Consultații admitere
Tipuri structurate de date

Ionescu Vlad-Sebastian și Coroiu Adriana 16 decembrie 2017

Întrebări grilă

1. Se dă următoarea structură și secvență de cod:

C++

```
struct Elev
{
    char nume[40];
    float medie
    float medieGenerala;
} Ex[1000], aux;

do
{
    s=0;
    for (i=0; i<=nrElevi-1; i++ )
      if ((strcmp(Ex[i].nume, Ex[i+1].nume)↩
          >0)
        {
            a=Ex[i];
            Ex[i]=Ex[i+1];
            Ex[i+1]=a;

            s=1;
        }
} while (s);
```

Pascal

```
type Elev = record

    nume : string[40];
    medie : double;
    medieGenerala : double;
end;

var Ex : array [1..1000] of Elev;
    aux : Elev;
    s, i, nrElevi : integer;
begin
nrElevi := 20;

 repeat
    s:= 0;
    for i := 0 to nrElevi-1 do
    begin
      if (compareStr(Ex[i].nume, Ex[i+1].↩
          nume)>0) then
        begin
            aux := Ex[i];
            Ex[i] := Ex[i+1];
            Ex[i+1] := aux;
            s:=1;
        end;
    end;
   until s = 1;
```

Selectați varianta care descrie efectul secvenței:

a. Se interschimbă valorile câmpurilor structurii

b. Se ordonează alfabetic crescător după câmpul **nume** elevii

c. Se ordonează in ordine crescatoare dupa **medieGenerala**

d. Se ordonează in ordine descrescatoare dupa **medieGenerala**

2. Se dă următoarea structură și urmatoarele secvente de cod:

C++

```
struct Punct {float x, y;} Pa, Pb, Pc;



secventa a)
float t(float a, float b, float c)
{
    float p = (a + b + c) / 2;
    return (sqrt(p*(p-a)*(p-b)*(p-c));
}



secventa b)
float t(float a, float b, float c)
{
    float p = (a - b - c) / 2;
    return (sqrt(p*(p+a)*(p+b)*(p+c));
}



secventa c)
float t(float a, float b, float c)
{
    float p = ((a-b)/(b+c));
    return (sqrt(p*(p-a)*(p-b)*(p-c));
}



secventa d)
float t(float a, float b, float c)
{
    float p = ((a+b)/(b-c));
    return (sqrt(p*(p-a)*(p-b)*(p-c));
}
```

Pascal

```
type Punct = record
    x : double;
    y : double;
end;
var Pa, Pb, Pc : Punct;

secventa a)
function t(a, b, c : double): double;
    var p, result : double;
    begin
        p := (a+b+c)/2;
        result := sqrt(p*(p-a)*(p-b)*(p-c));
        t := result;
    end;

secventa b)
function t(a, b, c : double): double;
    var p, result : double;
    begin
        p := (a-b-c)/2;
        result := sqrt(p*(p+a)*(p+b)*(p+c));
        t := result;
    end;

secventa c)
function t(a, b, c : double): double;
    var p, result : double;
    begin
        p := (a-b)/(b+c);
        result := sqrt(p*(p-a)*(p-b)*(p-c));
        t := result;
    end;

secventa d)
function t(a, b, c : double): double;
    var p, result : double;
    begin
        p := (a+b)/(b-c);
        result := sqrt(p*(p-a)*(p-b)*(p-c));
        t := result;
    end;
```

Care dintre secventele de cod de mai sus permit calcularea ariei unui triunghi determinat de cele 3 puncte date?

a. Secvența a)

b. Secvența b)

c. Secvența c)

d. Secvența d)

Problemă de modelare

Enunț

Considerăm o matrice A de numere naturale de dimensiune 2×2. Fie următoarea sumă:

$$S(n) = \left(A + A^2 + A^3 + ... + A^n\right) \text{ modulo } p \quad (1)$$

unde p poate fi orice număr prim. Pentru simplitate, vom considera $p = 666013$.

Avem $1 \leq n \leq 10^{18}$.

Operația X modulo p, unde X este o matrice, are ca rezultat o matrice cu elementele lui X care se iau fiecare modulo p.

Să se calculeze suma $S(n)$.

Analiza

- O primă rezolvare în $O(n)$ presupune un for de la 2 la n în care actualizăm suma (inițial A) și termenul curent (inițial A^2): la sumă adăugăm termenul curent, iar termenul curent îl înmulțim cu A. Acest lucru este implementat în funcția **calculeazaSFolosindBruteForce**.
- Observăm că n poate fi prea mare pentru o rezolvare în $O(n)$. Ne propunem să găsim rezolvări de complexitate mai bună.
- Observăm ca avem o progresie geometrică de rație A.
- O primă idee poate fi să folosim formula de sumă pentru o progresie geometrică.

