BABEȘ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Bachelor Degree Written Exam, September 3, 2024
Computer Science – English

VARIANT 2

REMARKS

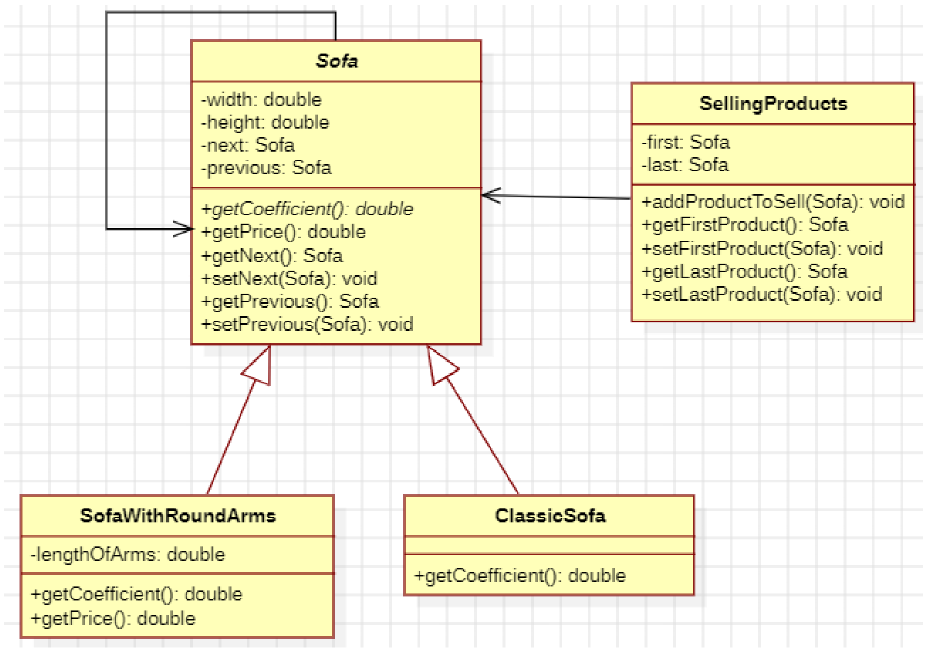
- All subjects are compulsory and full solutions are requested.
- The minimum passing grade is 5.00.
- The working time is 3 hours.

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#) may be used for **data types**.*

a) The following UML diagram is given, which contains the classes ***Sofa***, **SofaWithRoundArms**, **ClassicSofa**, **SellingProducts** (stores sofas for sale). Constructors are not indicated on the diagram.



- The attributes *width* and *height* in class ***Sofa*** and the attribute *lengthOfArms* (length of arms measured in meters) in class **SofaWithRoundArms** must have strictly positive values. Constructors must enforce these constraints.
- The abstract class ***Sofa*** has an abstract method *getCoefficient()*, which returns a coefficient that is applied to calculate the price of a sofa.
- The coefficient applied to calculate the price of an object of type **ClassicSofa** is 1.5, and for an object of type **SofaWithRoundArms** the coefficient is 2.

- The price of a sofa is calculated as the coefficient multiplied by the sum of *width* and *height*. For **SofaWithRoundArms**, the price of the arms is added to the sofa's price. One meter of arms costs 10 lei.
- The method *addProductToSell(Sofa)* in class **SellingProducts** adds the object passed as a parameter to the existing sequence of sofas. This addition operation will be made such that displaying the sequence of sofas in descending order of price can be done in linear time.

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

a1) Declare all classes, attributes, and methods as per the diagram above.

Implement only the following methods:

a2) The method *getPrice()* in class **SofaWithRoundArms.**
a3) The constructors of the ***Sofa*,** **SofaWithRoundArms** and **SellingProducts** classes.
a4) The method *addProductToSell(Sofa)* in class **SellingProducts**.
a5) The method *getCoefficient()* in class **ClassicSofa** and the method *setPrevious(Sofa)* in class ***Sofa***.

b) Define a function that receives as parameter an object *s* of type **SellingProducts** and displays, in linear time, the prices of sofas in *s*, in ascending order. Solutions that do not meet the required complexity will be awarded partial score.

c) Define a function that receives as parameters an object *s* of type **SellingProducts**, two real numbers *startPrice* and *endPrice*, and removes from *s* those sofas whose prices are in the range [*startPrice*, *endPrice*]. The implementation must have a time complexity of $O(n)$ (*n* is the number of sofas in the list associated to the object **SellingProducts**). Solutions that do not meet the required complexity will be awarded partial score.

d) Create an object of type **SellingProducts** and add 4 sofas to it (choose values for the unspecified attributes): two objects of type **ClassicSofa** and two of type **SofaWithRoundArms**, having 1.2 and respectively 0.9 metres of arms. Call the function at item b) and then the function at item c) (choose values for *startPrice* and *endPrice*).

e) State the *best-case*, *average-case* and *worst-case* time complexity for the method *addProductToSell(Sofa)* in class **SellingProducts**.

