BABEŞ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Bachelor Degree Written Exam
Computer Science – English

VARIANT 1

REMARKS

- All subjects are compulsory and full solutions are requested.
- The minimum passing grade is 5.00.
- The working time is 3 hours.

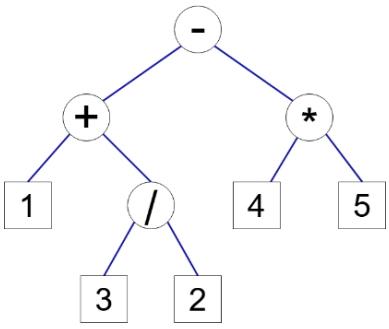
SUBJECT Algorithms and Programming

- **The programming language that is used must be indicated.**
- **The implementations are not required to deallocate dynamically allocated memory areas.**
- **Lack of appropriate programming style (suggestive variable names, indentation of code, comments, if necessary, readability of code) will result in a 10% deduction from the subject's score.**
- **Do not add additional attributes or methods, other than those mentioned in the statement, except for constructors and destructors, if applicable. Do not change the visibility of attributes specified in the statement.**
- **Do not use sorted containers, predefined sort or search operations.**
- **Existing libraries (from C++, Java, C#, Python) may be used for data types.**

An arithmetic expression can be represented as a binary tree as follows. If the expression consists of a single operand, the associated binary tree has a single node (the root node), which contains the respective operand as data. If the expression is of the form **E1** *op* **E2**, where *op* is a binary operator and **E1** and **E2** are arithmetic expressions, it is associated with a binary tree whose root node is labeled with the operator *op*, the left subtree is the binary tree associated with the expression **E1**, and the right subtree is the binary tree associated with the expression **E2**.

By traversing the binary tree associated with an arithmetic expression in *postorder*, its *postfixed form* is obtained. ***For example***, for the expression (1+3/2)-4*5

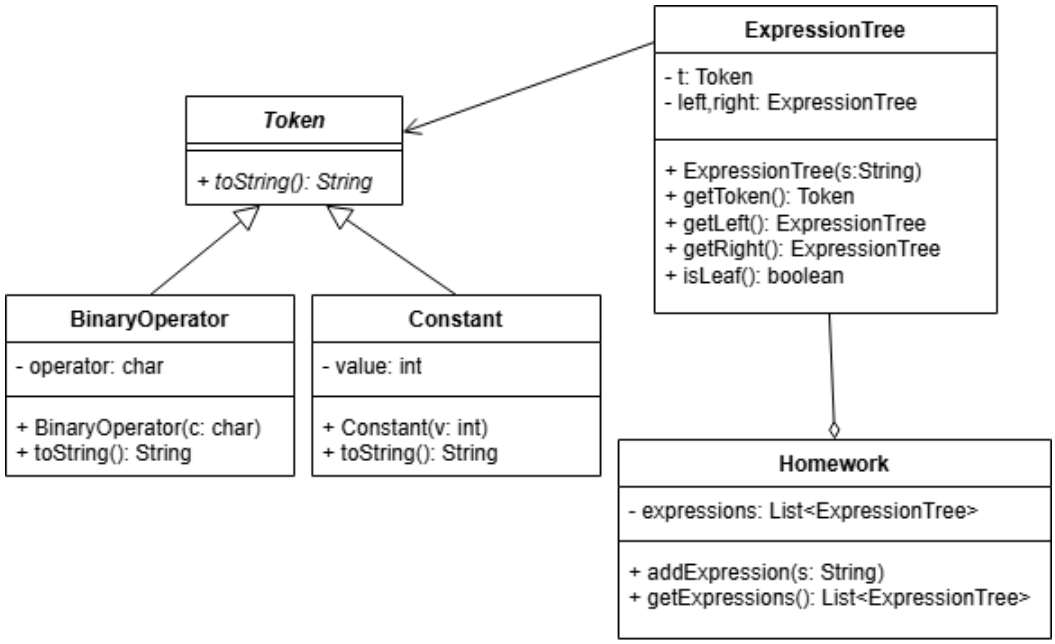
- the associated binary tree is



- the *postfix* form is '132/+45*-'.

1. **(3.5 points)** Write a function in one of the programming languages **Python, C++, Java** or **C#**, which receives as a parameter a string *s* having at most 100 characters, representing the **correct** *postfix* form of an arithmetic expression containing the binary operators '+' (addition), '-' (subtraction), '*' (multiplication), '/' (division), and as operands non-zero single-digit natural numbers. The function will return the value of the arithmetic expression. The algorithm must have time complexity $\theta(n)$, where n is the length of the string *s*. Solutions that do not meet the required complexity will be awarded partial score. **Example**: for the string '132/+45*-' the value **-17.5** will be returned. Auxiliary functions may be implemented.
2. **(1 point)** Indicate the *best-case*, *average-case*, and *worst-case* time complexity for the function from item **1**. Justify your answer.

