SEMINAR 3
STRUCTURI. UNIUNI. ENUMERĂRI. CÂMPURI DE BIŢI.

Probleme rezolvate

1. *Definiţi o structură în care să reţineţi datele unei persoane referitoare la CNP, nume, prenume, stare civilă, adresă.*

Rezolvare:

Definim mai întâi structura `Adresa` considerând diverşi membri care definesc adresa unei persoane. În cazul persoanelor care stau în Bucureşti, judeţul este înlocuit de sector. Modelăm acest fapt folosind o uniune în interiorul structurii. Membrul activ în această uniune va fi dat de numele oraşului.

```
struct Adresa
{
  char strada[50];
  unsigned short int numar_strada;
  char bloc[8];
  char scara[2];
  unsigned char etaj;
  unsigned short int apartament;
  char oras[30];
  union
  {
      char judet[30];
      char sector;
  };
  char cod_postal[6];
};
```

Definim apoi starea civilă ca o enumerare cu două posibile valori:

```
enum stare_civila {necasatorit, casatorit};
```

În final, definim structura Persoana cu membri nume, prenume, CNP, stare civila şi adresa.

```
struct Persoana
{
  char nume[30];
  char prenume[30];
  char CNP[13];
  enum stare_civila s;
  struct Adresa adresa;
};
```

2. *Definiţi o structură Camera care să permită memorarea numărului de ordine a unei camere dintr-o pensiune cu capacitatea de 10 de camere, numărul de paturi din cameră (există camere cu 1, 2 sau 3 paturi), dacă respectiva cameră este ocupată sau nu și dacă este achitată sau nu. Cum aţi defini această structură astfel încât să ocupe cât mai puţini octeţi în memorie?*

Rezolvare:

Definim structura *Camera* având ca membri numărul camerei, numărul de paturi (ambele de tipul `int`) și variabilele de tip `_Bool` ce indică dacă o camera este ocupată și achitată.

```
struct Camera
{
  int numar_camera;
  int numar_paturi;
  _Bool ocupata;
  _Bool achitata;
};
```

O astfel de structură ocupă în memorie 12 octeţi: câte 4 octeţi pentru fiecare membru de tip int și câte 1 octet pentru fiecare membru de tip `_Bool`. Alţi 2 octeţi sunt adăugaţi pentru aliniere. Alinierea are rolul de a spori viteza de citire a datelor din memorie, spre exemplu în cazul tablourilor cu elemente structuri de tip *Camera*. Folosind câmpurile de biţi putem defini structura *Camera* astel încât ea să ocupe un singur octet. Definim toţi cei 4 membri ai structurii ca fiind de tipul `unsigned char` cu o anumită dimensiune în biţi în funcţie de domeniul de valori.

```
struct Camera1
{
  unsigned char numar_camera:4;
  unsigned char numar_paturi:2;
  unsigned char ocupata:1;
  unsigned char achitata:1;
};
```

3. *Să se afișeze reprezentarea binară internă a unei valori de tip real (float sau double).*

Rezolvare:

Pentru a afișa reprezentarea binară internă a unei valori x de tip real (`float` sau `double`) vom folosi o uniune ce are ca membri valoarea x și un tablou de lungime sizeof(x) cu elemente de tipul `unsigned char`.

```
union
{
      float f;
      unsigned char c[sizeof(float)];
} tip_f;
```

```
union
{
	double d;
	unsigned char c[sizeof(double)];
} tip_d;
```

Astfel, afișarea reprezentării binare a valorii x revine la afișarea reprezentării interne a elementelor tabloului c. Vom folosi o funcție similară cu cea din seminarul trecut, folosind o mască m pentru afișarea reprezentării binare interne a fiecărui octet. Masca m, inițial, are cel mai semnificativ bit egal cu 1, iar restul biților vor fi nuli. După ce vom afișa valoarea bitului cel mai semnificativ al octetului curent, vom deplasa în m bitul egal cu 1 cu o poziție spre dreapta și vom afișa valoarea bitului corespunzător . Vom repeta acest procedeu până când vom afișa și bitul cel mai puțin semnificativ din reprezentarea binară internă a octetului curent.

```
void afiseazaScriereBinara(unsigned char a)
{
	unsigned char m = 1ULL << (8*sizeof(unsigned char)-1);
	unsigned char b;
	for(b = 0; b < 8*sizeof(unsigned char); b++){
		printf("%u" , (a & m) != 0);
		m = m >> 1;
	}
}
```