- Dar avem de-a face cu matrici și cu modulo, ceea ce complică folosirea formulei (dar o face imposibilă?)
- Încercăm să manipulăm suma astfel încât să obținem o expresie care se poate calcula mai eficient.
- Ne gândim că dacă putem exprima $S(n)$ în funcție de $S(n/2)$ (deocamdată nu ne gândim la detalii gen paritatea lui n), atunci putem obține o rezolvare în $O(log\, n)$. Complexitatea s-ar reduce deoarece, la fiecare pas, l-am înjumătăți pe n, până ajunge la 1. De câte ori îl putem înjumătăți pe n până să ajungă la 1? De aproximativ $log\ n$ ori.

  Această exprimare a lui $S(n)$, dacă e posibilă, s-ar realiza, probabil, printr-o operație gen "factor comun".
- Dar dacă îl dăm factor comun pe A, nu ne ajută cu nimic...
- O altă idee ar fi să înjumătățim numărul termenilor din sumă printr-o factorizare de genul:

$$A + A^2 + A^3 + ... + A^n = (X + Y)(A + ... + A^{n/2}) \tag{2}$$

- Astfel, ne propunem ca înmulțind cu X să obținem prima jumătate a sumei, iar înmulțind cu Y să obținem a doua jumătate.
- Se observă destul de ușor că funcționează $X = I$ și $Y = A^{n/2}$
- Rezultă:

$$S(2k) = (I + A^k)(A + ... + A^k) \tag{3}$$
$$= (I + A^k)S(k) \tag{4}$$
$$S(2k + 1) = S(2k) + A^{2k+1} \tag{5}$$

Considerente de implementare

- Avem nevoie de o funcție care ridică o matrice (de 2×2) la o anumită putere.
- Vom folosi algoritmul de exponențiere logaritmică.
- Vom folosi o structură $Matrice 2 \times 2$, pentru a putea lucra cu matrici așa cum am lucra și cu tipuri de date primitive.

- O implementare directă duce la complexitatea $O(log^2\, n)$, deoarece apelăm funcția de ridicare la putere la fiecare pas al recursivității principale. Această abordare este implementată în funcția **calculeazaSlnLogPatrat**.
- Implementând exponențierea logaritmică în funcția recursivă principală, putem obține complexitatea $O(log\, n)$. Această abordare este implementată în funcția **calculeazaSlnLog**.