3. **(4.5 points)** Consider the following UML diagram, which contains the classes **ExpressionTree** (represents the binary tree associated with a correct arithmetic expression), ***Token***, **BinaryOperator, Constant** and **Homework**.



- The class **ExpressionTree** represents the binary tree associated with an arithmetic expression whose operands are constants (non-zero natural numbers) and the operators are binary (+, -, *, /).
  - The constructor of the class **ExpressionTree** takes as a parameter a non-empty string *s* representing a correct arithmetic expression containing parentheses, binary operators, and constants as operands, and constructs the binary tree associated with it. *Implementation of this constructor is not required.*
  - The **isLeaf**() method returns true if the tree consists of a single node (it is associated with an arithmetic expression containing a single operand) and false otherwise.
  - The **getToken()** method returns the data stored in the root of the binary tree **A** associated with a non-empty expression, and the **getLeft()** and **getRight()** methods return the left subtree and the right subtree of tree **A**, respectively.
- The abstract class ***Token*** represents the data (binary operator or constant operand) that can be stored in a node of the tree. The class has an abstract method ***toString*()** that returns a string associated with the token: if the token is a binary operator, the character representing the operator is returned (e.g. '+'); if the token is a constant, the value of the constant converted into a string is returned (e.g. '123').
- The class **Homework** stores a list *expressions* of arithmetic expressions received as homework by a student. The **addExpression(s: String)** method in class **Homework** receives as parameter a string *s* representing a correct arithmetic expression and adds the binary tree constructed from *s* to the end of the list *expressions*. The **getExpressions()** method returns the list of binary trees associated with the arithmetic expressions received as homework.

Write a program that implements the following requirements, using one of the **C++, Java**, or **C#** programming languages:

a) Declare all classes, attributes, and methods as per the diagram above. Implement only the following methods:

a1) Constructor of the **BinaryOperator** class. The *operator* attribute must represent a binary arithmetic operator (+, -, *, /). The constructor must enforce this constraint.
a2) The method **isLeaf**() in class **ExpressionTree**.
a3) The method **addExpression**(**s***:* **String**) in class **Homework.**

b) Define a function that receives as parameter an object *e* of type **ExpressionTree** and returns the string representing the postfix form of the arithmetic expression associated with the tree stored in *e*.

c) Create an object *h* of type **Homework** in which the following arithmetic expressions to be evaluated are added: '(5-3)+(2*6)', '5-(3*4+5)/(6-2)' and '3*(5/(3+2)-4)+6'. For each arithmetic expression stored in *h*, display its postfix form, using the function from **b)**.

SUBJECT Databases **VARIANT 1**

Problem 1. (4 points)

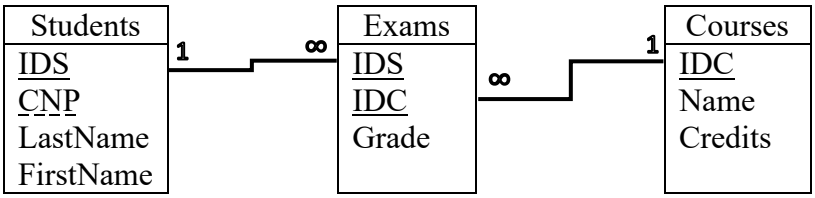
A towel manufacturing company stores information about customers, towels, and customer reviews related to towels in a relational database:

- A customer has an ID, a name, an email address (unique for each customer) and date of birth.
- A towel has an ID, a name, a description, a model, length and width.
- A material has an ID, a name, and a description. A material can be used to make multiple towels, and a towel can be made from multiple materials. Each material can be associated with each towel at most once. For each pair of towel and material (for example, the pair made of towel p1 and material m1), the percentage value of the material in the towel composition will also be stored.
- A customer can rate multiple towels, and a towel can receive ratings from multiple customers. Each customer can rate each towel at most once.

Create a relational BCNF schema for the database, rigorously highlighting the primary keys, candidate keys, and foreign keys. Create the schema in one of the ways indicated in the example below:

* example for tables Students, Courses, and Exams:

V1. Diagram with tables, primary keys underlined with a solid line, candidate keys underlined with a dashed line, relationships drawn directly between foreign keys and the corresponding primary / candidate keys (for instance, the relationship drawn between column IDS in Exams and column IDS in Students).