Programul principal afișează scrierea binară a numărului 7 stocat inițial într-o variabilă de tip `float` și apoi stocat într-o variabilă de tip `double`.

```
int main()
{
	tip_f.f = 7;
	int i;
	printf("%f ca float in baza 2 se scrie \n",tip_f.f);
	for(i = 0; i < sizeof(float); i++)
		afiseazaScriereBinara(tip_f.c[i]);

	tip_d.d = 7;
	printf("\n%f in baza 2 ca double se scrie \n",tip_f.f);
	for(i = 0; i < sizeof(double); i++)
		afiseazaScriereBinara(tip_f.c[i]);

	return 0;
}
```

Reprezentarea binară a numerelor mai mari de un octet (de tipul `int`, `float`, `double`) determină arhitectura unui procesor (little-endian sau big-endian). În reprezentarea little-endian cel mai puțin semnificativ octet din reprezentare este memorat primul, diferit de reprezentarea big-endian în care cel mai semnificativ octet este memorat primul.

4. *Definiţi o structură* `Produs` *care să permită memorarea numelui unui produs, a preţului său unitar şi a cantităţii existente din produsul respectiv într-un depozit. Scrieţi o funcţie care să ieftinească cu* 10% *toate produsele aflate într-un depozit al căror stoc depăşeşte un număr real* s *(stocul unui produs este egal cu produsul dintre preţ şi cantitate). Informaţiile despre produsele din depozitul respectiv sunt memorate într-un tablou* t *cu* n *elemente de tip* `Produs`*. Să se sorteze folosind funcţia* `qsort` *elementele unui tablou unidimensional* t *format din* n *elemente de tip* `Produs` *în ordinea descrescătoare a preţurilor unitare, iar produsele având acelaşi preţ unitar se vor ordona alfabetic.*

Rezolvare:

Definim iniţial structura *Produs* având ca membri nume, pret şi cantitate. Preţul unui produs este de tip `float` şi nu `int`, întrucât după ieftinirea cu 10% a unui preţ întreg am obţine tot un preţ întreg, ceea ce ar fi incorect.

```
struct Produs
{
   char nume[25];
   float pret;
   float cantitate;
};
```

Funcţia pentru ieftinirea produselor primeşte ca parametri numele tabloului cu elemente de tipul *Produs*, lungimea n a acestui tablou şi valoarea s a stocului pentru care se realizează reducerea de preţ.

```
void reducePretul(struct Produs t[],int n,float s)
{
   for(int i=0; i < n; i++)
       if (t[i].pret * t[i].cantitate > s)
             t[i].pret = 0.9 * t[i].pret;
}
```

Pentru sortarea cu funcţia `qsort` a tabloului t de elemente de tipul *Produs* avem nevoie de o funcţie comparator. Funcţia `cmpProduse` este scrisă mai jos şi implementează cerinţa problemei. Această funcţie compară două elemente de tipul *Produs* în funcţie de preţul lor, iar la preţ egal compară numele lor. Funcţia primeşte doi pointeri generici a şi b pe care îi converteşte la tipul struct *Produs*. Întrucât vrem ca tabloul sortat să fie sortat descrescător în funcţie de preţurile unitare, funcţia `cmpProduse` întoarce valoarea -1 dacă preţul unitar al produsului către care pointează a este mai mare decât preţul produsului către care pointează b. Valoarea -1 înseamnă că în tabloul sortat, produsul către care pointează a (în tabloul iniţial) se va regăsi înaintea (la stânga) produsului către care pointează b (în tabloul iniţial). Funcţia `cmpProduse` întoarce valoarea +1 dacă preţul unitar al produsului către care pointează a este mai mic decât preţul produsului către care pointează a. Dacă preţurile sunt egale atunci funcţia întoarce rezultatul funcţiei `strcmp` ce compară numele numele produselor în ordine lexicografică.

```
int cmpProduse(const void *a, const void* b)
{
  if (((struct Produs *)a)->pret == ((struct Produs *)b)->pret)
        return strcmp(((struct Produs *) a)->nume,((struct Produs *)b)->nume);
    if (((struct Produs *)a)->pret > ((struct Produs *)b)->pret)
        return -1;
    return +1;
}
```

Apelul funcției `qsort` pentru un tablou t de n elemente de tipul *Produs* arată astfel:

```
qsort(t, n, sizeof(struct Produs), cmpProduse);
```

5. *Fie $n \in \mathbb{N}$ și fie n triplete (A_i, B_i, C_i) de puncte în plan cu $i = 1, \dots, n$. Fiecare punct are coordonatele, abscisa și ordonata, numere reale. Să se afișeze aria cea mai mare a unui triunghi, dacă acesta există, sau mesajul „Nu există" dacă niciun triplet de puncte citit nu formează un triunghi.*

