

SEMINAR 3

STRUCTURI. UNIUNI. ENUMERĂRI. CÂMPURI DE BIȚI.

Probleme rezolvate

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

Rezolvare:

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

```
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 civilă și adresa.

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

2. Definiți 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 ați defini această structură astfel încât să ocupe cât mai puțini octeți 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 octeți: câte 4 octeți pentru fiecare membru de tip `int` și câte 1 octet pentru fiecare membru de tip `_Bool`. Alți 2 octeți sunt adăugați 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 biți putem defini structura *Camera* astfel încât ea să ocupe un singur octet. Definim toți cei 4 membri ai structurii ca fiind de tipul `unsigned char` cu o anumită dimensiune în biți în funcție 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. Maska 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. Definiți o structură *Produs* care să permită memorarea numelui unui produs, a prețului său unitar și a cantității existente din produsul respectiv într-un depozit. Scrieți o funcție care să ieftinească cu 10% toate produsele aflate într-un depozit al căror stoc depășește un număr real s (stocul unui produs este egal cu produsul dintre preț și cantitate). Informațiile despre produsele din depozitul respectiv sunt memorate într-un tablou t cu n elemente de tip *Produs*. Să se sorteze folosind funcția `qsort` elementele unui tablou unidimensional t format din n elemente de tip *Produs* în ordinea descrescătoare a prețurilor unitare, iar produsele având același preț unitar se vor ordona alfabetic.

Rezolvare:

Definim inițial structura *Produs* având ca membri nume, pret și cantitate. Prețul unui produs este de tip `float` și nu `int`, întrucât după ieftinirea cu 10% a unui preț întreg am obține tot un preț întreg, ceea ce ar fi incorect.

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

Funcția pentru ieftinirea produselor primește ca parametri numele tabloului cu elemente de tipul *Produs*, lungimea n a acestui tablou și valoarea s a stocului pentru care se realizează reducerea de preț.

```
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 funcția `qsort` a tabloului t de elemente de tipul *Produs* avem nevoie de o funcție comparator. Funcția `cmpProduse` este scrisă mai jos și implementează cerința problemei. Această funcție compară două elemente de tipul *Produs* în funcție de prețul lor, iar la preț egal compară numele lor. Funcția primește doi pointeri generici a și b pe care îi convertește la tipul struct *Produs*. Întrucât vrem ca tabloul sortat să fie sortat descrescător în funcție de prețurile unitare, funcția `cmpProduse` întoarce valoarea -1 dacă prețul unitar al produsului către care pointează a este mai mare decât prețul produsului către care pointează b . Valoarea -1 înseamnă că în tabloul sortat, produsul către care pointează a (în tabloul inițial) se va regăsi înaintea (la stânga) produsului către care pointează b (în tabloul inițial). Funcția `cmpProduse` întoarce valoarea +1 dacă prețul unitar al produsului către care pointează a este mai mic decât prețul produsului către care pointează b . Dacă prețurile sunt egale atunci funcția întoarce rezultatul funcției `strcmp` ce compară 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 funcțiilor 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 funcție de aria triunghiului. O posibilitate este folosirea funcției `qsort` din librăria `stdlib.h`, în acest caz avem nevoie de definiția unei funcții 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 cărți de joc uzual care conține 52 de cărți. Fiecare carte are o valoare din mulțimea ordonată {2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A} și o culoare din mulțimea {h, d, s, c} (hearts – inimă roșie, diamonds – romb, spades – inimă neagră, clubs – treflă). În jocul de poker, mâinile (mulțime de 5 cărți 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 valoroasă combinație posibilă, și descresc până la “*carte mare*”. Clasificarea este exemplificată în cele ce urmează pe baza unei figuri conținând exemple particulare din cele 10 clase posibile de mâini de poker ierarhizate.



1. **Chintă roială** – conține cărțile cu valorile 10 (decar), J (valet), Q (damă), K (popă), A (as) de aceeași culoare.
2. **Chintă de culoare** – conține cinci cărți de valori consecutive, toate de aceeași culoare.
3. **Careu** – conține patru cărți de aceeași valoare și o carte adițională.
4. **Full** – conține trei cărți de aceeași valoare și alte două cărți având o altă valoare, dar egale între ele.
5. **Culoare** – conține cinci cărți de aceeași culoare.
6. **Chintă** – conține cinci cărți de valori consecutive.
7. **Trei de un fel** – conține trei cărți având aceeași valoare, alături de alte două cărți de valori diferite.
8. **Două perechi** – conține două cărți având aceeași valoare, alte două cărți având aceeași valoare (dar diferită de valoarea primelor două cărți) și o a cincea carte de valoare diferită.
9. **O pereche** – conține două cărți având aceeași valoare, alături de alte trei cărți de valori diferite.
10. **Carte mare** - orice mână care nu intră în niciuna dintre clasele de mai sus.