Implementare

C++

```cpp
#include <iostream>

using namespace std;

const int p = 666013;

/*
Retine o matrice de 2x2 de numere intregi.
*/
struct Matrice2x2
{
    // Elementele matricei
    // trebuie long deoarece p*p > INT_MAX
    long x[2][2];

    /*
    Initializeaza o matrice de 2x2, cu elementele luate modulo p.
    Input:
    - xij: elementul de pe linia i si coloana j (mod p)
    Output: -
    */
    Matrice2x2(long x00, long x01, long x10, long x11)
    {
        x[0][0] = x00 % p;
        x[0][1] = x01 % p;
        x[1][0] = x10 % p;
        x[1][1] = x11 % p;
    }
};

/*
Returneaza matricea unitate de 2x2.
*/
Matrice2x2 getUnitate()
{
    return Matrice2x2(1, 0, 0, 1);
}
```

```
/*
Inmulteste doua matrici mod P.
Input:
- X: o matrice de 2x2 de intregi
- Y: o matrice de 2x2 de intregi
Output:
- X*Y (mod p)
*/
Matrice2x2 inmulteste(Matrice2x2 X, Matrice2x2 Y)
{
    return Matrice2x2(
        (X.x[0][0]*Y.x[0][0] + X.x[0][1]*Y.x[1][0]) % p,
        (X.x[0][0]*Y.x[0][1] + X.x[0][1]*Y.x[1][1]) % p,
        (X.x[1][0]*Y.x[0][0] + X.x[1][1]*Y.x[1][0]) % p,
        (X.x[1][0]*Y.x[0][1] + X.x[1][1]*Y.x[1][1]) % p
    );
}

/*
Aduna doua matrici mod P.
Input:
- X: o matrice de 2x2 de intregi
- Y: o matrice de 2x2 de intregi
Output:
- X + Y (mod p)
*/
Matrice2x2 aduna(Matrice2x2 X, Matrice2x2 Y)
{
    return Matrice2x2(
        (X.x[0][0] + Y.x[0][0]) % p,
        (X.x[0][1] + Y.x[0][1]) % p,
        (X.x[1][0] + Y.x[1][0]) % p,
        (X.x[1][1] + Y.x[1][1]) % p
    );
}

/*
Ridica o matrice la o putere data (mod p).
Input:
- X: matrice de 2x2 de intregi
- exponent: puterea la care sa se ridice X
Output:
- X la puterea exponent (mod p)
*/
Matrice2x2 ridicaLaPutere(Matrice2x2 X, long exponent)
{
    if (exponent == 0)
    {
        // matricea I
        return getUnitate();
    }

    Matrice2x2 jumatate = ridicaLaPutere(X, exponent / 2);
    Matrice2x2 jumatatePatrat = inmulteste(jumatate, jumatate);
```

```
    if (exponent % 2 == 0)
    {
        return jumatatePatrat;
    }

    return inmulteste(jumatatePatrat, X);
}

/*
Afiseaza o matrice de 2x2.
Input:
- X: matricea de afisat
Output: -
*/
Matrice2x2 afiseazaMatrice(Matrice2x2 X)
{
    cout << X.x[0][0] << " " << X.x[0][1] << endl << X.x[1][0] << " " << X.x↩
       [1][1] << endl;
}

/*
Citeste o matrice de 2x2.
Input: -
Output:
- matricea citita
*/
Matrice2x2 citesteMatrice()
{
    Matrice2x2 X(0, 0, 0, 0);
    for (int i = 0; i < 2; ++i)
    {
        for (int j = 0; j < 2; ++j)
        {
            cout << "Dati elementul de pe linia " << i << " si coloana " << j ↩
               << ": ";
            cin >> X.x[i][j];
        }
    }

    return X;
}

/*
Citeste n-ul din enunt.
Input: -
Output:
- valoarea citita
*/
long citesteN()
{
    cout << "Dati n (pana la cat se calculeaza suma): ";
    long n;
    cin >> n;
    return n;
```

```
}

/*
Calculeaza suma S(n), in complexitate O(n)
Input:
- A: matricea A din enunt.
- n: limita din enunt.
Output:
- S(n) (mod p).
*/
Matrice2x2 calculeazaSFolosindBruteForce(Matrice2x2 A, long n)
{
    Matrice2x2 rez = A;
    Matrice2x2 curent = inmulteste(A, A);
    for (int i = 2; i <= n; ++i)
    {
        rez = aduna(rez, curent);
        curent = inmulteste(curent, A);
    }

    return rez;
}

/*
Calculeaza suma S(n), in complexitate O(log^2 n)
Input:
- A: matricea A din enunt.
- n: limita din enunt.
Output:x
- S(n) (mod p).
*/
Matrice2x2 calculeazaSInLogPatrat(Matrice2x2 A, long n)
{
    // n >= 1 in enunt. Daca poate fi si 0?
    if (n == 1)
    {
        return A;
    }

    long k = n / 2;
    Matrice2x2 jumatate = calculeazaSInLogPatrat(A, k);

    // apeleaza ridicaLaPutere, care are complexitatea O(log n)
    // rezulta complexitate totala O(log^2 n)
    Matrice2x2 unitatePlusALaNpe2 = aduna(getUnitate(), ridicaLaPutere(A, k));
    Matrice2x2 tot = inmulteste(unitatePlusALaNpe2, jumatate);

    if (n % 2 == 0)
    {
        return tot;
    }

    return aduna(tot, ridicaLaPutere(A, n));
}
```

```
/*
Helper pentru calculeaza suma S(n), in complexitate O(log n).
Input:
- A: matricea A din enunt.
- n: limita din enunt.
- AlaPutereaNpe2: folosit pentru a calcula A^n in acelasi timp cu suma.
Output:
- S(n) (mod p).
*/
Matrice2x2 calculeazaSInLogHelper(Matrice2x2 A, long n, Matrice2x2 &↩
    AlaPutereaNpe2)
{
    if (n == 1)
    {
        AlaPutereaNpe2 = A;
        return A;
    }

    long k = n / 2;
    Matrice2x2 jumatate = calculeazaSInLogHelper(A, k, AlaPutereaNpe2);

    // nu se mai apeleaza functia de ridicare la putere => complexitatea ↩
       totala ramane O(log n)
    Matrice2x2 unitatePlusALanPe2 = aduna(getUnitate(), AlaPutereaNpe2);
    Matrice2x2 tot = inmulteste(unitatePlusALanPe2, jumatate);

    Matrice2x2 patrat = inmulteste(AlaPutereaNpe2, AlaPutereaNpe2);

    if (n % 2 == 0)
    {
        AlaPutereaNpe2 = patrat;
        return tot;
    }

    AlaPutereaNpe2 = inmulteste(patrat, A);
    return aduna(tot, AlaPutereaNpe2);
}

/*
Calculeaza suma S(n), in complexitate O(log n)
Input:
- A: matricea A din enunt.
- n: limita din enunt.
Output:
- S(n) (mod p).
*/
Matrice2x2 calculeazaSInLog(Matrice2x2 A, long n)
{
    Matrice2x2 temp(0, 0, 0, 0);
    return calculeazaSInLogHelper(A, n, temp);
}


int main()
{
```

```
    Matrice2x2 A = citesteMatrice();
    long n = citesteN();


    cout << "Matricea data este:" << endl;
    afiseazaMatrice(A);
    cout << endl << endl;

    cout << "Sumele calculate sunt:" << endl;
    cout << "-------Cu brute force-------" << endl;
    afiseazaMatrice(calculeazaSFolosindBruteForce(A, n));
    cout << "-------In O(log^2 n)--------" << endl;
    afiseazaMatrice(calculeazaSInLogPatrat(A, n));
    cout << "-------In O(log n)----------" << endl;
    afiseazaMatrice(calculeazaSInLog(A, n));
    cout << "--------------" << endl;

    return 0;
}
```

