Subiecte EXAMEN CSIE Structuri de date

Data Structures Structuri de Date (Academia de Studii Economice din Bucure□ti)

Implementaţi o aplicaţie în limbajul C ce rezolvă probleme de gestionare unei reţele de magazine de desfacere bunuri de larg consum.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Listă Simplă*** ce conţine date aferente unor magazine. În fiecare nod al listei, magazinele se stochează la nivel de adresă (elemente de tip ***Magazin****). Inserarea unui nod are loc astfel incât lista simplă să fie ordonată crescător in funcţie de id magazin (inserare nod în interiorul listei). Inserarea unui nod se implementează într-o funcţie care se apelează în secvenţa de creare a structurii ***Listă Simplă***.

   Structura ***Magazin*** este definită astfel:

```
struct Magazin {
      int id;
      char* denumire;
      char* localitate;
      float suprafata;
      int numar_angajati;
};
```

   Exemplu set de date pentru un magazin: **{11, „La Nicusor", „Nehoiu", 33.89, 2}**

   Lista simplă va conţine datele a cel puţin 10 magazine care se preiau ca input dintr-un fişier text. **(2p)**

2. Scrieţi şi apelaţi funcţia pentru modificarea denumirii unui magazin specificat prin denumire şi localitate. **(1p)**

3. Scrieţi şi apelaţi funcţia determină numărul mediu de angajaţi per magazin dintr-o localitate specificată. **(1p)**

4. Scrieţi secvenţa de cod care copiază datele din ***Lista Simplă*** creată anterior într-o structură ***Tabelă de Dispersie***, având ***Linear Probing*** ca mecanism de tratare a coliziunilor. Cheia de căutare este ***localitate***. În caz de coliziune, căutarea primei poziţii disponibile în ***Tabela de Dispersie*** se efectuează cu pasul ***-1*** (la stânga punctului de coliziune). Cele două structuri de date ***NU*** partajează zone de memorie heap. **(2p)**

5. Scrieţi şi apelaţi funcţia pentru modificarea localităţii (cheie de căutare) unui magazin în tabela de dispersie. Magazinul este specificat prin denumire şi localitate (cheie de căutare). **(2p)**

6. Scrieţi secvenţa de cod care dezalocă structurile ***Listă Simplă*** şi ***Tabelă de Dispersie*** create la punctele anterioare. **(1p + 1p)**

OBSERVAŢIE: Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.

Implementaţi o aplicaţie în limbajul C ce rezolvă probleme de gestionare a rapoarte de analize medicale.

1. Definiţi structura ***RaportAnalizeMedicale*** ce conţine: cod raport (***unsigned int***), cod pacient (***unsigned int***), numărul de analize medicale (***unsigned char***), denumiri analize medicale (***char*****), valori calculate/determinate per analiza medicala (***float****), valoare de referință per analiză medicala (***float****), data recoltarii probelor biologice (***char****).

2. Scrieţi şi apelaţi funcţia de inserare a unui raport de analize medicale într-o ***Tabelă de Dispersie.*** Creaţi structura ***Tabelă de Dispersie*** pentru cel puţin 10 rapoarte de analize medicale utilizând această funcţie de inserare. Cheia de căutare în tabelă este data recoltarii probelor biologice, iar mecanismul de tratare a coliziunilor este ***chaining***. Conţinutul tabelei de dispersie se afişează la consolă după creare. **(2p)**

3. Scrieţi şi apelaţi funcţia pentru determinarea numarului de analize medicale efectuate intr-o anumită perioadă de timp. Perioada de timp este specificata ca parametru de intrare. Rezultatul se afişează la consolă. **(1,5p)**

4. Scrieţi şi apelaţi funcţia care copiază într-o ***Lista Simpla*** datele (*cod pacient, valoare calculata/determinata, data recoltare probe biologice*) aferente unei analize medicale (din tabela de dispersie). Denumirea analizei medicale este specificata ca parametru de intrare. Cele două structuri de date ***NU*** partajează zone de memorie heap. Conţinutul listei simple se afişează la consolă după creare. **(2p)**

5. Scrieţi şi apelaţi funcţia pentru determinarea numarului de pacienti (diferiti) care au efectuat analize medicale salvate in lista simpla de la punctul 4). Rezultatul se afişează la consolă. **(1,5p)**

6. Scrieţi şi apelaţi funcţia pentru determinarea numărului de rapoarte medicale din tabela de dispersie pentru analize medicale efectuate la o data specificata ca parametru de intrare. Rezultatul se afişează la consola. **(1p)**

7. Scrieţi secvenţa de cod care dezalocă structurile ***Tabelă de Dispersie*** şi ***Lista Simpla*** create la punctele anterioare. **(1p + 1p)**

OBSERVAŢII:

- **Inseraţi comentarii cu privire la numărul cerinţei de implementare.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**

Bilet #1 / (120 min)	**06.06.2020**

Implementaţi o aplicaţie în limbajul C ce rezolvă probleme de gestionare a grădiniţelor pentru copii.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Tabelă de Dispersie*** ce conţine date aferente unor grădiniţe. Cheia de căutare este ***alfanumerică***, iar mecanismul de tratare a coliziunilor este ***Chaining***. Inserarea unei grădiniţe se implementează într-o funcţie care se apelează în secvenţa de creare a structurii ***Tabelă de Dispersie***.

   Structurile ***Gradinita*** se va defini astfel încât să conţină minim 5 câmpuri, din care minim două sunt declarate ca pointeri.

   Tabela de dispersie va conţine datele a cel puţin 10 gradiniţe care se preiau ca input dintr-un fişier text. **(2p)**

2. Scrieţi şi apelaţi funcţia pentru modificarea cheii de căutare pentru o grădiniţă stocată în tabela de dispersie. **(2p)**

3. Scrieţi secvenţa de cod care copiază o parte din grădiniţe din ***Tabela de Dispersie*** creată anterior într-o structură ***Lista Dublă***. Filtrarea grădiniţelor copiate se realizează pe baza unui câmp definit în structura ***Gradinita***. Cele două structuri de date ***NU*** partajează zone de memorie heap. **(3p)**

4. Scrieţi şi apelaţi funcţia pentru operaţia de ”rupere”/”spargere” a listei duble în două subliste. Nodul la care se efectuează „ruperea” este identificat pe baza unui câmp definit în structura ***Gradinita***. **(2p)**

5. Scrieţi secvenţa de cod care dezalocă structura ***Tabelă de Dispersie*** şi cele două structuri ***Listă Dublă*** create la punctele anterioare. **(1p)**

OBSERVAŢIE: Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.

Bilet #2 / (120 min)	**06.06.2020**

Implementaţi o aplicaţie în limbajul C ce rezolvă probleme de gestionare a angajaților unei instituții.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Listă Dublu Înlănțuită*** ce conţine date aferente unor angajați. Inserarea unui angajat se implementează printr-o funcţie care se apelează în secvenţa de adăugare noduri în structura ***Listă Dublu Înlănțuită***.

   Structura ***Angajat*** se va defini astfel încât să conţină minim 5 câmpuri, din care minim două sunt declarate ca pointeri.

   Lista dublu înlănțuită va conţine datele a cel puţin 10 angajați care se preiau ca input dintr-un fişier text. **(2p)**

2. Scrieţi şi apelaţi funcţia pentru stergerea tuturor nodurilor din lista dubla identificate pe baza unui criteriu din structura ***Angajat***. Pentru verificare, lista dublă este afișată înainte și după stergere, prin traversare în ambele sensuri. **(2p)**

3. Scrieţi secvenţa de cod care copiază o parte dintre angajații din ***Lista Dublu Înlănțuită*** creată anterior într-o structură ***Arbore Binar de Căutare***. Filtrarea angajaților copiați se realizează pe baza unui câmp definit în structura ***Angajat***. Cele două structuri de date ***NU*** partajează zone de memorie heap. **(3p)**

4. Scrieţi şi apelaţi funcţia pentru salvarea intr-un vector a nodurilor din ***Arbore Binar de Căutare*** plasate de la radacina pana la un anumit nod identificat pe baza cheii de cautare a arborelui din structura ***Angajat***. **(2p)**

5. Scrieţi secvenţa de cod care dezalocă structura ***Listă Dublu Înlănțuită***, ***Arborele Binar de Căutare*** şi a structurii ***Vector*** create la punctele anterioare. **(1p)**

OBSERVAŢIE: Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.

Bilet #4 / (120 min) | **06.06.2020**

Implementaţi o aplicaţie în limbajul C ce rezolvă probleme de gestionare a platformelor online destinate tele-conferințelor.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Min-HEAP*** ce conţine date aferente unor Platforme. Prioritatea în utilizarea platformelor este dată de prețul aferent licenței pentru respectiva platformă. Inserarea unei platforme se implementează într-o funcţie care se apelează în secvenţa de creare a structurii ***Min-HEAP***.

   Structura ***Platforma*** se va defini astfel încât să conțină minim 5 câmpuri, unul este prețul și minim alte două dintre celelalte sunt declarate ca pointeri la două tipuri diferite între ele.

   În cadrul structurii Min-HEAP sunt introduse cel puţin 10 platforme care se preiau ca input dintr-un fişier text. **(2p)**

2. Scrieţi şi apelaţi funcţia care modifcă prețul platformei cu prețul cel mai mic din cadrul structurii Min-HEAP. Platforma este reintrodusă în cadrul structurii cu noul preț. **(2,5p)**

3. Scrieţi secvenţa de cod care copiază primele n platforme din structura creată anterior într-o structură de tip Arbore Binar de Căutare. Valoarea n este primită ca parametru. Inserarea platformelor în arborele binar de căutare se realizează pe baza unui câmp definit în structura Platformă, dar nu câmpul preț. Cele două structuri de date ***NU*** partajează zone de memorie heap. Adică trebuie să faceți deep-copy. **(3p)**

4. Scrieţi şi apelaţi funcţia care afiseaza platformele aflate în nodurile care au un singur nod descendent. **(1p)**

5. Scrieţi secvenţa de cod care dezalocă structura ***Min-HEAP*** şi cele **Arborele Binar de Căutare** create la punctele anterioare. **(1,5p)**

OBSERVAŢIE: Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.

