respect to column boundary.

Text42. Real numbers A, B and an integer N are given. Create a text file that contains table of values of the function $(x)^{1/2}$ at points of the segment $[A, B]$ with the step $H = (B - A)/N$ (that is, at the points A, $A + H$, $A + 2 \cdot H$, …, B). The table consists of two columns, the first column contains arguments x, the second one contains the function values $(x)^{1/2}$. The width of the columns is 10 and 15 characters respectively; the width of the fractional part of arguments and function values is 4 and 8 respectively; all string representations of numbers are right-aligned with respect to column boundary.

Text43. Real numbers A, B and an integer N are given. Create a text file that contains table of values of the functions sin(x) and cos(x) at points of the segment $[A, B]$ with the step $H = (B - A)/N$ (that is, at the points A, $A + H$, $A + 2 \cdot H$, …, B). The table consists of three columns, the first column contains arguments x, the second and the third one contains the function values sin(x) and cos(x) respectively. The width of the first column is 8 characters, the width of the other columns is 12 characters; the width of the fractional part of arguments and function values is 4 and 8 respectively; all string representations of numbers are right-aligned with respect to column boundary.

Text44. A text file is given. Each line of the file represents an integer padded by several leading and trailing blank characters. Output the amount and sum of these integers.

Text45. A text file is given. Each line of the file represents an integer or real number padded by several leading and trailing blank characters. All real numbers have nonzero fractional part. Output the amount and sum of numbers with nonzero fractional part.

Text46. A text file is given. Each line of the file represents several integers and real numbers that are separated by one or more blank characters. All real numbers have nonzero fractional part. Create a binary file of real numbers that contains all given numbers with nonzero fractional part (in the same order).

Text47. A text file is given. Each line of the file represents an integer or real number padded by several leading and trailing blank characters. All real numbers have nonzero fractional part. Output the amount and sum of integers.

Text48. A text file is given. Each line of the file represents several integers and real numbers that are separated by one or more blank characters. All real numbers have nonzero fractional part. Create a binary file of integers that contains all integers (in the same order).

Text49. A text file and a binary file of integers are given. Add a string representation of each integer from the binary file to the end of the corresponding line of the text file. If the amount of integers is less than the amount of text lines then do not change remaining text lines.

Text50. A text file is given. Each line of the text file consists of a text string (of 30 characters length) and a representation of a real number. Create two binary files; the first one is a file of strings that contains text parts from the given text file, the second one is a file of real numbers that contains numbers from the given text file (in the same order).

Text51. A text file is given. The file contains a table of real numbers; the table consists of three columns. The width of each column and the alignment of numbers are arbitrary. Create three binary files of real numbers; each file must contain numbers from the corresponding table column (in the same order).

Text52. A text file is given. The file contains a table of real numbers; the table consists of three columns. A character-*separator* is placed between adjacent columns, before the first column and after the last one. The width of each column and the alignment of numbers are arbitrary; a character that is used as a separator is arbitrary too. Create a binary file of integers that contains the sum of numbers from each row of the table.

16.4. Additional tasks

Text53. Given a text file, create a new binary file of characters that contains all punctuation marks of the text (in the same order).

Text54. A text file is given. Create a new binary file of characters that contains all characters of the given text (without repetitions) including blank character and punctuation marks. The order of characters is determined by their first occurrence in the text.

Text55. A text file is given. Create a new binary file of characters that contains all characters of the given text (without repetitions) including blank character and punctuation marks. The characters must be in ascending order of their numeric values in the character set.

Text56. A text file is given. Create a new binary file of characters that contains all characters of the given text (without repetitions) including blank character and punctuation marks. The characters must be in descending order of their numeric values in the character set.

Text57. A text file is given. Find the amount of occurrences of each Latin small letter and create a new binary file of strings whose components have the following format: "<a letter>–<amount of its occurrences>" (for example, "a–25"). Letters that are absent in the text should not be included in the new file. Strings of the resulting file must be in alphabetic order.

Text58. A text file is given. Find the amount of occurrences of each Latin small letter and create a new binary file of strings whose components have the following format: "<a letter>–<amount of its occurrences>" (for example, "a–25"). Letters that are absent in the text should not be included in the new file. Strings of the new file must be in descending order of the amount of letter occurrences; if some letters have the equal amount of occurrences then the corresponding strings must be in alphabetic order.

Text59. A string S that consists of 10 digits and a text file that contains an English text are given. Encrypt the file using the right cyclic shift of any letter by $S_{K\ mod\ 10\ +\ 1}$ positions of the English alphabet, where K is the line position of the letter, characters S_N of the given string S are numbered beginning with 1, and "mod" denotes the operator of taking the remainder after integer division. For instance, letters that are placed in the line positions 1, 11, 21, … are shifted by S_1 positions of alphabet, letters that are placed in the line positions 2, 12, 22, … are shifted by S_2 positions of alphabet, and so on). Do not change blank characters and punctuation marks.

Text60. A string and a text file are given. The file contains an encrypted English text (the method of encryption is described in Text59), the string is the first decrypted line of the given text. Decrypt the file; if the given information is insufficient for decryption then do not change the given file.

17. Structured data types in procedures and functions

All tasks of this section require to write a procedure or function and then use it for input data processing. Parameters of any *function* are assumed to be an *input* ones; if the kind of a *procedure* parameter is not specified explicitly then this parameter is assumed to be an *input* one too.

17.1. Arrays processing

Any given array should be input in the following order: its size (one integer for one-dimensional arrays, two integers for *matrices*, that is, two-dimensional arrays) and then all its elements in ascending order of indices (the elements of matrices should be input *by rows*).

The size of any one-dimensional array as well as the amount of rows and columns of any matrix are assumed to be in the range 1 to 10 if the task does not specify them explicitly. The order number of the first element of one-dimensional array is assumed to be equal to 1; the order number of the first row (column) of matrix is assumed to be equal to 1 too.

Procedures and functions that perform array processing should use no additional array of the same size.

Param1. Write an integer function MinElem(A, N) that returns the value of the minimal element of an array A of N integers. Using this function, find the minimal elements of three given arrays A, B, C whose sizes are N_A, N_B, N_C respectively.

Param2. Write an integer function MaxNum(A, N) that returns the order number of the maximal element of an array A of N real numbers. Using this function, find the order numbers of the maximal elements of three given arrays A, B, C whose sizes are N_A, N_B, N_C respectively.

Param3. Write a procedure MinmaxNum(A, N, *NMin*, *NMax*) that finds the order numbers *NMin* and *NMax* of the minimal and the maximal element of an array A of N real numbers (integers *NMin* and *NMax* are output parameters). Using this procedure, find the order numbers of the minimal and the maximal elements of three given arrays A, B, C whose sizes are N_A, N_B, N_C respectively.

Param4. Write a procedure Inv(A, N) that changes the order of elements of an array A of N real numbers to inverse one (the array A is an input and output parameter). Using this procedure, change order of elements of arrays A, B, C whose sizes are N_A, N_B, N_C respectively.

Param5. Write a procedure Smooth1(A, N) that performs *smoothing* an array A of N real numbers as follows: each element A_K is replaced with the average of initial values of K first elements of the given array A. The array A is an input and output parameter. Using five calls of this procedure, perform smoothing a given array A of N real numbers five times successively; output array elements after each smoothing.

Param6. Write a procedure Smooth2(A, N) that performs *smoothing* an array A of N real numbers as follows: an element A_1 remains unchanged; elements A_K ($K = 2, \ldots, N$) is replaced with the average of initial values of elements A_{K-1} and A_K. The array A is an input and output parameter. Using five calls of this procedure, perform smoothing a given array A of N real numbers five times successively; output array elements after each smoothing.

Param7. Write a procedure Smooth3(A, N) that performs *smoothing* an array A of N real numbers as follows: each array element is replaced with the average of initial values of this element and its neighbors. The array A is an input and output parameter. Using five calls of this procedure, perform smoothing a given array A of N real numbers five times successively; output array elements after each smoothing.

Param8. Write a procedure RemoveX(A, N, X) that removes all elements equal an integer X from an array A of N integers. The array A and its size N are input and output parameters. Using this procedure, remove elements with given values X_A, X_B, X_C from three given arrays A, B, C of size N_A, N_B, N_C respectively and output the new size and elements of each changed array.

Param9. Write a procedure RemoveForInc(A, N) that removes some elements from an array A of N real numbers so that the values of elements being remained were in ascending order: the first element remains unchanged, the second element must be removed if its value is less than the value of the first one, the third element must be removed if its value is less than the value of the previous element being remained, and so on. For instance, the array of elements 5.5, 2.5, 4.6, 7.2, 5.8, 9.4 must be changed to 5.5, 7.2, 9.4. All procedure parameters are input and output ones. Using this procedure, change three given arrays A, B, C whose sizes are N_A, N_B, N_C respectively and output the new size and elements of each changed array.

Param10. Write a procedure DoubleX(A, N, X) that doubles occurrences of all elements equal an integer X for an array A of N integers. The array A and its size N are input and output parameters. Using this procedure, double occurrences of elements with given values X_A, X_B, X_C for three given arrays A, B, C of size N_A, N_B, N_C respectively and output the new size and elements of each changed array.

Param11. Write a procedure SortArray(A, N) that sorts an array A of N real numbers in ascending order. The array A is an input and output parameter. Using this procedure, sort three given arrays A, B, C of size N_A, N_B, N_C respectively.

Param12. Write a procedure SortIndex(A, N, I) that creates an *index array* I for an array A of N real numbers. The index array contains order numbers of elements of array A so that they correspond to array elements in ascending order of their values (the array A remains unchanged). The index array I is an output parameter. Using this procedure, create index arrays for three given arrays A, B, C of size N_A, N_B, N_C respectively.

Param13. Write a procedure Hill(A, N) that changes order of elements of an array A of N real numbers as follows: the minimal element of the array must be the first one, an element, whose value is the next to minimal value, must be the last one, an element with the next value must be the second one, and so on (as a result, the diagram of values of the array elements will be similar to a *hill*). The array A is an input and output parameter. Using this procedure, change three given arrays A, B, C of size N_A, N_B, N_C respectively.

Param14. Write a procedure Split1(A, N_A, B, N_B, C, N_C) that copies elements of an array A of N_A real numbers to arrays B and C so that the array B contains all elements of the array A with odd order numbers (1, 3, …) and the array C contains all elements of the array A with even order numbers (2, 4, …). The arrays B, C and their sizes N_B, N_C are output parameters. Apply this procedure to a given array A of size N_A and output the size and the elements for each of the resulting arrays B and C.

Param15. Write a procedure Split2(A, N_A, B, N_B, C, N_C) that copies elements of an array A of N_A integers to arrays B and C so that the array B contains all elements whose values are even numbers and the array C contains all elements whose values are odd numbers (in the same order). The arrays B, C and their sizes N_B, N_C are output parameters. Apply this procedure to a given array A of size N_A and output the size and the elements for each of the resulting arrays B and C.

Param16. Write a procedure ArrayToMatrRow(A, K, M, N, B) that copies elements of an array A of K real numbers to an $M \times N$ matrix B (by rows). "Superfluous" array elements must be ignored; if the size of the array is less than the amount of matrix elements then zero value must be assigned to remaining matrix elements. Two-dimensional array B is an output parameter. Having input an array A of size K, integers M, N and using this procedure, create a matrix B and output its elements.

Param17. Write a procedure ArrayToMatrCol(A, K, M, N, B) that copies elements of an array A of K real numbers to an $M \times N$ matrix B (by columns). "Superfluous" array elements must be ignored; if the size of the array is less than the amount of matrix elements then zero value must be assigned to remaining matrix elements. Two-dimensional array B is an output parameter. Having input an array A of size K, integers M, N and using this procedure, create a matrix B and output its elements.

Param18. Write a procedure Chessboard(M, N, A) that creates an $M \times N$ matrix A whose elements are integers 0 and 1, which are arranged in "chessboard" order, and $A_{1,1} = 0$. Two-dimensional array A is an output parameter. Having input integers M, N and using this procedure, create an $M \times N$ matrix A.

Param19. Write a real-valued function Norm1(A, M, N) that computes the *norm* of an $M \times N$ matrix A of real numbers using the formula

$$\text{Norm1}(A, M, N) = \max \{|A_{1,J}| + |A_{2,J}| + \ldots + |A_{M,J}|\},$$

where the maximum is being found over $J = 1, \ldots, N$. Having input an $M \times N$ matrix A, output Norm1(A, K, N), $K = 1, \ldots, M$.

Param20. Write a real-valued function Norm2(A, M, N) that computes the *norm* of an $M \times N$ matrix A of real numbers using the formula

$$\text{Norm2}(A, M, N) = \max \{|A_{I,1}| + |A_{I,2}| + \ldots + |A_{I,N}|\},$$

where the maximum is being found over $I = 1, \ldots, M$. Having input an $M \times N$ matrix A, output Norm2(A, K, N), $K = 1, \ldots, M$.

Param21. Write a real-valued function SumRow(A, M, N, K) that returns the sum of elements in K-th row of an $M \times N$ matrix A of real numbers (if K is out of the range 1 to M then the function returns 0). Output the return value of SumRow(A, M, N, K) for a given $M \times N$ matrix A and three given integers K.

Param22. Write a real-valued function SumCol(A, M, N, K) that returns the sum of elements in K-th column of an $M \times N$ matrix A of real numbers (if K is out of the range 1 to N then the function returns 0). Output the return value of SumCol(A, M, N, K) for a given $M \times N$ matrix A and three given integers K.

Param23. Write a procedure SwapRow(A, M, N, K_1, K_2) that exchanges K_1-th and K_2-th row of an $M \times N$ matrix A of real numbers. The matrix A is an input and output parameter; if K_1 or K_2 are out of the range 1 to M then the matrix remains unchanged. Having input an $M \times N$ matrix A and two integers K_1, K_2 and using this procedure, exchange K_1-th and K_2-th row of the matrix A.

Param24. Write a procedure SwapCol(A, M, N, K_1, K_2) that exchanges K_1-th and K_2-th column of an $M \times N$ matrix A of real numbers. The matrix A is an input and output parameter; if K_1 or K_2 are out of the range 1 to N then the matrix remains unchanged. Having input an $M \times N$ matrix A and two integers K_1, K_2 and using this procedure, exchange K_1-th and K_2-th column of the matrix A.

Param25. Write a procedure Transp(A, M) that *transposes* a real-valued square matrix A of order M (that is, reflects its elements about the main diagonal). The matrix A is an input and output parameter. Using this procedure, transpose the given matrix A of order M.

Param26. Write a procedure RemoveRows(A, M, N, K_1, K_2) that removes rows with the order numbers in the range K_1 to K_2 from an $M \times N$ matrix A of real numbers (integers K_1 and K_2 are assumed to satisfy the double inequality $1 < K_1 \le K_2$). If $K_1 > M$ then the matrix remains unchanged, if $K_2 > M$ then rows with numbers from K_1 to M must be removed. Two-dimensional array A and integers M, N are input and output parameters. Having input an $M \times N$ matrix A and two integers K_1, K_2 and using this procedure, remove rows with the order numbers in the range K_1 to K_2 from the given matrix and output a new order and elements of the resulting matrix.

Param27. Write a procedure RemoveCols(A, M, N, K_1, K_2) that removes columns with the order numbers in the range K_1 to K_2 from an $M \times N$ matrix A of real numbers (integers K_1 and K_2 are assumed to satisfy the double inequality $1 < K_1 \le K_2$). If $K_1 > N$ then the matrix remains unchanged, if $K_2 > N$ then rows with numbers from K_1 to N must be removed. Two-dimensional array A and integers M, N are input and output parameters. Having input an $M \times N$ matrix A and two integers K_1, K_2 and using this procedure, remove columns with the order numbers in the range K_1 to K_2 from the given matrix and output a new order and elements of the resulting matrix.

Param28. Write a procedure RemoveRowCol(A, M, N, K, L) that removes K-th row and L-th column simultaneously from an $M \times N$ matrix A of real numbers (integers K and L are assumed to satisfy the inequalities $M > 1$, $N > 1$). If $K > M$ or $L > N$ then the matrix remains unchanged. Two-dimensional array A and integers M, N are input and output parameters. Having input an $M \times N$ matrix A and two integers K, L, apply this procedure to the given matrix and output a new order and elements of the resulting matrix.

Param29. Write a procedure SortCols(A, M, N) that rearrange columns of an $M \times N$ matrix A of integers in ascending *lexicographic order* (that is, for comparison of two columns their first distinct elements with the equal order numbers must be compared). Two-dimensional array A is an input and output parameter. Using this procedure, sort columns of a given $M \times N$ matrix A.

17.2. Strings processing

Param30. Write an integer function IsIdent(S) that indicates whether a string S is a valid *identifier*, that is, a nonempty string that does not begin with a digit and contains Latin letters, digits, and a character “_” only. If S is a valid identifier then the function returns 0. If S is an empty string or begins with a digit then the function returns −1 or −2 respectively. If S contains invalid characters then the function returns the order number of the first invalid character. Using this function, check five given strings.

Param31. Write a string function FillStr(S, N) that returns a string of length N containing repeating copies of the *template string* S (the last copy of S may be contained partially in the resulting string). Having input an integer N and five template strings and using this function, create five resulting strings of length N.

Param32. Write a procedure UpCaseLat(S) that converts all Latin small letters of a string S to uppercase (others characters of S must remain unchanged). A string S is an input and output parameter. Using this procedure, process five given strings.

Param33. Write a procedure LowCaseLat(S) that converts all Latin capital letters of a string S to lowercase (others characters of S must remain unchanged). A string S is an input and output parameter. Using this procedure, process five given strings.

Param34. Write a procedure TrimLeftC(S, C) that removes all leading characters equal a character C from a string S. A string S is an input and output parameter. Having input a character C and five strings and using this procedure, process the given strings.

Param35. Write a procedure TrimRightC(S, C) that removes all trailing characters equal a character C from a string S. A string S is an input and output parameter. Having input a character C and five strings and using this procedure, process the given strings.

Param36. Write a string function InvStr(S, K, N) that returns an inverted substring of a string S. The substring contains N characters of S (starting at a character with the order number K) in inverse order. If K is greater than the length of S then the function returns an empty string; if the length of S is less than $K + N$ then all characters of S starting at its K-th character must be inverted. Output return values of this function for a given string S and each of three pairs of positive integers (K_1, N_1), (K_2, N_2), (K_3, N_3).

Param37. Write an integer function PosSub(S_0, S, K, N) that searches for the first occurrence of a string S_0 within a substring of a string S (the substring contains N characters of S starting at a character with the order number K). The function returns the order number of the first character of this occurrence within S. If K is greater than the length of S then the function returns 0; if the length of S is less than $K + N$ then all characters of S starting at its K-th character must be analyzed. If the required substring of S does not contain occurrences of S_0 then the function returns 0. Output return values of this function for given strings S_0, S and each of three pairs of positive integers (K_1, N_1), (K_2, N_2), (K_3, N_3).

Param38. Write an integer function PosLast(S_0, S) that searches for the last occurrence of a string S_0 within a string S and returns the order number of the first character of this occurrence. If the string S does not contain occurrences of S_0 then the function returns 0. Output return values of this function for five given pairs of strings (S_0, S).

Param39. Write an integer function PosK(S_0, S, K) that searches for K-th occurrence ($K > 0$) of a string S_0 within a string S and returns the order number of the first character of this occurrence. The string S is assumed to contain no overlapping occurrences of the substring S_0. If the string S does not contain occurrences of S_0 then the function returns 0. Output return values of this function for five given triples (S_0, S, K).

Param40. Write a string function WordK(S, K) that returns K-th word of a string S (a *word* is defined as a character sequence that does not contain blank characters and is bounded by blank characters or the string beginning/end). If the amount of words in the string S is less than K then the function returns an empty string. Having input a string S and three positive integers K_1, K_2, K_3 and using this function, extract words with the given order numbers from the given string.

Param41. Write a procedure SplitStr(*S*, *W*, *N*) that creates an array *W* of all words being contained in a string *S*. The array *W* of strings and its size *N* are output parameters. A *word* is defined as a character sequence that does not contain blank characters and is bounded by blank characters or the string beginning/end; the string *S* is assumed to contain no more than 10 words. Using this function, output the amount *N* of words in the given string *S* and also all these words.