Rezolvare:

Definim structurile `punct2D` (având membri coordonatele punctelor) și `triunghi` (având membri cele trei puncte care sunt vârfurile triunghiului, lungimile celor trei laturi, o variabilă booleană care validează sau nu existența triunghiului, perimetrul și semiperimetrul folosit la calculul ariei):

```
typedef struct {
  float abscisa;
  float ordonata;
} punct2D;

typedef struct{
  punct2D  A, B, C;
  float AB, AC, BC;
  _Bool triunghiValid;
  float perimetru;
  float arie;
}  triunghi;
```

Citirea tripletelor (vârfurilor triunghiurilor) se face citind rând pe rând abscisa și ordonata fiecărei punct. Toate cele n triplete sunt stocate în tabloul L ce are elemente triunghiuri:

```
void citestePuncte(triunghi L[],int n)
{
    for(int i = 0; i < n; i++)
        scanf("%f %f %f %f %f %f",&L[i].A.abscisa, &L[i].A.ordonata,
            &L[i].B.abscisa,&L[i].B.ordonata,&L[i].C.abscisa, &L[i].C.ordonata);
}
```

Calcul ariei fiecărui triunghi se face folosind formula lui Heron pe baze semiperimetrului (detalii aici https://ro.wikipedia.org/wiki/Formula_lui_Heron). Triunghiul este valid dacă vârfurile nu sunt coliniare sau identice, condiție ce este echivalentă cu aceea că suma oricăror două laturi este diferită de a treia.

```
void verificaTriunghi(triunghi L[],int n)
{
    for(int i = 0; i < n; i++)
    {
        L[i].AB = sqrt(pow(L[i].A.abscisa - L[i].B.abscisa,2) +
            pow(L[i].A.ordonata - L[i].B.ordonata,2));
        L[i].AC = sqrt(pow(L[i].A.abscisa - L[i].C.abscisa,2) +
            pow(L[i].A.ordonata - L[i].C.ordonata,2));
        L[i].BC = sqrt(pow(L[i].B.abscisa - L[i].C.abscisa,2) +
            pow(L[i].B.ordonata - L[i].C.ordonata,2));
        //validare triunghi
        if ((L[i].AB + L[i].BC == L[i].AC) ||
            (L[i].AB + L[i].AC == L[i].BC) ||
            (L[i].AC + L[i].BC == L[i].AB))
            L[i].triunghiValid = 0;
        else
            L[i].triunghiValid = 1;
        if(L[i].triunghiValid)
        {   L[i].perimetru = L[i].AB + L[i].AC + L[i].BC;
            float sp = L[i].perimetru/2;
            L[i].arie=sqrt(sp * (sp - L[i].AB)  * (sp - L[i].BC) * (sp - L[i].AC));
        }
    }
}
```

Afișarea ariei maxime sau a mesajului „Nu exista” se face imediat:

```
void afiseazaArie(triunghi L[], int n)
{
    _Bool valid = 0;
    float arieMaxima = 0;
    for(int i = 0; i < n; i++)
    {
        if(L[i].triunghiValid)
        {
            valid = 1;
            if (arieMaxima < L[i].arie)
                arieMaxima = L[i].arie;
        }
    }
    valid ? printf("Aria maxima este %f\n",arieMaxima) : printf("Nu exista \n");
}
```

Programul principal este reprezentat de apeluri ale funcţiilor de mai sus:

```
int main()
{
    int n;
    printf("n = "); scanf("%d",&n);

    triunghi T[n];
    citestePuncte(T,n);
    verificaTriunghi(T,n);
    afiseazaArie(T,n);
    return 0;
}
```

O altă rezolvare presupune sortarea tablouri de triunghiuri luând în considerare numai triunghiurile valide. Sortarea se face descrescător în funcţie de aria triunghiului. O posibilitate este folosirea funcţiei `qsort` din librăria `stdlib.h`, în acest caz avem nevoie de definiţia unei funcţii comparator:

```
int cmpTriunghiuri(const void *a, const void *b)
{
 triunghi *pa = (triunghi *) a;
 triunghi *pb = (triunghi *) b;
 if(pa->triunghiValid == pb->triunghiValid)
 {
    if (pa->triunghiValid ==0)
        return 0;
    return pb->arie - pa->arie;
}
if (pa->triunghiValid)
    return -1;
return 1;
}
```