Bilet #7 / (120 min) | **14.06.2020**

Implementaţi o aplicaţie în limbajul C ce rezolvă gestiunea camerelor de cazare dintr-un hotel.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Arbore binar de căutare*** care reprezintă un hotel cu mai multe camere. Cheia de inserare se alege din lista de câmpuri a structurii ***Camera.*** Inserarea unei camere se implementează în funcţia principală pentru un număr de minim 10 elemente citite dintr-un fisier de intrare. **(2p)**

   Structura ***Camera*** se va defini astfel încât să conţină minim 5 câmpuri din care următoarele sunt obligatorii: *tip camera, etaj*. De asemenea, minim două câmpuri sunt definite ca variabile pointer.

2. Scrieţi şi apelaţi funcţia pentru determinarea numărului de camere de pe fiecare etaj din hotel. Funcţia returnează o structură în care sunt stocate rezultatele. (**2p**)

3. Scrieţi secvenţa de cod care copiază o parte din camerele din ***Arborele binar de căutare*** creat la cerinţa 1) într-o structură ***Lista Dublă***. Filtrarea camerelor copiate se realizează pe baza unui câmp definit în structura ***Camera***. Cele două structuri de date ***NU*** partajează zone de memorie heap. **(3p)**

4. Scrieţi şi apelaţi funcţia pentru inserarea unei camere în ***Lista Dublă*** după un nod specificat pe baza unui câmp al structurii ***Camera***. Validarea operaţiei se realizează prin traversarea listei înainte şi după apelul funcţiei de inserare. **(2p)**

5. Scrieţi secvenţa de cod care dezalocă structura ***Arbore binar de cautare, Lista Dublă*** şi toate structurile auxiliare utilizate în implementarea cerinţelor. **(1p)**

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerintele trebuie apelate si demonstrate in functia main() pentru a fi evaluate.**

Bilet #13 / (120 min) | **15.06.2020**

Implementaţi o aplicaţie în limbajul C care implementează soluţii la probleme de depozitare a cerealelor.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Arbore Binar de Căutare***. Cheia de inserare se alege din lista de câmpuri a structurii ***Depozit.*** Inserarea unui depozit se implementează în funcţia principală pentru un număr de minim 10 elemente citite dintr-un fisier de intrare. **(2p)**

   Structura ***Depozit*** se va defini astfel încât să conţină minim 5 câmpuri din care minim două câmpuri sunt definite ca variabile pointer.

2. Scrieţi şi apelaţi funcţia pentru determinarea depozitelor plasate pe un nivel specificat în arborele creat la punctul 1). Funcţia returnează un vector de depozite care **NU** partajează memorie heap cu arborele binar de căutare. (**2p**)

3. Scrieţi şi apelaţi funcţia pentru salvarea într-un vector a drumului de la un nod către rădăcina arborelui creat la punctul 1). Nodul este identificat prin cheia de căutare. **(2p)**

4. Scrieţi şi apelaţi funcţia pentru determinarea nivelului cu numărul maxim de noduri. Funcţia returnează numărul nivelului. **(2p)**

5. Scrieţi şi apelaţi funcţia pentru ştergerea tuturor nodurilor frunză din arborele binar de căutare. **(1p)**

6. Scrieţi secvenţa de cod care dezalocă structurile ***Arbore binar de cautare, Vectori*** şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este necesar). **(1p)**

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerintele trebuie apelate si demonstrate in functia main() pentru a fi evaluate.**

Bilet #11 / (120 min) | **15.06.2020**

Implementaţi o aplicaţie în limbajul C care implementează soluţii la probleme colaborative din cadrul unei echipe de dezvoltare software.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Arbore AVL*** care reprezintă structura de mapare pe task-uri a unui proiect software. Cheia de inserare se alege din lista de câmpuri a structurii ***Task.*** Inserarea unui task se implementează în funcţia principală pentru un număr de minim 10 elemente citite dintr-un fisier de intrare. **(2p)**

   Structura ***Task*** se va defini astfel încât să conţină minim 5 câmpuri din care minim două câmpuri sunt definite ca variabile pointer.

2. Scrieţi şi apelaţi funcţia pentru „spargerea" structurii creată la punctul 1) în doi arbori binari de căutare. Nodul în care are loc „spargerea" se identifică pe baza unui câmp din structura ***Task***. Dupa spargere, se calculează înălţimile celor doi arbori rezultaţi. (**2p**)

3. Scrieţi secvenţa de cod care copiază task-urile din unul din cei doi arbori creaţi la cerinţa 2) într-o structură ***Lista Dublă***. Inserarea task-urilor în ***Lista Dublă*** are loc astfel încât să se păstreze nodurile sortate după un câmp al structurii ***Task*** (diferit faţă de cel utilizat la punctul 2)). Cele două structuri de date ***NU*** partajează zone de memorie heap. **(3p)**

4. Scrieţi şi apelaţi funcţia pentru extragerea într-un vector a tuturor task-urilor din ***Lista Dublă*** care îndeplinesc un anumit criteriu. Criteriul utilizat este diferit faţă de cele utilizate anterior (punctele 2) si 3)) **(2p)**

5. Scrieţi secvenţa de cod care dezalocă structurile ***Arbori binari de căutare, Listă Dublă, Vector*** şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este necesar). **(1p)**

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerintele trebuie apelate si demonstrate in functia main() pentru a fi evaluate.**

Bilet #10 / (120 min)	15.06.2020

Implementaţi o aplicaţie în limbajul C ce rezolvă probleme de gestionare a facturilor de platit la nivelul unei gospodării.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Heap (coadă de priorități)*** ce conţine date aferente unor facturi. Inserarea unei noi facturi se implementează printr-o funcţie care se apelează în secvenţa de adăugare elemente în structura ***Heap***.

   Structura ***Factura*** se va defini astfel încât să conţină minim 5 câmpuri, dintre care minim două sunt declarate ca pointeri.

   Coada de priorități va conţine datele a cel puţin 10 facturi care se preiau ca input dintr-un fişier text, prioritatea fiind dată de numărul de zile rămase până la scadență. **(2p)**

2. Scrieţi şi apelaţi funcţia pentru eliminarea tuturor facturilor din structura ***Heap*** care mai au mai puțin de trei zile până la scadență. Pentru verificare, structura ***Heap*** este afișată înainte și după stergere. **(2p)**

3. Scrieţi secvenţa de cod care copiază o parte dintre facturile din structura ***Heap*** creată anterior într-o structură ***Arbore Binar de Căutare***. Filtrarea facturilor copiate se realizează pe baza unui câmp definit în structura ***Factura***. Cele două structuri de date ***NU*** partajează zone de memorie heap. **(3p)**

4. Scrieţi şi apelaţi funcţia pentru salvarea într-o ***Listă simplu înlănțuită*** a nodurilor din ***Arborele Binar de Căutare*** plasate de la rădăcina până la un anumit nod identificat pe baza cheii de căutare a arborelui din structura ***Factura***. **(2p)**

5. Scrieţi secvenţa de cod care dezalocă structura ***Heap***, ***Arborele Binar de Căutare*** şi ***Listă simplu înlănțuită*** create la punctele anterioare. **(1p)**

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerintele trebuie apelate si demonstrate in functia main() pentru a fi evaluate.**

Bilet #23 / (120 min) | **03.07.2020**

Implementaţi o aplicaţie în limbajul C care rezolva managementul coletelor livrate de o firma de curierat.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Coada de prioritati*** care sa stocheze pachetele livrate, in memoria HEAP. Structura va fi populata cu cel putin 10 inregistrari citite din fisier. Structura ***Colet*** se va defini astfel încât să conţină 5 câmpuri din care minim unul sa fie pointer de tip ***char (localitate)*** iar unul sa fie de tip int (**cost livrare**). **(2p)**
   Cerinţe de implementare:
   - Definire structură ***Colet*** *(campurile au sens iar datele din fisier sunt corelate)*. ***(0,25p)***
   - String-urile preluate din fişier trebuie să accepte prezenţa simbolului ***blank***. ***(0,25p)***
   - Absenţă memory leaks. ***(0,25p)***
   - Implementare logică de creare structură ***Coada de prioritati***. ***(0,50p)***
   - Testare implementare, populare completă şi corectă a structurii ***Coada de prioritati***. ***(0,75p)***

2. Scrieţi şi apelaţi funcţia pentru procesarea unui colet conform prioritatii. Informatia utila se returnează în ***main()*** de catre functia de procesare prin tipul de retur al funcţiei. **(1p)**
   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Implementare logică pentru procesarea coletului. ***(0,25p)***
   - Testarea implementarii in functia ***main()***. ***(0,50p)***

3. Scrieţi şi apelaţi funcţia pentru schimbarea unei prioritati aferente unui colet identificat in mod unic pe baza unui atribut din structura ***Colet***. Noua ordine a elementelor este afisata in functia ***main()***. **(1.5p)**
   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Implementare logică pentru schimbarea prioritatii in cadrul structurii. ***(0,75p)***
   - Testarea implementarii in functia ***main()***. ***(0,50p)***

4. Scrieţi şi apelaţi funcţia pentru stergerea unui colet din structura ***Coada de prioritati***, identificat in mod unic pe baza unui atribut din structura ***Colet***. Stergerea coletului se realizeaza cu redimensionarea structurii in scopul optimizarii spatiului de memorie aferent acesteia. **(2p)**
   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Realocare dinamică a structurii. ***(0,25p)***
   - Absenţă memory leaks. ***(0,25p)***
   - Implementare logică pentru stergerea unui colet. ***(0,75p)***
   - Testarea implementarii in functia ***main()***. ***(0,50p)***