Param42. Write a string function CompressStr(*S*) that *compresses* a string *S* and returns the compressed string. The string compression must be carried out as follows: each substring consisting of 4 or more equal characters *C* is replaced by the string "C{*K*}", where *K* is the amount of characters *C* (the string being compressed is assumed to contain no braces "{}"). For example, the string "bbbccccce" must be compressed to "bbbc{5}e". Using this function, compress five given strings.

Param43. Write a string function DecompressStr(*S*) that restores a string, which was compressed by a function CompressStr (see Param42). An input parameter *S* is a compressed string; the function returns the decompressed value of the string *S*. Using this function, restore five given compressed strings.

Param44. Write a string function DecToBin(*N*) that returns a string containing the *binary representation* of a nonnegative integer *N*. The string consists of characters "0", "1" and does not contain leading zeros (except for the representation of zero number). Using this function, output binary representations of five given integers.

Param45. Write a string function DecToHex(*N*) that returns a string containing the *hexadecimal representation* of a nonnegative integer *N*. The string consists of characters "0"–"9", "A"–"F" and does not contain leading zeros (except for the representation of zero number). Using this function, output hexadecimal representations of five given integers.

Param46. Write an integer function BinToDec(*S*) that returns a nonnegative integer whose binary representation is contained in a string parameter *S*. The parameter *S* consists of characters "0", "1" and does not contain leading zeros (except for the representation of zero number). Using this function, output five integers whose binary representations are given.

Param47. Write an integer function HexToDec(*S*) that returns a nonnegative integer whose hexadecimal representation is contained in a string parameter *S*. The parameter *S* consists of characters "0"–"9", "A"–"F" and does not contain leading zeros (except for the representation of zero number). Using this function, output five integers whose hexadecimal representations are given.

17.3. Files processing

Param48. Write an integer function IntFileSize(*S*) that returns the amount of components in a binary file of integers called *S*. If the required file does not exist then the function returns −1. Using this function, output the amount of components for three binary files of integers with given names.

Param49. Write an integer function LineCount(*S*) that returns the amount of lines in a text file called *S*. If the required file does not exist then the function returns −1. Using this function, output the amount of lines for three text files with given names.

Param50. Write a procedure InvIntFile(*S*) that changes the order of components of a binary file of integers called *S* to inverse one. If the required file does not exist then the procedure performs no actions. Using this procedure, process three binary files of integers with given names.

Param51. Write a procedure AddLineNumbers(*S*, *N*, *K*, *L*) that adds the number of each line of a text file called *S* to the beginning of this line; the first line receives the number *N*, the second line receives the number $N + 1$, and so on. Any number is right-aligned within *K* first character positions of a line and is separated from the following text by *L* blank characters ($K > 0$, $L > 0$). If a line is empty then its number should not contain trailing blank characters. Apply this procedure to a given text file using given values of parameters *N*, *K*, *L*.

Param52. Write a procedure RemoveLineNumbers(*S*) that removes order numbers (with leading and trailing blank characters) from the beginning of each line of a text file called *S* (the format of order numbers is described in Param51). If file lines do not contain order numbers then the procedure performs no actions. Apply this procedure to a given text file.

Param53. Write a procedure SplitIntFile(S_0, K, S_1, S_2) that copies first *K* (≥ 0) components of an existing file called S_0 to a new file called S_1 and other components of this file to a new file called S_2. All files are assumed to be binary files of integers; one of the resulting files may be empty. Apply this procedure to a given file called S_0 using given values of K, S_1, S_2.

Param54. Write a procedure SplitText(S_0, K, S_1, S_2) that copies first *K* (≥ 0) lines of an existing text file called S_0 to a new text file called S_1 and other lines of this file to a new text file called S_2 (one of the resulting files may be empty). Apply this procedure to a given file called S_0 using given values of K, S_1 S_2.

Param55. Write a procedure StringFileToText(*S*) that converts a binary file of strings called *S* to a new text file with the same name. Using this procedure, convert two given files of strings with the names S_1, S_2 to text files.

Param56. Write a procedure TextToStringFile(*S*) that converts a text file called *S* to a new binary file of strings with the same name. Using this procedure, convert two given text files with the names S_1, S_2 to binary files of strings.

Param57. Write a procedure EncodeText(*S*, *K*) that encrypts the contents of a text file called *S* using the right cyclic shift of any Latin letter by *K* positions of the English alphabet ($0 < K < 10$). For instance, if $K = 3$ then the letter “A” is encoded by the letter “D”, “a” is encoded by “d”, “B” is encoded by “E”, “z” is encoded by “c”, and so on. Blank characters and punctuation marks should not be changed. Having input an integer *K* and using this procedure, encrypt a text file with the given name.

Param58. Write a procedure DecodeText(*S*, *K*) that decrypts the contents of a text file called *S* provided that this file was encrypted by the method described in Param57 with using an integer parameter *K*. Having input an integer *K* and using this procedure, decrypt a text file with the given name.

17.4. Records processing

Fields of each *date* record should be input/output in the following order: *Day*, *Month*, *Year*. Fields of each *2D-point* record should be input/output in the following order: *X* (an *x*-coordinate), *Y* (an *y*-coordinate).

Param59. Define a new type called TDate that is a record with three integer fields: *Day* (a day number), *Month* (a month number), *Year* (a year number). Also write a logical function LeapYear(*D*) with a parameter *D* of TDate type. The function returns True if a year of date *D* is a leap year, and False otherwise. Output the return values of this function for five given dates (all dates are assumed to be correct). Note that a year is a *leap year* if it is divisible by 4 except for years that are divisible by 100 and are not divisible by 400.

Param60. Using the TDate type and the LeapYear function (see Param59), write an integer function DaysInMonth(*D*) with a parameter *D* of TDate type. The function returns the amount of days for the month of date *D*. Output the return values of this function for five given dates (all dates are assumed to be correct).

Param61. Using the TDate type and the DaysInMonth function (see Param59 and Param60), write an integer function CheckDate(*D*) with a parameter *D* of TDate type. If the date *D* is a correct date then the function returns 0; if the date *D* contains an invalid month number or invalid day number for the correct month then the function returns 1 or 2 respectively. Output the return values of this function for five given dates.

Param62. Using the TDate type and the DaysInMonth and CheckDate functions (see Param59−Param61), write a procedure PrevDate(*D*) that changes a correct date *D* (of TDate type) to the previous one; if *D* contains an invalid date then it remains unchanged. The parameter *D* is an input and output parameter. Apply this procedure to five given dates.

Param63. Using the TDate type and the DaysInMonth and CheckDate functions (see Param59−Param61), write a procedure NextDate(*D*) that changes a correct date *D* (of TDate type) to the next one; if *D* contains an invalid date then it remains unchanged. The parameter *D* is an input and output parameter. Apply this procedure to five given dates.

Param64. Define a new type called TPoint that is a record with two real-valued fields: *X* (an *x*-coordinate), *Y* (an *y*-coordinate). Also write a real-valued function Leng(A, B) that returns the length of a segment AB (A and B are input parameters of TPoint type):

$$|AB| = ((A.X - B.X)^2 + (A.Y - B.Y)^2)^{1/2}.$$

Using this function, output lengths of segments AB, AC, AD provided that the coordinates of points A, B, C, D are given.

Param65. Using the TPoint type and the Leng function (see Param64), define a new type called TTriangle that is a record with three fields A, B, C (triangle vertices) of TPoint type, and write a real-valued function Perim(T) that returns the perimeter of a triangle T (T is an input parameter of TTriangle type). Using this function, find perimeters of triangles ABC, ABD, ACD provided that the coordinates of points A, B, C, D are given.

Param66. Using the TPoint and TTriangle types and the Leng and Perim functions (see Param64 and Param65), write a real-valued function Area(T) that returns the area of a triangle T (T is an input parameter of TTriangle type):

$$S_{ABC} = (p\cdot(p - |AB|)\cdot(p - |AC|)\cdot(p - |BC|))^{1/2},$$

where p is the *half-perimeter*. Using this function, find the areas of triangles ABC, ABD, ACD provided that the coordinates of points A, B, C, D are given.

Param67. Using the TPoint and TTriangle types and the Leng and Area functions (see Param64–Param66), write a real-valued function Dist(P, A, B) that returns the distance $D(P, AB)$ between a point P and a line AB:

$$D(P, AB) = 2\cdot S_{PAB}/|AB|,$$

where S_{PAB} is the area of the triangle PAB (parameters P, A, B are input parameters of TPoint type). Using this function, find the distance between a point P and each of lines AB, AC, BC provided that the coordinates of points P, A, B, C are given.

Param68. Using the TPoint and TTriangle types and the Dist function (see Param64, Param65, Param67), write a procedure Alts(T, h_1, h_2, h_3) that evaluates the altitudes h_1, h_2, h_3 drawn from the vertices $T.A$, $T.B$, $T.C$ of a triangle T (T is an input parameter of TTriangle type, h_1, h_2, h_3 are output real-valued parameters). Using this procedure, evaluate the altitudes of each of triangles ABC, ABD, ACD provided that the coordinates of points A, B, C, D are given.

Param69. Using the TPoint type and the Leng function (see Param64), write a real-valued function PerimN(P, N) that returns the perimeter of a polygon with N (> 2) vertices. The polygon vertices have the TPoint type; an array P contains all vertices in order of walk. Using this function, find the perimeters of three polygons provided that the amount of vertices and the coordinates of all vertices are given for each polygon.

Param70. Using the TPoint and TTriangle types and the Area function (see Param64–Param66), write a real-valued function AreaN(P, N) that returns the area of a convex polygon with N (> 2) vertices. The polygon vertices have the TPoint type; an array P contains all vertices in order of walk. Using this function, find the areas of three polygons provided that the amount of vertices and the coordinates of all vertices are given for each polygon.

18. Recursion

18.1. Simple algorithms

It should be noted that all tasks of this subsection can be solved by means of simple iterative algorithms without using of recursion; moreover, in some cases using of recursion leads to inefficient algorithms (see Recur4 and Recur6). Nevertheless, tasks of such a kind allow to receive easily an initial experience in developing of recursive algorithms.

Recur1. Write a recursive real-valued function Fact(N) that returns the value of N-*factorial*:

$$N! = 1\cdot 2\cdot \ldots \cdot N,$$

where N (> 0) is an integer parameter. Using this function, output factorials of five given integers.

Recur2. Write a recursive real-valued function Fact2(N) that returns the value of *double factorial* of N:

$$N!! = N\cdot(N-2)\cdot(N-4)\cdot\ldots,$$

where N (> 0) is an integer parameter; the last factor of the product equals 2 if N is an even number, and 1 otherwise. Using this function, output double factorials of five given integers.

Recur3. Write a recursive real-valued function PowerN(X, N) that returns the power X^N ($X \neq 0$ is a real number, N is an integer) calculated as follows:

$$X^0 = 1,$$
$$X^N = (X^{N\ div\ 2})^2 \text{ if } N \text{ is a positive even number,}$$
$$X^N = X\cdot X^{N-1} \text{ if } N \text{ is a positive odd number,}$$
$$X^N = 1/X^{-N} \text{ if } N < 0,$$

where "div" denotes the operator of *integer division*. Using this function, output powers X^N for a given real number X and five given integers N.

Recur4. Write a recursive integer function Fib1(N) that returns the Fibonacci number F_N (N is a positive integer). The *Fibonacci numbers F_K* are defined as:

$$F_1 = F_2 = 1, \qquad F_K = F_{K-2} + F_{K-1}, \quad K = 3, 4, \ldots .$$

Using the function Fib1, find the Fibonacci numbers F_N for five given integers N; output the value of each Fibonacci number and also the amount of the recursive function calls, which are required for its calculation.

Recur5. Write a recursive integer function Fib2(N) that returns the Fibonacci number F_N (N is a positive integer). The *Fibonacci numbers F_K* are defined as:

$$F_1 = F_2 = 1, \qquad F_K = F_{K-2} + F_{K-1}, \quad K = 3, 4, \ldots .$$

The integer N is assumed to be not greater than 20. Decrease the amount of recursive calls of the function Fib2 (in comparison with the Fib1 function from the task Recur4) by means of using an additional array of integers that should store the Fibonacci numbers *having been calculated*. Using the Fib2 function, output the Fibonacci numbers F_N for five given integers N.

Recur6. Write a recursive integer function Combin1(N, K) that returns $C(N, K)$ (the *number of combinations* of N objects taken K at a time) using the following recursive relations (N and K are integers, $N > 0$, $0 \le K \le N$):

$$C(N, 0) = \mathrm{C}(N, N) = 1,$$
$$C(N, K) = C(N-1, K) + C(N-1, K-1) \quad \text{if } 0 < K < N.$$

Using the function Combin1, find the numbers $C(N, K)$ for a given integer N and five given integers K; output the value of each number and also the amount of the recursive function calls, which are required for its calculation.

Recur7. Write a recursive integer function Combin2(N, K) that returns $C(N, K)$ (the *number of combinations* of N objects taken K at a time) using the following recursive relations (N and K are integers, $N > 0$, $0 \le K \le N$):

$$C(N, 0) = \mathrm{C}(N, N) = 1,$$
$$C(N, K) = C(N-1, K) + C(N-1, K-1) \quad \text{if } 0 < K < N.$$

The integer N is assumed to be not greater than 20. Decrease the amount of recursive calls of the function Combin2 (in comparison with the Combin1 function from the task Recur6) by means of using an additional two-dimensional array of integers that should store the numbers $C(N, K)$ *having been calculated*. Using the Combin2 function, output the numbers $C(N, K)$ for a given integer N and five given integers K.

Recur8. Write a recursive real-valued function RootK(X, K, N) that returns an approximate value of a K-th root of X using the following formulas:

$$Y_0 = 1, \qquad Y_{N+1} = Y_N - (Y_N - X/(Y_N)^{K-1})/K,$$

where X (> 0) is a real number, K (> 1), N (> 0) are integers, Y_N denotes RootK(X, K, N) for a fixed values of X and K. Using this function, output approximate values of a K-th root of X for a given X, K and six integers N.

Recur9. Write a recursive integer function GCD(A, B) that returns the *greatest common divisor* of two positive integers A and B. Use the *Euclidean algorithm*:

$$\mathrm{GCD}(A, B) = \mathrm{GCD}(B, A \bmod B), \quad \text{if } B \ne 0; \qquad \mathrm{GCD}(A, 0) = A,$$

where “mod” denotes the operator of taking the remainder after integer division. Using this function, find the greatest common divisor for each of pairs (A, B), (A, C), (A, D) provided that integers A, B, C, D are given.

Recur10. Write a recursive integer function DigitSum(K) that returns the sum of digits of an integer K (the loop statements should not be used). Using this function, output the sum of digits for each of five given integers.

Recur11. Write a recursive integer function MaxElem(A, N) that returns the maximal element of an array A of N integers ($1 \le N \le 10$; the loop statements should not be used). Using this function, output the maximal elements of three given arrays A, B, C whose sizes are N_A, N_B, N_C respectively.

Recur12. Write a recursive integer function DigitCount(*S*) that returns the amount of digit characters in a string *S* (the loop statements should not be used). Using this function, output the amount of digit characters for each of five given strings.

Recur13. Write a recursive logical function Palindrome(*S*) that returns True if a string *S* is a *palindrome* (i. e., it is read equally both from left to right and from right to left), and False otherwise; the loop statements should not be used. Output return values of this function for five given string parameters.

18.2. Parsing of expressions

Input strings are assumed to contain no blank characters in all tasks of this subsection.

The loop statements should not be used for solving these tasks.

Recur14. Given a string *S* that represents a correct expression of integer type, output the value of this expression. The expression is defined as follows:

```
<expression> ::= <digit> | <expression> + <digit> |
                 <expression> − <digit>
```

Recur15. Given a string *S* that represents a correct expression of integer type, output the value of this expression. The expression is defined as follows:

```
<expression> ::= <term> | <expression> + <term> |
                 <expression> − <term>
<term>       ::= <digit> | <term> * <digit>
```

Recur16. Given a string *S* that represents a correct expression of integer type, output the value of this expression. The expression is defined as follows:

```
<expression> ::= <term> | <expression> + <term> |
                 <expression> − <term>
<term>       ::= <element> | <term> * <element>
<element>    ::= <digit> | (<expression>)
```

Recur17. Given a string *S* that represents a correct expression of integer type, output the value of this expression. The expression is defined as follows:

```
<expression> ::= <digit> |
                 (<expression><operator><expression>)
<operator>   ::= + | − | *
```

Recur18. A nonempty string *S* that represents an expression of integer type is given (see the expression definition in Recur17). Output True if the given expression is a correct one, otherwise output False.

Recur19. A nonempty string *S* that represents an expression of integer type is given (see the expression definition in Recur17). Output 0 if the given expression is a correct one, otherwise output the order number of its first character that is invalid, superfluous or missing.

Recur20. Given a string *S* that represents a correct expression of integer type, output the value of this expression. The expression is defined as follows (functions M and m return their maximal and minimal argument respectively):

<expression> : := <digit> | M(<expression> , <expression>) |
m(<expression> , <expression>)

Recur21. Given a string *S* that represents a correct expression of logical type, output the value of this expression. The expression is defined as follows ("T" means True, "F" means False):

<expression> : := T | F | And(<expression> , <expression>) |
Or(<expression> , <expression>)

Recur22. Given a string *S* that represents a correct expression of integer type, output the value of this expression. The expression is defined as follows (functions M and m return their maximal and minimal argument respectively):

<expression> : := <digit> | M(<arguments>) | m(<arguments>)
<arguments> : := <expression> | <expression> , <arguments>

Recur23. Given a string *S* that represents a correct expression of logical type, output the value of this expression. The expression is defined as follows ("T" means True, "F" means False):

<expression> : := T | F | And(<arguments>) | Or(<arguments>)
<arguments> : := <expression> | <expression> , <arguments>

Recur24. Given a string *S* that represents a correct expression of logical type, output the value of this expression. The expression is defined as follows ("T" means True, "F" means False):

<expression> : := T | F | And(<arguments>) |
Or(<arguments>) | Not(<expression>)
<arguments> : := <expression> | <expression> , <arguments>

18.3. Backtracking

Recur25. A tree of depth *N* is given. Each internal node of the tree has *K* (< 10) children that are numbered from 1 (the most left child) to *K* (the most right child). The number of the tree root is 0. Create a text file (with a given name) whose lines contain paths from the root to all tree leaves. Paths must be ordered from the *most left path* ("011...1") to the *most right path* (for instance, "033...3" provided that K = 3); the last nodes of path must be changed faster than the first ones.

Recur26. A tree of depth *N* is given. Each internal node of the tree has *K* (< 10) children that are numbered from 1 (the most left child) to *K* (the most right child). The number of the tree root is 0. Create a text file (with a given name) whose lines contain paths from the root to all tree leaves; each path must satisfy the following additional condition: adjacent nodes of the path have different numbers. The order of paths must be the same as in Recur25.

Recur27. A tree of depth *N* is given (*N* is an even number). Each internal node of the tree has two children; the left child “A” with the weight 1 and the right child “B” with the weight −1. The tree root “C” has the weight 0. Create a text file (with a given name) whose lines contain paths from the root to all tree leaves; each path must satisfy the following additional condition: the total weight of all path nodes is equal to 0. The order of paths must be the same as in Recur25.

Recur28. A tree of depth *N* is given; see the description of tree nodes in Recur27. Create a text file (with a given name) whose lines contain paths from the root to all tree leaves; each path must satisfy the following additional condition: the total weight of any initial part of the path nodes is nonnegative. The order of paths must be the same as in Recur25.

Recur29. A tree of depth *N* is given. Each internal node of the tree has three children; the left child ”A” with the weight 1, the middle child ”B” with the weight 0, the right child ”C” with the weight −1. The tree root ”D” has the weight 0. Create a text file (with a given name) whose lines contain paths from the root to all tree leaves; each path must satisfy two additional conditions: the total weight of any initial part of the path nodes is nonnegative, and the total weight of all path nodes equals 0. The order of paths must be the same as in Recur25.

Recur30. A tree of depth *N* is given; see the description of tree nodes in Recur29. Create a text file (with a given name) whose lines contain paths from the root to all tree leaves; each path must satisfy two additional conditions: adjacent nodes of the path have different letters, and the total weight of all path nodes equals 0. The order of paths must be the same as in Recur25.