O mână este dată printr-un șir de caractere conținând cele cinci cărți separate printr-un spațiu. Fiecare carte din cele cinci este reprezentată de valoarea sa (numerele de la 2 la 10 sau caracterele J, Q, K, A) și 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 cărților într-o mână nu contează. Măinile *8h 10h 9h Qh Jh* și *10h 8h 9h Qh Jh* sunt identice. Prin excepție, un as (A) poate lua și valoarea 1 astfel încât o mână ce conține cărțile A, 2, 3, 4, 5 este o chintă (posibil chintă de culoare dacă cărțile au aceeași culoare).

Definiți structura *Carte* care să permită memorarea valorii și a culorii unei cărți de joc.

Scrieți o funcție care citește de la tastatură un șir de caractere reprezentând o mână și extrage cele 5 cărți într-un vector cu elemente de tip *Carte*.

Scrieți o funcție prin care clasificați o mână de poker într-una din cele 10 clase.

Rezolvare:

Definim structura *Carte* care permite memorarea valorii și a culorii unei cărți de joc:

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

Funcția *citesteMana* primește ca argument un tablou unidimensional de 5 elemente de tip *Carte*, citește de la tastatură cele 5 cărți în formatul așteptat și face conversia din acest format într-un format convenabil. Formatul convenabil ales asignează valorile între 2 și 14 pentru cărțile cu valorile 2, 3, ..., 9, 10, J = 11, Q = 12, K = 13, A = 14. Pentru reprezentarea culorilor păstrăm formatul inițial și anume valorile posibile de tip `char {h, d, s, c}`.

```
void citesteMana(Carte mana[5])
{
    char text[5][4];
    int i, j;
    printf("Dati mana de 5 carti in formatul cunoscut: ");
    scanf("%s %s %s %s %s", text[0], text[1], text[2], text[3], text[4]);
    char v[] = "**23456789*JQKA"; //tablou ajutator
    //determina culoarea si valoarea fiecărei carti
    for(i=0; i<5; i++)
    {
        if (strlen(text[i]) == 3) //cazul cu un decar - are 3 caractere
        {
```

```

        mana[i].valoare = 10;
        mana[i].culoare = text[i][2];
    }
    else
    {
        mana[i].culoare = text[i][1];
        for(j = 0; j <= 15; j++)
            if(v[j] == text[i][0])
            {
                mana[i].valoare = j;
            }
    }
}
return;
}

```

Clasificarea unei mâini de poker se face pe baza a doi vectori de frecvență, unul ce numără frecvențele valorilor cărților de la 2 la 14 (considerate în cele ce urmează între 0 și 12) ce apar în mână și celălalt ce numără frecvențele culorilor cărților. Spre exemplu, chinta roială se definește ca o mână pentru care vectorul de frecvență al culorilor cărților apare un element de 5 ori (sunt 5 cărți de aceeași culoare) și de asemenea pentru care vectorul de frecvență al valorilor avem ultimele 5 elemente (corespunzătoare valorilor 8-10, 9-J, 10-Q, 11-K, 12-A) egale cu 1. Un careu poate fi definit ca mâna pentru care vectorul de frecvență al valorilor cărților are un element cu frecvența 4. Folosim funcțiile `eCuloare`, `eChinta`, `eChintaMare`, `eCareu`, `eTrei`, `calculeazaNumarPerechi` pentru a încadra o mână de cărți într-una din cele 10 clase posibile. Pentru simplitate definim aceste clase cu ajutorul unei enumerări.

```

enum clasaMana{cartemare, opereche, douaperechi, treideunfel, chinta, culoare,
full, careu, chintadeculoare, chintaroiala};

int clasificaMana(Carte mana[5])
{
    int frecventaValoare[13] = {0}; //pentru valorile 2,...,10,11=J,12=Q,13=K14=A
    int frecventaCuloare[4] = {0};
    int i;
    for(i=0; i<5; i++)
    {
        frecventaValoare[mana[i].valoare-2]++;
        switch(mana[i].culoare)
        {
            case 'h': frecventaCuloare[0]++; break;
            case 'd': frecventaCuloare[1]++; break;
            case 's': frecventaCuloare[2]++; break;
            case 'c': frecventaCuloare[3]++; break;
        }
    }
}

//chinta roiala
if(eCuloare(frecventaCuloare) && eChintaMare(frecventaValoare))
    return chintaroiala;