SUBJECT Databases

Problem 1. (4 points)

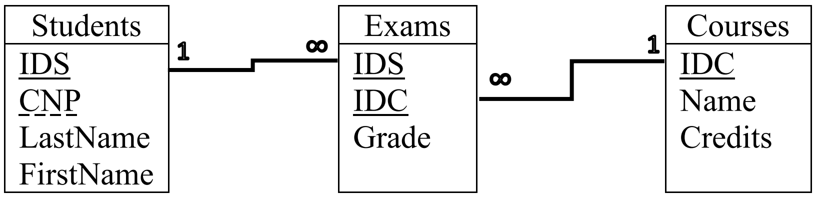
The organizers of some seashell collecting competitions are storing the following information in a relational database:

- A participant has an ID, a name, an email address and a country of origin.
- A beach has an ID, a name, a description and a city.
- A competition has an ID, a name, a date it is scheduled for, a start time, an end time and an associated beach. Many competitions can take place on the same beach.
- A type of seashell has an ID, a name and a description.
- During each competition, the participants collect seashells of different types from the beach where the competition takes place. Only the total number of seashells collected by each participant in each competition, for each seashell type, is stored in the database (for example, participant *p* collected: 10 seashells of type *t1* and 15 seashells of type *t2* in competition *c1*; 50 seashells of type *t1*, 15 seashells of type *t2* and 20 seashells of type *t3* in competition *c2*, etc.).

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[<u>IDS</u>, CNP, LastName, FirstName]
Courses[<u>IDC</u>, Name, Credits]
Exams[<u>IDS, IDC</u>, 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 private kindergarten:

Groups[<u>GroupID</u>, GroupName, RoomNumber]
Parents[<u>ParentID</u>, ParentName, Address, PhoneNumber]
Children[<u>ChildID</u>, ChildName, *ParentID*, DateOfBirth, *GroupID*]
FeePayments[<u>PaymentID</u>, *ChildID*, PaymentDate, 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, for each group for which at least one fee has been paid: the ID, the name of the group and the total amount of fees paid for the children in the group.

b. Consider the following instances for relations Groups, Parents and Children:

Groups:

Group ID	**Group Name**	**RoomNumber**
1	Buburuze	1
2	Veverițe	5
3	Albinuțe	8

Parents:

Parent ID	**Parent Name**	**Address**	**PhoneNumber**
1	P1	A1	1111111111
2	P2	A2	2222222222
3	P3	A3	3333333333

Children:

Child ID	**Child Name**	**Parent ID**	**DateOfBirth**	**Group ID**
1	C1	1	2018.02.12	1
2	C2	2	2018.06.25	1
3	C3	2	2018.06.25	1
4	C4	3	2018.06.30	1
5	C5	1	2020.04.15	2
6	C6	1	2022.06.05	3

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.ParentName, p.Address
FROM Parents p
   INNER JOIN Children c ON p.ParentID = c.ParentID
WHERE c.GroupID = (SELECT g.GroupID
                          FROM Groups g
                          WHERE g.GroupName = 'Buburuze')
INTERSECT
SELECT p.ParentName, p.Address
FROM Parents p
   INNER JOIN Children c ON p.ParentID = c.ParentID
GROUP BY p.ParentID, p.ParentName, p.Address
HAVING COUNT(DISTINCT c.GroupID) = 1
```

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

- {ChildID} → {ParentID}
- {ParentID} → {GroupID}.

SUBJECT Operating Systems

Problem 1 (5 points). Answer the following questions about the execution of the program `./a.out` compiled from the source code below, considering that all necessary includes are present, the `fork` system call and the `execlp` system call are executed successfully, and the FIFOs `fa` and `fb` have been created beforehand. The `bash` command with the `-c` option executes the command in the string given as the value of the `-c` option.

```
 1  int main(int argc, char** argv) {
 2    int fa, fb, k; char s[32];
 3
 4    if(fork() == 0) {
 5      execlp("bash","bash","-c","sort <fa >>fb",NULL);
 6      exit(0);
 7    }
 8    if(fork() == 0) {
 9      fa = open("fa", O_WRONLY);
10      for(k=1; k<argc; k++) {
11        write(fa, argv[k], strlen(argv[k]));
12        write(fa, "\n", 1);
13      }
14      close(fa);
15      exit(0);
16    }
17    if(fork() == 0) {
18      fb = open("fb", O_RDONLY);
19      while((k = read(fb, s, 32)) > 0) {
20        write(1, s, k);
21      }
22      close(fb);
23      exit(0);
24    }
25    wait(NULL); wait(NULL); wait(NULL);
26    return 0;
27  }
```

a) What will the execution below display? Justify your answer.

```
./a.out a d b c
```

b) How is the execution affected if the IF block on lines 8-16 is moved between lines 3 and 4? Justify your answer.

c) What will the execution below display if the digit `1` on line 20 is changed to `2`? Justify your answer.