5. Scrieţi şi apelaţi functia pentru procesarea tuturor coletelor in ordinea corecta a prioritatilor si contabilizarea acestora pe centri de profit, specifici localitatilor de destinatie. Centri de profit identificati sunt salvati intr-un vector ca elemente de tipul (numeLocalitate, totalLivrari). Vectorul se returneaza in ***main()*** prin tipul de retur si este afisat. **(2.5p)**
   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Absenţă memory leaks. ***(0,25p)***
   - Alocare dinamică a vectorului pe centri de profit. ***(0,25p)***
   - Implementare logică pentru calculul centrilor de profit. ***(1,00p)***
   - Testare implementare, populare completă şi corectă a vectorului, afisare. ***(0,75p)***

6. Scrieţi şi apelaţi funcţiile care dezalocă structura ***Coada de prioritati***, precum şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este cazul). **(1p)**
   Cerinţe de implementare:
   - Definire funcţii cu parametri de I/O definiţi complet şi corect. ***(0,15p)***
   - Absenţă memory leaks. ***(0,15p)***
   - Actualizare variabile de gestionare a structurilor în funcţia ***main()***. ***(0,20p)***
   - Implementare logică de dezalocare a structurilor de date. ***(0,30p)***
   - Testare implementare, dezalocare completă şi corectă a structurilor. ***(0,20p)***
   - Absenţă dezalocări structuri auxiliare utilizate. ***(-0,20p)***

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările nu trebuie să conţină variabile definite la nivel global sau statice.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerinţele trebuie apelate şi demonstrate in functia main() pentru a fi evaluate.**
- **Art. 72 (1) Pentru următoarele fapte, studenţii vor fi exmatriculaţi fără drept de reînmatriculare în Academia de Studii Economice din Bucureşti:**
  (c) încercarea de promovare prin fraudă a examenelor sau a altor evaluări;

Bilet #7 / (150 min) | **22.06.2021**

Implementaţi o aplicaţie în limbajul C care implementează soluţii la probleme de gestionare a rezervarilor la o sala de spectacole.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de date de tip ***Arbore Binar de Cautare,*** *in continuare* ***ABC,*** ce conţine date aferente rezervarilor de tip ***Spectacol*.*** Cheia de căutare utilizată este atributul ***idSpectacol***, iar articolul este compus din urmatoarele atribute obligatorii ***pretBilet(float)***, ***numeClient(char*)***, ***dataSustinere(char[10])***, precum si din alte 2 atribute definite la alegere. Inserarea unei rezervari se implementează **într-o funcţie** care se apelează în secvenţa de creare a structurii ***ABC***. Strucura conţine minim 10 inregistrari incarcate in aplicatie dintr-un fisier de intrare. ***(2p)***

   Cerinţe de implementare:
   - Definire structură ***Spectacol***. ***(0,25p)***
   - String-urile preluate din fişier trebuie să accepte prezenţa simbolului ***blank***. ***(0,25p)***
   - Absenţă memory leaks. ***(0,25p)***
   - Implementare logică de creare structură ***ABC***. ***(0,75p)***
   - Populare completă şi corectă a structurii ***ABC*** cu date de intrare din fisier. ***(0,25p)***
   - Testare implementare cu afisarea la consola a continutului structurii ***ABC***. ***(0,25p)***

2. Scrieţi şi apelaţi funcţia pentru determinarea rezervarilor din structura creată la cerinta 1) care se desfasoara la aceeasi data specificata ca parametru de intrare al functiei. Rezervarile identificate sunt salvate într-un vector şi ***NU*** partajează zone de memorie heap cu structura ***ABC***. Vectorul se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei. **(2p)**

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Realizare deep-copy a rezervarilor în vector. ***(0,25p)***
   - Implementare logică de determinare şi salvare a rezervarilor în vector. ***(1,00p)***
   - Populare completă şi corectă a vectorului. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut dupa apel. ***(0,25p)***

3. Scrieţi şi apelaţi funcţia pentru determinarea ***costului total aferent fiecarui client*** pentru ***toate rezervarile*** atribuite. Valorile identificate sunt salvate într-un vector in care fiecare element contine perechea de valori ***(numeClient, costTotal).*** Vectorul si dimensiunea acestuia se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei. **(2.5p)**

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Determinare dimensiune masiv. ***(0,25p)***
   - Implementare logică de determinare clienti si cost total rezervari. ***(1,5p)***
   - Populare completă şi corectă a vectorului. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut dupa apel. ***(0,25p)***

4. Scrieţi şi apelaţi funcţia pentru transformarea structurii ABC de la cerinta 1) in doua structuri complementare de tip arbore binar de cautare, pe baza unuia din campurile optionale definite la alegere. Campul utilizat trebuie sa aiba semnificatie binara pentru a putea crea cei doi arbori binari de cautare complementari. Structurile rezultate se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei iar continutul acestora este afisat la consola. **(2.5p)**

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Determinare elemente pentru inserarea in structurile complementare. ***(0,25p)***
   - Implementare logică de creare arbori binari de cautare ***fara noi alocari de memorie***. ***(1,25p)***
   - Populare completă şi corectă a structurilor. ***(0,5p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut dupa apel. ***(0,25p)***

5. Scrieţi şi apelaţi funcţiile care dezalocă structurile **2x*ABC*** si ***2xVectori*** precum şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este cazul). **(1p)**

   Cerinţe de implementare:
   - Definire funcţii cu parametri de I/O definiţi complet şi corect. ***(0,15p)***
   - Absenţă memory leaks. ***(0,15p)***
   - Actualizare variabile de gestionare a structurilor în funcţia ***main()***. ***(0,20p)***
   - Implementare logică de dezalocare a structurilor de date. ***(0,30p)***
   - Testare implementare, dezalocare completă şi corectă a structurilor prin apel de functii si afisare la consola a rezultatelor obtinute la apel. ***(0,20p)***
   - Absenţă dezalocări structuri auxiliare utilizate. ***(-0,20p)***

Bilet #7 / (150 min) | **22.06.2021**

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările nu trebuie să conţină variabile definite la nivel global sau variabile statice.**
- **Implementarile nu trebuie sa contina structuri predefinite (ex STL, 3rd party libraries etc).**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerinţele trebuie apelate şi demonstrate in functia main() pentru a fi evaluate.**
- **Art. 72 (1) Pentru următoarele fapte, studenţii vor fi exmatriculaţi fără drept de reînmatriculare în Academia de Studii Economice din Bucureşti:**
    - **(c) încercarea de promovare prin fraudă a examenelor sau a altor evaluări;**

Implementaţi o aplicaţie în limbajul C ce rezolvă probleme de gestionare a unor vouchere emise de un agent economic.

1. Definiţi structura ***Voucher*** ce conţine: nr voucher (***unsigned int***), nume beneficiar (***char****), dată expirare (ex: ***char****), valoare (***float***).

   Creaţi o structură ***listă dublă*** cu cel puţin 5 vouchere ale căror date sunt preluate dintr-un fişier text. ***(1 p)***

2. Afişaţi elementele din lista dublă creată mai sus prin traversarea structurii în ambele sensuri. La consolă se vor afişa următoarele date: ***nr voucher***, ***dată expirare***. ***(0,5 p)***

3. Implementaţi funcţia care returnează valoarea voucher-elor din lista dublă creată mai sus şi a căror dată de expirare este în luna specificată ca parametru de intrare al funcţiei. Funcţia implementată se apelează în funcţia ***main()***, iar rezultatul apelului se afişează în consola de execuţie a aplicaţiei. ***(1,5 p)***

4. Implementaţi funcţia care returnează numărul beneficiarilor cu cel puţin 2 apariţii în lista dublă creată mai sus. Funcţia implementată se apelează în funcţia ***main()***, iar rezultatul apelului se afişează în consola de execuţie a aplicaţiei. ***(1,5 p)***

5. Implementaţi funcţia care salvează într-un vector voucher-ele din lista dublă de mai sus care au valoarea peste un prag specificat ca parametru de intrare al funcţiei. Datele salvate în vector sunt sortate în funcţie cu criteriul ***valoare voucher***. Funcţia implementată se apelează în funcţia ***main()***, iar rezultatul apelului (vector sortat) se afişează în consola de execuţie a aplicaţiei. ***(1,5 p)***

Următoarele aspecte trebuie considerate cu privire la implementare:
- Implementarea ***NU*** determină apariţia de ***memory leaks***.
- Funcţiile implementate ***NU*** conţin apeluri de funcţii standard privind operaţii de ***I/O*** cu dispozitive standard (ex: ***printf, scanf*** etc). Excepţie face funcţia ***main()***.
- Funcţiile implementate care nu se apelează în funcţia main() ***NU*** vor face obiectul evaluării.
- Secvenţele de cod sursă comentate ***NU*** vor face obiectul evaluării.

Bilet #20 / (120 min) | **03.07.2020**

Implementaţi o aplicaţie în limbajul C care implementează soluţii la probleme de gestionare a unei reţele de calculatoare.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Graf*** implementată prin ***Listă de Adiacenţă (Listă de Liste).*** Graful este orientat şi ponderat, conţine minim 6 hosturi şi 13 muchii/arce. Hosturile şi muchiile/arcele sunt preluate dintr-un fişier de intrare.
   Structura ***Host*** se va defini astfel încât să conţină minim 5 câmpuri din care minim două câmpuri sunt definite ca variabile pointer. Cel puţin unul din cei doi pointeri este de tip ***char***.

   Cerinţe de implementare:
   - Definire structură ***Host***. ***(0,25p)***
   - String-urile preluate din fişier trebuie să accepte prezenţa simbolului ***blank***. ***(0,25p)***
   - Absenţă memory leaks. ***(0,25p)***
   - Implementare logică de creare structură ***Listă de Adiacenţă*** pentru un graf orientat şi ponderat. ***(1,25p)***
   - Testare implementare, populare completă şi corectă a structurii ***Listă de Adiacenţă***. ***(1,00p)***

2. Scrieţi şi apelaţi funcţia pentru determinarea arcelor din structura creată la punctul 1) şi a căror pondere este mai mare decât o valoare specificată prin parametru al funcţiei. Arcele identificate sunt salvate într-un vector ca elemente de tipul ***(varf_src, varf_dst)***, unde ***varf_src*** este vârf sursă ala arcului, iar ***varf_dst*** este cel destinaţie al arcului. Vectorul se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei.