Pascal

```
Program structuri;
const p = 666013;

{Retine o matrice de 2x2 de numere intregi.}
type Matrice2x2 = record
{ Elementele matricei trebuie long deoarece p*p > INT_MAX}
    elems:array[0..10,0..10] of longint;
    linii: longint;
    coloane: longint;
end;
 {Initializeaza o matrice de 2x2, cu elementele luate modulo p.
    Input:
    - xij: elementul de pe linia i si coloana j (mod p)
    Output: -}

function initializareMatrice (x00, x01, x10, x11:longint):Matrice2x2;
var x: Matrice2x2;
   begin
        x.elems[0,0] := x00 mod p;
        x.elems[0,1] := x01 mod p;
        x.elems[1,0] := x10 mod p;
        x.elems[1,1] := x11 mod p;
    initializareMatrice:=x;
  end;

  {Returneaza matricea unitate de 2x2.}
function getUnitate():matrice2x2;
   var result : Matrice2x2;
   begin
       result:=initializareMatrice(1, 0, 0, 1);
       getUnitate := result;
```

```
   end;

{Inmulteste doua matrici mod P.
    Input:
    - X: o matrice de 2x2 de intregi
    - Y: o matrice de 2x2 de intregi
    Output:
    - X*Y (mod p)}

function inmulteste(X : Matrice2x2;Y : Matrice2x2) : Matrice2x2;
var Res: Matrice2x2;
begin
        Res := initializareMatrice(0, 0, 0, 0);
        Res.elems[0,0] := (X.elems[0,0]*Y.elems[0,0] + X.elems[0,1]*Y.elems↩
           [1,0]) mod p;
        Res.elems[0,1] := (X.elems[0,0]*Y.elems[0,1] + X.elems[0,1]*Y.elems↩
           [1,1]) mod p;
        Res.elems[1,0] := (X.elems[1,0]*Y.elems[0,0] + X.elems[1,1]*Y.elems↩
           [1,0]) mod p;
        Res.elems[1,1] := (X.elems[1,0]*Y.elems[0,1] + X.elems[1,1]*Y.elems↩
           [1,1]) mod p;
        inmulteste := Res;
end;

{Aduna doua matrici mod P.
Input:
- X: o matrice de 2x2 de intregi
- Y: o matrice de 2x2 de intregi
Output:
- X + Y (mod p)}

function aduna( X : Matrice2x2; Y: Matrice2x2):Matrice2x2 ;
var Res : Matrice2x2;
begin
        Res := initializareMatrice(0, 0, 0, 0);
        Res.elems[0,0] := (X.elems[0,0] + Y.elems[0,0]) mod p;
        Res.elems[0,1] := (X.elems[0,1] + Y.elems[0,1]) mod p;
        Res.elems[1,0] := (X.elems[1,0] + Y.elems[1,0]) mod p;
        Res.elems[1,1] := (X.elems[1,1] + Y.elems[1,1]) mod p;
        aduna:= Res;
end;

{Ridica o matrice la o putere data (mod p).
Input:
- X: matrice de 2x2 de intregi
- exponent: puterea la care sa se ridice X
Output:
- X la puterea exponent (mod p)}

function ridicaLaPutere( X : Matrice2x2; exponent : integer): Matrice2x2;
var jumatate, jumatatePatrat, Res :  Matrice2x2;
begin
    if (exponent = 0) then ridicaLaPutere := getUnitate()
    else begin
```

```
        jumatate := ridicaLaPutere(X, exponent div 2);
        jumatatePatrat := inmulteste(jumatate, jumatate);

        if (exponent mod 2 = 0)
            then Res := jumatatePatrat
            else Res := inmulteste(jumatatePatrat, X);
        ridicaLaPutere:=Res;
    end;
end;

{Citeste o matrice de 2x2.
Input: -
Output:
- matricea citita}
procedure citesteMatrice(var m:Matrice2x2);
var i,j:integer;
begin
   for i:=0 to 1 do begin
     for j:=0 to 1 do begin
        write('m[',i,',',j,']=');
        readln(m.elems[i,j]);
     end;
   end;
end;

{Afiseaza o matrice de 2x2.
Input:
- X: matricea de afisat
Output: -}
procedure afiseazaMatrice(var m:Matrice2x2);
var i,j:integer;
begin
  writeln;
  for i:=0 to 1 do
  begin
      for j:=0 to 1 do
         write(m.elems[i,j]:2,',');
      writeln;
  end;
end;

{Citeste n-ul din enunt.
Input: -
Output:
- valoarea citita }
procedure citesteNumar(n:integer);
begin
 write('"Dati n (pana la cat se calculeaza suma): ');
 readln(n);
end;

{Calculeaza suma S(n), in complexitate O(n)
Input:
- A: matricea A din enunt.
- n: limita din enunt.
```

```
Output:
- S(n) (mod p).}
function calculeazaSFolosindBruteForce(var A: Matrice2x2; n:longint):↩
   Matrice2x2;
var res,res1,curent1, curent : Matrice2x2; i:integer;
begin
    res := A;
    curent := inmulteste(A, A);
    for  i := 2 to n do
        begin
        res1 := aduna(res, curent);
        curent1 := inmulteste(curent, A);
        res:=res1;
        curent:=curent1;
        end;
calculeazaSFolosindBruteForce:=res;
 end;

 {Calculeaza suma S(n), in complexitate O(log^2 n)
Input:
- A: matricea A din enunt.
- n: limita din enunt.
Output:
- S(n) (mod p).}
function calculeazaSInLogPatrat(var A : Matrice2x2; n : longint) : Matrice2x2;
var res, jumatate, unitatePlusALaNpe2, tot : Matrice2x2;
    k: longint;
begin
     // n >= 1 in enunt. Daca poate fi si 0?
    if (n = 1) then calculeazaSInLogPatrat := A
    else begin
        k := n div 2;
        jumatate := calculeazaSInLogPatrat(A, k);
        // apeleaza ridicaLaPutere, care are complexitatea O(log n)
        // rezulta complexitate totala O(log^2 n)
        unitatePlusALaNpe2 := aduna(getUnitate(), ridicaLaPutere(A, k));
        tot := inmulteste(unitatePlusALaNpe2, jumatate);
        if (n mod 2 = 0) then res := tot
                         else res := aduna(tot, ridicaLaPutere(A, n));

        calculeazaSInLogPatrat := res;
    end;
end;


{Helper pentru calculeaza suma S(n), in complexitate O(log n).
Input:
- A: matricea A din enunt.
- n: limita din enunt.
- AlaPutereaNpe2: folosit pentru a calcula A^n in acelasi timp cu suma.
Output:
- S(n) (mod p).}
 function calculeazaSInLogHelper(var A : Matrice2x2; n :longint; var ↩
    AlaPutereaNpe2:Matrice2x2) : Matrice2x2;
var jumatate, unitatePlusALanPe2, tot, patrat, result : Matrice2x2;
```

```
    k : longint;
begin
    if (n = 1) then
        begin
           AlaPutereaNpe2 := A;
           calculeazaSInLogHelper := AlaPutereaNpe2;
        end
    else begin
        k := n div 2;
        jumatate := calculeazaSInLogHelper(A, k, AlaPutereaNpe2);
        // nu se mai apeleaza functia de ridicare la putere => complexitatea ↩
           totala ramane O(log n)
        unitatePlusALanPe2 := aduna(getUnitate(), AlaPutereaNpe2);
        tot := inmulteste(unitatePlusALanPe2, jumatate);

        patrat := inmulteste(AlaPutereaNpe2, AlaPutereaNpe2);

        if (n mod 2 = 0) then
        begin
            AlaPutereaNpe2 := patrat;
            calculeazaSInLogHelper := tot;
       end
       else
       begin
           AlaPutereaNpe2 := inmulteste(patrat, A);
           calculeazaSInLogHelper := aduna(tot, AlaPutereaNpe2);
       end;
    end;
end;

{Calculeaza suma S(n), in complexitate O(log n)
Input:
- A: matricea A din enunt.
- n: limita din enunt.
Output:
- S(n) (mod p).}
function calculeazaSInLog(var A:Matrice2x2; n:longint): Matrice2x2;
var temp, res : Matrice2x2 ;
begin
    temp:=initializareMatrice(0, 0, 0, 0);
    res := calculeazaSInLogHelper(A, n,temp);
    calculeazaSInLog := res;
end;

var A,aux: Matrice2x2;
    n : longint;

begin
    citesteMatrice(A);
    citesteNumar(n);
    write('Matricea data este:');
    afiseazaMatrice(A);
    writeln('Sumele calculate sunt:');
    write ( '-------Cu brute force-------' );
    aux:=calculeazaSFolosindBruteForce(A, n);
```

```
    afiseazaMatrice(aux);
    write('-------In O(log^2 n)--------');
    aux:=calculeazaSInLogPatrat(A, n);
    afiseazaMatrice(aux);
    write ( '-------In O(log n)----------' );
    aux:=calculeazaSInLog(A, n);
    afiseazaMatrice(aux);
    writeln ( '--------------');
end.
```