V2.
Students[IDS, CNP, LastName, FirstName]
Courses[IDC, Name, Credits]
Exams[IDS, IDC, Grade]

Primary keys are underlined with a solid line, and candidate keys are underlined with a dashed line.
{IDS} in Exams is a foreign key referencing {IDS} in Students. {IDC} in Exams is a foreign key referencing {IDC} in Courses.

Problem 2. (5 points)

Consider the following relations from a database about a company that offers board games for rent:

Categories(CategoryID, Name)
BoardGames(BoardGameID, Name, *CategoryID*, ReleaseYear, MinNoOfPlayers, MaxNoOfPlayers, PricePerDay)
Persons(PersonID, Name, PhoneNumber)
Rentals(RentalID, *BoardGameID*, *PersonID*, RentalDate, NoOfDays, PaymentAmount)

Primary keys are underlined. Foreign keys are written in italics and have the same name as the columns they reference.

a. Write an SQL query that returns the ID and the name of board games that can be played individually (see the MinNoOfPlayers attribute), were rented in 2024 at least once, and have a release year equal to the most recent release year of all board games that meet the mentioned conditions (see the ReleaseYear attribute).

b. Consider the following instances for relations BoardGames, Categories, Persons and Rentals:

Persons:

PersonID	Name	PhoneNumber
1	P1	1111111111
2	P2	2222222222
3	P3	3333333333

Categories:

CategoryID	Name
1	strategy game
2	family game
3	cooperation game

BoardGames:

BoardGameID	Name	CategoryID	Release Year	MinNoOf Players	MaxNoOf Players	PricePerDay
1	JS1	1	2020	3	6	15
2	JS2	2	2016	2	4	5
3	JS3	1	2020	2	5	12
4	JS4	3	2024	3	10	10

Rentals:

RentalID	BoardGameID	PersonID	RentalDate	NoOfDays	PaymentAmount
1	1	1	15.05.2025	3	45
2	2	1	18.05.2025	5	25
3	3	2	10.05.2025	2	24
4	3	2	02.06.2025	3	36
5	1	3	25.05.2025	1	15
6	2	3	03.06.2025	4	20
7	3	3	10.06.2025	2	24
8	4	2	20.06.2025	3	30

b1. Write the result of evaluating the query below on the given instances. Provide only the values of the tuple(s) and the names of the columns in the result without describing all the steps of evaluating the query.

```
SELECT P.PersonID, P.Name, MIN(PaymentAmount) MinSum
FROM Persons P
    INNER JOIN Rentals R ON P.PersonID = R.PersonID
    INNER JOIN BoardGames BG ON R.BoardGameID = BG.BoardGameID
    INNER JOIN Categories C ON BG.CategoryID = C.CategoryID
WHERE C.Name = 'strategy game'
GROUP BY P.PersonID, P.Name
HAVING COUNT(DISTINCT R.BoardGameID) =
    (SELECT COUNT(*)
     FROM BoardGames BG2
          INNER JOIN Categories C2 ON BG2.CategoryID = C2.CategoryID
     WHERE C2.Name = 'strategy game'
     )
```

b2. Explain whether the following functional dependencies are satisfied or not by the data in the Rentals instance:

- {RentalID} → {BoardGameID}
- {PersonID} → {RentalDate}.

VARIANT 1

SUBJECT Operating Systems

Problem 1 (5 points). Answer the following questions about the execution of the program below, assuming that all necessary includes are present.

```
1  int main() {
2      int k;
3      if(fork() < 0) {
4          printf("abc\n");
5      } else if(fork() == 0) {
6          return 1;
7      } else {
8          return 2;
9      }
10     while((k = wait(NULL)) > 0) {
11         printf("%d\n", k);
12     }
13     return 3;
14 }
```

a) Explain in detail the functioning of line 3.

b) Explain in detail the functioning of lines 10-12.

c) What will be displayed in the console if both `fork` calls fail? Justify your answer.

d) What will be displayed in the console if both `fork` calls execute successfully? Justify your answer.

e) What will the initial process display in the console if line 8 is removed and both `fork` calls execute successfully? Justify your answer.