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Alocare dinamică a vectorului de arce. ***(0,25p)***
   - Implementare logică de determinare a arcelor. ***(1,00p)***
   - Testare implementare, populare completă şi corectă a vectorului. ***(0,50p)***

3. Scrieţi şi apelaţi funcţia pentru determinarea host-urilor din structura creată la punctul 1) şi care îndeplinesc un criteriu specificat (câmp structură ***Host***) prin parametru de intrare al funcţiei. Host-urile identificate sunt salvate într-un vector care ***NU*** partajează zone de memorie heap cu structura ***Listă de Adiacenţă***. Vectorul se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei.

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Alocare dinamică a vectorului de host-uri. ***(0,25p)***
   - Realizare deep-copy a departamentelor în vectorul rezultat. ***(0,25p)***
   - Implementare logică de determinare a host-urilor fără partajare de zone de memorie heap. ***(0,75p)***
   - Testare implementare, populare completă şi corectă a vectorului. ***(0,50p)***

4. Scrieţi şi apelaţi funcţia pentru eliminarea unor host-uri din vectorul obţinut la punctul 3). Condiţia de eliminare este ca cel puţin un arc incident spre exteriorul host-ului să aibă pondere minimă în graful stocat prin ***Lista de Adiacenţă*** creată la punctul 1). Vectorul se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei.

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Realocare dinamică a vectorului de host-uri. ***(0,25p)***
   - Absenţă memory leaks. ***(0,25p)***
   - Implementare logică de eliminare a host-urilor din vectorul de intrare. ***(0,75p)***
   - Testare implementare, populare completă şi corectă a vectorului. ***(0,50p)***

5. Scrieţi şi apelaţi funcţiile care dezalocă structurile ***Listă de Adiacenţă (Listă de Liste)*** şi ***Vectori***, precum şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este cazul).

   Cerinţe de implementare:
   - Definire funcţii cu parametri de I/O definiţi complet şi corect. ***(0,15p)***
   - Absenţă memory leaks. ***(0,15p)***
   - Actualizare variabile de gestionare a structurilor în funcţia ***main()***. ***(0,20p)***
   - Implementare logică de dezalocare a structurilor de date. ***(0,30p)***
   - Testare implementare, dezalocare completă şi corectă a structurilor. ***(0,20p)***
   - Absenţă dezalocări structuri auxiliare utilizate. ***(-0,20p)***

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările nu trebuie să conţină variabile definite la nivel global sau statice.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerinţele trebuie apelate şi demonstrate in functia main() pentru a fi evaluate.**
- **Art. 72 (1) Pentru următoarele fapte, studenţii vor fi exmatriculaţi fără drept de reînmatriculare în Academia de Studii Economice din Bucureşti:**
  (c) încercarea de promovare prin fraudă a examenelor sau a altor evaluări;

Bilet #11 / (120 min) | **15.06.2020**

Implementaţi o aplicaţie în limbajul C care implementează soluţii la probleme colaborative din cadrul unei echipe de dezvoltare software.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Arbore AVL*** care reprezintă structura de mapare pe task-uri a unui proiect software. Cheia de inserare se alege din lista de câmpuri a structurii ***Task.*** Inserarea unui task se implementează în funcţia principală pentru un număr de minim 10 elemente citite dintr-un fisier de intrare. **(2p)**

   Structura ***Task*** se va defini astfel încât să conţină minim 5 câmpuri din care minim două câmpuri sunt definite ca variabile pointer.

2. Scrieţi şi apelaţi funcţia pentru „spargerea" structurii creată la punctul 1) în doi arbori binari de căutare. Nodul în care are loc „spargerea" se identifică pe baza unui câmp din structura ***Task***. Dupa spargere, se calculează înălţimile celor doi arbori rezultaţi. (**2p**)

3. Scrieţi secvenţa de cod care copiază task-urile din unul din cei doi arbori creaţi la cerinţa 2) într-o structură ***Lista Dublă***. Inserarea task-urilor în ***Lista Dublă*** are loc astfel încât să se păstreze nodurile sortate după un câmp al structurii ***Task*** (diferit faţă de cel utilizat la punctul 2)). Cele două structuri de date ***NU*** partajează zone de memorie heap. **(3p)**

4. Scrieţi şi apelaţi funcţia pentru extragerea într-un vector a tuturor task-urilor din ***Lista Dublă*** care îndeplinesc un anumit criteriu. Criteriul utilizat este diferit faţă de cele utilizate anterior (punctele 2) si 3)) **(2p)**

5. Scrieţi secvenţa de cod care dezalocă structurile ***Arbori binari de căutare, Listă Dublă, Vector*** şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este necesar). **(1p)**

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerintele trebuie apelate si demonstrate in functia main() pentru a fi evaluate.**

Arbore AVL- Imobil

1. Definiti structura **Imobil** ce contine: **idImpobil** (unsigned int), **numeStrada** (char*), **nrStrada** (unsigned int), **numarul de apartamente din bloc**, un vector cu **numarul de locatari din fiecare apartament**, **valoare totala de plata la intretinere** pentru intreg blocul in expresie monetara. (1p)

2. Definiti funtia de inserare a unui imobil intr-un **Arbore AVL**. Inserarea se realizeaza pe baza cheii idImobil. Apelati functia pentru cel putin 5 imobile stocate intr-un fisier de intrare. (2p)

3. Definiti functia de traversare in **inordine** a arborelui. Rezultatul executiei functiei este afisat in consola aplicatiei. (1p)

4. Definiti functia care determina numarul de imobile de pe o anumita strada (precizata ca parametru de intrare). Rezultatul executiei functiei este afisat in consola aplicatiei. (2p)

5. Definiti functia care returneaza valoarea de plata la intretinere pentru o persoana care locuieste in bloc. Suma totala de plata la intretinere se imparte la numarul total de locatari. (2p)

6. Definiti functia care returneaza valorile de plata pentru fiecare apartament dintr-un bloc, primit ca parametru. Valoarea de plata pentru fiecare apartament este construita pe baza valorii totale de plata si a numarului de locatari din fiecare apartament. (2p)

Tabela de dispersie – Proiect

1. Definiti structura **Proiect** ce contine: **cod proiect** (unsigned int), **titlu proiect** (char*), **beneficiar** (char*), **numar de executanti** (unsigned char), **buget alocat** (float).

   Creati o **tabela de dispersie** cu cel putin 5 proiecte ale caror date sunt preluate dintr-un fisier text. Cheia de cautare in tabela de dispersie este **beneficiar**. (1p)

2. Afisati elementele din tabela creata mai sus prin traversarea intregii structuri conform mecanismului de tartare a coliziunilor. La consola de executie se vor afisa urmatoarele date: **cod proiect, beneficiar.** (0,5p)

3. Implementati functia care returneaza bugetul total de investitii al unui beneficiar specificat ca parametru de intrare al functiei. Se iau in considerare proiectele salvate in tabela de dispersie create mai sus. Functia implementata se apeleaza in functia main(), iar rezultatul apelului se afiseaza in consola de executie. (1,5p)

4. Implementati functia care modifica denumirea unui beneficiar pentru proiectele salvate in tabela de dispersie creata mai sus. Beneficiarul a carui denumire este modificata reprezinta parametru de intrare al functiei, impreuna cu noua denumire a sa. Functia implementata se apeleaza in functia main(), iar tabela de dispersie se va afisa la consola conform cerintei. (1,5p)

5. Implementati functia care salveaza intr-o **lista simpla** proiectele (din tabela de mai sus) cu un budget alocat peste un nivel specificat ca parametru de intrare al functiei. Datele sunt preluate din tabela de dispersie create mai sus si nu sunt partajate zone de memorie heap intre cele doua structuri (tabela si lista). Functia implementata se apeleaza in functia main(), iar rezultatul apelului (lista simpla) se afiseaza in consola de executie a aplicatiei. (1,5p)

Tabela de dispersie – Avion

1. Definiti structura **Avion** ce contine: **cod avion** (unsigned int), **model avion** (char*), **nr calatori** (int), **pretul biletului platit de fiecare calator** (int*) (1p)

2. Definiti functia de inserare a unui avion intr-o Tabela de Dispersie. Inserarea se realizeaza pe baza cheii cod avion. Apelati functia pentru cel putin 5 avioane stocate intr-un fisier de intrare. (2p)

3. Definiti functia de listare a continutului tabelei de dispersie. Rezultatul executiei functiei este afisat in consola aplicatiei. (1p)

4. Definiti functia care cauta avioane stocate in tabela de dispersie. Avioanele se identifica pe baza codului de avion si trebuie sa fie incasari de pe bilete mai mari decat o valoare precizata ca parametru de intrare a functiei. Avioanele identificate sunt returnate si afisate la consola aplicatiei dupa apelul functiei. (3p)

5. Definiti functia care salveaza intr-un **vector** id-urile avioanelor stocate in tabela de dispersie avand acelasi model precizat ca parametru de intrare. Avioanele identificate sunt afisate in consola aplicatiei prin consultarea vectorului rezultat. (3p)

Arbore ABC-Evaluare

1. Definiti structura **Evaluare** ce contine urmatoarele informatii: **nume angajat** (char*), **marca** (unsigned int), **department** (char*), **nota evaluare** (unsigned int), **data evaluare** (char*). (1p)

2. Aplicatia trebuie sa incarce datele despre angajati intr-o structura de tip arbore **binar de cautare echilibrat**. Datele de intrare sunt preluate dintr-un fisier text ce contine minim 5 inregistrari. (2p)