Problemă de programare

Enunț

Scrieți o aplicație pentru o patiserie care să țină evidența:

- **Prajiturilor**: idPrajitura, denumirePrajitura, cantitatePrajitura
- **Clientilor**: idClient, numeClient, adresaClient
- **Comenzilor**: idComanda, idPrajitura, idClient, an

Aplicația va gestiona o listă de Prajituri, o listă de Clienti și o listă de Comenzi cu următoarele funcționalități:

- Adaugă Prăjitură / Client
- Afișare Prăjituri și Clienți
- Creează rapoarte:
    1. Prăjiturile cu cantitatea mai mare decât o cantitate precizată
    2. Adăugare comandă
    3. Afișare comenzi

Analiza

Identificarea entităților

- **Prajitura**
    - **id**: intreg
    - **denumire**: string

    - **cantitate**: real
- **SirPrajituri**:
    - **sir**: Prajitura []
    - **nr**: intreg
- **Client**:
    - **id**: intreg
    - **nume**: string
    - **adresa**: string
- **SirClienti**:
    - **sir**: Client []
    - **nr**: intreg
- **Comanda**:
    - **idComanda**: intreg
    - **idClient**: intreg
    - **idPrajitura**: intreg
    - **an**: intreg
- **SirComenzi**:
    - **sir**: Comanda []
    - **nr**: intreg