```

```

//chinta de culoare
if(eCuloare(frecventaCuloare) && eChinta(frecventaValoare))
    return chintadeculoare;

//careu
if(eCareu(frecventaValoare))
    return careu;

int nrPerechi = calculeazaNumarPerechi(frecventaValoare);
//full
if(eTrei(frecventaValoare) && (nrPerechi==1))
    return full;

//culoare
if(eCuloare(frecventaCuloare))
    return culoare;

//chinta
if(eChinta(frecventaValoare))
    return chinta;

//treideunfel
if(eTrei(frecventaValoare))
    return treideunfel;

//doua perechi
if(nrPerechi==2)
    return douaperechi;

//o pereche
if(nrPerechi==1)
    return opereche;

return cartemare;
}

```

Funcțiile eCuloare, eChinta, eChintaMare, eCareu, eTrei, calculeazaNumarPerechi se folosesc pentru clasificarea unei mâini din funcția anterioară. În funcție de specificul lor, ele primesc ca parametru vectorul de frecvență al valorilor sau al culorilor cărților. Spre exemplu funcția eChinta verifică dacă cele 5 valori ale cărților sunt consecutive (condiția pentru chintă) și de asemenea tratează cazul extrem când avem o mână formată din cărțile cu valorile 2, 3, 4, 5, A (=13, socotit aici ca excepție = 1) care de asemenea formează o chintă. De reținut este că ordinea cărților nu contează, lucrul realizat de implementarea cu vectori de frecvență.

```

int eCuloare(int c[])//c aici e vectorul de culori
{
    int i;
    for(i=0;i<4;i++)
        if (c[i]==5)

```

```

        return 1;
    return 0;
}

int eChinta(int v[]) //v aici e vectorul de frecventa al valorilor cu 13 componente
{
    //verificam 2-6, 3-7,..., 9-14 si apoi cazul extrem 2,3,4,5,13(As)
    int i, j;
    for(i=0;i<9;i++)
        if ((v[i]==1) && (v[i+1]==1) && (v[i+2]==1) && (v[i+3]==1) && (v[i+4]==1))
            return 1;
    //cazul extrem
    if ((v[0]==1) && (v[1]==1) && (v[2]==1) && (v[3]==1) && (v[12]==1))
        return 1;

    //altfel nu e chinta
    return 0;
}

int eChintaMare(int v[])
{
    //verificam 9-13
    if ((v[8]==1) && (v[9]==1) && (v[10]==1) && (v[11]==1) && (v[12]==1))
        return 1;
    //altfel nu e chinta mare
    return 0;
}

int eCareu(int v[])
{
    int i;
    for(i=0;i<13;i++)
        if(v[i]==4)
            return 1;
    return 0;
}

int eTrei(int v[])
{
    int i;
    for(i=0;i<13;i++)
        if(v[i]==3)
            return 1;
    return 0;
}

int calculeazaNumarPerechi(int v[])
{
    int i;
    int nrPerechi = 0;
    for(i=0;i<13;i++)
        if(v[i]==2)
            nrPerechi++;
}

```

```

    return nrPerechi;
}

```

Programul principal revine la o serie de apeluri ale funcțiilor descrise mai sus `citesteMana` și `clasificaMana`. Pentru fiecare clasa se afișează mesajul corespunzător clasei, acest mesaj este implementat de funcția `afiseazaMesaj`.

```

int main()
{
    Carte manaCarti[5];
    citesteMana (manaCarti);
    afiseazaMesaj (clasificaMaini (manaCarti));
    printf("\n");
    return 0;
}

void afiseazaMesaj (int nr)
{
    switch (nr)
    {
        case 0: printf("Carte mare");break;
        case 1: printf("O pereche");break;
        case 2: printf("Doua perechi");break;
        case 3: printf("Trei de un fel");break;
        case 4: printf("Chinta");break;
        case 5: printf("Culoare");break;
        case 6: printf("Full");break;
        case 7: printf("Careu");break;
        case 8: printf("Chinta de culoare");break;
        case 9: printf("Chinta roiala");break;
    }
}

```

Probleme propuse

1. Considerăm în planul xOy dreptunghiuri cu laturile paralele cu axele Ox și Oy . Definiți o structură *Dreptunghi* prin care să puteți reprezenta aceste dreptunghiuri.