3. Aplicatia permite cautarea evaluarii unui angajat in arborele binar echilibrat. (2p)

4. Aplicatia permite stergerea evaluarii unui angajat din arborele binar echilibrat. (2p)

5. Aplicatia permite salvarea intr-un fisier text a datelor despre evaluarile angajatilor de la un anumit department. (2p)

6. Aplicatia permite dezalocarea tuturor structurilor utilizate la terminarea executiei programului (1p)

Arbore AVL -Depozit

1. Se considera structura **Depozit** cu urmatoarele atribute: **codDepozit** (int), **locatie** (char*), **capacitate** (int), **nrCategoriiCereale** (int), **nomenclatorCereale** (denumiri cereale)(char**). Sa se implementeze functia de inserare a unui depozit intr-o structura **Arbore AVL**. Functia se apeleaza pentru minim 5 depozite. (2p)

2. Implementati functia care determina si intoarce in apelator numarul de depozite care pot stoca secara. Functia se apeleaza, iar rezultatul este afisat la consola. (1p)

3. Implementati functia care creeaza si intoarce in apelator multimea de categorii de cereale (denumiri cereale) care pot fi stocate in depozitele din Arborele AVL. Functia se apeleaza, iar rezultatul functiei (nomenclator cereale) se afiseaza la consola. (2p)

4. Implementati functia care adauga o categorie de cereale la nomenclatorul de cereale existent pentru un depozit specificat prin cod (parametru de intrare al functiei) (1p)

5. Implementati functia care afiseaza codurile pentru depozitele plasate pe drumul de la radacina la un depozit specificat prin cod (parametru de intrare al functiei). (1p)

6. Implementati functia care creeaza o structura **Stiva** prin copierea datelor din depozitele plasate pe drumul de la radacina la un depozit specificat prin cod (parametru de intrare a functiei). Functia returneaza structura Stiva in apelator. Stiva se afiseaza la consola prin extragerea depozitelor (cu dezalocare noduri stiva). (3p)

LDI -Vagon

Se defineste structura **Vagon** ce contine urmatoarele campuri: **numarVagon** (int), **firmaTransport** (char*), **numarBileteVandute** (int), **capacitateVagon** (int).

1. Creati o structura **lista dublu inlantuita** cu cel putin 5 vagoane associate unui tren, ale caror date sunt preluate dintr-un fisier text. (1p)

2. Sa se afiseze elementele din lista dubla, prin traversarea in ambele sensuri. (0,5p)

3. Implementati functia care sterge vagonul/vagoanele cu numarul minim de bilete vandute. Functia implementata se apeleaza in main(), iar rezultatul apelului se afiseaza in consola de executie a aplicatiei, prin traversarea in ambele sensuri. (1,5p)

4. Implementati functia care salveaza intr-o structura de tip **coada de prioritati** vagoanele din lista dubla, astfel incat vagonul cu cel mai mic grad de ocupare sa fie primul. Gradul de ocupare se calculeaza pentru fiecare vagon. Structura coada de prioritati NU partajeaza zone de memorie heap cu structura lista dubla. Functia implementata se apeleaza in main(), iar rezultatul apelului se afiseaza in consola de execurie a aplicatiei. (1,5p)

5. Implementati functia care modifica numarul de bilete vandute pentru un vagon din coada de prioritati al carui numar este specificat ca parametru. Noul numar de bilete este primit, de asemenea, ca parametru. Functia implementata se apeleaza in main(), iar rezultatul apelului se afiseaza in consola de executie a aplicatiei. (1,5p)

Arbore ABC – Repository de librarii software

Implementati o aplicatie in limbajul C ce rezolva gestiunea librariilor software dintr-un repository.

1. Scrieti secventa de cod sursa pentru crearea unei structuri de tip **Arbore binar de cautare** care reprezinta un repository de librarii software. Cheia de inserare este alfanumerica. Inserarea unei librarii se implementeaza in functia principala pentru un numar de minim 10 elemente citite dintr-un fisier de intrare. (2p)
   Structura **Librarie** se va defini astfel incat sa contina minim 3 campuri, din care unu este un masiv alocat dinamic.

2. Scrieti si apelati functia pentru copierea tuturor librariilor, dupa un anumit criteriu, intr-un arbore binar de cautare diferit care sa partajeze informatia utila la nivelul memoriei HEAP. Filtrarea librariilor se realizeaza pe baza unui camp definit in structura Librarie. (2p)

3. Scrieti si apelati functia care sterge librariile din repository pe baza unui anumit criteriu, diferit de cel implementat anterior si le insereaza intr-o structura care sa reflecte ordinea in care acestea au fost sterse. Filtrarea librariilor sterse se realizeaza pe baza unui camp definit in structura Librarii, altul fata de cel implementat anterior. Toate elementele sunt afisate la consola dupa ce toate au fost sterse. (3p)

4. Scrieti si apelati functia care insereaza intr-un vector in ordine descrescatoare a cheilor elementele din repository care n-au descendenti. (2p)

5. Scrieti secventa de cod care dezaloca structura Arbore binar de cautare si toate structurile auxiliare utilizate in implementarea cerintelor. (1p)

Tabela de dispersie – Dosar Candidat

Se defineste structura **DosarCandidat** ce contine urmatoarele campuri: **numeCandidat** (char*), **programStudiu** (char*), **medieBac** (float), **codDosar** (int).

1. Creati o **tabela de dispersie** cu cel putin 5 dosare ale unor candidati ale caror date sunt preluate dintr-un fisier text. Cheia de cautare in tabela este **numeCandidat** iar mecanismul de tratare a coliziunilor este **chaining**. (1p)

2. Sa se afiseze elementele din tabela de dispersie, cu evidentierea pozitiilor ocupate in cadrul structurii. (0,5)

3. Implementati functia care determina numarul de candidati din tabela de dispersie create mai sus care au optat pentru un anumit program de studiu specificat ca parametru de intrare al functiei. Functia implementata se apeleaza in main(), iar rezultatul apelului se afiseaza la consola de executie a aplicatiei. (1,5p)

4. Implementati functia care salveaza intr-o structura de tip **lista de liste** candidatii din tabela de dispersie a caror medie de bacalaureat este sub un anumit prag specificat ca parametru de intrare al functiei, grupati pe fiecare program de studiu. Structura lista de liste NU partajeaza zone de memorie heap cu structura tabela de dispersie. Functia implementata se apeleaza in main(), iar rezultatul apelului se afiseaza in consola de executie a aplicatiei. (2p)

5. Implementati functia care sterge din lista de liste candidatul/candidatii cu cea mai mica medie de bacalaureat. Functia implementata se apeleaza in main(), iar rezultatul apelului se afiseaza in consola de executie a aplicatiei.

Tabela de dispersie- Task

O companie IT care realizeaza produse software implementeaza un sistem colaborativ cu privire loa task-urile asignate echipelor de programatori. In acest sens, structura unui task este: **id task** (string char*), **data asignarii** (string char*), **nume inginer software assignat** (string char*), **nivel complexitate** (numeric), **stare task** (valoare in multimea {deschis, in lucru, duplicat, rezolvat, inchis}). Gestionarea task-urilor se realizeaza pe baza unei aplicatii care implementeaza urmatoarele cerinte:

1. Datele aferente task-urilor sunt preluate dintr-un fisier text (1p) sau de la consola (0,5p);
2. Aplicatia incarca datele task-urilor intr-o structura de date dinamica de tip **tabela de dispersie**, in care mecanismul de evitare a coliziunilor este **chaining**; cheia de cautare este numele inginerului software assignat, iar functia hash trebuie sa tina cont de tipul asociat cheii (2,5 p);
3. Pentru siguranta, aplicatia contine o functie de afisare la consola pentru tabela de dispersie creata la punctul 2) cu evidentierea grupelor de inregistrari in punctele de coliziune (1,5 p);
4. Aplicatia modifica un numar arbitrar de task-uri din tabela de dispersie prin modificarea inginerului software assignat cu repozitionarea task-ului in cadrul tabelei; dupa efectuarea modificarilor, se apeleaza functia de afisare pentru confirmare, conform punctului 3); (2,5 p)
5. Aplicatia elimina task-urile cu starea “inchis” din tabela de dispersie (1p);
6. Aplicatia salveaza in fisierul cu date de intrare toate modificarile effectuate pe task-uri la punctul 4) (1,5 p);

Tabela de dispersie- Tichet

Implementati o aplicatie in limbajul C ce rezolva probleme de gestionare a tichetelor deschise intr-o companie.

1. Definiti structura Tichet ce contine: cod tichet (char*), descriere (char*), nume emitent (char*), data deschidere, stare (new/open/progress/fixed/closed)
2. Definiti functia de inserare a unui tichet intr-o Tabela de Dispersie. Creati o tabela de disperstie cu cel putin 5 tichete. Cheia de cautare in tabela este nume emitent, iar mecanismul de tartare a coliziunilor este chaining. (3p)
3. Definiti functia de traversare a structurii tabela de dispersie create la punctul 2. Vor fi afisate in consola aplicatiei doar listele de tichete stocate in tabela. (1p)
4. Definiti functia care determina numarul de tichete deschise de acelasi angajat (din tabela creata la punctul 2) al carui nume este specificat. Rezultatul executiei functiei este afisat in consola aplicatiei. (2p)
5. Definiti functia care determina tichetele (din tabela creata la punctul 2) deschise de un angajat specificat intr-o perioada de timp specificata. Rezultatul apelului de functie se afiseaza in consola aplicatiei. (2p)
6. Definiti functia care modifica starea X a unui tichet in starea Y. Tichetul este identificat pe baza codului, iar X si Y sunt cunoscute la run-time. Rezultatul apelului de functie se afiseaza in consola aplicatiei. (2p)

Coada de prioritati – Rezervare

Se considera codul sursa din Bilet_08.cpp care implementeaza o aplicatie pentru centralizarea rezervarilor clientilor unui restaurant. Rezervarile ajung in sistem prin doua modalitati: apel telefonic, respective aplicatie mobile. Rezervarile sunt onorate in ordinea prioritatii, data de durata rezervarii (numar de ore in care o masa este rezervata), indifferent de canalul pe care acestea sunt transmise.