Programul principal de modifică astfel:

```
int main()
{
    int n;
    printf("n = "); scanf("%d",&n);
    triunghi T[n];
    citestePuncte(T,n);
    verificaTriunghi(T,n);
    qsort(T,n,sizeof(triunghi),cmpTriunghiuri);
    T[0].triunghiValid ? printf("Aria maxima este %f\n",T[0].arie) : printf("Nu
exista \n");

    return 0;
}
```

6. Considerăm un pachet de cărţi de joc uzual care conţine 52 de cărţi. Fiecare carte are o valoare din mulţimea ordonată {2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A} şi o culoare din mulţimea {h, d, s, c} (hearts – inimă roşie, diamonds – romb, spades – inimă neagră, clubs – treflă). În jocul de poker, mâinile (mulţime de 5 cărţi de joc diferite între ele) sunt clasificate într-un mod clar. În total sunt 10 clase diferite de mâini de poker care pot exista. Acestea pornesc de la "*chinta roială*", care este cea mai valoroasa combinaţie posibilă, şi descresc până la "*carte mare*". Clasificarea este exemplificată în cele ce urmează pe baza unei figuri conţinând exemple particulare din cele 10 clase posibile de mâini de poker ierarhizate.

1. ***Chintă roială*** – conţine cărţile cu valorile 10 (decar), J (valet), Q (damă), K (popă), A (as) de aceeaşi culoare.

2. ***Chintă de culoare*** – conţine cinci cărţi de valori consecutive, toate de aceeaşi culoare.

3. ***Careu*** – conţine patru cărţi de aceeaşi valoare şi o carte adiţională.

4. ***Full*** – conţine trei cărţi de aceeaşi valoare şi alte două cărţi având o altă valoare, dar egale între ele.

5. ***Culoare*** – conţine cinci cărţi de aceeaşi culoare.

6. ***Chintă*** – conţine cinci cărţi de valori consecutive.

7. ***Trei de un fel*** – conţine trei cărţi având aceeaşi valoare, alături de alte două cărţi de valori diferite.

8. ***Două perechi*** – conţine două cărţi având aceeaşi valoare, alte două cărţi având aceeaşi valoare (dar diferită de valoarea primelor două cărţi) şi o a cincea carte de valoare diferită.

9. ***O pereche*** – conţine două cărţi având aceeaşi valoare, alături de alte trei cărţi de valori diferite.

10. ***Carte mare*** - orice mână care nu intră în niciuna dintre clasele de mai sus.

O mână este dată printr-un şir de caractere conţinând cele cinci cărţi separate printr-un spaţiu. Fiecare carte din cele cinci este reprezentată de valoarea sa (numerele de la 2 la 10 sau caracterele J, Q, K, A) şi de culoarea sa. Spre exemplu, mâinile din figura de mai sus se reprezintă astfel:
10h Jh Qh Kh Ah
5s 6s 7s 8s 9s

Ah Ac Ad As 2h
As Ad Ac Kh Ks
2h 4h 6h 8h Kh
5h 6c 7d 8s 9h
Ac Ah As 2d 7c
Kd Kc Qh Qs Jd
Ac Ah 9s 8d 7c
Ah 8s 6d 6c 2h

Ordinea cărţilor într-o mână nu contează. Mâinile *8h 10h 9h Qh Jh* şi *10h 8h 9h Qh Jh* sunt identice. Prin excepţie, un as (A) poate lua şi valoarea 1 astfel încât o mâna ce conţine cărţile A, 2, 3, 4, 5 este o chintă (posibil chintă de culoare dacă cărţile au aceeaşi culoare).

Definiţi structura *Carte* care să permită memorarea valorii şi a culorii unei cărţi de joc.

Scrieţi o funcţie care citeşte de la tastatură un şir de caractere reprezentând o mână şi extragele cele 5 cărţi într-un vector cu elemente de tip *Carte*.

Scrieţi o funcţie prin care clasificaţi o mână de poker într-una din cele 10 clase.

Rezolvare:

Definim structura *Carte* care permite memorarea valorii şi a culorii unei cărţi de joc:

```
typedef struct
      {
            int valoare;
            char culoare;
} Carte;
```

De adăugat ...

Probleme propuse

1. Considerăm în planul xOy dreptunghiuri cu laturile paralele cu axele Ox şi Oy. Definiţi o structură *Dreptunghi* prin care să puteţi reprezenta aceste dreptunghiuri.

Scrieţi o funcţie care primeşte un tablou unidimensional t de n elemente de tip *Dreptunghi* şi le sortează folosind funcţia `qsort` pe baza ariei lor.