Problem 2 (4 points). Answer the following questions about the execution of the `./a.sh` script below.

```
1  #!/bin/bash
2
3  N=$1
4  D=$2
5  shift 2
6
7  for A in $*; do
8      if [ $N -gt 0 ]; then
9          mkdir -p $D/$A
10         $0 `expr $N - 1` $D/$A $*
11     else
12         echo abc > $D/$A
13     fi
14 done
```

a) Explain in detail the functioning of line 10.

b) What values will variable `A` take when executing the command below?

```
./a.sh 1 a b c d
```

c) Considering that directory `x` does not exist initially, what will be displayed in the console after executing the commands below?

```
./a.sh 1 x a b
find x -type f | sort
```

d) Considering that directory `y` does not exist initially, what will be displayed in the console after executing the commands below?

```
./a.sh 2 y a b
cat `find y -type f` | wc -l
```

BAREM INFORMATICĂ
VARIANTA 1

Subiect Algoritmică și Programare

Oficiu – **1p**

Cerința **1**. **– 3.5p**
- signatura – **0.2p**
- implementare având complexitate timp $\theta(n)$ – **3.2p**
  * soluție având complexitate timp $O(n^2)$ – **2**p
- returnare rezultat – **0.1p**

Cerința **2**. **– 1p**
- caz favorabil **0.4p** din care
  complexitate – 0.2
  justificare – 0.2
- caz mediu **0.4p** din care
  complexitate – 0.2
  justificare – 0.2
- caz defavorabil **0.2p** din care
  complexitate – 0.1
  justificare – 0.1

Cerința **3.a) – 2.25p**

Definirea clasei Token – **0.2p** din care
- clasă abstractă – 0.1
- metoda **toString** – 0.1

Definirea clasei BinaryOperator– **0.65p** din care
- relația de moștenire – 0.15
- atribut – 0.1
- **constructor (a1)** – 0.3
- metoda **toString** – 0.1

Definirea clasei Constant– **0.45p** din care
- relația de moștenire – 0.15
- atribut – 0.1
- constructor – 0.1
- metoda **toString** – 0.1

Definirea clasei ExpressionTree – **0.55p** din care
- atribute– 0.1
- constructor – 0.1
- metode **getToken, getLeft, getRight** – 0.15
- metoda **isLeaf (a2) –** 0.2

Definirea clasei Homework – **0.4p** din care
- atribut – 0.1
- metoda **getExpressions** – 0.1p
- metoda **addExpression (a3)** – 0.2p

Funcția **3.b)** – **1.75p**
- signatura – **0.1p**
- implementare parcurgere postordine – **1.55p**
- returnare rezultat – **0.1p**

Funcția principală **3.c)** – **0.5p**
- construire obiect **h** – 0.05p
- adăugare expresii aritmetice în **h** - 0.25p
- parcurgere listă expresii memorate în **h** și afișare formă postfixată – 0.2p din care
  apel funcție b) pentru expresie– 0.1p
  afișare formă postfixată expresie - 0.1 p

BAREM INFORMATICĂ
VARIANTA 1

Subiect Baze de date

Oficiu – **1p**

Problema 1. Punctaj - 4p

- relații cu atribute corecte, chei primare, chei candidat: **3p**
- legături modelate corect (chei externe): **1p**

Problema 2. Punctaj - 5p

- **a** - rezolvarea completă a interogării: **2.5p**

- **b1** - rezultat evaluare interogare:

CodPersoana	Nume	MinSum
3	P3	15

  - coloane – **0.5p**
  - valori tuplu – **1p**

- **b2 -** {CodInchiriere} → {CodJocDeSocietate} este satisfăcută – **0.25p**; **0.25p** explicație
  - {CodPersoana} → {DataInchirierii} nu este satisfăcută – **0.25p**; **0.25p** explicație

Notă: La specializările Informatică engleză și Informatică maghiară se iau în considerare versiunile traduse în limbile corespunzătoare.

VARIANTA 1

SUBIECT Sisteme de operare

Barem:

1p – oficiu

Problema 1 (5p)

1p – a) Se rulează `fork` şi dacă eşuează condiţia va fi adevărată. Dacă `fork` nu eşuează, se creează un proces fiu şi condiţia va fi evaluată ca falsă şi de procesul părinte şi de procesul fiu.

1p – b) Câtă vreme `wait` returnează succes, se tipăreşte PID-ul procesului fiu care tocmai s-a încheiat.

1p – c) Se va afişa `abc` pentru că procesul iniţial va intra în primul if, dar nu va intra în `while` pentru că neavând procese fiu, `wait` va returna eroare.

1p – d) Nu se va afişa nimic, pentru că procesul iniţial şi primul proces fiu se încheie la linia 8, al doilea proces fiu şi procesul nepot se incheie la linia 6

1p – e) Afişează PID-urile celor două procese fiu pe care le creează la liniile 3 şi 5.

Problema 2 (4p)

1p – a) Scriptul Shell se relansează în execuţie, cu primul argument decrementat, al doilea argument directorul tocmai creat şi restul argumentelor identice cu cele rămase după shift.

1p – b) Va lua valorile `b`, `c` şi `d`

1p – c) Se va afişa: `x/a/a  x/a/b  x/b/a  x/b/b`

1p – d) Se va afişa cifra 8