1. Reprezentati graphic structura de tip coada de prioritati in care prioritatea este data de durata rezervarii unei mese. (1 p)
2. Scrieti functia pentru traversarea structurii de tip coada de prioritati. In consola se afiseaza datele aferente fiecarei rezervari. Rezultatul executiei este afisat la consola. (1p)
3. Modificati functia inserare astfel incat sa produca rezultatul correct (inserarea unei noi rezervari). Testarea functiei este realizata prin afisarea la consola a structurii in urma inserarii. (1p)
4. Scrieti functia pentru determinarea volumului de memorie ocupata aferent tuturor rezervarilor stocate in coada de prioritati. Rezultatul executiei functiei (volum memorie) este afisat la consola. Functia nu include operatii I/O cu tastatura/monitorul. (1p)
5. Scrieti functia pentru determinarea rezervarilor a caror durata depaseste 3 ore. Rezervarile rezultate sunt puse intr-un vector alocat dinamic. Functia nu include operatii I/O cu tastatura/monitorul. (1p)
6. Scrieti functia care permite generarea unui raport intr-un fisier text privind toate rezervarile effectuate de un anumit client, pe baza datelor din structura coada de prioritati. (2p)
7. Scrieti functia care extrage o rezervare din coada de prioritati. Rezultatul executiei functiei este afisat prin apelul functiei de traversare a structurii (punctul 2). Functia nu include operatii I/O cu tastatura/monitorul.

Coada de prioritati – Autocar

Implementati o aplicatie in limbajul C ce rezolva probleme de gestionare a locurilor disponibile la nivelul unei curse de autocar din cadrul unei companii de servicii turistice.

1. Definiti structura **LocAutocar** ce contine urmatoarele campuri: indicativ loc autocar (char*), numar rand (int), pret loc (float), stare loc (rezervat/disponibil). Definiti structura **CursaAutocar** care contine atributele: capacitate locuri (int), distanta km (float).
2. Sa se citeasca atributele unei curse de autocar si sa se creeze structura **coada de prioritate**, la nivelul unei curse de autocar, care sa cuprinda un set de minim 10 locuri citite dintr-un fisier. (3p)
3. Sa se scrie functia care mareste prioritatea unui pasager din perspectiva locului rezervat, daca acesta doreste sa avanseze spre locurile din fata autocarului. (1p)
4. Sa se scrie functia care returneaza costul mediu per kilometru al unui loc din autocar. (1p)
5. Sa se scrie functia care returneaza valoarea incasata pentru toate locurile rezervate la nivelul unei curse de autocar. (2p)
6. Sa se scrie functia care afiseaza la consola un raport privind distributia locurilor libere la nivelul fiecarui rand din autocar. (2p)

ABC -Examen

Sa se creeze un program in limbajul C ce rezolva problemele de gestionare a examenelor.

1. Definiti structura **Examen** ce consine: materie (char*), codExamen (unsigned int), numarul de credite ale materiei, numarul de student examinari, un vector cu notele obtinute de studentii examinati. (1p)
2. Definiti functia de inserare a unui examen intr-un **arbore binar de cautare**. Inserarea se realizeaza pe baza cheii **codExamen**. Apelati functia pentru cel putin 5 examene stocate intr-un fisier de intrare. (2p)
3. Definiti functia de traversare completa a arborelui in postordine. Rezultatul executiei functiei este afisat in consola aplicatiei. (1p)
4. Definiti functia care sterge un examen pe baza valorii codExamen primita ca parametru. Functia returneaza radacina arborelui actualizata sau nu (examenul este identificat sau nu in structura). (2p)
5. Definiti functia care returneaza promovabilitatea unui examen primit ca parametru prin codExamen. Rezultatul functiei este afisat dupa apelul functiei. (2p)
6. Definiti functia care determina examenele plasate pe un nivel specificat in arbore. Functia returneaza succesiunea de coduri de examen. Rezultatul functiei este afisat dupa apelul acesteia. (2p)

Reteta – Lista Dubla

Implementati o aplicatie in limbajul C ce rezolva probleme de gestionare a retetelor de medicamente emise de medici.

1. Definiti structura **Reteta** ce contine: numarul retetei (unsigned int), numarul de medicamente prescrise (unsigned char), lista de medicamente prescrise (char**), cantitatile medicamentelor prescrise (unsigned char*), preturile medicamentelor prescrise (float*), procentele de compensare ale medicamentelor prescrise exprimate in procente aplicate la pretul de vanzare (unsigned char*), numele medicului care a emis reteta (char*).
2. Scrieti si apelati functia de inserare a unei retete intr-o **Lista Dubla**. Creati structura **Lista Dubla** pentru cel putin 10 retele utilizand aceasta functie de inserare. Continutul listei duble se afiseaza la consola dupa creare prin parcurgerea listei double in ambele sensuri. (2p)
3. Scrieti si apelati functia pentru determinarea incasarilor aferente unui medicament (din lista dubla) specificat ca parametru de intrare. Rezultatul se afiseaza la consola. (1,5p)
4. Scrieti si apelati functia care copiaza intr-un **Vector** retetele emise de un medic (din lista dubla) specificat ca parametru de intrare. Cele doua structuri de date **NU** partajeaza zone de memorie heap. Continutul vectorului se afiseaza la consola dupa creare. (2p)
5. Scrieti si apelati functia pentru determinarea valorii compensate a medicamentelor din vectorul create la punctul 4). Rezultatul se afiseaza la consola. (1,5p)
6. Scrieti si apelati functia pentru determinarea cantitatii vandute pentru un medicament din lista dubla. Denumirea medicamentului este specificata ca parametru de intrare. Rezultatul se afiseaza la consola. (1p)
7. Scrieti secventa de cod care dezaloca structurile **Lista Dubla** si **Vector** creare la punctele anterioare. (1p + 1p)

Depozit – Graf

Sa se creeze un program in limbajul C ce rezolva probleme legate de gestionarea depozitelor unei companii de logistica

1. Definiti structura **Depozit** ce contine: id depozit (char*), capacitate de stocare (volum exprimat in metri cubi), locatie (coordonate GPS), lista categorii materiale ce pot fi stocate in depozit (char*), procent volum disponibil pentru depozitare. (1p)
2. Definiti functia de creare a unei structuri de tip Graf Depozite. Apelati functia pentru cel putin 7 depozite stocate intr-un fisier de intrare alaturi de legaturile directe (infrastructura rutiera) dintre acestea. (2p)
3. Definiti functia de afisare completa a grafului de depozite, inclusiv legaturile directe dintre acestea. Rezultatul executiei functiei este afisat la consola aplicatiei. (1p)
4. Definiti functia care traverseaza Depth-First graful de depozite. Functia returneaza succesiunea de id-uri depozite aferenta traversarii. Rezultatul functiei (succesiune id-uri) este afisat la consola dupa terminarea executiei acesteia. (2p)
5. Definiti functia care determina depozitele avand incarcare minima de 60%. Functia returneaza succesiunea de id-uri depozite care satisfac aceasta conditie. Rezultatul functiei (succesiune id-uri) este afisat la consola dupa terminarea executiei acesteia. (2p)
6. Definiti functia care determina lista tuturor materialelor care pot fi stocate in depozitele companiei de logistica. Functia returneaza lista materialelor care este afisata la consola dupa terminarea executiei functiei. (2p)

Bicicleta – ABC

Sa se creeze un program in limbajul C ce rezolva probleme de utilizare a bicicletelor publice intr-un oras civilizat.

1. Definiti structura **Bicicleta** ce contine: numarul de identificare (int), durata de utilizare intr-o zi (minute) (int), id statie de parcare (int), munar de utilizari pe zi (int), numele utilizatorului (char*). (1p)
2. Definiti functia de inserare a unei biciclete intr-o structura de tip **arbore binar**, in care inserarea elementelor se face in ordine crescatoare a duratei de utilizare. Apelati functia pentru cel putin 5 bibiclete stocate intr-un fisier de intrare. (2p)
3. Definiti functia de traversare a structurii de tip arborescent, iar rezultatul executiei functiei este afisat in consola aplicatiei, incluzand toate detaliile structurii **Bicicleta** (1p)
4. Definiti functia care calculeaza numarul total de minute in care bicicletele au fost utilizate pe parcursul unei zile. Rezultatul executiei functiei este afisat in consola aplicatiei. (2p)
5. Definiti functia care calculeaza valoarea totala incasata de compania ce inchiriaza bicicletele in cadrul unei zile, stiind ca tariful pe minut este de 2 lire, iar primele 10 minute sunt gratuite pentru fiecare utilizare. Se iau in considerare toate bicicletele stocate in structura de tip arborescent. Rezultatul executiei functiei este afisat in consola aplicatiei. (2p)
6. Definiti functia care determina numarul distinct de statii de parcare prin care au trecut bicicletele stocate in arbore. Rezultatul functiei este afisat in consola aplicatiei. (2p)

Bilet #15 / (120 min) | **16.06.2020**

Implementaţi o aplicaţie în limbajul C care implementează soluţii la probleme de localizare geografică.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de tip ***Graf*** implementată prin ***Listă de Adiacenţă.*** Inserarea unei muchii în graf se implementează în funcţia principală pentru un număr de minim 6 localităţi citite dintr-un fişier de intrare. **(3p)**

   Structura ***Localitate*** se va defini astfel încât să conţină minim 5 câmpuri din care minim două câmpuri sunt definite ca variabile pointer.