19. Dynamic data structures (based on pointers)

All numbers mentioned in tasks of this group are of integer type. All pointers are of PNode type; they point to records of TNode type. In the tasks of this group the Data, Next, and Prev fields of the TNode record are used, therefore one can assume that the PNode and TNode types are defined as follows:

[Pascal]

```
type
  PNode = ^TNode;
  TNode = record
    Data: integer;
    Next: PNode;
    Prev: PNode;
  end;
```

[C++]
```
struct TNode
{
  int Data;
  TNode* Next;
  TNode* Prev;
};
typedef TNode* PNode;
```
In the introductory tasks and in the tasks devoted to *stacks* and *queues* only the Data and Next fields of the TNode record are used. In the tasks devoted to *lists* all fields (Data, Next, Prev) of the TNode record are used.

Words “pointer” (to some data) and “address” (of some data) are used as synonyms since variables of pointer type are intended for storing *addresses*.

The order number of the first node of a list is assumed to be equal to 1.

In C++ programs the delete p operator or DeleteNode(p) function call should be used to free the memory that a pointer p (of the PNode type) addresses.

19.1. Nodes and chains of nodes

Dynamic1. An address P_1 of a record of TNode type is given. The record consists of the *Data* field (of integer type) and the *Next* field (of PNode type that refers to a variable of TNode type). The given record is linked by its *Next* field with the next record of the same type. Output the value of the *Data* field for each record and the address P_2 of the record that follows the given one.

Dynamic2. An address P_1 of a record of TNode type is given. The record is linked by its *Next* field with the next record of the same type, that record is linked with the next one, and so on, until the last record whose *Next* field equals nil (as a result, we obtain a *chain* of linked records). Output the value of the *Data* field for each chain component, the chain *length* (that is, the amount of its components) and the address of the last chain component.

19.2. Stack

In these tasks a *stack* structure is implemented by a chain of linked components (*nodes*) of TNode type. The Next field of the last node equals nil. The first node is said to be a *top* of the stack. The pointer to the top of the stack provides access to the stack data (if the stack is empty then this pointer equals nil). The value of the Data field of a stack component is considered as the *value of this component*.

Dynamic3. An integer *D* and a pointer P_1 to the top of a nonempty stack are given. Push a component with the value *D* onto the stack and output the address P_2 of a new top of the stack.

Dynamic4. An integer *N* (> 0) and a sequence of *N* integers are given. Create a stack that contains *N* components with the given values (a component with the last value must be the top of the stack) and output a pointer to the top of the stack.

Dynamic5. A pointer P_1 to the top of a nonempty stack is given. Pop the top component off the stack and output its value D and the address P_2 of a new top of the stack. If the stack will be empty after popping the component then P_2 must be equal to nil. After popping the component release the memory allocated for this component.

Dynamic6. A pointer P_1 to the top of a stack is given; the stack contains at least ten components. Pop the first nine components off the stack and output their values and the address P_2 of a new top of the stack. After popping components release the memory allocated for these components.

Dynamic7. A pointer P_1 to the top of a stack is given (if the stack is empty then P_1 equals nil). Pop all components off the stack and output their values. Also output the amount of popped components (if the stack is empty then output 0). After popping components release the memory allocated for these components.

Dynamic8. Two pointers P_1 and P_2 that refer to the tops of two nonempty stacks are given. Move all components from the first stack into the second one (as a result, all components of the first stack will be contained within the second stack in inverse order). Output the address of a new top of the second stack. Do not use operations of allocating and freeing memory.

Dynamic9. Two pointers P_1 and P_2 that refer to the tops of two nonempty stacks are given. Move components from the first stack into the second one until the value of the top component of the first stack is equal to an even number (as a result, all components having been moved will be contained within the second stack in inverse order). If the first stack contains no components with even values then move all its components. Output the address of a new top for each stack (if the first stack will be empty then output nil for this stack). Do not use operations of allocating and freeing memory.

Dynamic10. A pointer P_1 to the top of a nonempty stack is given. Create two new stacks by moving the given stack components whose values are even (odd) numbers into the first (second) new stack respectively. As a result, all components having been moved will be contained within each new stack in inverse order; one of the new stacks may be empty. Output the address of the top for each new stack (if one of the new stacks will be empty then output nil for this stack). Do not use operations of allocating and freeing memory.

Dynamic11. A pointer P_1 to the top of a stack is given (if the stack is empty then P_1 equals nil). Also an integer N (> 0) and a sequence of N integers are given. Define a new type called TStack that is a record with one field, *Top*, of PNode type (the field refers to the top of a stack). Also write a procedure Push(S, D) that pushes a new component with the value D onto a stack S (a record S of TStack type is an input and output parameter, an integer D is an input parameter). Using this procedure, push all elements of the given sequence onto the given stack (the last number must be the value of the top component). Output the address of a new top of the stack.

Dynamic12. A pointer P_1 to the top of a stack is given; the stack contains at least five components. Using the TStack type (see Dynamic11), write an integer function Pop(*S*) that pops the top component off a stack *S*, releases memory allocated for this component and returns its value (a record *S* of TStack type is an input and output parameter). Using this function, pop five components off the given stack and output their values. Also output a pointer that refers to a new top of the stack (if the stack will be empty then this pointer must be equal to nil).

Dynamic13. A pointer P_1 to the top of a stack is given. Using the TStack type (see Dynamic11), write two functions: a logical function StackIsEmpty(*S*) that returns True if a stack *S* is empty, and False otherwise, and an integer function Peek(*S*) that returns the value of the top component of the stack *S*. A record *S* of TStack type is an input parameter for each function. Using these functions and the Pop function from the task Dynamic12, pop five components (or all stack components if their amount is less than five) off the given stack and output their values. Also output the return value of the StackIsEmpty function for the resulting stack. At last, in the case of the nonempty resulting stack, output the value and the address of its top component.

19.3. Queue

In these tasks a *queue* structure is implemented by a chain of linked components (*nodes*) of TNode type. The Next field of the last node equals nil. The first node is said to be a *head* of the queue; the head is located at the *front* of the queue. The last node is said to be a *tail* of the queue; the tail is located at the *end* of the queue. It is convenient to store not only a pointer to the queue head but also a pointer to the queue tail because it accelerates adding a new component to the end of a queue. If a queue is empty then these pointers equal nil. The value of the Data field of a queue component is considered as the *value of this component*.

Dynamic14. A sequence of 10 integers is given. Create a queue that contains components with the given values (a component with the first value must be the head of the queue, a component with the last value must be the tail of the queue) and output pointers P_1 and P_2 to the head and tail of the queue respectively.

Dynamic15. A sequence of 10 integers is given. Create two queues; the first one must contain the given integers with odd order numbers (1, 3, …, 9), the second one must contain the given integers with even order numbers (2, 4, …, 10). Output pointers to the head and tail of the first queue and then output pointers to the head and tail of the second one.

Dynamic16. A sequence of 10 integers is given. Create two queues; the first one must contain the given integers with odd values (in the same order), the second one must contain the given integers with even values (in the same order). Output pointers to the head and tail of the first queue and then output pointers to the head and tail of the second one (if one of the queues will be empty then output nil twice for this queue).

Dynamic17. An integer D and pointers P_1 and P_2 to the head and tail of a queue are given (if the queue is empty then the pointers equal nil). Add a component with the value D to the end of the queue and output the new addresses of the head and tail of the queue.

Dynamic18. An integer D and pointers P_1 and P_2 to the head and tail of a queue are given; the queue contains at least two components. Add a component with the value D to the end of the queue and remove the first component from the front of the queue. Output the value of the component being removed and also output the new addresses of the head and tail of the queue. After removing the component release the memory allocated for this component.

Dynamic19. An integer N (> 0) and pointers P_1 and P_2 to the head and tail of a nonempty queue are given. Remove N initial components from the queue and output their values (if the queue contains less than N components then remove all its components). Also output the new addresses of the head and tail of the queue (if the resulting queue will be empty then output nil twice). After removing components release the memory allocated for them.

Dynamic20. Pointers P_1 and P_2 to the head and tail of a nonempty queue are given. Remove components from the front of the queue until the value of the head of the queue is equal to an even number; output values of all components being removed (if the queue contains no components with even values then remove all its components). Also output the new addresses of the head and tail of the queue (if the resulting queue will be empty then output nil twice). After removing components release the memory allocated for them.

Dynamic21. Two queues are given; pointers P_1 and P_2 refer to the head and tail of the first one, pointers P_3 and P_4 refer to the head and tail of the second one (if some queue is empty then the corresponding pointers equal nil). Move all components from the first queue (starting with its first component) to the end of the second one. Output the new addresses of the head and tail of the second queue. Do not use operations of allocating and freeing memory.

Dynamic22. An integer N (> 0) and two nonempty queues are given; pointers P_1 and P_2 refer to the head and tail of the first one, pointers P_3 and P_4 refer to the head and tail of the second one. Move N initial components of the first queue to the end of the second one (if the first queue contains less than N components then move all its components). Output the new addresses of the head and tail of the first queue and then output the new addresses of the head and tail of the second one (if the first queue will be empty then output nil twice for this queue). Do not use operations of allocating and freeing memory.

Dynamic23. Two nonempty queues are given; pointers P_1 and P_2 refer to the head and tail of the first one, pointers P_3 and P_4 refer to the head and tail of the second one. Move initial components of the first queue to the end of the second one until the value of the head of the first queue is equal to an even number (if the first queue contains no components with even values then move all its components). Output the new addresses of the head and tail of the first queue and then output the new addresses of the head and tail of the second one (if the first queue will be empty then output nil twice for this queue). Do not use operations of allocating and freeing memory.

Dynamic24. Two nonempty queues are given; pointers P_1 and P_2 refer to the head and tail of the first one, pointers P_3 and P_4 refer to the head and tail of the second one. The queues contain the equal amount of components. Combine the given queues into a new one; the resulting queue must contain alternating components of the given queues starting with the head of the first one. Output pointers to the head and tail of the resulting queue. Do not use operations of allocating and freeing memory.

Dynamic25. Two nonempty queues are given; pointers P_1 and P_2 refer to the head and tail of the first one, pointers P_3 and P_4 refer to the head and tail of the second one. The values of components of each given queue are in ascending order. Combine the given queues into a new one; the values of components of the resulting queue must be in ascending order too. Output pointers to the head and tail of the resulting queue. Do not use operations of allocating and freeing memory; do not change the *Data* fields.

Dynamic26. Pointers P_1 and P_2 to the head and tail of a queue are given (if the queue is empty then the pointers equal nil). Also an integer N (> 0) and a sequence of N integers are given. Define a new type called TQueue that is a record with two fields, *Head* and *Tail*, of PNode type (the fields refer to the head and tail of a queue respectively). Also write a procedure Enqueue(Q, D) that adds a new component with the value D to the end of a queue Q (a record Q of TQueue type is an input and output parameter, an integer D is an input parameter). Using this procedure, add all elements of the given sequence to the end of the given queue. Output the new addresses of the head and tail of the queue.

Dynamic27. Pointers P_1 and P_2 to the head and tail of a queue are given; the queue contains at least five components. Using the TQueue type (see Dynamic26), write an integer function Dequeue(Q) that removes the first component from the front of a queue Q, releases memory allocated for this component and returns its value (a record Q of TQueue type is an input and output parameter). Using this function, remove five initial components from the front of the given queue and output their values. Also output the new addresses of the head and tail of the queue (if the queue will be empty then output nil twice).

Dynamic28. Pointers P_1 and P_2 to the head and tail of a queue are given. Using the TQueue type (see Dynamic26), write a logical function QueueIsEmpty(Q) that returns True if a queue Q is empty, and False otherwise (a record Q of TQueue type is an input parameter). Using this function and also the Dequeue function from the task Dynamic27, remove five initial components (or all queue components if their amount is less than five) from the front of the given queue and output their values. Also output the return value of the QueueIsEmpty function for the resulting queue and the new addresses of the head and tail of this queue (if the queue will be empty then output nil twice).

19.4. Doubly linked list

In these tasks a *doubly linked list* structure is implemented by a chain of components (*nodes*) of TNode type; these nodes are linked with both the next node and the previous one. The Next field of the last node and the Prev field of the first node are equal to nil. Though storing of address of some list node is sufficient to provide access to any list node, it is convenient to store *three* pointers (to the *first*, *last*, and *current* list node) because they accelerate list operations. If a list is empty then these pointers equal nil. The value of the Data field of a list component is considered as the *value of this component*.

Dynamic29. An address P_2 of a record of TNode type is given. The record consists of the following fields: *Data* (of integer type), *Prev*, *Next* (each of PNode type that refers to a variable of TNode type). The given record is linked by its *Prev* and *Next* field with the previous and next record of the same type respectively. Output the values of the *Data* field for the previous and next record, and also output the addresses P_1 and P_3 of these records.

Dynamic30. A pointer P_1 to the beginning of a chain of records is given; the records have TNode type and are linked by their *Next* fields. Using the *Prev* field of the TNode record, transform the given (*singly linked*) chain into the *doubly linked* chain whose components are linked not only with the next ones (by the *Next* field) but also with the previous ones (by the *Prev* field). The *Prev* field of the first chain component must be equal to nil. Output the address of the last component of the resulting chain.

Dynamic31. A pointer P_0 to one of the components of a nonempty doubly linked list is given. Output the amount N of the list components and also pointers P_1 and P_2 to the first and last component respectively.

Dynamic32. Two integers D_1, D_2 and a pointer P_0 to one of the components of a nonempty doubly linked list are given. Insert a new component with the value D_1 at the beginning of the list; insert a new component with the value D_2 at the end of the list. Output the addresses of the first and last list component.

Dynamic33. An integer D and a pointer P_0 to one of the components of a nonempty doubly linked list are given. Insert a new component with the value D before the given list component. Output the address of the component being inserted.

Dynamic34. An integer D and a pointer P_0 to one of the components of a nonempty doubly linked list are given. Insert a new component with the value D after the given list component. Output the address of the component being inserted.

Dynamic35. Pointers P_1 and P_2 to the first and last component of a doubly linked list are given. The list contains at least two components. Double occurrences of the first and last list component (new components must be inserted before the existing ones with the same value). Output the address of the first component of the resulting list.

Dynamic36. Pointers P_1 and P_2 to the first and last component of a doubly linked list are given. The list contains at least two components. Double occurrences of the first and last list component (new components must be inserted after the existing ones with the same value). Output the address of the last component of the resulting list.

Dynamic37. A pointer P_1 to the first component of a nonempty doubly linked list is given. Double occurrences of all list components with odd order numbers (new components must be inserted before the existing ones with the same value). Output the address of the first component of the resulting list.

Dynamic38. A pointer P_1 to the first component of a nonempty doubly linked list is given. Double occurrences of all list components with odd order numbers (new components must be inserted after the existing ones with the same value). Output the address of the last component of the resulting list.

Dynamic39. A pointer P_1 to the first component of a nonempty doubly linked list is given. Double occurrences of all list components with odd values (new components must be inserted before the existing ones with the same value). Output the address of the first component of the resulting list.

Dynamic40. A pointer P_1 to the first component of a nonempty doubly linked list is given. Double occurrences of all list components with odd values (new components must be inserted after the existing ones with the same value). Output the address of the last component of the resulting list.

Dynamic41. A pointer P_0 to one of the components of a nonempty doubly linked list is given. Remove this component from the list and output the addresses of its previous and next component in the list (one or both these components may be absent; output nil for each absent component). After removing the component release the memory allocated for this component.

Dynamic42. A pointer P_1 to the first component of a doubly linked list is given. The list contains at least two components. Remove all components with odd order numbers from the list and output the address of the first component of the resulting list. After removing components release the memory allocated for them.

Dynamic43. A pointer P_1 to the first component of a nonempty doubly linked list is given. Remove all components with odd values from the list and output the address of the first component of the resulting list (if this list will be empty then output nil). After removing components release the memory allocated for them.

Dynamic44. A pointer P_0 to one of the components of a nonempty doubly linked list is given. Move this component to the end of the list and output the addresses of the first and last component of the resulting list. Do not use operations of allocating and freeing memory; do not change the *Data* fields.

Dynamic45. A pointer P_0 to one of the components of a nonempty doubly linked list is given. Move this component to the beginning of the list and output the addresses of the first and last component of the resulting list. Do not use operations of allocating and freeing memory; do not change the *Data* fields.

Dynamic46. An integer K (> 0) and a pointer P_0 to one of the components of a nonempty doubly linked list are given. Move this component by K positions forward in the list (if the list contains less than K components after the given component then move it to the end of the list). Output the addresses of the first and last component of the resulting list. Do not use operations of allocating and freeing memory; do not change the *Data* fields.

Dynamic47. An integer K (> 0) and a pointer P_0 to one of the components of a nonempty doubly linked list are given. Move this component by K positions backward in the list (if the list contains less than K components before the given component then move it to the beginning of the list). Output the addresses of the first and last component of the resulting list. Do not use operations of allocating and freeing memory; do not change the *Data* fields.

Dynamic48. Pointers P_X and P_Y to different components of a doubly linked list are given. The component with the address P_X precedes the component with the address P_Y in the list but need not be adjacent with it. Exchange the given components in the list and output the address of the first component of the resulting list. Do not use operations of allocating and freeing memory; do not change the *Data* fields.

Dynamic49. A pointer P_1 to the first component of a nonempty doubly linked list is given. Rearrange list components by moving all components with odd order numbers to the end of the list (in the same order). Output the address of the first component of the resulting list. Do not use operations of allocating and freeing memory; do not change the *Data* fields.

Dynamic50. A pointer P_1 to the first component of a nonempty doubly linked list is given. Rearrange list components by moving all components with odd values to the end of the list (in the same order). Output the address of the first component of the resulting list. Do not use operations of allocating and freeing memory; do not change the *Data* fields.

Dynamic51. Two nonempty doubly linked lists are given; pointers P_1 and P_2 refer to the first and last component of the first list, a pointer P_0 refers to one of the components of the second list. Combine the given lists by inserting all components of the first list (in the same order) before the given component of the second list. Output the addresses of the first and last component of the combined list. Do not use operations of allocating and freeing memory.

Dynamic52. Two nonempty doubly linked lists are given; pointers P_1 and P_2 refer to the first and last component of the first list, a pointer P_0 refers to one of the components of the second list. Combine the given lists by inserting all components of the first list (in the same order) after the given component of the second list. Output the addresses of the first and last component of the combined list. Do not use operations of allocating and freeing memory.

Dynamic53. Pointers P_X and P_Y to different components of a doubly linked list are given. The component with the address P_X precedes the component with the address P_Y in the list but need not be adjacent with it. Move list components that are located between the given components (including these components) to a new list (in the same order). Output the addresses of the first components of the changed and new list. If the changed list will be empty then output nil for this list. Do not use operations of allocating and freeing memory.

Dynamic54. Pointers P_X and P_Y to different components of a doubly linked list are given. The component with the address P_X precedes the component with the address P_Y in the list but need not be adjacent with it. Move list components that are located between the given components (not including these components) to a new list (in the same order). Output the addresses of the first components of the changed and new list. If the new list will be empty then output nil for this list. Do not use operations of allocating and freeing memory.

Dynamic55. A pointer P_1 to the first component of a nonempty doubly linked list is given. Transform this list to the *circular* one by assigning the address of the first component to the *Next* field of the last component and the address of the last component to the *Prev* field of the first component. Output the address of the component that has been the last component of the given list.

Dynamic56. Pointers P_1 and P_2 to the first and last component of a doubly linked list are given. The amount of list components is an even number. Split the list into two *circular* lists (see Dynamic55); the first (second) resulting list must contain the first (second) half of components of the given list respectively. Output the pointers P_3 and P_4 to two middle components of the given list; the component with the address P_3 must be contained in the first resulting circular list, the component with the address P_4 must be contained in the second one. Do not use operations of allocating and freeing memory.

Dynamic57. An integer K (> 0) and pointers P_1 and P_2 to the first and last component of a nonempty doubly linked list are given. Perform a *cyclic shift* of all list components by K positions forward (that is, from the beginning toward the end of the list). Output the addresses of the first and last component of the resulting list. The required shift should be performed as follows: transform the given list to the circular one (see Dynamic55) and then break this circular list at the position that corresponds to the given value of K. Do not use operations of allocating and freeing memory.