Implementare

C++

```
#include <iostream>

using namespace std;

/*
Structura pentru o prajitura.
*/
struct Prajitura {
```

```
    int id;
    string denumire;
    float cantitate;

    /*
    Constructor fara parametri: variabilele raman neinitializate.
    */
    Prajitura() { }

    /*
    Construieste o Prajitura cu anumiti parametri.
    Input:
    - id: id-ul prajiturii.
    - denumire: denumirea.
    - cantitate: cantitatea.
    */
    Prajitura(int id, string denumire, float cantitate)
    {
        this->id = id;
        this->denumire = denumire;
        this->cantitate = cantitate;
    }
} ;

/*
Structura pentru un sir de prajituri.
*/
struct SirPrajituri {
    int nr = 0;
    Prajitura sir[100];

} ;

/*
Structura pentru un client.
*/
struct Client
{
    int id;
    string nume, adresa;

    /*
    Constructor fara parametri: variabilele raman neinitializate.
    */
    Client() { }

    /*
    Construieste un client cu anumiti parametri.
    Input:
    - id: id-ul clientului.
    - nume: numele clientului.
    - adresa: adresa clientului.
    */
    Client(int id, string nume, string adresa)
    {
```

```
        this->id = id;
        this->nume = nume;
        this->adresa = adresa;
    }
};

/*
Structura pentru un sir de clienti.
*/
struct SirClienti
{
    int nr = 0;
    Client sir[100];

} ;

/*
Structura pentru o comanda.
*/
struct Comanda
{
    int idComanda, idClient, idPrajitura, an;

    /*
    Constructor fara parametri: variabilele raman neinitializate.
    */
    Comanda() { }

    /*
    Construieste o comanda cu anumiti parametri.
    Input:
    - idComanda: id-ul comenzii.
    - idClient: id-ul clientului care a facut comanda.
    - idPrajitura: id-ul prajiturii care a fost comandata.
    - an: anul comenzii.
    */
    Comanda(int idComanda, int idClient, int idPrajitura, int an)
    {
        this->idComanda = idComanda;
        this->idClient = idClient;
        this->idPrajitura = idPrajitura;
        this->an = an;
    }
};

/*
Structura pentru un sir de comenzi.
*/
struct SirComenzi {
    int nr = 0;
    Comanda sir[100];

} ;

/*
```

```
Adauga o prajitura in sirul prajiturilor.
Input:
- prajituri: sirul de prajituri.
- id: id-ul prajiturii. Trebuie sa fie unic. TODO: eroare daca nu este unic.
- denumire: denumirea prajiturii.
- cantitate: cantitatea prajiturii.
*/
void adaugaPrajitura(SirPrajituri &prajituri, int id, string denumire, float ↩
    cantitate)
{
    prajituri.sir[prajituri.nr++] = Prajitura(id, denumire, cantitate);
}

/*
Adauga un client in sirul de clienti.
Input:
- clienti: sirul clientilor.
- id: id-ul clientului. Trebuie sa fie unic. TODO: eroare daca nu este unic.
- nume: numele clientului.
- adresa: adresa clientului.
*/
void adaugaClient(SirClienti &clienti, int id, string nume, string adresa)
{
    clienti.sir[clienti.nr++] = Client(id, nume, adresa);
}

/*
Afiseaza un sir de prajituri.
Input:
- prajituri: sirul care sa se afiseze.
*/
void afisarePrajituri(SirPrajituri prajituri)
{
    for (int i = 0; i < prajituri.nr; ++i)
    {
        Prajitura p = prajituri.sir[i];
        cout << "Prajitura #" << p.id << ": " << p.denumire << ", cantitate=" ↩
            << p.cantitate << endl;
    }
}

/*
Afiseaza un sir de clienti.
Input:
- clienti: sirl care sa se afiseze.
*/
void afisareClienti(SirClienti clienti)
{
    for (int i = 0; i < clienti.nr; ++i)
    {
        Client c = clienti.sir[i];
        cout << "Clientul #" << c.id << ": " << c.nume << ", adresa=" << c.↩
            adresa << endl;
    }
}
```

```
/*
Determina prajiturile cu o cantitate mai mare decat un numar dat.
Input:
- prajituri: sirul de prajituri.
- cantitateMinima: numarul relativ la care se determina prajiturile cu o ↩
   cantitate minima.
Output:
- un sir de prajituri cu cantitate > cantitateMinima.
*/
SirPrajituri getPrajituriCuCantitateMaiMareDecat(SirPrajituri prajituri, float↩
    cantitateMinima)
{
    SirPrajituri rezultat;
    for (int i = 0; i < prajituri.nr; ++i)
    {
        if (prajituri.sir[i].cantitate > cantitateMinima)
        {
            rezultat.sir[rezultat.nr++] = prajituri.sir[i];
        }
    }

    return rezultat;
}

/*
Determina o prajitura cu un id dat.
Input:
- prajituri: sirul de prajituri.
- id: id-ul dat.
Output:
- o prajitura cu id-ul id.
*/
Prajitura getPrajituraCuId(SirPrajituri prajituri, int id)
{
    for (int i = 0; i < prajituri.nr; ++i)
    {
        if (prajituri.sir[i].id == id)
        {
            return prajituri.sir[i];
        }
    }
}

/*
Determina un client cu un id dat.
Input:
- clienti: sirul de clienti.
- id: id-ul dat.
Output:
- un client cu id-ul id.
*/
Client getClientCuId(SirClienti clienti, int id)
{
    for (int i = 0; i < clienti.nr; ++i)
```

```
    {
        if (clienti.sir[i].id == id)
        {
            return clienti.sir[i];