2. Scrieţi şi apelaţi funcţia pentru determinarea localităţilor din ***Graf*** cu număr maxim de conexiuni cu alte localităţi. Funcţia returnează un vector care conţine denumirea localităţilor cu maximul de conexiuni. (**2p**)

3. Scrieţi secvenţa de cod care copiază datele localităţilor din structura ***Graf*** creată la punctul 1) într-o structură ***Arbore Binar de Căutare***. Sunt considerate localităţile a căror denumire încep cu o literă specificată. Cele două structuri de date ***NU*** partajează zone de memorie heap. **(2p)**

4. Scrieţi şi apelaţi funcţia pentru extragerea într-un vector a localităţilor plasate în nodurile frunză în ***Arborele Binar de Căutare*** creat la punctul 3). Nodurile frunză sunt şterse din structura arborescentă. **(2p)**

5. Scrieţi secvenţa de cod care dezalocă structurile ***Listă de Adiacenţă, Vectori, Arbore Binar de Căutare*** şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este necesar). **(1p)**

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerintele trebuie apelate si demonstrate in functia main() pentru a fi evaluate.**

Bilet #6 / (150 min) | **11.06.2021**

Implementaţi o aplicaţie în limbajul C care implementează soluţii la probleme de gestionare a comenzilor online ale unui restaurant.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de date de tip ***Arbore binar de cautare*** ce conţine date aferente comenzilor de mancare**.** Cheia de căutare utilizată este ***id_comanda***. Inserarea unei comenzi se implementează într-o funcţie care se apelează în secvenţa de creare a structurii ***Arbore binar de cautare***. Structura ***Arbore binar de cautare*** conţine minim 10 comenzi incarcate in aplicatie dintr-un fisier de intrare.
   Structura ***Comanda*** se va defini astfel încât să conţină minim 7 câmpuri, astfel: ***timp_livrare (int)***, ***cod_client (int), id_comanda (int)***; celelalte 4 campuri sunt definite la alegere, din care minim unul este de tip ***char****.

   Cerinţe de implementare:
   - Definire structură ***Comanda***. ***(0,25p)***
   - String-urile preluate din fişier trebuie să accepte prezenţa simbolului ***blank***. ***(0,25p)***
   - Absenţă memory leaks. ***(0,25p)***
   - Implementare logică de creare structură ***Arbore binar de cautare***. ***(0,75p)***
   - Populare completă şi corectă a structurii ***Arbore binar de cautare*** cu date de intrare din fisier. ***(0,25p)***
   - Testare implementare cu afisarea la consola a continutului structurii ***Arbore binar de cautare***. ***(0,25p)***

2. Scrieţi şi apelaţi funcţia pentru determinarea comenzilor din structura creată la cerinta 1) care au timpul de livrare mai mare decat o valoare specificata ca parametru de intrare al functiei. Comenzile identificate sunt salvate într-un vector şi ***NU*** partajează zone de memorie heap cu structura ***Arbore binar de cautare***. Vectorul se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei.

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Realizare deep-copy a comenzilor în vector. ***(0,25p)***
   - Implementare logică de determinare şi salvare a comenzilor în vector. ***(1,00p)***
   - Populare completă şi corectă a vectorului. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut la apel. ***(0,25p)***

3. Scrieţi şi apelaţi funcţia pentru determinarea comenzilor cu cea mai mare prioritate de servire din ***Arborele binar de cautare***. Implementarea presupune copierea comenzilor intr-o structura ***Heap*****,** unde prioritatea este data de timpul de livrare**.** Arborele si structura Heap ***NU*** partajează zone de memorie. Structura ***Heap*** se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei.

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Implementare mecanism filtrare ***Heap***. ***(0,50p)***
   - Implementare inserare element in ***Heap***. ***(0,50p)***
   - Implementare extragere element din ***Heap***. ***(0,50p)***
   - Populare completă şi corectă a structurii ***Heap***. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut la apel. ***(0,25p)***

4. Scrieţi şi apelaţi funcţia pentru determinarea valorii totale a comenzilor la nivel de client. Se iau in considerare comenzile salvate in structura ***Arbore binar de cautare*** de la la cerinta 1). Un client poate avea mai multe comenzi la restaurant. Perechile de valori ***(cod_client, suma_totala)*** sunt salvate intr-un vector. Vectorul si dimensiunea acestuia se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei.

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Determinare valori ***(cod_client, suma_totala)*** pentru comenzile obtinute pe baza structurii de la cerinta 1). ***(0,75p)***
   - Implementare logică de creare vector cu valori ***(cod_client, suma_totala)***. ***(1,25p)***
   - Populare completă şi corectă a vectorului. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut la apel. ***(0,25p)***

5. Scrieţi şi apelaţi funcţiile care dezalocă structurile ***Arbore binar de cautare, Heap*** si ***2 x Vectori*** precum şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este cazul).

   Cerinţe de implementare:
   - Definire funcţii cu parametri de I/O definiţi complet şi corect. ***(0,15p)***
   - Absenţă memory leaks. ***(0,15p)***
   - Actualizare variabile de gestionare a structurilor în funcţia ***main()***. ***(0,20p)***
   - Implementare logică de dezalocare a structurilor de date. ***(0,30p)***
   - Testare implementare, dezalocare completă şi corectă a structurilor prin apel de functii si afisare la consola a rezultatelor obtinute la apel. ***(0,20p)***
   - Absenţă dezalocări structuri auxiliare utilizate. ***(-0,20p)***

Bilet #6 / (150 min) | **11.06.2021**

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările nu trebuie să conţină variabile definite la nivel global sau statice.**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerinţele trebuie apelate şi demonstrate in functia main() pentru a fi evaluate.**
- **Art. 72 (1) Pentru următoarele fapte, studenţii vor fi exmatriculaţi fără drept de reînmatriculare în Academia de Studii Economice din Bucureşti:**
  (c) încercarea de promovare prin fraudă a examenelor sau a altor evaluări;

Implementaţi o aplicaţie în limbajul C care gestioneaza evaluarile studentilor in cadrul unei sesiuni de examinare.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de date de tip ***Tabelă de Dispersie*** ce conţine date aferente evaluarilor folosind ca mecanism de tratare a coliziunilor, ***Chaining***. Tabela de dispersie stocheaza elemente de tip ***Evaluare*.*** Cheia de cautare utilizata este *denumireExamen***(char*).** Alte attribute necesare in cadrul structurii Evaluare: *numeStudent*(**char***), *semestru* (valori valide 1/2), *notaFinala*(**float**) si alte 2 atribute la alegere. Tabela de dispersie conţine minim 10 evaluari incarcate in aplicatie dintr-un fisier de intrare. ***(2p)***

   Cerinţe de implementare:
   - Definire structură ***Evaluare***. ***(0,25p)***
   - String-urile preluate din fişier trebuie să accepte prezenţa simbolului ***blank***. ***(0,25p)***
   - Absenţă memory leaks. ***(0,25p)***
   - Implementare logică de creare structură ***Tabelă de Dispersie*** cu ***Chaining***. ***(0,75p)***
   - Populare completă şi corectă a structurii ***Tabelă de Dispersie*** cu date de intrare din fisier. ***(0,25p)***
   - Testare implementare cu afisarea la consola a continutului structurii ***Tabelă de Dispersie***. ***(0,25p)***

2. Scrieţi şi apelaţi funcţia pentru determinarea evaluarilor din structura creată la cerinta 1) prin filtrarea acestora pe baza valorii unuia din atributele optionale. Valoarea este trimisa ca parametru functiei. Evaluarile sunt salvate intr-un vector fara ca acesta sa partajeze zone de memorie heap cu elementele din tabela de dispersie. Vectorul si dimensiunea acestuia sunt returnate in ***main()*** prin tipul de retur sau lista de parametri ai funcţiei. **(2p)**

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Realizare deep-copy a produselor în vector. ***(0,25p)***
   - Implementare logică de determinare şi salvare a produselor în vector. ***(1p)***
   - Populare completă şi corectă a vectorului. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut la apel. ***(0,25p)***

3. Scrieţi şi apelaţi funcţia pentru determinarea numarului de examene promovate pentru fiecare cluster de coliziuni din ***Tabela de Dispersie***. Rezultatul se va stoca intr-un vector in care fiecare element contine perechea de valori (**index cluster, numar examene promovate**). Vectorul si dimensiunea acestuia se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei. **(2.5p)**

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Determinare dimensiune vector. ***(0,25p)***
   - Implementare logică de determinare a elementelor în vector. ***(1,25p)***
   - Populare completă şi corectă a vectorului. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut la apel. ***(0,50p)***

4. Scrieţi şi apelaţi funcţia pentru crearea unei matrice care stocheaza evaluarile grupate dupa atributul semestru. Structura **partajeaza** zone de memorie heap cu tabela de dispersie la nivelul evaluarilor stocate. Structura rezultata este returnata in ***main()*** iar continutul este afisat la consola. **(2.5p)**

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Determinare elementelor ce trebuie inserate in matrice. ***(0,25p)***
   - Implementare logică de creare a matricei; fiecare din cei 2 vectori (pentru sem.1/2) sunt alocati dinamic. ***(1p)***
   - Populare completă şi corectă a matricei cu partajarea memorie heap cu tabela de dispersie. ***(0,50p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut la apel. ***(0,50p)***

5. Scrieţi şi apelaţi funcţiile care dezalocă structura principala ***Tabelă de Dispersie, vectorii rezultati*** in implementarea cerintelor precum şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este cazul). **(1p)**

   Cerinţe de implementare:
   - Definire funcţii cu parametri de I/O definiţi complet şi corect. ***(0,15p)***
   - Absenţă memory leaks. ***(0,15p)***
   - Actualizare variabile de gestionare a structurilor în funcţia ***main()***. ***(0,20p)***
   - Implementare logică de dezalocare a structurilor de date. ***(0,30p)***
   - Testare implementare, dezalocare completă şi corectă a structurilor prin apel de functii si afisare la consola a rezultatelor obtinute la apel. ***(0,20p)***
   - Absenţă dezalocări structuri auxiliare utilizate. ***(-0,20p)***

NOTES:

- **Projects with compilation errors won't be evaluated.**
- **Implementations must not contain globally defined or static variables.**
- **Implementations must not use predefined structures such as STL or 3rd party libraries.**
- **Plagiarized implementations will be evaluated with 0 points, regardless of the source.**
- **All requirements must be called and demonstrated in the main () function to be evaluated.**
- **Art. 72 (1) For the following facts, students will be expelled without the right to re-enroll in the Academy of Economic Studies in Bucharest:**
  - **(c) attempting to fraudulently pass examinations or other assessments;**

Bilet #4 / (150 min) | **11.06.2021**

Implementaţi o aplicaţie în limbajul C care implementează soluţii la probleme de gestionare a a conturilor bancare deschise la o bancă comercială.