Dynamic58. An integer K (> 0) and pointers P_1 and P_2 to the first and last component of a nonempty doubly linked list are given. Perform a *cyclic shift* of all list components by K positions backward (that is, from the end toward the beginning of the list). Output the addresses of the first and last component of the resulting list. The required shift should be performed as follows: transform the given list to the circular one (see Dynamic55) and then break this circular list at the position that corresponds to the given value of *K*. Do not use operations of allocating and freeing memory.

Dynamic59. Pointers P_1, P_2, and P_3 to the first, last, and current component of a doubly linked list are given (if the list is empty then the pointers equal nil). Also an integer *N* (> 0) and a sequence of *N* integers are given. Define a new type called TList that is a record with three fields—*First*, *Last*, *Current*—of PNode type (the fields refer to the *first*, *last*, and *current* component of a doubly linked list respectively). Also write a procedure InsertLast(*L*, *D*) that inserts a new component with the value *D* at the end of a list *L* (a record *L* of TList type is an input and output parameter, an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert all elements of the given sequence at the end of the given list (in the same order). Output the new addresses of the first, last, and current component of the resulting list.

Dynamic60. Pointers P_1, P_2, and P_3 to the first, last, and current component of a doubly linked list are given (if the list is empty then the pointers equal nil). Also an integer *N* (> 0) and a sequence of *N* integers are given. Using the TList type (see Dynamic59), write a procedure InsertFirst(*L*, *D*) that inserts a new component with the value *D* at the beginning of a list *L* (a record *L* of TList type is an input and output parameter, an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert all elements of the given sequence at the beginning of the given list (a component with the last value must be the first component of the resulting list). Output the new addresses of the first, last, and current component of the resulting list.

Dynamic61. Pointers P_1, P_2, and P_3 to the first, last, and current component of a nonempty doubly linked list and five integers are given. Using the TList type (see Dynamic59), write a procedure InsertBefore(*L*, *D*) that inserts a new component with the value *D* before the current component of a list *L* (a record *L* of TList type is an input and output parameter, an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert five given integers into the given list. Output the new addresses of the first, last, and current component of the resulting list.

Dynamic62. Pointers P_1, P_2, and P_3 to the first, last, and current component of a nonempty doubly linked list and five integers are given. Using the TList type (see Dynamic59), write a procedure InsertAfter(*L*, *D*) that inserts a new component with the value *D* after the current component of a list *L* (a record *L* of TList type is an input and output parameter, an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert five given integers into the given list.

Output the new addresses of the first, last, and current component of the resulting list.

Dynamic63. Pointers P_1, P_2, and P_3 to the first, last, and current component of a nonempty doubly linked list are given. Using the TList type (see Dynamic59), write three procedures: a procedure ToFirst(*L*) makes the first component of a list *L* the current one; a procedure ToNext(*L*) makes the component, which follows the current component of a list *L*, the new current one (provided that such a component exists); a procedure SetData(*L*, *D*) assigns a new integer value *D* to the current component of a list *L*. Also write a logical function IsLast(*L*) that returns True if the current component of a list *L* is the last component, and False otherwise. A record *L* of TList type is an input and output parameter of the ToFirst and ToNext procedure and is an input parameter of the SetData procedure and the IsLast function. Using these procedures and function, assign zero value to the list components with odd order numbers. Output the amount of list components and also output addresses of the first, last, and current component of the resulting list (the current component should be the last one).

Dynamic64. Pointers P_1, P_2, and P_3 to the first, last, and current component of a nonempty doubly linked list are given. Using the TList type (see Dynamic59), write two procedures: a procedure ToLast(*L*) makes the last component of a list *L* the current one; a procedure ToPrev(*L*) makes the component, which precedes the current component of a list *L*, the new current one (provided that such a component exists). Also write two functions: an integer function GetData(*L*) returns the value of the current component of a list *L*; a logical function IsFirst(*L*) returns True if the current component of a list *L* is the first component, and False otherwise. A record *L* of TList type is an input and output parameter of the ToLast and ToPrev procedure and is an input parameter of the GetData and IsFirst function. Using these procedures and functions, browse all list components from the end toward the beginning of the list and output their values that are even numbers. Also output the amount of list components.

Dynamic65. Pointers P_1, P_2, and P_3 to the first, last, and current component of a doubly linked list are given. The list contains at least five components. Using the TList type (see Dynamic59), write an integer function DeleteCurrent(*L*) that removes the current component of a list *L*, releases memory allocated for the component being removed, and returns the value of this component (a record *L* of TList type is an input and output parameter). If the next component of the list *L* exists then it becomes the new current component, otherwise the last component becomes the new current one. Using this function, remove five components from the given list and output their values. Also output the new addresses of the first, last, and current component of the resulting list (if the resulting list will be empty then output nil three times).

Dynamic66. Pointers P_1, P_2, and P_3 to the first, last, and current component of a nonempty doubly linked list are given. Using the TList type (see Dynamic59), write a procedure SplitList(L_1, L_2) that moves some components of a list L_1 into a new list L_2: components between the current and last component inclusively must be moved (as a result, the list L_1 will be split into two parts; the first part may be empty). A record L_1 of TList type is an input and output parameter, a record L_2 of the same type is an output parameter. The first component of each nonempty resulting list becomes the current component of this list. The procedure should not use operations of allocating and freeing memory. Using this procedure, split the given list into two lists and output the addresses of the first, last, and current component of each resulting list.

Dynamic67. Pointers to the first, last, and current component of two nonempty doubly linked lists are given. Using the TList type (see Dynamic59), write a procedure AddList(L_1, L_2) that inserts all components of a list L_1 (in the same order) at the end of a list L_2; as a result, the list L_1 will be empty. The first component being inserted becomes the current component of the list L_2. Records L_1 and L_2 of TList type are input and output parameters. The procedure should not use operations of allocating and freeing memory. Using this procedure, insert the first given list at the end of the second one and output the addresses of the first, last, and current component of the combined list.

Dynamic68. Pointers to the first, last, and current component of two nonempty doubly linked lists are given. Using the TList type (see Dynamic59), write a procedure InsertList(L_1, L_2) that inserts all components of a list L_1 (in the same order) before the current component of a list L_2; as a result, the list L_1 will be empty. The first component being inserted becomes the current component of the list L_2. Records L_1 and L_2 of TList type are input and output parameters. The procedure should not use operations of allocating and freeing memory. Using this procedure, insert the first given list before the current component of the second one and output the addresses of the first, last, and current component of the combined list.

Dynamic69. Pointers to the first, last, and current component of two doubly linked lists are given; the second list may be empty. Using the TList type (see Dynamic59), write a procedure MoveCurrent(L_1, L_2) that removes the current component from a list L_1 and inserts this component after the current component of a list L_2. If the next component of the list L_1 exists then it becomes the new current component of this list, otherwise the last component becomes the new current one; the component being inserted becomes the current component of the list L_2. Records L_1 and L_2 of TList type are input and output parameters. The procedure should not use operations of allocating and freeing memory. Using this procedure, move the current component of the first given list into the second one and output the addresses of the first, last, and current component of each resulting list (if the first resulting list will be empty then output nil three times for this list).

19.5. List with the barrier component

In these tasks a *doubly linked list* structure is implemented by a *circular* doubly linked chain of nodes with an additional *barrier node* (the *barrier component* of a list). This barrier node is linked with the first and last “true” list component by the Next and Prev field respectively; similarly, the first/last “true” list component is linked with the barrier node by the Prev/Next field respectively. Such a list implementation allows to store only *two* pointers (to the barrier and current list component) for list processing. The *Data* field of the barrier component may be of any value; for definiteness the value of this field is considered to equal zero. Both some “true” list component and the barrier component may be the current component. An *empty* list in this implementation is represented as a single barrier node *linked with itself*; the current component of an empty list is always the barrier component.

Dynamic70. Pointers P_1 and P_2 to the first and last component of a doubly linked list are given (a doubly linked list is implemented by a chain of linked nodes of TNode type, the *Prev* field of the first node and the *Next* field of the last node are equal to nil); if the list is empty then the pointers equal nil. Transform the given list into the circular list (see Dynamic55) with a *barrier component*. The barrier component has zero value and is linked with the first and last component of the given list by the *Next* and *Prev* field respectively (if the given list is empty then the *Next* and *Prev* field of the barrier component must refer to the barrier component itself). Output the address of the barrier component of the resulting list. Use the operation of allocating memory only for creation of the barrier component.

Dynamic71. Pointers P_1 and P_2 to the barrier and current component of a doubly linked list are given (see the *barrier component* definition in Dynamic70). Move the given list components that are between the current and last component (inclusively) to a new list with the barrier component. If the current component of the given list is its barrier component then the new list must be empty (that is, it must contain the barrier component only). Output the address of the barrier component of the new list. Use the operation of allocating memory only for creation of the barrier component of the new list.

Dynamic72. Pointers P_1 and P_2 to the barrier components of two doubly linked lists are given (see the *barrier component* definition in Dynamic70). Combine the given lists by linking the last component of the first list with the first component of the second list. Use the barrier component of the first list as the barrier component of the combined list. Output the addresses of the first and last component of the combined list (if the combined list will be empty then output the address of its barrier component twice). After removing the superfluous barrier component (of the second list) release the memory allocated for this component.

Dynamic73. Pointers P_1 and P_2 to the barrier components of two doubly linked lists are given (see the *barrier component* definition in Dynamic70). Combine the given lists by linking the last component of the first list with the first component of the second list. Use the barrier component of the second list as the barrier component of the combined list. Output the addresses of the first and last component of the combined list (if the combined list will be empty then output the address of its barrier component twice). After removing the superfluous barrier component (of the first list) release the memory allocated for this component.

Dynamic74. Pointers P_1 and P_2 to the barrier and current component of a doubly linked list are given (see the *barrier component* definition in Dynamic70). Also an integer N (> 0) and a sequence of N integers are given. Define a new type called TListB that is a record with two fields, *Barrier* and *Current*, of PNode type (the fields refer to the *barrier* and *current* component of a doubly linked list respectively). Also write a procedure LBInsertLast(*L*, *D*) that inserts a new component with the value *D* at the end of a list *L* (a record *L* of TListB type is an input and output parameter, an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert all elements of the given sequence at the end of the given list (in the same order). Output the new address of the current component of the resulting list.

Dynamic75. Pointers P_1 and P_2 to the barrier and current component of a doubly linked list are given. Also an integer N (> 0) and a sequence of N integers are given. Using the TListB type (see Dynamic74), write a procedure LBInsertFirst(*L*, *D*) that inserts a new component with the value *D* at the beginning of a list *L* (a record *L* of TListB type is an input and output parameter, an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert all elements of the given sequence at the beginning of the given list (a component with the last value must be the first component of the resulting list). Output the new address of the current component of the resulting list.

Dynamic76. Pointers P_1 and P_2 to the barrier and current component of a doubly linked list and five integers are given. Using the TListB type (see Dynamic74), write a procedure LBInsertBefore(*L*, *D*) that inserts a new component with the value *D* before the current component of a list *L* (a record *L* of TListB type is an input and output parameter, an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert five given integers into the given list. Output the new address of the current component of the resulting list.

Dynamic77. Pointers P_1 and P_2 to the barrier and current component of a doubly linked list and five integers are given. Using the TListB type (see Dynamic74), write a procedure LBInsertAfter(*L*, *D*) that inserts a new component with the value *D* after the current component of a list *L* (a record *L* of TListB type is an input and output parameter, an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert five given integers into the given list. Output the new address of the current component of the resulting list.

Dynamic78. Pointers P_1 and P_2 to the barrier and current component of a doubly linked list are given. Using the TListB type (see Dynamic74), write three procedures: a procedure LBToFirst(*L*) makes the first component of a list *L* the current one; a procedure LBToNext(*L*) makes the component, which follows the current component of a list *L*, the new current one; a procedure LBSetData(*L*, *D*) assigns a new integer value *D* to the current component of a list *L* (provided that the current component is not the barrier one). Also write a logical function IsBarrier(*L*) that returns True if the current component of a list *L* is the barrier component, and False otherwise. A record *L* of TListB type is an input and output parameter of the LBToFirst and LBToNext procedure and is an input parameter of the LBSetData procedure and the IsBarrier function. Using these procedures and function, assign zero value to the list components with odd order numbers. Output the amount of list components and the address of the current component of the resulting list (the current component should be the barrier one). The components are numbered from the first component, which has the order number 1; the barrier component is not numbered and should not be counted.

Dynamic79. Pointers P_1 and P_2 to the barrier and current component of a doubly linked list are given. Using the TListB type (see Dynamic74), write two procedures: a procedure LBToLast(*L*) makes the last component of a list *L* the current one; a procedure LBToPrev(*L*) makes the component, which precedes the current component of a list *L*, the new current one. Also write an integer function LBGetData(*L*) that returns the value of the current component of a list *L*. A record *L* of TListB type is an input and output parameter of the LBToLast and LBToPrev procedure and is an input parameter of the LBGetData function. Using these procedures and function and also the IsBarrier function from the task Dynamic78, browse all list components from the end toward the beginning of the list and output their values that are even numbers. Also output the amount of list components. The barrier component should not be processed and counted.

Dynamic80. Pointers P_1 and P_2 to the barrier and current component of a nonempty doubly linked list are given; the current component is not the barrier one. Using the TListB type (see Dynamic74), write an integer function LBDeleteCurrent(*L*) that removes the current component of a list *L*, releases memory allocated for the component being removed, and returns the value of this component (a record *L* of TListB type is an input and output parameter). If the next component of the list *L* is not the barrier one then it becomes the new current component, otherwise the previous component becomes the new current one. If the current component is the barrier one then the function performs no actions and returns 0. Using this function and also the IsBarrier function from the task Dynamic78, remove five components from the given list (or all components if their amount is less than five) and output their values. Also output the new address of the current component of the resulting list.

20. Dynamic data structures (based on objects)

All numbers mentioned in tasks of this group are of integer type. All objects are of Node type; this class is defined in Programming Taskbook. In the tasks of this group the Data, Next, and Prev properties of the Node class are used. Therefore one can assume that the Node class contains the following public members:

[C#]

// Constructors:
```
public Node();
public Node(int aData);
public Node(int aData, Node aNext);
public Node(int aData, Node aNext, Node aPrev);
```
// Properties (available to read and to write):
```
public int Data;
public Node Next;
public Node Prev;
```
// Method that releases resources used by the Node object:
```
public void Dispose();
```
[VB.NET]

' Constructors:
```
Public Sub New()
Public Sub New(aData As Integer)
Public Sub New(aData As Integer, aNext As Node)
Public Sub New(aData As Integer, aNext As Node, _
  aPrev As Node)
```
' Properties (available to read and to write):
```
Public Property Data() As Integer
Public Property Next() As Node
Public Property Prev() As Node
```
' Method that releases resources used by the Node object:
```
Public Sub Dispose() Implements IDisposable.Dispose
```
[Java]

// Constructors:
```
Node();
Node(int aData);
Node(int aData, Node aNext);
Node(int aData, Node aNext, Node aPrev);
```
// Accessors to properties:
```
int getData();
void setData(int value);
Node getNext();
void setNext(Node value);
Node getPrev();
```

```
  void setPrev(Node value);
```
// Method that releases resources used by the Node object:
```
  void dispose();
```
[Python]

# Constructor:
```
  Node(data = 0, next = None, prev = None)
```
# Properties (available to read and to write):
```
  Data
  Next
  Prev
```
# Method that releases resources used by the Node object:
```
  dispose()
```
[Ruby]

# Constructors:
```
  Node.new()
  Node.new(data)
  Node.new(data, next)
  Node.new(data, next, prev)
```
# Properties (available to read and to write):
```
  data
  next
  prev
```
# Method that releases resources used by the Node object:
```
  dispose()
```

In the introductory tasks and in the tasks devoted to *stacks* and *queues* the Prev property of the Node class is not used. In the tasks devoted to *lists* all properties (Data, Next, Prev) of the Node class are used.

All these languages use the reference object model; that is, any object variable is a *reference* to the object instance. Therefore, the expression “output the reference to a node” means that you should output the value of a corresponding variable of the Node type.

The order number of the first node of a list is assumed to be equal to 1.

20.1. Nodes and chains of nodes

ObjDyn1. An object A_1 of the Node class is given. The class contains public properties *Data* (of integer type) and *Next* (of Node type). The given object is linked by its *Next* property with the next object of the same type (that is, the *Next* property of the given object contains the *reference* to the next object). Output the value of the *Data* property for each object, and also output a reference to the object A_2 that follows the given object.

ObjDyn2. An object A_1 of Node type is given. The object is linked by its *Next* property with the next object of the same type, that object is linked with the next one, and so on, until the last object whose *Next* property equals null (as a result, we obtain a *chain* of linked objects). Output the value of the *Data* property for each chain component, the chain *length* (that is, the amount of its components) and a reference to the last chain component.

20.2. Stack

In these tasks a *stack* structure is implemented by a chain of linked components (*nodes*) that are instances of the Node class. The Next property of the last node equals null. The first node is said to be a *top* of the stack. The stack data can be accessed by means of the object variable (of the Node class) that refers to the top of the stack; if the stack is empty then this variable equals null. The value of the Data property of a stack component is considered as the *value of this component*.

ObjDyn3. An integer D and the top A_1 of a nonempty stack are given. Push a component with the value D onto the stack and output a reference A_2 to a new top of the stack.

ObjDyn4. An integer N (> 0) and a sequence of N integers are given. Create a stack that contains N components with the given values (a component with the last value must be the top of the stack) and output a reference to the top of the stack.

ObjDyn5. The top A_1 of a nonempty stack is given. Pop the top component off the stack and output its value D and a reference A_2 to a new top of the stack. If the stack will be empty after popping the component then A_2 must be equal to null. After popping the component release resources allocated for this component; for this purpose call its Dispose method.

ObjDyn6. The top A_1 of a stack is given; the stack contains at least ten components. Pop the first nine components off the stack and output their values and a reference A_2 to a new top of the stack. After popping components release resources allocated for these components; for this purpose call the Dispose method for each of them.

ObjDyn7. The top A_1 of a stack is given (if the stack is empty then A_1 equals null). Pop all components off the stack and output their values. Also output the amount of popped components (if the stack is empty then output 0). After popping components release resources allocated for these components; for this purpose call the Dispose method for each of them.

ObjDyn8. The tops A_1 and A_2 of two nonempty stacks are given. Move all components from the first stack into the second one (as a result, all components of the first stack will be contained within the second stack in inverse order). Output a reference to a new top of the second stack. Do not create new instances of the Node class.

ObjDyn9. The tops A_1 and A_2 of two nonempty stacks are given. Move components from the first stack into the second one until the value of the top component of the first stack is equal to an even number (as a result, all components having been moved will be contained within the second stack in inverse order). If the first stack contains no components with even values then move all its components. Output a reference to a new top for each stack (if the first stack will be empty then output null for this stack). Do not create new instances of the Node class.

ObjDyn10. The top A_1 of a nonempty stack is given. Create two new stacks by moving the given stack components whose values are even (odd) numbers into the first (second) new stack respectively. As a result, all components having been moved will be contained within each new stack in inverse order; one of the new stacks may be empty. Output a reference to the top for each new stack (if one of the new stacks will be empty then output null for this stack). Do not create new instances of the Node class.

ObjDyn11. The top A_1 of a stack is given (if the stack is empty then A_1 equals null). Also an integer N (> 0) and a sequence of N integers are given. Define a new class called IntStack that contains the following members:

• a private field, *top*, of Node type (this field refers to the top of the stack);

• a constructor with the parameter *aTop* of Node type (this parameter refers to the top of some existing stack);

• a procedure Push(D) that pushes a new component with the value D onto the stack (an integer D is an input parameter);

• a procedure Put that output a reference to the *top* field by means of the Put method of the PT class (this procedure has no parameters).

Using the Push method, push all elements of the given sequence onto the given stack (the last number must be the value of the top component). Using the Put method of the IntStack class, output a reference to a new top of the stack.

ObjDyn12. The top A_1 of a stack is given; the stack contains at least five components. Include an integer function Pop in the IntStack class (see ObjDyn11); this function pops the top component off the stack, calls the Dispose method for this component, and returns the value of the popped component. The function has no parameters. Using this function, pop five components off the given stack and output their values. Also output a reference to a new top of the stack (if the stack will be empty then this reference must be equal to null).