        }
    }
}

/*
Adauga o comanda de prajitura.
Input;
- comenzi: sirul de comenzi.
- idComanda: id-ul comenzii.
- idClient: id-ul clientului (se presupune ca exista).
- idPrajitura: id-ul prajiturii (se presupune ca exista).
- an: anul comenzii.
*/
void adaugaComanda(SirComenzi& comenzi, int idComanda, int idClient, int ↩
    idPrajitura, int an)
{
    comenzi.sir[comenzi.nr++] = Comanda(idComanda, idClient, idPrajitura, an);
}

/*
Afiseaza toate comenzile.
Input;
- comenzi: sirul comenzilor.
- clienti: sirul clientilor.
- prajituri: sirul prajiturilor.
*/
void afiseazaComenzi(SirComenzi comenzi, SirClienti clienti, SirPrajituri ↩
    prajituri)
{
    for (int i = 0; i < comenzi.nr; ++i)
    {
        Comanda cmd = comenzi.sir[i];
        Prajitura p = getPrajituraCuId(prajituri, cmd.idPrajitura);
        Client c = getClientCuId(clienti, cmd.idClient);

        cout << "Comanda #" << cmd.idComanda << ", din anul " << cmd.an << ", ↩
            la clientul " << c.nume << ", cu adresa "
            << c.adresa << ", pentru prajitura " << p.denumire << endl;
    }
}

/*
Adauga date initiale.
Input;
- comenzi: sirul comenzilor.
- clienti: sirul clientilor.
- prajituri: sirul prajiturilor.
*/
void populareCuDate(SirClienti &clienti, SirPrajituri &prajituri, SirComenzi &↩
    comenzi)
{
```

```
    adaugaClient(clienti, 1, "Vlad", "adrVlad");
    adaugaClient(clienti, 2, "Adriana", "adrAdriana");
    adaugaClient(clienti, 3, "Maria", "adrMaria");

    adaugaPrajitura(prajituri, 1, "ecler", 10);
    adaugaPrajitura(prajituri, 2, "rulada", 20);
    adaugaPrajitura(prajituri, 3, "lamaita", 30);
    adaugaPrajitura(prajituri, 4, "nucsoara", 40);
    adaugaPrajitura(prajituri, 5, "dobos", 50);
    adaugaPrajitura(prajituri, 6, "cheesecake", 60);

    adaugaComanda(comenzi, 1, 1, 1, 2017);
    adaugaComanda(comenzi, 2, 1, 2, 2017);
    adaugaComanda(comenzi, 3, 1, 3, 2016);
    adaugaComanda(comenzi, 4, 2, 5, 2017);
    adaugaComanda(comenzi, 5, 2, 6, 2017);
    adaugaComanda(comenzi, 6, 2, 4, 2016);
    adaugaComanda(comenzi, 7, 2, 3, 2017);
    adaugaComanda(comenzi, 8, 3, 1, 2017);
    adaugaComanda(comenzi, 9, 3, 3, 2016);
    adaugaComanda(comenzi, 10, 3, 4, 2017);
    adaugaComanda(comenzi, 11, 3, 5, 2017);
    adaugaComanda(comenzi, 12, 3, 6, 2016);
}

/*
Afiseaza meniul cu optiunile utilizatorului.
*/
void afisareMeniu()
{
    cout << "1. Adauga client." << endl;
    cout << "2. Adauga prajitura." << endl;
    cout << "3. Afisare client." << endl;
    cout << "4. Afisare prajituri." << endl;
    cout << "5. Raport prajituri cu o anumita cantitate minima." << endl;
    cout << "6. Adauga comanda." << endl;
    cout << "7. Afisare comenzi." << endl;
    cout << "0. Iesire." << endl;

    cout << endl;

}

/*
Porneste interfata cu utilizatorul.
Input:
- comenzi: sirul comenzilor.
- clienti: sirul clientilor.
- prajituri: sirul prajiturilor.
*/
void rulareInterfata(SirClienti &clienti, SirPrajituri &prajituri, SirComenzi ↩
   &comenzi)
{
    while (true)
    {
```

```
        afisareMeniu();

        int optiune;
        cin >> optiune;
        int id;
        string nume, adresa;
        string denumire;
        float cantitate;

        int idClient, idPrajitura, an;

        switch (optiune)
        {
        case 1:
            cout << "Dati id-ul, numele si adresa, separate prin spatii: ";
            cin >> id >> nume >> adresa;
            adaugaClient(clienti, id, nume, adresa);
            break;
        case 2:
            cout << "Dati id-ul, denumirea si cantitatea, separate prin spatii↩
                : ";
            cin >> id >> denumire >> cantitate;
            adaugaPrajitura(prajituri, id, denumire, cantitate);
            break;
        case 3:
            afisareClienti(clienti);
            break;
        case 4:
            afisarePrajituri(prajituri);
            break;
        case 5:
            cout << "Dati cantitatea minima: ";
            cin >> cantitate;
            afisarePrajituri(getPrajituriCuCantitateMaiMareDecat(prajituri, ↩
                cantitate));
            break;
        case 6:
            cout << "Dati id-ul comenzii, clientului, prajiturii si anul ↩
                separate prin spatii: ";
            cin >> id >> idClient >> idPrajitura >> an;
            adaugaComanda(comenzi, id, idClient, idPrajitura, an);
            break;
        case 7:
            afiseazaComenzi(comenzi, clienti, prajituri);
            break;
        case 0:
            return;
        default:
            cout << "Comanda invalida!" << endl << endl;
        }
    }
}

int main()
{
```

```
    SirClienti clienti;
    SirPrajituri prajituri;
    SirComenzi comenzi;


    populareCuDate(clienti, prajituri, comenzi);

    rulareInterfata(clienti, prajituri, comenzi);

    return 0;
}
```