1. Scrieţi secvenţa de cod sursă pentru crearea unei structuri de date de tip ***Tabelă de Dispersie*** ce conţine date aferente conturilor bancare***.*** Cheia de căutare utilizată este ***nume_client***, iar mecanismul de tratare a coliziunilor este ***Chaining***. Inserarea unui cont bancar se implementează într-o funcţie care se apelează în secvenţa de creare a structurii ***Tabelă de Dispersie***. Tabela de dispersie conţine minim 10 conturi bancare incarcate in aplicatie dintr-un fisier de intrare.
   Structura ***ContBancar*** se va defini astfel încât să conţină minim 7 câmpuri, astfel: ***nume_client (char*)***, ***sold (float), valuta_cont (char*)***; celelalte 4 campuri sunt definite la alegere.

   Cerinţe de implementare:
   - Definire structură ***ContBancar***. ***(0,25p)***
   - String-urile preluate din fişier trebuie să accepte prezenţa simbolului ***blank***. ***(0,25p)***
   - Absenţă memory leaks. ***(0,25p)***
   - Implementare logică de creare structură ***Tabelă de Dispersie*** cu ***Chaining***. ***(0,75p)***
   - Populare completă şi corectă a structurii ***Tabelă de Dispersie*** cu date de intrare din fisier. ***(0,25p)***
   - Testare implementare cu afisarea la consola a continutului structurii ***Tabelă de Dispersie***. ***(0,25p)***

2. Scrieţi şi apelaţi funcţia pentru determinarea conturilor bancare din structura creată la cerinta 1) care au valuta specificata ca parametru de intrare al functiei. Conturile bancare identificate sunt salvate într-un vector şi ***NU*** partajează zone de memorie heap cu structura ***Tabelă de Dispersie***. Vectorul se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei.

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Realizare deep-copy a conturilor bancare în vector. ***(0,25p)***
   - Implementare logică de determinare şi salvare a conturilor în vector. ***(1,00p)***
   - Populare completă şi corectă a vectorului. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut la apel. ***(0,25p)***

3. Scrieţi şi apelaţi funcţia pentru determinarea numarului si a dimensiunilor (exprimate ca numar de conturi bancare) pentru toate cluster-ele de coliziuni din ***Tabela de Dispersie***. Cluster-ele identificate sunt salvate într-un vector in care fiecare element contine perechea de valori ***(dimensiune_cluster, index_tabela).*** Vectorul si dimensiunea acestuia se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei.

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Determinare numar cluster-e. ***(0,25p)***
   - Determinare dimensiuni si indecsi cluster-e. ***(0,25p)***
   - Implementare logică de determinare şi salvare a cluster-elor în vector. ***(1,00p)***
   - Populare completă şi corectă a vectorului. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut la apel. ***(0,25p)***

4. Scrieţi şi apelaţi funcţia pentru determinarea soldurilor bancare totale la nivel de client. Se iau in considerare conturile bancare salvate in vectorul de la la cerinta 2). Un client poate avea deschise mai multe conturi bancare avand aceeasi valuta. Perechile de valori ***(nume_client, sold_total)*** sunt salvate intr-un vector. Vectorul si dimensiunea acestuia se returnează în ***main()*** prin tipul de retur sau lista de parametri ai funcţiei.

   Cerinţe de implementare:
   - Definire funcţie cu parametri de I/O definiţi complet şi corect. ***(0,25p)***
   - Determinare valori ***(nume_client, sold_total)*** pentru conturile bancare obtinute la cerinta 2). ***(0,75p)***
   - Implementare logică de creare vector cu valori ***(nume_client, sold_total)***. ***(1,25p)***
   - Populare completă şi corectă a vectorului. ***(0,25p)***
   - Testare implementare prin apel de functie si afisare la consola a rezultatului obtinut la apel. ***(0,25p)***

5. Scrieţi şi apelaţi funcţiile care dezalocă structurile ***Tabelă de Dispersie*** si ***3xVectori*** precum şi toate structurile auxiliare utilizate în implementarea cerinţelor (dacă este cazul).

   Cerinţe de implementare:
   - Definire funcţii cu parametri de I/O definiţi complet şi corect. ***(0,15p)***
   - Absenţă memory leaks. ***(0,15p)***
   - Actualizare variabile de gestionare a structurilor în funcţia ***main()***. ***(0,20p)***
   - Implementare logică de dezalocare a structurilor de date. ***(0,30p)***
   - Testare implementare, dezalocare completă şi corectă a structurilor prin apel de functii si afisare la consola a rezultatelor obtinute la apel. ***(0,20p)***
   - Absenţă dezalocări structuri auxiliare utilizate. ***(-0,20p)***

Bilet #4 / (150 min) | **11.06.2021**

MENTIUNI:

- **Proiectele cu erori de compilare nu vor fi evaluate.**
- **Implementările nu trebuie să conţină variabile definite la nivel global sau variabile statice.**
- **Implementarile nu trebuie sa contina structuri predefinite (ex STL, 3rd party libraries etc).**
- **Implementările plagiate vor fi evaluate cu 0 puncte, indiferent de sursă.**
- **Toate cerinţele trebuie apelate şi demonstrate in functia main() pentru a fi evaluate.**
- **Art. 72 (1) Pentru următoarele fapte, studenţii vor fi exmatriculaţi fără drept de reînmatriculare în Academia de Studii Economice din Bucureşti:**
  (c) încercarea de promovare prin fraudă a examenelor sau a altor evaluări;

Implement a C application for managing the airplanes' landings in an airport with only one single runway. For this, you can use a Binary Search Tree structure (BST). Structure ***Flight***, which is the useful information of a tree node, is created with the following attributes: airplane code(char*), ***landing time (unsigned short) – expressed in minutes***, coming from (char*), no. of passengers (unsigned short).

1. Print the reversed order of landings. **0.5p**
2. Find the landing that has the maximum number of passengers and return it for display in the main section. **1p**
3. Print the planes that land in a given time interval [x; y]; x and y represent the number of minutes given as parameters to the function. **1p**
4. Find the *next plane to land* and remove it; print the remaining elements after. **1p**
5. How many planes are scheduled to land at times <= t (t given value as a parameter). **1p**
6. Save all the entries starting from the root of the tree all the way to a given leaf (given as a parameter by its landing time) in an array of pointers to be displayed (the array doesn't share memory space with the BST implementation)**. 1.5p**

The following items should be considered for the implementation:

- ***Projects with compilation issues are NOT going to be evaluated;***
- ***Functions that are not tested in the main() function are not taken into account at evaluation;***
- ***Source code that is commented is NOT going to be evaluated;***

Subject #2E | 15.06.2022

Implement a C application for managing the queue of processes that need to be executed by a CPU. For this, you can use a Queue structure that handles processes with the following structure: process name (char*), priority (unsigned short), needed memory (unsigned int), and execution time (unsigned short).

1. Print all the processes in the queue without changing the queue structure and only by using queue-specific operations. **0.5p**
2. Process ***only*** the processes with a specific priority (value given as a parameter to the function); processing a process means removing it from the initial structure, displaying it at the console, and releasing the occupied memory at the end. The ones that do not meet the criteria will be kept in the queue. **1p**
3. Extract all the processes and save them into a circular simple linked list by inserting them in descending order by the amount of memory that they need. **1p (-0.5 if it is not ordered)**
4. Determine which are the processes from the newly created list that have the lowest resource consumption in terms of memory and time and display them to the console. **1p**
5. Delete the processes from the previous list that have the needed memory between [base-x; base+x]; *base* and *x* are two given parameters to the function; **1p**
6. Split the previous list into two separate lists based on a given process name, so that the first list will include all the processes from the start of the list up to the given process and the second list will include the rest of the processes; print the two newly created lists; **1.5p**

The following items should be considered for the implementation:

- ***Projects with compilation issues are NOT going to be evaluated;***
- ***Functions that are not tested in the main() function are not taken into account at evaluation;***
- ***Source code that is commented is NOT going to be evaluated;***

Implement a C-language application that handles hotel room management.

1. Write the source code sequence to create a Binary Search Tree structure that represents a hotel with different types of rooms. The insertion key is a **composite key** made of two fields: floor and room number. The insertion of a room is implemented in the main function for at least 10 elements taken from an input file. (**1,5p**)

   The Room structure will be defined so that it should be made of the following fields: floor (Roman numerals), room number, room type, price per night.

2. Write and call the function for extracting the subtree whose root is received as a parameter by specifying the composite insertion key. After extraction, the initially remaining tree will be displayed, as well as the extracted one. (**1p**)

3. Write and call the function for determining the total income for a certain type of room on a certain floor of the hotel, values received as parameters, assuming that the hotel is fully booked. (**1,5p**)

4. Write and call the function to display all the elements in the tree structure grouped by each level in the tree. (**2p**)

5. Write and call the function to display all the rooms grouped by each floor of the hotel, having O(n) complexity for the binary search tree structure. (**3p**)

6. Write the code sequences that free the Binary Search Tree structure along with all the auxiliary structures used for implementing the requirements. (**1p**)

NOTES:

- **Projects with compilation errors will not be evaluated.**
- **Plagiarized implementations will be evaluated with 0 points, regardless of the source.**
- **All requirements must be called and demonstrated in the main () function to be evaluated.**