ObjDyn13. The top A_1 of a stack is given. Include two functions in the IntStack class (see ObjDyn11): a logical function IsEmpty returns true if the stack is empty, and false otherwise; an integer function Peek returns the value of the top component of the stack. The functions have no parameters. Using these functions and the Pop function from the task ObjDyn12, pop five components (or all stack components if their amount is less than five) off the given stack and output their values. Also output the return value of the IsEmpty function for the resulting stack. At last, in the case of the nonempty resulting stack, output the value of its top component and a reference to this component.

20.3. Queue

In these tasks a *queue* structure is implemented by a chain of linked components (*nodes*) that are instances of the Node class. The Next property of the last node equals null. The first node is said to be a *head* of the queue; the head is located at the *front* of the queue. The last node is said to be a *tail* of the queue; the tail is located at the *end* of the queue. It is convenient to store not only a reference to the queue head but also a reference to the queue tail because it accelerates adding a new component to the end of a queue. If a queue is empty then these references equal null. The value of the Data property of a queue component is considered as the *value of this component*.

ObjDyn14. A sequence of 10 integers is given. Create a queue that contains components with the given values (a component with the first value must be the head of the queue, a component with the last value must be the tail of the queue) and output references A_1 and A_2 to the head and tail of the queue respectively.

ObjDyn15. A sequence of 10 integers is given. Create two queues; the first one must contain the given integers with odd order numbers (1, 3, …, 9), the second one must contain the given integers with even order numbers (2, 4, …, 10). Output references to the head and tail of the first queue and then output references to the head and tail of the second one.

ObjDyn16. A sequence of 10 integers is given. Create two queues; the first one must contain the given integers with odd values (in the same order), the second one must contain the given integers with even values (in the same order). Output references to the head and tail of the first queue and then output references to the head and tail of the second one (if one of the queues will be empty then output null twice for this queue).

ObjDyn17. An integer D and references A_1 and A_2 to the head and tail of a queue are given (if the queue is empty then the references equal null). Add a component with the value D to the end of the queue and output references to the head and tail of the resulting queue.

ObjDyn18. An integer D and references A_1 and A_2 to the head and tail of a queue are given; the queue contains at least two components. Add a component with the value D to the end of the queue and remove the first component from the front of the queue. Output the value of the component being removed and also output references to the head and tail of the resulting queue. After removing the component call its Dispose method.

ObjDyn19. An integer N (> 0) and references A_1 and A_2 to the head and tail of a nonempty queue are given. Remove N initial components from the queue and output their values (if the queue contains less than N components then remove all its components). Also output references to the head and tail of the resulting queue (if the queue will be empty then output null twice). After removing components call the Dispose method for each of them.

ObjDyn20. References A_1 and A_2 to the head and tail of a nonempty queue are given. Remove components from the front of the queue until the value of the head of the queue is equal to an even number; output values of all components being removed (if the queue contains no components with even values then remove all its components). Also output references to the head and tail of the resulting queue (if the queue will be empty then output null twice). After removing components call the Dispose method for each of them.

ObjDyn21. Two queues are given; references A_1 and A_2 refer to the head and tail of the first one, references A_3 and A_4 refer to the head and tail of the second one (if some queue is empty then the corresponding references equal null). Move all components from the first queue (starting with its first component) to the end of the second one. Output references to the head and tail of the changed second queue. Do not create new instances of the Node class.

ObjDyn22. An integer N (> 0) and two nonempty queues are given; references A_1 and A_2 refer to the head and tail of the first one, references A_3 and A_4 refer to the head and tail of the second one. Move N initial components of the first queue to the end of the second one (if the first queue contains less than N components then move all its components). Output references to the head and tail of the first queue and then output references to the head and tail of the second one (if the first queue will be empty then output null twice for this queue). Do not create new instances of the Node class.

ObjDyn23. Two nonempty queues are given; references A_1 and A_2 refer to the head and tail of the first one, references A_3 and A_4 refer to the head and tail of the second one. Move initial components of the first queue to the end of the second one until the value of the head of the first queue is equal to an even number (if the first queue contains no components with even values then move all its components). Output references to the head and tail of the first queue and then output references to the head and tail of the second one (if the first queue will be empty then output null twice for this queue). Do not create new instances of the Node class.

ObjDyn24. Two nonempty queues are given; references A_1 and A_2 refer to the head and tail of the first one, references A_3 and A_4 refer to the head and tail of the second one. The queues contain the equal amount of components. Combine the given queues into a new one; the resulting queue must contain alternating components of the given queues starting with the head of the first one. Output references to the head and tail of the resulting queue. Do not create new instances of the Node class.

ObjDyn25. Two nonempty queues are given; references A_1 and A_2 refer to the head and tail of the first one, references A_3 and A_4 refer to the head and tail of the second one. The values of components of each given queue are in ascending order. Combine the given queues into a new one; the values of components of the resulting queue must be in ascending order too. Output references to the head and tail of the resulting queue. Do not create new instances of the Node class; do not change the *Data* properties.

ObjDyn26. References A_1 and A_2 to the head and tail of a queue are given (if the queue is empty then the references equal null). Also an integer N (> 0) and a sequence of N integers are given. Define a class called IntQueue that contains the following members:

• two private fields, *head* and *tail*, of Node type (these fields refer to the *head* and *tail* of the queue respectively);

• a constructor with the parameters *aHead* and *aTail* of Node type (these parameters refer to the head and tail of some existing queue);

• a procedure Enqueue(D) that adds a new component with the value D to the end of the queue (an integer D is an input parameter);

• a procedure Put that output references to the *head* and *tail* fields by means of the Put method of the PT class (this procedure has no parameters).

Using the Enqueue method, add all elements of the given sequence to the end of the given queue. Using the Put method of the IntQueue class, output references to the head and tail of the resulting queue.

ObjDyn27. References A_1 and A_2 to the head and tail of a queue are given; the queue contains at least five components. Include an integer function Dequeue in the IntQueue class (see ObjDyn26); this function removes the first component from the front of the queue, calls the Dispose method for this component, and returns the value of the removed component. The function has no parameters. Using this function, remove five initial components from the front of the given queue and output their values. Also output references to the head and tail of the queue (if the queue will be empty then output null twice).

ObjDyn28. References A_1 and A_2 to the head and tail of a queue are given. Include a logical function IsEmpty in the IntQueue class (see ObjDyn26); this function returns true if the queue is empty, and false otherwise. The function has no parameters. Using this function and also the Dequeue function from the task ObjDyn27, remove five initial components (or all queue components if their amount is less than five) from the front of the given queue and output their values. Also output the return value of the IsEmpty function for the resulting queue and references to the head and tail of this queue (if the queue will be empty then output null twice).

20.4. Doubly linked list

In these tasks a *doubly linked list* structure is implemented by a chain of components (*nodes*) of the Node class; these nodes are linked with both the next node and the previous one. The Next property of the last node and the Prev property of the first node are equal to null. Though storing of a reference to some list node is sufficient to provide access to any list node, it is convenient to store *three* references (to the *first*, *last*, and *current* list node) because they accelerate list operations. If a list is empty then these references equal null. The value of the Data property of a list component is considered as the *value of this component*.

ObjDyn29. An object A_2 of the Node class is given. The class contains public properties *Data* (of integer type), *Prev* and *Next* (each of Node type). The given object is linked by its *Prev* and *Next* properties with the previous and next object of the same type respectively (that is, the *Prev* and *Next* properties of the given object contain *references* to the previous and next object). Output the values of the *Data* property for the previous and next object, and also output references A_1 and A_3 to the previous and next object.

ObjDyn30. A reference A_1 to the beginning of a chain of objects is given; objects have the Node type and are linked by their *Next* property. Using the *Prev* property of the Node class, transform the given (*singly linked*) chain into the *doubly linked* chain whose components are linked not only with the next ones (by the *Next* property) but also with the previous ones (by the *Prev* property). The *Prev* property of the first chain component must be equal to null. Output a reference A_2 to the last component of the resulting chain.

ObjDyn31. A reference A_0 to one of the components of a nonempty doubly linked list is given. Output the amount N of the list components and also references A_1 and A_2 to the first and last component respectively.

ObjDyn32. Two integers D_1, D_2 and a reference A_0 to one of the components of a nonempty doubly linked list are given. Insert a new component with the value D_1 at the beginning of the list; insert a new component with the value D_2 at the end of the list. Output references to the first and last list component.

ObjDyn33. An integer D and a reference A_0 to one of the components of a nonempty doubly linked list are given. Insert a new component with the value D before the given list component. Output a reference to the component being inserted.

ObjDyn34. An integer D and a reference A_0 to one of the components of a nonempty doubly linked list are given. Insert a new component with the value D after the given list component. Output a reference to the component being inserted.

ObjDyn35. References A_1 and A_2 to the first and last component of a doubly linked list are given. The list contains at least two components. Double occurrences of the first and last list components (new components must be inserted before the existing ones with the same value). Output a reference to the first component of the resulting list.

ObjDyn36. References A_1 and A_2 to the first and last component of a doubly linked list are given. The list contains at least two components. Double occurrences of the first and last list components (new components must be inserted after the existing ones with the same value). Output a reference to the last component of the resulting list.

ObjDyn37. A reference A_1 to the first component of a nonempty doubly linked list is given. Double occurrences of all list components with odd order numbers (new components must be inserted before the existing ones with the same value). Output a reference to the first component of the resulting list.

ObjDyn38. A reference A_1 to the first component of a nonempty doubly linked list is given. Double occurrences of all list components with odd order numbers (new components must be inserted after the existing ones with the same value). Output a reference to the last component of the resulting list.

ObjDyn39. A reference A_1 to the first component of a nonempty doubly linked list is given. Double occurrences of all list components with odd values (new components must be inserted before the existing ones with the same value). Output a reference to the first component of the resulting list.

ObjDyn40. A reference A_1 to the first component of a nonempty doubly linked list is given. Double occurrences of all list components with odd values (new components must be inserted after the existing ones with the same value). Output a reference to the last component of the resulting list.

ObjDyn41. A reference A_0 to one of the components of a nonempty doubly linked list is given. Remove this component from the list and output references to its previous and next component in the list (one or both these components may be absent; output null for each absent component). After removing the component call its Dispose method.

ObjDyn42. A reference A_1 to the first component of a doubly linked list is given. The list contains at least two components. Remove all components with odd order numbers from the list and output a reference to the first component of the resulting list. After removing components call the Dispose method for each of them.

ObjDyn43. A reference A_1 to the first component of a nonempty doubly linked list is given. Remove all components with odd values from the list and output a reference to the first component of the resulting list (if this list will be empty then output null). After removing components call the Dispose method for each of them.

ObjDyn44. A reference A_0 to one of the components of a nonempty doubly linked list is given. Move this component to the end of the list and output references to the first and last component of the resulting list. Do not create new instances of the Node class; do not change the *Data* properties.

ObjDyn45. A reference A_0 to one of the components of a nonempty doubly linked list is given. Move this component to the beginning of the list and output references to the first and last component of the resulting list. Do not create new instances of the Node class; do not change the *Data* properties.

ObjDyn46. An integer K (> 0) and a reference A_0 to one of the components of a nonempty doubly linked list are given. Move this component by K positions forward in the list (if the list contains less than K components after the given component then move it to the end of the list). Output references to the first and last component of the resulting list. Do not create new instances of the Node class; do not change the *Data* properties.

ObjDyn47. An integer K (> 0) and a reference A_0 to one of the components of a nonempty doubly linked list are given. Move this component by K positions backward in the list (if the list contains less than K components before the given component then move it to the beginning of the list). Output references to the first and last component of the resulting list. Do not create new instances of the Node class; do not change the *Data* properties.

ObjDyn48. References A_X and A_Y to different components of a doubly linked list are given. The component A_X precedes the component A_Y in the list but need not be adjacent with it. Exchange the given components in the list and output a reference to the first component of the resulting list. Do not create new instances of the Node class; do not change the *Data* properties.

ObjDyn49. A reference A_1 to the first component of a nonempty doubly linked list is given. Rearrange list components by moving all components with odd order numbers to the end of the list (in the same order). Output a reference to the first component of the resulting list. Do not create new instances of the Node class; do not change the *Data* properties.

ObjDyn50. A reference A_1 to the first component of a nonempty doubly linked list is given. Rearrange list components by moving all components with odd values to the end of the list (in the same order). Output a reference to the first component of the resulting list. Do not create new instances of the Node class; do not change the *Data* properties.

ObjDyn51. Two nonempty doubly linked lists are given; references A_1 and A_2 refer to the first and last component of the first list, a reference A_0 refers to one of the components of the second list. Combine the given lists by inserting all components of the first list (in the same order) before the given component of the second list. Output references to the first and last component of the combined list. Do not create new instances of the Node class.

ObjDyn52. Two nonempty doubly linked lists are given; references A_1 and A_2 refer to the first and last component of the first list, a reference A_0 refers to one of the components of the second list. Combine the given lists by inserting all components of the first list (in the same order) after the given component of the second list. Output references to the first and last component of the combined list. Do not create new instances of the Node class.

ObjDyn53. References A_X and A_Y to different components of a doubly linked list are given. The component A_X precedes the component A_Y in the list but need not be adjacent with it. Move list components that are located between the given components (including these components) to a new list (in the same order). Output references to the first components of the changed and new list. If the changed list will be empty then output null for this list. Do not create new instances of the Node class.

ObjDyn54. References A_X and A_Y to different components of a doubly linked list are given. The component A_X precedes the component A_Y in the list but need not be adjacent with it. Move list components that are located between the given components (not including these components) to a new list (in the same order). Output references to the first components of the changed and new list. If the new list will be empty then output null for this list. Do not create new instances of the Node class.

ObjDyn55. A reference A_1 to the first component of a nonempty doubly linked list is given. Transform this list to the *circular* one by assigning the first component reference to the *Next* property of the last component and the last component reference to the *Prev* property of the first component. Output a reference to the component that has been the last component of the given list.

ObjDyn56. References A_1 and A_2 to the first and last component of a doubly linked list are given. The amount of list components is an even number. Split the list into two *circular* lists (see ObjDyn55); the first (second) resulting list must contain the first (second) half of components of the given list respectively. Output references A_3 and A_4 to two middle components of the given list; the object A_3 must be contained in the first resulting circular list, the object A_4 must be contained in the second one. Do not create new instances of the Node class.

ObjDyn57. An integer K (> 0) and references A_1 and A_2 to the first and last component of a nonempty doubly linked list are given. Perform a *cyclic shift* of all list components by K positions forward (that is, from the beginning toward the end of the list). Output references to the first and last component of the resulting list. The required shift should be performed as follows: transform the given list to the circular one (see ObjDyn55) and then break this circular list at the position that corresponds to the given value of K. Do not create new instances of the Node class.

ObjDyn58. An integer K (> 0) and references A_1 and A_2 to the first and last component of a nonempty doubly linked list are given. Perform a *cyclic shift* of all list components by K positions backward (that is, from the end toward the beginning of the list). Output references to the first and last component of the resulting list. The required shift should be performed as follows: transform the given list to the circular one (see ObjDyn55) and then break this circular list at the position that corresponds to the given value of K. Do not create new instances of the Node class.

ObjDyn59. References A_1, A_2, and A_3 to the first, last, and current component of a doubly linked list are given (if the list is empty then the references equal null). Also an integer N (> 0) and a sequence of N integers are given. Define a new class called IntList that contains the following members:

- three private fields—*first, last, current*—of Node type (these fields refer to the *first*, *last*, and *current* component of the list respectively);
- a constructor with the parameters *aFirst*, *aLast*, *aCurrent* of Node

type (these parameters refer to the first, last, and current component of some existing list);
• a procedure InsertLast(D) that inserts a new component with the value D at the end of the list (an integer D is an input parameter; the component being inserted becomes the current component of the list);
• a procedure Put that output references to the fields *first*, *last*, and *current* by means of the Put method of the PT class (this procedure has no parameters).

Using the InsertLast method, insert all elements of the given sequence at the end of the given list (in the same order). Using the Put method of the IntList class, output references to the first, last, and current component of the resulting list.

ObjDyn60. References A_1, A_2, and A_3 to the first, last, and current component of a doubly linked list are given (if the list is empty then the references equal null). Also an integer N (> 0) and a sequence of N integers are given. Include a procedure InsertFirst(D) in the IntList class (see ObjDyn59); this procedure inserts a new component with the value D at the beginning of the list (an integer D is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert all elements of the given sequence at the beginning of the given list (a component with the last value must be the first component of the resulting list). Output references to the first, last, and current component of the resulting list.

ObjDyn61. References A_1, A_2, and A_3 to the first, last, and current component of a nonempty doubly linked list and five integers are given. Include a procedure InsertBefore(D) in the IntList class (see ObjDyn59); this procedure inserts a new component with the value D before the current component of the list (an integer D is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert five given integers into the given list. Output references to the first, last, and current component of the resulting list.

ObjDyn62. References A_1, A_2, and A_3 to the first, last, and current component of a nonempty doubly linked list and five integers are given. Include a procedure InsertAfter(D) in the IntList class (see ObjDyn59); this procedure inserts a new component with the value D after the current component of the list (an integer D is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert five given integers into the given list. Output references to the first, last, and current component of the resulting list.

ObjDyn63. References A_1, A_2, and A_3 to the first, last, and current component of a nonempty doubly linked list are given. Include four methods in the IntList class (see ObjDyn59): a procedure ToFirst makes the first component of the list the current one; a procedure ToNext makes the component, which follows the current component of the list, the new current one (provided that such a component exists); a procedure SetData(D) assigns a new integer value D to the current component of the list (an integer D is an input parameter); a logical function IsLast returns true if the current component of the list is the last component, and false otherwise. The ToFirst, ToNext, IsLast methods have no parameters. Using these methods, assign zero value to the list components with odd order numbers. Output the amount of list components and also output references to the first, last, and current component of the resulting list (the current component should be the last one).

ObjDyn64. References A_1, A_2, and A_3 to the first, last, and current component of a nonempty doubly linked list are given. Include four methods in the IntList class (see ObjDyn59): a procedure ToLast makes the last component of the list the current one; a procedure ToPrev makes the component, which precedes the current component of the list, the new current one (provided that such a component exists); an integer function GetData returns the value of the current component of the list; a logical function IsFirst returns true if the current component of the list is the first component, and false otherwise. All these methods have no parameters. Using these methods, browse all list components (from the end toward the beginning of the list) and output their values that are even numbers. Also output the amount of list components.

ObjDyn65. References A_1, A_2, and A_3 to the first, last, and current component of a doubly linked list are given. The list contains at least five components. Include an integer function DeleteCurrent in the IntList class (see ObjDyn59); this function removes the current component of the list, calls the Dispose method for this component, and returns the value of the removed component. The function has no parameters. If the next component of the list exists then it becomes the new current component, otherwise the last component becomes the new current one. Using this function, remove five components from the given list and output their values. Also output references to the first, last, and current component of the resulting list (if the resulting list will be empty then output null three times).

ObjDyn66. References A_1, A_2, and A_3 to the first, last, and current component of a nonempty doubly linked list are given. Include a procedure Split(L_1, L_2) as a *class method* in the IntList class (see ObjDyn59); this procedure moves some components of a list L_1 into a new list L_2: components between the current and last component inclusively must be moved (as a result, the list L_1 will be split into two parts; the first part may be empty). An object L_1 of IntList type is an input parameter, an object L_2 of the same type is an output parameter. The first component of each nonempty resulting list becomes the current component of this list. The procedure should not create new instances of the Node class. Using this procedure, split the given list into two lists and output references to the first, last, and current component of each resulting list.

ObjDyn67. References to the first, last, and current component of two nonempty doubly linked lists are given. Include a procedure Add(L_1, L_2) as a *class* *method* in the IntList class (see ObjDyn59); this procedure inserts all components of a list L_1 (in the same order) at the end of a list L_2; as a result, the list L_1 will be empty. The first component being inserted becomes the current component of the list L_2. Objects L_1 and L_2 of IntList type are input parameters. The procedure should not create new instances of the Node class. Using this procedure, insert the first given list at the end of the second one and output references to the first, last, and current component of the combined list.