Scrieţi o funcţie care primeşte un tablou unidimensional t de n elemente de tip *Dreptunghi* şi calculează intersecţia tuturor dreptunghiurilor din tablou.

2. Definiţi o structură *Meteo* care să permită memorarea temperaturilor medii lunare dintr-o ţară în primele n luni ale unui anumit an ($1 \leq n \leq 12$). Scrieţi o funcţie care să calculeze temperatura medie anuală înregistrată într-o anumită ţară. Informaţiile referitoare la temperaturile lunare înregistrate în mai multe ţări de-a lungul mai multor ani vor fi transmise funcţiei prin intermediul unui tablou unidimensional t cu n elemente de tip *Meteo*.

Folosind funcţia `qsort` din biblioteca `stdlib.h`, sortaţi elementele unui tablou unidimensional t format din n elemente de tip *Meteo* în ordinea descrescătoare a temperaturilor medii anule, iar în cazul unor temperaturi medii anuale egale ţările se vor ordona alfabetic. Implementaţi funcţia comparator corespunzătoare şi scrieţi apelul funcţiei `qsort`.

3. Definiţi o structură *Elev* care să permită memorarea numelui şi prenumelui unui elev, precum şi a notelor, mediei şi rezultatului pe care acesta le-a obţinut la Bacalaureat.
Scrieţi o funcţie care să calculeze rezultatele obţinute de n elevi ale căror date sunt memorate într-un tablou unidimensional t cu elemente de tip *Elev*. Un elev promovează examenul de Bacalaureat dacă toate notele obţinute sunt mai mari sau egale decât 5 şi media este mai mare sau egală decât 6.

Folosind funcţia `qsort`, sortaţi elementele unui tablou unidimensional t format din n elemente de tip Elev în ordinea descrescătoare a rezultatelor obţinute (se consideră că nepromovat ≼ promovat), iar în cazul unor rezultate egale elevii se vor ordona descrescător în funcţie de medii, iar la medii egale în funcţie de nume şi apoi de prenume. Implementaţi funcţia comparator corespunzătoare şi scrieţi apelul funcţiei `qsort`.

4. Considerăm mulţimea numerele complexe cu coeficienţi numere raţionale, adică:
$\mathbb{Q}[i] = \{z \in \mathbb{C} \mid z = \frac{a}{b} + i * \frac{c}{d}, cu\ a, b, c, d \in \mathbb{Z}\ şi\ (a, b) = 1, (c, d) = 1\}$.
Scrieţi funcţii care să adune, scadă, înmulţească, împartă două numere z_1, z_2 din $\mathbb{Q}[i]$.

5. Sunteţi rugat să scrieţi un program în limbajul C pentru a ajuta secretariatul Facultăţii de Matematică şi Informatică în calcularea situaţiilor şcolare dupa primul an. Un student promovează primul an dacă suma creditelor examenelor promovate este ≥ jumătate din suma tututor creditelor. În primul an fiecare student are de susţinut 12 examene obligatorii. În catalog, pentru fiecare student, în dreptul fiecărei discipline, este trecută o notă (între 1 şi 10) sau ‚a' (absent). O disciplină este considerată promovată dacă nota la acea disiplina este cel puţin 5. La sfârşit de an se calculează mediile numai pentru studenţii integralişti, acei studenţi care promovează toate disciplinele din an. Pentru fiecare student se calculează punctajul de clasificare ca suma ponderată dintre notele la disciplinele promovate şi creditele aferente disciplinei.

Definiţi o structură *Student* care să permită memorarea numelui, prenumelui, grupei corespunzătoare unui student, notelor, mediei, punctajului de clasificare şi statutului la sfârşitul primului an universitar (promovat sau nu).

Scrieţi o funcţie care să calculeze media, punctajul de clasificare, statutul pentru n studenţi ale căror date sunt memorate într-un tablou unidimensional t cu elemente de tip $Student$.

6. Considerăm implementarea unui arbore binar ca o structură de date în care fiecare nod frunză are o valoare de tip double iar fiecare nod intern conţine doi pointeri către cei doi fii, fără să conţină vreo valoare. Este următoarea definiţie a structurii NOD eficientă din punct de vedere al memoriei ocupate în contextul problemei?

```
struct NOD {
  struct NOD *left;
  struct NOD *right;
  float data;
};
```

Puteţi scrie o altă definiţie a structurii NOD mai eficiente din punct de vedere al memoriei ocupate? Cum influenţează alinierea spaţiul de memorie ocupat?