Scrieți o funcție care primește un tablou unidimensional t de n elemente de tip *Dreptunghi* și le sortează folosind funcția `qsort` pe baza ariei lor.

Scrieți o funcție care primește un tablou unidimensional t de n elemente de tip *Dreptunghi* și calculează intersecția tuturor dreptunghiurilor din tablou.

2. Definiți o structură *Meteo* care să permită memorarea temperaturilor medii lunare dintr-o țară în primele m luni ale unui anumit an ($1 \leq m \leq 12$). Scrieți o funcție care să calculeze temperatura medie anuală înregistrată într-o anumită țară. Informațiile referitoare la temperaturile lunare înregistrate în mai multe țări de-a lungul mai multor ani vor fi transmise funcției prin intermediul unui tablou unidimensional t cu n elemente de tip *Meteo*.

Folosind funcția `qsort` din biblioteca `stdlib.h`, sortați elementele unui tablou unidimensional t format din n elemente de tip *Meteo* în ordinea descrescătoare a temperaturilor medii anuale, iar în cazul unor temperaturi medii anuale egale țările se vor ordona alfabetic. Implementați funcția comparator corespunzătoare și scrieți apelul funcției `qsort`.

3. Definiți o structură *Elev* care să permită memorarea numelui și prenumelui unui elev, precum și a notelor, mediei și rezultatului pe care acesta le-a obținut la Bacalaureat.

Scrieți o funcție care să calculeze rezultatele obținute 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 obținute sunt mai mari sau egale decât 5 și media este mai mare sau egală decât 6.

Folosind funcția `qsort`, sortați elementele unui tablou unidimensional t format din n elemente de tip *Elev* în ordinea descrescătoare a rezultatelor obținute (se consideră că nepromovat \leq promovat), iar în cazul unor rezultate egale elevii se vor ordona descrescător în funcție de medii, iar la medii egale în funcție de nume și apoi de prenume. Implementați funcția comparator corespunzătoare și scrieți apelul funcției `qsort`.

4. Considerăm mulțimea numerele complexe cu coeficienți numere raționale, adică:

$$\mathbb{Q}[i] = \{z \in \mathbb{C} \mid z = \frac{a}{b} + i * \frac{c}{d}, \text{ cu } a, b, c, d \in \mathbb{Z} \text{ și } (a, b) = 1, (c, d) = 1\}.$$

Scrieți funcții care să adune, scadă, înmulțească, împartă două numere z_1, z_2 din $\mathbb{Q}[i]$.

5. Sunteți rugat să scrieți un program în limbajul C pentru a ajuta secretariatul Facultății de Matematică și Informatică în calcularea situațiilor școlare după primul an. Un student promovează primul an dacă suma creditelor examenelor promovate este \geq jumătate din suma tuturor creditelor. În primul an fiecare student are de susținut 12 examene obligatorii. În catalog, pentru fiecare student, în dreptul fiecărei discipline, este trecută o notă (între 1 și 10) sau „a” (absent). O disciplină este considerată promovată dacă nota la acea disciplină este cel puțin 5. La sfârșit de an se calculează mediile numai pentru studenții integraliști, acei studenți care promovează toate disciplinele din an. Pentru fiecare student se calculează punctajul de clasificare ca suma ponderată dintre notele la disciplinele promovate și creditele aferente disciplinei.

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

Scrieți o funcție care să calculeze media, punctajul de clasificare, statutul pentru n studenți 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 conține doi pointeri către cei doi fii, fără să conțină vreo valoare. Este următoarea definiție a structurii *NOD* eficientă din punct de vedere al memoriei ocupate în contextul problemei?

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

Puteți scrie o altă definiție a structurii *NOD* mai eficientă din punct de vedere al memoriei ocupate? Cum influențează alinierea spațiului de memorie ocupat?

7. Considerăm problema clasificărilor mâinilor de poker descrisă în problema 6 din cele rezolvate. Scrieți o funcție care primește ca parametri două mâini de poker a doi jucători și decide cine câștigă. Aveți în vedere că atunci când cele două mâini de poker sunt în clase egale (spre exemplu ambele sunt chintă) se aplică reguli de departajare (detalii aici: <https://www.pokerstars.ro/poker/games/rules/hand-rankings/>).