ObjDyn68. References to the first, last, and current component of two nonempty doubly linked lists are given. Include a procedure Insert(L_1, L_2) as a *class method* in the IntList class (see ObjDyn59); this procedure inserts all components of a list L_1 (in the same order) before the current component of a list L_2; as a result, the list L_1 will be empty. The first component being inserted becomes the current component of the list L_2. Objects L_1 and L_2 of IntList type are input parameters. The procedure should not create new instances of the Node class. Using this procedure, insert the first given list before the current component of the second one and output references to the first, last, and current component of the combined list.

ObjDyn69. References to the first, last, and current component of two doubly linked lists are given; the second list may be empty. Include a procedure MoveCurrent(L_1, L_2) as a *class method* in the IntList class (see ObjDyn59); this procedure removes the current component from a list L_1 and inserts this component after the current component of a list L_2. If the next component of the list L_1 exists then it becomes the new current component of this list, otherwise the last component becomes the new current one; the component being inserted becomes the current component of the list L_2. Objects L_1 and L_2 of IntList type are input parameters. The procedure should not create new instances of the Node class. Using this procedure, move the current component of the first given list into the second one and output references to the first, last, and current component of each resulting list (if the first resulting list will be empty then output null three times for this list).

20.5. List with the barrier component

In these tasks a *doubly linked list* structure is implemented by a *circular* doubly linked chain of nodes with an additional *barrier node* (the *barrier component* of a list). This barrier node is linked with the first and last “true” list component by the Next and Prev property respectively; similarly, the first/last “true” list component is linked with the barrier node by the Prev/Next property respectively. Such a list implementation allows to store only two references (to the *barrier* and *current* list component) for list processing. The *Data* property of the barrier component may be of any value; for definiteness the value of this property is considered to equal zero. Both some “true” list component and the barrier component may be the current component. An empty list in this implementation is represented as a single barrier node *linked with itself*; the current component of an empty list is always the barrier component.

ObjDyn70. References A_1 and A_2 to the first and last component of a doubly linked list are given (a doubly linked list is implemented by a chain of linked nodes of Node type, the *Prev* property of the first node and the *Next* property of the last node are equal to null); if the list is empty then the references equal null. Transform the given list into the circular list (see ObjDyn55) with a *barrier component*. The barrier component has zero value and is linked with the first and last component of the given list by the *Next* and *Prev* property respectively (if the given list is empty then the *Next* and *Prev* properties of the barrier component must refer to the barrier component itself). Output a reference to the barrier component of the resulting list. Do not create new instances of the Node class except the barrier component.

ObjDyn71. References A_1 and A_2 to the barrier and current component of a doubly linked list are given (see the *barrier component* definition in ObjDyn70). Move the given list components that are between the current and last component (inclusively) to a new list with the barrier component. If the current component of the given list is its barrier component then the new list must be empty (that is, it must contain the barrier component only). Output a reference to the barrier component of the new list. Do not create new instances of the Node class except the barrier component of the new list.

ObjDyn72. References A_1 and A_2 to the barrier components of two doubly linked lists are given (see the *barrier component* definition in ObjDyn70). Combine the given lists by linking the last component of the first list with the first component of the second list. Use the barrier component of the first list as the barrier component of the combined list. Output references to the first and last component of the combined list (if the combined list will be empty then output a reference to its barrier component twice). After removing the superfluous barrier component (of the second list) call its Dispose method.

ObjDyn73. References A_1 and A_2 to the barrier components of two doubly linked lists are given (see the *barrier component* definition in ObjDyn70). Combine the given lists by linking the last component of the first list with the first component of the second list. Use the barrier component of the second list as the barrier component of the combined list. Output references to the first and last component of the combined list (if the combined list will be empty then output a reference to its barrier component twice). After removing the superfluous barrier component (of the first list) call its Dispose method.

ObjDyn74. References A_1 and A_2 to the barrier and current component of a doubly linked list are given (see the *barrier component* definition in ObjDyn70). Also an integer N (> 0) and a sequence of N integers are given. Define a new class called IntListB that contains the following members:

- two private fields, *barrier* and *current*, of Node type (these fields refer to the *barrier* and *current* component of the list respectively);

• a constructor with the parameters *aBarrier* and *aCurrent* of Node type (these parameters refer to the barrier and current component of some existing list);
• a procedure InsertLast(*D*) that inserts a new component with the value *D* at the end of the list (an integer *D* is an input parameter; the component being inserted becomes the current component of the list);
• a procedure Put that output a reference to the *current* field by means of the Put method of the PT class (this procedure has no parameters).

Using the InsertLast method, insert all elements of the given sequence at the end of the given list (in the same order). Using the Put method of the IntListB class, output a reference to the current component of the resulting list.

ObjDyn75. References A_1 and A_2 to the barrier and current component of a doubly linked list are given. Also an integer N (> 0) and a sequence of N integers are given. Include a procedure InsertFirst(*D*) in the IntListB class (see ObjDyn74); this procedure inserts a new component with the value *D* at the beginning of the list (an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert all elements of the given sequence at the beginning of the given list (a component with the last value must be the first component of the resulting list). Output a reference to the current component of the resulting list.

ObjDyn76. References A_1 and A_2 to the barrier and current component of a doubly linked list and five integers are given. Include a procedure InsertBefore(*D*) in the IntListB class (see ObjDyn74); this procedure inserts a new component with the value *D* before the current component of the list (an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert five given integers into the given list. Output a reference to the current component of the resulting list.

ObjDyn77. References A_1 and A_2 to the barrier and current component of a doubly linked list and five integers are given. Include a procedure InsertAfter(*D*) in the IntListB class (see ObjDyn74); this procedure inserts a new component with the value *D* after the current component of the list (an integer *D* is an input parameter). The component being inserted becomes the current component of the list. Using this procedure, insert five given integers into the given list. Output a reference to the current component of the resulting list.

ObjDyn78. References A_1 and A_2 to the barrier and current component of a doubly linked list are given. Include four methods in the IntListB class (see ObjDyn74): a procedure ToFirst makes the first component of the list the current one; a procedure ToNext makes the component, which follows the current component of the list, the current one; a procedure SetData(*D*) assigns a new integer value *D* to the current component of the list provided that the current component is not the barrier one (an integer *D* is an input parameter); a logical function IsBarrier returns true if the current component of the list is the barrier component, and false otherwise. The ToFirst, ToNext, IsBarrier methods have no parameters. Using these methods, assign zero value to the list components with odd order numbers. Output the amount of list components and a reference to the current component of the resulting list (the current component should be the barrier one). The components are numbered from the first component, which has the order number 1; the barrier component is not numbered and should not be counted.

ObjDyn79. References A_1 and A_2 to the barrier and current component of a doubly linked list are given. Include three methods in the IntListB class (see ObjDyn74): a procedure ToLast makes the last component of the list the current one; a procedure ToPrev makes the component, which precedes the current component of the list, the new current one; an integer function GetData returns the value of the current component of the list. All these methods have no parameters. Using these methods and also the IsBarrier function from the task ObjDyn78, browse all list components from the end toward the beginning of the list and output their values that are even numbers. Also output the amount of list components. The barrier component should not be processed and counted.

ObjDyn80. References A_1 and A_2 to the barrier and current component of a nonempty doubly linked list are given; the current component is not the barrier one. Include an integer function DeleteCurrent in the IntListB class (see ObjDyn74); this function removes the current component of the list, calls the Dispose method for this component, and returns the value of the removed component. The function has no parameters. If the next component of the list is not the barrier one then it becomes the new current component, otherwise the previous component becomes the new current one. If the current component is the barrier one then the function performs no actions and returns 0. Using this function and also the IsBarrier function from the task ObjDyn78, remove five components from the given list (or all components if their amount is less than five) and output their values. Also output a reference to the current component of the resulting list.

21. Binary trees (based on pointers)

All numbers mentioned in the tasks of this group are of integer type. All pointers are of PNode type; they point to records of TNode type. In the tasks of this group the Data, Left, Right, and Parent fields of the TNode record are used, therefore one can assume that the PNode and TNode types are defined as follows:

[Pascal]

```
type
  PNode = ^TNode;
  TNode = record
    Data: integer;
    Left: PNode;
    Right: PNode;
```

```
      Parent: PNode;
    end;
```

[C++]

```
  struct TNode
  {
    int Data;
    TNode* Left;
    TNode* Right;
    TNode* Parent;
  };
  typedef TNode* PNode;
```

In the most of the tasks only the Data, Left, and Right fields of the TNode record are used. The Parent field is required in the tasks devoted to doubly linked trees.

The value of the Data field of a variable of TNode type is considered as the *value of the corresponding tree node*.

In C++ programs the delete p operator or DeleteNode(p) function call should be used to free the memory that a pointer p (of the PNode type) addresses.

21.1. Analysis

Tree1. An address P_1 of a record of TNode type is given. The record consists of the *Data* field (of integer type) and the *Left* and *Right* fields (of PNode type). The given record (*a tree root*) is linked by its *Left* and *Right* fields with records of the same type (named the left and right *child nodes* respectively). Output the Data fields of the tree root and its left and right children. Also output the addresses of left and right child nodes.

Tree2. An address P_1 of a record of TNode type (a tree root) is given. This record is linked by its *Left* and *Right* fields with records of the same type (child nodes); they, in turn, are linked with their own child nodes and so on, until records whose *Left* and *Right* fields are equal to nil. Some of the nodes may have one field (*Left* or *Right*) equals nil. Output the amount of tree nodes.

Tree3. A pointer P_1 to the root of a nonempty tree and an integer K are given. Output the amount of nodes whose value equals K.

Tree4. A pointer P_1 to the root of a nonempty tree is given. Output the sum of values of all tree nodes.

Tree5. A pointer P_1 to the root of a nonempty tree is given. Output the amount of left child nodes (the tree root should not be counted).

Tree6. A pointer P_1 to the root of a nonempty tree is given. Nodes without children are called *the external nodes* or *the leaf nodes* (*the leaves*). Output the amount of leaf nodes.

Tree7. A pointer P_1 to the root of a nonempty tree is given. Output the sum of values of all tree leaves.

Tree8. A pointer P_1 to the root of a tree is given, the tree contains at least two nodes. Output the amount of tree leaves that are the right child nodes.

Tree9. A pointer P_1 to the root of a nonempty tree is given. The root node is said to be on the *zero level*, its child nodes — on the *first level*, and so on. Output the *depth* of the tree (that is, the maximal level of tree nodes). For example, the depth of a tree containing only a root node is equal to 0.

Tree10. A pointer P_1 to the root of a nonempty tree is given. For each tree level (including the zero one) output the amount of its nodes. The tree depth is assumed to be not greater than 10.

Tree11. A pointer P_1 to the root of a nonempty tree is given. For each tree level (including the zero one) output the sum of values of its nodes. The tree depth is assumed to be not greater than 10.

Tree12. A pointer P_1 to the root of a nonempty tree is given. Using the recursive algorithm named *inorder tree walk* output the values of all tree nodes as follows: output the left subtree (using inorder tree walk), then output the root node, then output the right subtree (using inorder tree walk too).

Tree13. A pointer P_1 to the root of a nonempty tree is given. Using the recursive algorithm named *preorder tree walk* output the values of all tree nodes as follows: output the root node, then output the left subtree (using preorder tree walk), then output the right subtree (using preorder tree walk too).

Tree14. A pointer P_1 to the root of a nonempty tree is given. Using the recursive algorithm named *postorder tree walk* output the values of all tree nodes as follows: output the left subtree (using postorder tree walk), then output the right subtree (using postorder tree walk too), then output the root node.

Tree15. A pointer P_1 to the root of a nonempty tree and an integer N (> 0) are given. The value of N is not greater than the amount of tree nodes. Output the values of tree nodes whose order numbers are not greater than N (the tree nodes are numbered from 1 using inorder tree walk — see Tree12).

Tree16. A pointer P_1 to the root of a nonempty tree and an integer N (> 0) are given. The value of N is not greater than the amount of tree nodes. Output the values of tree nodes whose order numbers are N or greater (the tree nodes are numbered from 1 using postorder tree walk — see Tree14).

Tree17. A pointer P_1 to the root of a nonempty tree and two integers N_1, N_2 ($0 < N_1 < N_2$) are given. The value of N_2 is not greater than the amount of tree nodes. Output the values of tree nodes whose order numbers are in the range N_1 to N_2 (the tree nodes are numbered from 1 using preorder tree walk — see Tree13).

Tree18. A pointer P_1 to the root of a nonempty tree and an integer L (≥ 0) are given. Using tree walk of any type (see Tree12−Tree14) output values of all nodes of the level L. Also output the amount N of these nodes. If the given tree does not contain nodes of level L then output 0.

Tree19. A pointer P_1 to the root of a nonempty tree is given. Output the maximal value of the tree nodes and the amount of nodes with this value.

Tree20. A pointer P_1 to the root of a nonempty tree is given. Output the minimal value of the tree nodes and the amount of leaves with this value (the amount may be equal to 0).

Tree21. A pointer P_1 to the root of a nonempty tree is given. Output the minimal value of its leaves.

Tree22. A pointer P_1 to the root of a tree is given, the tree contains at least two nodes. Output the maximal value of its *internal nodes* (that is, nodes with children).

Tree23. A pointer P_1 to the root of a nonempty tree is given. Using preorder tree walk, find the first tree node with the minimal value and output its address P_2.

Tree24. A pointer P_1 to the root of a nonempty tree is given. Using inorder tree walk, find the last node with the maximal odd value and output its address P_2. If the tree does not contain nodes with odd values then output nil.

21.2. Creation

Tree25. An integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes and assign values of the given sequence to tree nodes in order of their creation. Each node of the tree (except for the root) should be a right child. Output the address of the tree root.

Tree26. An integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes and assign values of the given sequence to tree nodes in order of their creation. Each internal node of the tree should have one child: the root has a left child, which has a right child, which has a left child, and so on. Output the address of the tree root.

Tree27. An integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes and assign values of the given sequence to tree nodes in order of their creation. Each internal node of the tree should have one child: an internal node whose value is an odd number has a left child, otherwise it has a right child. Output the address of the tree root.

Tree28. An even integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes; left child nodes of the tree should be leaves, right child nodes should be internal ones. For each internal node create a left child at first, then create a right one (if it exists). Assign values of the given sequence to tree nodes in order of their creation. Output the address of the tree root.

Tree29. An even integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes. Inner node whose value is an odd number should have a left child leaf, otherwise it should have a right child leaf. For each internal node create a child leaf node at first, and then create a child internal node (if it exists). Assign values of the given sequence to tree nodes in order of their creation. Output the address of the tree root.

Tree30. An integer N (> 0) is given. Create a tree that satisfies the following conditions: the value of root node equals N; if the value of a node is an even number K then this node has only a left child whose value equals $K/2$; if the value of a node equals 1 then this node is a leaf; if the value of a node is another odd number K then this node has a left child whose value equals $K/2$ and has a right child whose value equals $K - K/2$ ("/" denotes the operator of integer division). Output the address of the tree root.

Tree31. Two positive integers L, N ($N > L$) and a sequence of N integers are given. Create a tree of depth L. Use elements of the given sequence as node values; add new nodes using the following algorithm: for each node of the level not greater than L create the node itself, then its left subtree of corresponding depth, and finally its right subtree. If less than N nodes are required to create an L-depth tree then do not use the rest of elements of the given sequence. Output the address of the tree root.

Tree32. An integer N (> 0) and a sequence of N integers are given. Create *a balanced tree* with N nodes (that is, a binary tree which satisfies the following condition: for each tree node the amount of nodes of its left subtree differs at most on 1 from the amount of nodes of its right subtree) and output the address of the tree root. Use elements of the given sequence as node values; create the tree by means of the following recursive algorithm: create a root node, then repeat the algorithm twice: for creating the left subtree with $N/2$ nodes and for creating the right subtree with $N - 1 - N/2$ nodes ("/" denotes the operator of integer division).

Tree33. An integer N (> 0) is given. Create a balanced tree with N nodes and output the address of the tree root. The value of each node should be equal to its level (for example, the root value is 0, the value of its children is 1, and so on). Create the balanced tree by means of the recursive algorithm described in Tree32.

Tree34. An address P_1 of the root of a nonempty tree is given. Create a copy of the tree and output the address P_2 of its root.

21.3. Changing

Tree35. A pointer P_1 to the root of a nonempty tree is given. Double the value of each tree node.

Tree36. A pointer P_1 to the root of a nonempty tree is given. Halve the value of each tree node whose initial value is an even number.

Tree37. A pointer P_1 to the root of a nonempty tree is given. Add 1 to the value of each tree leaf and subtract 1 from the value of each internal node.

Tree38. A pointer P_1 to the root of a nonempty tree is given. For each tree node with two child swap values of its child nodes (that is, swap values of *Data* fields of child nodes).

Tree39. A pointer P_1 to the root of a nonempty tree is given. Swap child nodes of each internal node in the tree (that is, swap values of its *Left* and *Right* field).

Tree40. A pointer P_1 to the root of a nonempty tree is given. Remove all nodes from the tree (except the root), release the memory allocated for removed nodes, and assign nil to the *Left* and *Right* fields of the root.

Tree41. A pointer P_1 to the root of a nonempty tree is given, the tree contains at least two nodes. Remove all tree leaves and assign nil to the *Left* and *Right* fields of their parents. Release the memory allocated for removed nodes.

Tree42. A pointer P_1 to the root of a nonempty tree is given. Remove all nodes whose value is less than the root value, together with all their descendants. Release the memory allocated for removed nodes.

Tree43. A pointer P_1 to the root of a nonempty tree is given. Apply the following action to each tree node that has two child nodes: if the node value is an even number then remove its right child, otherwise remove its left child. Use preorder tree walk; each node should be removed together with all its descendants. Release the memory allocated for removed nodes.

Tree44. A pointer P_1 to the root of a nonempty tree is given. Add two child nodes to each tree leaf; the values of left and right child nodes should be equal to 10 and 11 respectively.

Tree45. A pointer P_1 to the root of a nonempty tree is given. Add one child node to each thee leaf; if the leaf value is an odd number then its child should be a left node, otherwise its child should be a right one. Value of created child node should be equal to value of its parent.

Tree46. A pointer P_1 to the root of a nonempty tree is given. For each tree node with one child add another child node (a leaf). Value of created child node should be equal to doubled value of its parent.

Tree47. A pointer P_1 to the root of a nonempty tree is given. Transform the given tree to a *perfect tree* by adding some new nodes (a perfect tree is a binary tree whose all leaves are at the same level). Do not change the initial depth of the tree; value of all new nodes should be equal to -1.

21.4. Doubly linked binary trees

Tree48. An address P_1 of a tree node is given. Tree node is a record of TNode type containing the *Data* field (of integer type) and the *Left*, *Right*, and *Parent* fields (of PNode type). The *Left* and *Right* fields point to the left and right child nodes respectively, the *Parent* field points to the parent node (the *Parent* field of the root node equals nil). Output pointers P_L, P_R to the left and right child of the given node, P_0 to its parent, and P_2 to its *sibling* (siblings are nodes that have the same parent). If some of required nodes are not exist then output nil for each absent node.

Tree49. A pointer P_1 to the root of a tree is given. Tree nodes are represented by records of TNode type; they are linked by the *Left* and *Right* fields of TNode record. Using the *Parent* field of TNode record, transform the given tree into *a doubly linked tree* whose each node is connected not only with its child nodes (by the *Left* and *Right* fields) but also with its parent node (by the *Parent* field). The *Parent* field of the root node should be equal to nil.

Tree50. A pointer P_1 to some node of a doubly linked tree is given. Output the pointer P_2 to the tree root.

Tree51. Pointers P_1, P_2, P_3 to three nodes of a doubly linked tree are given. Output the level of each node (the level of the root equals 0).

Tree52. Pointers P_1 and P_2 to two different nodes of a doubly linked tree are given. Output *the degree of relationship* of the node P_1 to the node P_2 (the degree of relationship equals −1 if the node P_2 is not in the chain of ancestors of the node P_1; otherwise it equals $L_1 - L_2$, where L_1 and L_2 are the levels of nodes P_1 and P_2 respectively).

Tree53. Pointers P_1 and P_2 to two different nodes of a doubly linked tree are given. Find the nearest mutual ancestor of the nodes P_1 and P_2 and output its pointer P_0.

Tree54. A pointer P_1 to the node of a doubly linked tree is given. Create a copy of the given tree and output a pointer P_2 to the root of the created tree.