```
./a.out y z x > /dev/null
```

d) How will the execution be affected if line 14 is commented out? Justify your answer.

e) How will the execution be affected if lines 4, 6, and 7 are commented out? Justify your answer.

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

```
 1  #!/bin/bash
 2
 3  RE='^(.*)([+-]) (-?[0-9]+) (-?[0-9]+)(.*)$'
 4  L="$1"
 5  while true; do
 6    if echo $L | grep -q -v -E "$RE"; then
 7      exit 1
 8    fi
 9
10    A=`echo $L | sed -E "s/$RE/\1/"`
11    B=`echo $L | sed -E "s/$RE/\3 \2 \4/"`
12    C=`echo $L | sed -E "s/$RE/\5/"`
13    L="$A `expr $B` $C"
14
15    echo "A='$A'  B='$B'  C='$C'"
16
17    if echo $L | grep -E -q "^ *-?[0-9]+ *$"; then
18      break
19    fi
20  done
21  echo $L
```

a) What will the execution below display?

```
./a.sh "+ 1 2"
```

b) What will be the values of variables A, B, and C at each iteration of the `while` loop and what will be the output of the execution below?

```
./a.sh "+ + 1 2 3"
```

c) What will be the values of variables A, B, and C at each iteration of the while loop and what will be the output of the command below? Justify your answer.

```
./a.sh "+ + -1 2 -3 4"; echo $?
```

d) Explain in detail the functioning of lines 17-19.

BAREM INFORMATICĂ
VARIANTA 2

Subiect Algoritmică şi Programare

Oficiu – 1p

Cerința **a) – 4.6p**

- Definirea clasei Sofa – **0.55p** din care
  - clasă abstractă – 0.05
  - atribute - 0.1
  - **constructor (a3)** - 0.1
  - metoda **setPrevious (a5)** – 0.1
  - metode **getCoefficient**, **getPrice**, **getNext**, **setNext**, **getPrevious** - 0.2
- Definirea clasei SofaWithRoudArms– **0.85p** din care
  - relația de moștenire – 0.2
  - atribut – 0.1
  - **constructor (a3)** – 0.2
  - metoda **getPrice (a2)** – 0.25
  - metoda **getCoefficient** - 0.1
- Definirea clasei ClassicSofa – **0.4p** din care
  - relația de moștenire – 0.25
  - metoda **getCoefficient (a5)** - 0.15
- Definirea clasei SellingProducts– **2.8p** din care
  - atribute– 0.2
  - **constructor (a3)** - 0.1
  - metoda **addProductToSell (a4)** – 2.5
    - adăugare listă vidă – 0.3
    - adăugare început – 0.3
    - adăugare sfârșit – 0.3
    - inserare în listă – 1.6

Funcția **b) - 1.1p**

- signatura - 0.1
- afișare în $\theta(n)$ - 1p

  * afișare în complexitate timp mai mare decât cea cerută– 0.25

Funcția **c) - 1.9p**

- signatura - 0.1
- ștergere în $O(n)$ – 1.8
  - ștergere început – 0.3
  - ștergere sfârșit – 0.3
  - ștergere în listă – 1.2

  * ștergere în complexitate timp mai mare decât cea cerută– 0.5

Funcția principală **d) – 0.4p**

- construire obiecte – 0.2p
- apel funcție b) – 0.1p
- apel funcție c) - 0.1 p

Cerința **e) – 1p**

- favorabil (0.25p)
- mediu (0.5p)
- defavorabil (0.25p)

BAREM INFORMATICĂ
VARIANTA 2

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:

NumeParinte	**Adresa**
P2	A2
P3	A3

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

- **b2** - {CodCopil}→{CodParinte} este satisfăcută – **0.25p**; **0.25p** explicație
  - {CodParinte}→{CodGrupa} 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.

BAREM INFORMATICĂ
VARIANTA 2

<u>Subiect: Sisteme de Operare</u>

1p – oficiu

Problema 1 (5p)

1p – a) a b c d, pentru că sort citeşte din fa ce scrie programul şi apoi programul citeşte din fb ce afişează sort
1p – b) nicicum, cele trei procese fiu sunt concurente şi ordinea creării lor nu are nicciun efect asupra execuţiei
1p – c) x y z, pentru ca deşi redirecţionăm ieşirea standard în /dev/null, programul scrie în ieşirea de eroare
1p – d) nicicum, procesul fiu se încheie şi FIFO-ul e închis automat, astfel sort nu ramâne blocat la citire
1p – e) se va bloca fără a afişa nimic, pentru că nu poate deschide FIFO-urile, procesele fiu nemaifiind create din cauză că execlp suprascrie codul procesului părinte

Problema 2 (4p)

1p – a) 3 – scriptul calculează suma celor două numere
1p – b) A='+ ' B='1 + 2' C=' 3'
A='' B='3 + 3' C=''
Se afişează 6
1p – c) A='+ ' B='-1 + 2' C=' -3 4'
A='' B='1 + -3' C=' 4'
Se va afişa 1 pentru că $L nu se va potrivi cu expresia regulară de la linia 6 şi se va face exit 1, iar $? va conţine această valoare
1p – d) Dacă variabila L conţine rezultatul final, adică o valoare întreagă cu eventuale spaţii înainte şi după, valoarea de adevăr a comenzii grep va fi TRUE şi ca urmare se intră in IF şi se va executa comanda break, astfel încheindu-se cu succes execuţia scriptului