Tree55. A pointer P_1 to the non-root node of a doubly linked tree is given. If the node P_1 has a sibling then remove the sibling together with all its descendants from the tree and release the memory allocated for removed nodes, if the node P_1 has no sibling then create it and all its descendants as a copy of the subtree with the root P_1. Output the pointer P_0 to the parent of P_1.

Tree56. Two positive integers L, N ($N > L$) and a sequence of N integers are given. Create a doubly linked tree of depth L. Use elements of the given sequence as node values; add new nodes using the following algorithm: for each node of the level not greater than L create the node itself, then its left subtree of corresponding depth, and finally its right subtree. If less than N nodes are required to create an L depth tree then do not use the rest of elements of the given sequence. Output the address of the tree root.

21.5. Binary search trees

Tree57. A pointer P_1 to the root of a nonempty tree is given. It the tree is *a search tree*, that is, values of its nodes form a non-decreasing sequence in inorder tree walk, then output nil; otherwise output the address of the first node (in inorder tree walk) that breaks the search-tree property.

Tree58. A pointer P_1 to the root of a nonempty tree is given. It the tree is *a non-recurrent search tree*, that is, values of its nodes form an increasing sequence in inorder tree walk, then output nil; otherwise output the address of the first node (in inorder tree walk) that breaks the search-tree property.

Tree59. A pointer P_1 to the root of a nonempty non-recurrent search tree and an integer K are given. If the tree contains a node whose value equals K then output the address P_2 of this node, otherwise output nil. Also output the amount N of tree nodes that were checked during the search.

Tree60. A pointer P_1 to the root of a nonempty search tree and an integer K are given. Output the amount C of tree nodes whose value equals K. Also output the amount N of tree nodes that were checked during the search.

Tree61. A pointer P_1 to the root of a search tree and an integer K are given (if the tree is empty then P_1 = nil). Add a new node with the value K to the tree so that the tree still remains a search tree. Output the pointer P_2 to the root of the resulting tree. Use the following recursive algorithm for a tree with the root P: if P = nil then create a leaf with the value K and assign the address of the leaf to the pointer P; if the tree root exists then repeat the algorithm for the left subtree in case K is less than the root value or for the right subtree otherwise.

Tree62. A pointer P_1 to the root of a non-recurrent search tree and an integer K are given (if the tree is empty then P_1 = nil). Add a new node with the value K to the tree so that the tree still remains a non-recurrent search tree. Do not change the given tree if it already contains a node with the value K. Output the pointer P_2 to the root of the resulting tree. Use the following recursive algorithm for a tree with the root P: if P = nil then create a leaf with the value K and assign the address of the leaf to the pointer P; if the tree root exists then repeat the algorithm for the left subtree in case K is less than the root value or for the right subtree in case K is greater than the root value.

Tree63. An integer N (> 0), a sequence of N integers and a pointer P_1 to the root of a search tree are given (if the tree is empty then P_1 = nil). Add N new nodes with values from the given sequence to the tree so that the tree still remains a search tree. Output the pointer P_2 to the root of the resulting tree. Use the recursive algorithm described in Tree61 to add each new node.

Tree64. An integer N (> 0), a sequence of N integers and a pointer P_1 to the root of a non-recurrent search tree are given (if the tree is empty then P_1 = nil). Add N new nodes with values from the given sequence to the tree so that the tree still remains a non-recurrent search tree. Output the pointer P_2 to the root of the resulting tree. Use the recursive algorithm described in Tree62 to add each new node.

Tree65. An integer N (> 0) and a sequence of N integers are given. Sort the sequence by creating a search tree (use the recursive algorithm described in Tree61 to add each new node). Output the pointer P_1 to the root of the created tree. Also output elements of the sorted sequence using the inorder tree walk.

Tree66. An integer N (> 0) and a sequence of N integers are given. Sort all different elements of the sequence by creating a non-recurrent search tree (use the recursive algorithm described in Tree62 to add each new node). Output the pointer P_1 to the root of the created tree. Also output elements of the sorted sequence using the inorder tree walk.

Tree67. Two pointers are given: P_1 to the root of a nonempty search tree and P_2 to one of its nodes with no more than one child. Remove the node P_2 from the tree so that the tree still remains a search tree (if the node P_2 has a child then link the child with the parent of the node P_2). If the resulting tree is not empty then output the pointer P_3 to its root, otherwise output nil.

Tree68. Two pointers are given: P_1 to the root of a nonempty search tree and P_2 to one of its nodes with two children. Remove the node P_2 from the tree so that the tree still remains a search tree. Use the following algorithm: find the node P with the maximal value in the left subtree of the node P_2, then assign its value to the node P_2, and finally remove the node P as in Tree67 (because the node P should have no more than one child).

Tree69. Two pointers are given: P_1 to the root of a nonempty search tree and P_2 to one of its nodes with two children. Remove the node P_2 from the tree so that the tree still remains a search tree. Use the following algorithm: find the node P with the minimal value in the right subtree of the node P_2, then assign its value to the node P_2, and finally remove the node P as in Tree67 (because the node P should have no more than one child).

Tree70. A pointer P_1 to a node of a doubly linked search tree is given. Remove the node P_1 from the tree so that the tree still remains a doubly linked search tree. If the resulting tree is not empty then output the pointer P_2 to its root, otherwise output nil. If the node P_1 has two children then use the algorithm described in Tree68 for its removing.

Tree71. A pointer P_1 to a node of a doubly linked search tree is given. Remove the node P_1 from the tree so that the tree still remains a doubly linked search tree. If the resulting tree is not empty then output the pointer P_2 to its root, otherwise output nil. If the node P_1 has two children then use the algorithm described in Tree69 for its removing.

21.6. Binary parse trees

Tree72. A string *S* that represents a nonempty tree is given. The tree representation is defined as follows (blank characters are not used):

```
<tree>   : := <empty string> |
             <node>(<left subtree>,<right subtree>)
<node> : := <digit>
```

For example, "4(2(,),6(,7(3(,),)))". Create a tree represented by the string *S* and output the pointer to its root.

Tree73. A pointer P_1 to the root of a nonempty tree is given. Output the string that describes the tree using the representation specified in Tree72.

Tree74. A string *S* that represents a nonempty tree is given. The tree representation is defined as follows (blank characters are not used, the node representation depends on presence of subtrees of the node):

```
<tree>   ::= <node> |
             <node>(<left subtree>,<right subtree>) |
             <node>(<left subtree>) |
             <node>(,<right subtree>)
<node> ::= <digit>
```

For example, “4(2,6(,7(3)))”. Create a tree represented by the string S and output the pointer to its root.

Tree75. A pointer P_1 to the root of a nonempty tree is given. Output the string that describes the tree using the representation specified in Tree74.

Tree76. A string S that represents a correct expression of integer type is given. The expression is defined as follows (blank characters are not used):

```
<expression> ::= <digit> |
                 (<expression><operator><expression>)
<operator>   ::= + | − | *
```

Create a tree that represents the given expression (*a parse tree*): each internal node corresponds to one of the arithmetic operators and equals −1 for addition, −2 for subtraction, and −3 for multiplication; a left subtree of a node-operator represents its left operand and a right subtree represents its right operand; leaf nodes represent digits. Output the pointer to the root of the created tree.

Tree77. A string S that represents a correct expression of integer type is given. The expression is defined as follows (*the parenthesis-free preorder format*):

```
<expression> ::= <digit> |
                 <operator> <expression> <expression>
<operator>   ::= + | − | *
```

Expressions are separated from each other and from the operators by one blank character. Create a parse tree for the given expression and output the pointer to its root. See the description of parse tree structure in Tree76; a left subtree of the node-operator corresponds to its first operand and a right subtree corresponds to its second operand.

Tree78. A string S that represents a correct expression of integer type is given. The expression is defined as follows (*the parenthesis-free postorder format*):

```
<expression> ::= <digit> |
                 <expression> <expression> <operator>
<operator>   ::= + | − | *
```

Expressions are separated from each other and from the operators by one blank character. Create a parse tree for the given expression and output the pointer to its root. See the description of parse tree structure in Tree76; a left subtree of the node-operator corresponds to its first operand and a right subtree corresponds to its second operand.

Tree79. A pointer P_1 to the root of a nonempty parse tree is given (see the description of parse tree structure in Tree76). Output the value of expression that corresponds to the given tree.

Tree80. A pointer P_1 to the root of a nonempty parse tree is given (see the description of parse tree structure in Tree76). Output the string representation of expression that corresponds to the given tree. Use the expression format specified in the same task:

```
<expression> ::= <digit> |
                 (<expression><operator><expression>)
<operator>   ::= + | − | *
```

Tree81. A pointer P_1 to the root of a nonempty parse tree is given. Output the string representation of expression that corresponds to the given tree. Use the parenthesis-free preorder format (see Tree77).

Tree82. A pointer P_1 to the root of a nonempty parse tree is given. Output the string representation of expression that corresponds to the given tree. Use the parenthesis-free postorder format (see Tree78).

Tree83. A string S that represents a correct expression of integer type is given. The expression is defined as follows (blank characters are not used, functions M and m return their maximal and minimal argument respectively):

```
<expression> ::= <digit> | M(<expression> , <expression>) |
                 m(<expression> , <expression>)
```

Create a parse tree for the given expression: each internal node corresponds to one of two available functions and equals −1 for the function M and −2 for the function m; a left subtree of a node-function represents its first argument and a right subtree represents its second argument; leaf nodes represent digits. Output the pointer to the root of the created tree.

Tree84. A pointer P_1 to the root of a nonempty parse tree is given (see the description of parse tree structure in Tree83). Output the value of expression that corresponds to the given tree.

Tree85. A pointer P_1 to the root of a nonempty parse tree is given (see the description of parse tree structure in Tree83). Output the string representation of expression that corresponds to the given tree. Use the expression format specified in the same task:

```
<expression> ::= <digit> | M(<expression> , <expression>) |
                 m(<expression> , <expression>)
```

21.7. General trees

Tree86. In a *general tree* a node may have more than two child nodes arranged in fixed order (from left to right). A general tree may be represented by linked records of the TNode type as follows: the *Left* field of any node points to its leftmost child whereas the *Right* field points to the nearest right sibling of this node. The tree root has no siblings, therefore its *Right* field always equals nil. A pointer P_1 to the root of a nonempty binary tree is given. Create a general tree that corresponds to the given binary tree and output the pointer P_2 to the root of the created general tree.

Tree87. A pointer P_1 to the root of a nonempty general tree is given. Each node has no more than two child nodes. Create a binary tree corresponding to the given general tree and output the pointer P_2 to the root of the created binary tree. First child of any node of general tree should be the left child of the correspondent node of binary tree.

Tree88. A pointer P_1 to the root of a nonempty general tree is given. Output *the depth* of the tree (that is, the maximal level of tree nodes). All siblings are assumed to be on the same level; the level of the root equals 0.

Tree89. A pointer P_1 to the root of a nonempty general tree is given. For each tree level (including the zero one) output the amount of its nodes. The tree depth is assumed to be not greater than 10.

Tree90. A pointer P_1 to the root of a nonempty general tree is given. For each tree level (including the zero one) output the sum of values of its nodes. The tree depth is assumed to be not greater than 10.

Tree91. A pointer P_1 to the root of a nonempty general tree and an integer L (≥ 0) are given. Output the values of nodes of the level L and their amount N (nodes must be ordered from left to right). If nodes of the level L are absent then output 0.

Tree92. A pointer P_1 to the root of a nonempty general tree is given. Output the values of all nodes using *inorder tree walk* as follows: output the first (i. e., the most left) subtree using inorder tree walk, then output the root value, then output all other subtrees (from left to right) using inorder tree walk too.

Tree93. A pointer P_1 to the root of a nonempty general tree is given. Output the values of all nodes using *postorder tree walk* as follows: output all subtrees (from left to right) using postorder tree walk, then output the root value.

Tree94. A pointer P_1 to the root of a nonempty general tree and an integer N (≥ 0) are given. Output the amount of nodes that have exactly N child nodes (this amount may be equal to 0).

Tree95. A pointer P_1 to the root of a nonempty general tree is given. Output the pointer P_2 to the first node that has the maximal amount of child nodes. Use inorder tree walk (see Tree92).

Tree96. A pointer P_1 to the root of a nonempty general tree is given. Output the pointer P_2 to the last node that has the maximal sum of values of child nodes. Use postorder tree walk (see Tree93).

Tree97. A pointer P_1 to the root of a nonempty general tree is given. In each set of siblings find the maximal value (that is, the maximal *Data* field) and assign this value to each node of the set.

Tree98. A pointer P_1 to the root of a nonempty general tree is given. In each set of siblings inverse the order of the node values (that is, swap the *Data* fields of the first and the last sibling, then swap the *Data* fields of the second and the last but one sibling, and so on).

Tree99. A string *S* that represents a nonempty general tree is given. The tree representation is defined as follows (blank characters are not used, siblings are ordered from left to right):

```
<tree>               ::= <node> |
                         <node>(<list of subtrees>)
<list of subtrees>   ::= <tree> |
                         <tree>,<list of subtrees>
<node>               ::= <digit>
```

For example, "3(2,7(6,4,5),8(4(2,3),5(1)))". Create a tree represented by the string *S* and output the pointer to its root.

Tree100. A pointer P_1 to the root of a nonempty general tree is given. Output the string that describes the tree using the representation specified in Tree99.

22. Binary trees (based on objects)

All numbers mentioned in tasks of this group are of integer type. All objects are of Node type; this class is defined in Programming Taskbook. In the tasks of this group the Data, Left, Right, and Parent properties of the Node class are used. Therefore one can assume that the Node class contains the following public members:

[C#]

```
// Constructors:
  public Node();
  public Node(int aData);
  public Node(Node aLeft, Node aRight, int aData);
  public Node(Node aLeft, Node aRight, int aData, Node aParent);
// Properties (available to read and to write):
  public int Data;
  public Node Left;
  public Node Right;
  public Node Parent;
// Method that releases resources used by the Node object:
  public void Dispose();
```

[VB.NET]

```
' Constructors:
  Public Sub New()
  Public Sub New(aData As Integer)
  Public Sub New(aLeft As Node, aRight As Node, _
    aData As Integer)
  Public Sub New(aLeft As Node, aRight As Node, _
    aData As Integer, aParent As Node)
```

' Properties (available to read and to write):
```
Public Property Data() As Integer
Public Property Left() As Node
Public Property Right() As Node
Public Property Parent() As Node
```
' Method that releases resources used by the Node object:
```
Public Sub Dispose() Implements IDisposable.Dispose
```
[Java]

// Constructors:
```
Node();
Node(int aData);
Node(Node aLeft, Node aRight, int aData);
Node(Node aLeft, Node aRight, int aData, Node aParent);
```
// Accessors to properties:
```
int getData();
void setData(int value);
Node getLeft();
void setLeft(Node value);
Node getRight();
void setRight(Node value);
Node getParent();
void setParent(Node value);
```
// Method that releases resources used by the Node object:
```
void dispose();
```
[Python]

# Constructors:
```
Node(data = 0)
Node.for_tree(data = 0, left = None, right = None, parent = None)
```
# Properties (available to read and to write):
```
Data
Left
Right
Parent
```
# Method that releases resources used by the Node object:
```
dispose()
```
[Ruby]

# Constructors:
```
Node.new()
Node.new(data)
Node.new(left, right, data)
Node.new(left, right, data, parent)
```
# Properties (available to read and to write):
```
data
left
right
parent
```

# Method that releases resources used by the Node object:

```
dispose()
```

In the most of the tasks only the Data, Left, and Right properties of the Node class are used. The Parent property is required in the tasks devoted to doubly linked trees.

The value of the Data property of a Node object is considered as the *value of the corresponding tree node*.

All these languages use the reference object model; that is, any object variable is a *reference* to the object instance. Therefore, the expression “output the reference to a tree node” means that you should output the value of a corresponding variable of the Node type.

22.1. Analysis

ObjTree1. An object A_1 of Node type is given. The object has public properties named *Data* (of integer type), *Left* and *Right* (of Node type). The *Left* and *Right* properties of the given object (*a tree root*) contain references to the left and right *child nodes* respectively. Output the *Data* properties of the tree root and its left and right children. Also output the references to the left and right child nodes.

ObjTree2. An object A_1 of Node type (a tree root) is given. This object is linked by its *Left* and *Right* properties with objects of the same type (child nodes); they, in turn, are linked with their own child nodes and so on, until objects whose *Left* and *Right* properties are equal to null. Some of the nodes may have one property (*Left* or *Right*) equals null. Output the amount of tree nodes.

ObjTree3. A root A_1 of a nonempty tree and an integer K are given. Output the amount of nodes whose value equals K.

ObjTree4. A root A_1 of a nonempty tree is given. Output the sum of values of all tree nodes.

ObjTree5. A root A_1 of a nonempty tree is given. Output the amount of left child nodes (the tree root should not be counted).

ObjTree6. A root A_1 of a nonempty tree is given. Nodes without children are called *the external nodes* or *the leaf nodes* (*the leaves*). Output the amount of leaf nodes.

ObjTree7. A root A_1 of a nonempty tree is given. Output the sum of values of all tree leaves.

ObjTree8. A root A_1 of a tree is given, the tree contains at least two nodes. Output the amount of tree leaves that are the right child nodes.

ObjTree9. A root A_1 to the root of a nonempty tree is given. The root node is said to be on the *zero level*, its child nodes — on the *first level* and so on. Output the *depth* of the tree (that is, the maximal level of tree nodes). For example, the depth of a tree containing only a root node is equal to 0.

ObjTree10. A root A_1 of a nonempty tree is given. For each tree level (including the zero one) output the amount of its nodes. The tree depth is assumed to be not greater than 10.

ObjTree11. A root A_1 of a nonempty tree is given. For each tree level (including the zero one) output the sum of values of its nodes. The tree depth is assumed to be not greater than 10.

ObjTree12. A root A_1 of a nonempty tree is given. Using the recursive algorithm named *inorder tree walk* output the values of all tree nodes as follows: output the left subtree (using inorder tree walk), then output the root node, then output the right subtree (using inorder tree walk too).

ObjTree13. A root A_1 of a nonempty tree is given. Using the recursive algorithm named *preorder tree walk* output the values of all tree nodes as follows: output the root node, then output the left subtree (using preorder tree walk), then output the right subtree (using preorder tree walk too).

ObjTree14. A root A_1 of a nonempty tree is given. Using the recursive algorithm named *postorder tree walk* output the values of all tree nodes as follows: output the left subtree (using postorder tree walk), then output the right subtree (using postorder tree walk too), then output the root node.

ObjTree15. A root A_1 of a nonempty tree and an integer N (> 0) are given. The value of N is not greater than the amount of tree nodes. Output the values of tree nodes whose order numbers are not greater than N (the tree nodes are numbered from 1 using inorder tree walk — see ObjTree12).

ObjTree16. A root A_1 of a nonempty tree and an integer N (> 0) are given. The value of N is not greater than the amount of tree nodes. Output the values of tree nodes whose order numbers are N or greater (the tree nodes are numbered from 1 using postorder tree walk — see ObjTree14).

ObjTree17. A root A_1 of a nonempty tree and two integers N_1, N_2 ($0 < N_1 < N_2$) are given. The value of N_2 is not greater than the amount of tree nodes. Output the values of tree nodes whose order numbers are in the range N_1 to N_2 (the tree nodes are numbered from 1 using preorder tree walk — see ObjTree13).

ObjTree18. A root A_1 of a nonempty tree and an integer L (≥ 0) are given. Using tree walk of any type (see ObjTree12−ObjTree14) output values of all nodes of the level L. Also output the amount N of these nodes. If the given tree does not contain nodes of level L then output 0.

ObjTree19. A root A_1 of a nonempty tree is given. Output the maximal value of the tree nodes and the amount of nodes with this value.

ObjTree20. A root A_1 to the root of a nonempty tree is given. Output the minimal value of the tree nodes and the amount of leaves with this value (the amount may be equal to 0).

ObjTree21. A root A_1 of a nonempty tree is given. Output the minimal value of its leaves.

ObjTree22. A root A_1 of a tree is given, the tree contains at least two nodes. Output the maximal value of its *internal nodes* (that is, nodes with children).

ObjTree23. A root A_1 of a nonempty tree is given. Using preorder tree walk, find the first tree node with the minimal value and output its reference A_2.

ObjTree24. A root A_1 of a nonempty tree is given. Using inorder tree walk, find the last node with the maximal odd value and output its reference A_2. If the tree does not contain nodes with odd values then output null.

22.2. Creation

ObjTree25. An integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes and assign values of the given sequence to tree nodes in order of their creation. Each node of the tree (except for the root) should be a right child. Output the reference to the tree root.

ObjTree26. An integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes and assign values of the given sequence to tree nodes in order of their creation. Each internal node of the tree should have one child: the root has a left child, which has a right child, which has a left child, and so on. Output the reference to the tree root.

ObjTree27. An integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes and assign values of the given sequence to tree nodes in order of their creation. Each internal node of the tree should have one child: an internal node whose value is an odd number has a left child, otherwise it has a right child. Output the reference to the tree root.

ObjTree28. An even integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes; left child nodes of the tree should be leaves, right child nodes should be internal ones. For each internal node create a left child at first, then create a right one (if it exists). Assign values of the given sequence to tree nodes in order of their creation. Output the reference to the tree root.

ObjTree29. An even integer N (> 0) and a sequence of N integers are given. Create a tree with N nodes. Inner node whose value is an odd number should have a left child leaf, otherwise it should have a right child leaf. For each internal node create a child leaf node at first, and then create a child internal node (if it exists). Assign values of the given sequence to tree nodes in order of their creation. Output the reference to the tree root.

ObjTree30. An integer N (> 0) is given. Create a tree that satisfies the following conditions: the value of root node equals N; if the value of a node is an even number K then this node has only a left child whose value equals $K/2$; if the value of a node equals 1 then this node is a leaf; if the value of a node is another odd number K then this node has a left child whose value equals $K/2$ and has a right child whose value equals $K - K/2$ ("/" denotes the operator of integer division). Output the reference to the tree root.

ObjTree31. Two positive integers L, N ($N > L$) and a sequence of N integers are given. Create a tree of depth L. Use elements of the given sequence as node values; add new nodes using the following algorithm: for each node of the level not greater than L create the node itself, then its left subtree of corresponding depth, and finally its right subtree. If less than N nodes are required to create an L-depth tree then do not use the rest of elements of the given sequence. Output the reference to the tree root.

ObjTree32. An integer N (> 0) and a sequence of N integers are given. Create *a balanced tree* with N nodes (that is, a binary tree which satisfies the following condition: for each tree node the amount of nodes of its left subtree differs at most on 1 from the amount of nodes of its right subtree) and output the reference to the tree root. Use elements of the given sequence as node values; create the tree by means of the following recursive algorithm: create a root node, then repeat the algorithm twice: for creating the left subtree with $N/2$ nodes and for creating the right subtree with $N - 1 - N/2$ nodes ("/" denotes the operator of integer division).

ObjTree33. An integer N (> 0) is given. Create a balanced tree with N nodes and output the reference to the tree root. The value of each node should be equal to its level (for example, the root value is 0, the value of its children is 1, and so on). Create the balanced tree by means of the recursive algorithm described in ObjTree32.

ObjTree34. An root A_1 of a nonempty tree is given. Create a copy of the tree and output the reference A_2 to its root.

22.3. Changing

ObjTree35. A root A_1 of a nonempty tree is given. Double the value of each tree node.

ObjTree36. A root A_1 of a nonempty tree is given. Halve the value of each tree node whose initial value is an even number.

ObjTree37. A root A_1 of a nonempty tree is given. Add 1 to the value of each tree leaf and subtract 1 from the value of each internal node.

ObjTree38. A root A_1 of a nonempty tree is given. For each tree node with two child swap values of its child nodes (that is, swap values of *Data* properties of child nodes).

ObjTree39. A root A_1 of a nonempty tree is given. Swap child nodes of each internal node in the tree (that is, swap values of its *Left* and *Right* property).

ObjTree40. A root A_1 of a nonempty tree is given. Remove all nodes from the tree (except the root) and call the Dispose method for each removed node (assign null to the *Left* and *Right* properties of the root).

ObjTree41. A root A_1 of a nonempty tree is given, the tree contains at least two nodes. Remove all tree leaves and assign null to the *Left* and *Right* properties of their parents. Call the Dispose method for each removed node.

ObjTree42. A root A_1 of a nonempty tree is given. Remove all nodes whose value is less than the root value, together with all their descendants. Call the Dispose method for each removed node.

ObjTree43. A root A_1 of a nonempty tree is given. Apply the following action to each tree node that has two child nodes: if the node value is an even number then remove its right child, otherwise remove its left child. Use preorder tree walk; each node should be removed together with all its descendants. Call the Dispose method for each removed node.

ObjTree44. A root A_1 of a nonempty tree is given. Add two child nodes to each tree leaf; the values of left and right child nodes should be equal to 10 and 11 respectively.

ObjTree45. A root A_1 of a nonempty tree is given. Add one child node to each thee leaf; if the leaf value is an odd number then its child should be a left node, otherwise its child should be a right one. Value of created child node should be equal to value of its parent.

ObjTree46. A root A_1 of a nonempty tree is given. For each tree node with one child add another child node (a leaf). Value of created child node should be equal to doubled value of its parent.

ObjTree47. A root A_1 of a nonempty tree is given. Transform the given tree to a *perfect tree* by adding some new nodes (a perfect tree is a binary tree whose all leaves are at the same level). Do not change the initial depth of the tree; value of all new nodes should be equal to -1.

22.4. Doubly linked binary trees

ObjTree48. A node A_1 of a tree is given. It is an object of Node type that has public properties named *Data* (of integer type), *Left*, *Right*, and *Parent* (of Node type). The *Left* and *Right* properties contain references to the left and right child nodes respectively, the *Parent* property contains reference to the parent node (the *Parent* property of the root node equals null). Output references A_L and A_R to the left and right child of the given node, A_0 to its parent, and A_2 to its *sibling* (siblings are nodes that have the same parent). In some of required nodes are not exist then output null for each absent node.

ObjTree49. A root A_1 of a tree is given. Tree nodes are represented by object of Node type; they are linked by the *Left* and *Right* properties of Node class. Using the *Parent* property of Node class, transform the given tree into *a doubly linked tree* whose each node is connected not only with its child nodes (by the *Left* and *Right* properties) but also with its parent node (by the *Parent* property). The *Parent* property of the root node should be equal to null.

ObjTree50. A reference A_1 to some node of a doubly linked tree is given. Output the reference A_2 to the tree root.

ObjTree51. References P_1, P_2, P_3 to three nodes of a doubly linked tree are given. Output the level of each node (the level of the root equals 0).

ObjTree52. References A_1 and A_2 to two different nodes of a doubly linked tree are given. Output *the degree of relationship* of the node A_1 to the node A_2 (the degree of relationship equals -1 if the node A_2 is not in the chain of ancestors of the node A_1; otherwise it equals $L_1 - L_2$, where L_1 and L_2 are the levels of nodes A_1 and A_2 respectively).

ObjTree53. References A_1 and A_2 to two different nodes of a doubly linked tree are given. Find the nearest mutual ancestor of the nodes A_1 and A_2 and output its reference A_0.

ObjTree54. A reference A_1 to the node of a doubly linked tree is given. Create a copy of the given tree and output a reference A_2 to the root of the created tree.

ObjTree55. A reference A_1 to the non-root node of a doubly linked tree is given. If the node A_1 has a sibling then remove the sibling together with its descendants from the tree and call the Dispose method for each removed node. If the node A_1 has no sibling then create it and all its descendants as a copy of a the subtree with the root A_1. Output the reference A_0 to the parent of A_1.

ObjTree56. Two positive integers L, N ($N > L$) and a sequence of N integers are given. Create a doubly linked tree of depth L. Use elements of the given sequence as node values; add new nodes using the following algorithm: for each node of the level not greater than L create the node itself, then its left subtree of corresponding depth, and finally its right subtree. If less than N nodes are required to create an L depth tree then do not use the rest of elements of the given sequence. Output the reference to the tree root.

22.5. Binary search trees

ObjTree57. A root A_1 of a nonempty tree is given. It the tree is *a search tree*, that is, values of its nodes form a non-decreasing sequence in inorder tree walk, then output null; otherwise output the reference to the first node (in inorder tree walk) that breaks the search-tree property.

ObjTree58. A root A_1 of a nonempty tree is given. It the tree is *a non-recurrent search tree*, that is, values of its nodes form an increasing sequence in inorder tree walk, then output null; otherwise output the reference to the first node (in inorder tree walk) that breaks the search-tree property.

ObjTree59. A root A_1 of a nonempty non-recurrent search tree and an integer K are given. If the tree contains a node whose value equals K then output the reference A_2 to this node, otherwise output null. Also output the amount N of tree nodes that were checked during the search.

ObjTree60. A root A_1 of a nonempty search tree and an integer K are given. Output the amount C of tree nodes whose value equals K. Also output the amount N of tree nodes that were checked during the search.

ObjTree61. A root A_1 of a search tree and an integer K are given (if the tree is empty then P_1 = null). Add a new node with the value K to the tree so that the tree still remains a search tree. Output the reference A_2 to the root of the resulting tree. Use the following recursive algorithm for a tree with the root A: if A = null then create a leaf with the value K and assign the leaf reference to the object A; if the tree root exists then repeat the algorithm for the left subtree in case K is less than the root value or for the right subtree otherwise.

ObjTree62. A root A_1 of a non-recurrent search tree and an integer K are given (if the tree is empty then A_1 = null). Add a new node with the value K to the tree so that the tree still remains a non-recurrent search tree. Do not change the given tree if it already contains a node with the value K. Output the reference A_2 to the root of the resulting tree. Use the following recursive algorithm for a tree with the root A: if A = null then create a leaf with the value K and assign the leaf reference to the object A; if the tree root exists then repeat the algorithm for the left subtree in case K is less than the root value or for the right subtree in case K is greater than the root value.

ObjTree63. An integer N (> 0), a sequence of N integers and the root A_1 of a search tree are given (if the tree is empty then A_1 = null). Add N new nodes with values from the given sequence to the tree so that the tree still remains a search tree. Output the reference A_2 to the root of the resulting tree. Use the recursive algorithm described in ObjTree61 to add each new node.

ObjTree64. An integer N (> 0), a sequence of N integers and the root A_1 of a non-recurrent search tree are given (if the tree is empty then A_1 = null). Add N new nodes with values from the given sequence to the tree so that the tree still remains a non-recurrent search tree. Output the reference A_2 to the root of the resulting tree. Use the recursive algorithm described in ObjTree62 to add each new node.

ObjTree65. An integer N (> 0) and a sequence of N integers are given. Sort the sequence by creating a search tree (use the recursive algorithm described in ObjTree61 to add each new node). Output the reference A_1 to the root of the created tree. Also output elements of the sorted sequence using the inorder tree walk.

ObjTree66. An integer N (> 0) and a sequence of N integers are given. Sort all different elements of the sequence by creating a non-recurrent search tree (use the recursive algorithm described in ObjTree62 to add each new node). Output the reference A_1 to the root of the created tree. Also output elements of the sorted sequence using the inorder tree walk.

ObjTree67. Two references are given: A_1 to the root of a nonempty search tree and A_2 to one of its nodes with no more than one child. Remove the node A_2 from the tree so that the tree still remains a search tree (if the node A_2 has a child then link the child with the parent of the node A_2). If the resulting tree is not empty then output the reference A_3 to its root, otherwise output null.

ObjTree68. Two references are given: A_1 to the root of a nonempty search tree and A_2 to one of its nodes with two children. Remove the node A_2 from the tree so that the tree still remains a search tree. Use the following algorithm: find the node A with the maximal value in the left subtree of the node A_2, then assign its value to the node A_2, and finally remove the node A as in ObjTree67 (because the node A should have no more than one child).

ObjTree69. Two references are given: A_1 to the root of a nonempty search tree and A_2 to one of its nodes with two children. Remove the node A_2 from the tree so that the tree still remains a search tree. Use the following algorithm: find the node A with the minimal value in the right subtree of the node A_2, then assign its value to the node A_2, and finally remove the node A as in ObjTree67 (because the node A should have no more than one child).

ObjTree70. A reference A_1 to a node of a doubly linked search tree is given. Remove the node A_1 from the tree so that the tree still remains a doubly linked search tree. If the resulting tree is not empty then output the reference A_2 to its root, otherwise output null. If the node A_1 has two children then use the algorithm described in ObjTree68 for its removing.

ObjTree71. A reference A1 to a node of a doubly linked search tree is given. Remove the node A1 from the tree so that the tree still remains a doubly linked search tree. If the resulting tree is not empty then output the reference A2 to its root, otherwise output null. If the node A1 has two children then use the algorithm described in ObjTree69 for its removing.

22.6. Binary parse trees

ObjTree72. A string S that represents a nonempty tree is given. The tree representation is defined as follows (blank characters are not used):

```
<tree>   ::= <empty string> |
             <node>(<left subtree>,<right subtree>)
<node>   ::= <digit>
```

For example, “4(2(,),6(,7(3(,),)))”. Create a tree represented by the string S and output the reference to its root.

ObjTree73. A root A_1 of a nonempty tree is given. Output the string that describes the tree using the representation specified in ObjTree72.

ObjTree74. A string S that represents a nonempty tree is given. The tree representation is defined as follows (blank characters are not used, the node representation depends on presence of subtrees of the node):

```
<tree>   ::= <node> |
             <node>(<left subtree>,<right subtree>) |
             <node>(<left subtree>) |
             <node>(,<right subtree>)
<node>   ::= <digit>
```

For example, “4(2,6(,7(3)))”. Create a tree represented by the string S and output the reference to its root.

ObjTree75. A root A_1 of a nonempty tree is given. Output the string that describes the tree using the representation specified in ObjTree74.

ObjTree76. A string S that represents a correct expression of integer type is given. The expression is defined as follows (blank characters are not used):

```
<expression> ::= <digit> |
                 (<expression><operator><expression>)
<operator>   ::= + | − | *
```

Create a tree that represents the given expression (*a parse tree*): each internal node corresponds to one of the arithmetic operators and equals −1 for addition, −2 for subtraction, and −3 for multiplication; a left subtree of a node-operator represents its left operand and a right subtree represents its right operand; leaf nodes represent digits. Output the reference to the root of the created tree.

ObjTree77. A string S that represents a correct expression of integer type is given. The expression is defined as follows (*the parenthesis-free preorder format*):

```
<expression> ::= <digit> |
                 <operator> <expression> <expression>
<operator>   ::= + | − | *
```

Expressions are separated from each other and from the operators by one blank character. Create a parse tree for the given expression and output the reference to its root. See the description of parse tree structure in ObjTree76; a left subtree of the node-operator corresponds to its first operand and a right subtree corresponds to its second operand.

ObjTree78. A string S that represents a correct expression of integer type is given. The expression is defined as follows (*the parenthesis-free postorder format*):

```
<expression> ::= <digit> |
                 <expression> <expression> <operator>
<operator>   ::= + | − | *
```

Expressions are separated from each other and from the operators by one blank character. Create a parse tree for the given expression and output the reference to its root. See the description of parse tree structure in ObjTree76; a left subtree of the node-operator corresponds to its first operand and a right subtree corresponds to its second operand.

ObjTree79. A root A_1 of a nonempty parse tree is given (see the description of parse tree structure in ObjTree76). Output the value of expression that corresponds to the given tree.

ObjTree80. A root A_1 of a nonempty parse tree is given (see the description of parse tree structure in ObjTree76). Output the string representation of expression that corresponds to the given tree. Use the expression format specified in the same task:

<expression> : := <digit> |
(<expression><operator><expression>)
<operator> : := + | − | *

ObjTree81. A root A_1 of a nonempty parse tree is given. Output the string representation of expression that corresponds to the given tree. Use the parenthesis-free preorder format (see ObjTree77).

ObjTree82. A root A_1 of a nonempty parse tree is given. Output the string representation of expression that corresponds to the given tree. Use the parenthesis-free postorder format (see ObjTree78).

ObjTree83. A string *S* that represents a correct expression of integer type is given. The expression is defined as follows (blank characters are not used, functions M and m return their maximal and minimal argument respectively):

<expression> : := <digit> | M(<expression> , <expression>) |
m(<expression> , <expression>)

Create a parse tree for the given expression: each internal node corresponds to one of two available functions and equals −1 for the function M and −2 for the function m; a left subtree of a node-function represents its first argument and a right subtree represents its second argument; leaf nodes represent digits. Output the reference to the root of the created tree.

ObjTree84. A root A_1 of a nonempty parse tree is given (see the description of parse tree structure in ObjTree83). Output the value of expression that corresponds to the given tree.

ObjTree85. A root A_1 of a nonempty parse tree is given (see the description of parse tree structure in ObjTree83). Output the string representation of expression that corresponds to the given tree. Use the expression format specified in the same task:

<expression> : := <digit> | M(<expression> , <expression>) |
m(<expression> , <expression>)

22.7. General trees

ObjTree86. In a *general tree* a node may have more than two child nodes arranged in fixed order (from left to right). A general tree may be represented by linked records of the Node class as follows: the *Left* property of any node refers to its leftmost child whereas the *Right* property refers to the nearest right sibling of this node. The tree root has no siblings, therefore its *Right* property always equals null. A root A_1 of a nonempty binary tree is given. Create a general tree that corresponds to the given binary tree and output the reference A_2 to the root of the created general tree.

ObjTree87. A root A_1 of a nonempty general tree is given. Each node has no more than two child nodes. Create a binary tree corresponding to the given general tree and output the reference A_2 to the root of the created binary tree. First child of any node of general tree should be the left child of the correspondent node of binary tree.

ObjTree88. A root A_1 of a nonempty general tree is given. Output *the depth* of the tree (that is, the maximal level of tree nodes). All siblings are assumed to be on the same level; the level of the root equals 0.

ObjTree89. A root A_1 of a nonempty general tree is given. For each tree level (including the zero one) output the amount of its nodes. The tree depth is assumed to be not greater than 10.

ObjTree90. A root A_1 of a nonempty general tree is given. For each tree level (including the zero one) output the sum of values of its nodes. The tree depth is assumed to be not greater than 10.

ObjTree91. A root A_1 of a nonempty general tree and a integer L (≥ 0) are given. Output the values of nodes of the level L and their amount N (nodes must be ordered from left to right). If nodes of the level L are absent then output 0.

ObjTree92. A root A_1 of a nonempty general tree is given. Output the values of all nodes using *inorder tree walk* as follows: output the first (i. e., the most left) subtree using inorder tree walk, then output the root value, then output all other subtrees (from left to right) using inorder tree walk too.

ObjTree93. A root A_1 of a nonempty general tree is given. Output the values of all nodes using *postorder tree walk* as follows: output all subtrees (from left to right) using postorder tree walk, then output the root value.

ObjTree94. A root A_1 of a nonempty general tree and an integer N (≥ 0) are given. Output the amount of nodes that have exactly N child nodes (this amount may be equal to 0).

ObjTree95. A root A_1 of a nonempty general tree is given. Output the reference A_2 to the first node that has the maximal amount of child nodes. Use inorder tree walk (see ObjTree92).

ObjTree96. A root A_1 of a nonempty general tree is given. Output the reference A_2 to the last node that has the maximal sum of values of child nodes. Use postorder tree walk (see ObjTree93).

ObjTree97. A root A_1 of a nonempty general tree is given. In each set of siblings find the maximal value (that is, the maximal *Data* property) and assign this value to each node of the set.

ObjTree98. A root A_1 of a nonempty general tree is given. In each set of siblings inverse the order of the node values (that is, swap the *Data* properties of the first and the last sibling, then swap the *Data* properties of the second and the last but one sibling, and so on).

ObjTree99. A string S that represents a nonempty general tree is given. The tree representation is defined as follows (blank characters are not used, siblings are ordered from left to right):

```
<tree>              : := <node> |
                         <node>(<list of subtrees>)
<list of subtrees> : := <tree> |
                         <tree>,<list of subtrees>
<node>              : := <digit>
```

For example, “3(2,7(6,4,5),8(4(2,3),5(1)))”. Create a tree represented by the string *S* and output the reference to its root.

ObjTree100. A root A_1 of a nonempty general tree is given. Output the string that describes the tree using the representation specified in ObjTree99.

Contents