Pascal

```
Program prajituradecraciun;
{Structura pentru o prajitura.}
type Prajitura=record
    denumire:string[50];
    id:integer;
    cantitate: double;
end;

{Structura pentru un sir de prajituri.}
type SirPrajituri=record
    sir:array[1..100] of Prajitura;
    nr:integer;
end;

{Structura pentru un client.}
type Client=record
    id:integer;
    nume,adresa:string[50];
end;

{Structura pentru un sir de clienti.}
type SirClienti=record
    nr:integer;
    sir:array[1..100] of Client;
end;

{Structura pentru o comanda.}
type Comanda=record
    idComanda,idClient,idPrajitura,an:integer;
end;

{Structura pentru un sir de comenzi.}
type SirComenzi=record
    nr:integer;
    sir:array[1..100] of Comanda;
end;


{Adauga un client in sirul de clienti.
```

```
Input:
- clienti: sirul clientilor.
- id: id-ul clientului. Trebuie sa fie unic. TODO: eroare daca nu este unic.
- nume: numele clientului.
- adresa: adresa clientului.}
procedure adaugaClient(var clienti:SirClienti;idC:integer;numeC:string;adresaC:string);
    var i:integer;
    begin
    inc(clienti.nr);
    i:=clienti.nr;
    clienti.sir[i].id:=idC;
    clienti.sir[i].nume:=numeC;
    clienti.sir[i].adresa:=adresaC;
end;


{Adauga o prajitura in sirul prajiturilor.
Input:
- prajituri: sirul de prajituri.
- id: id-ul prajiturii. Trebuie sa fie unic. TODO: eroare daca nu este unic.
- denumire: denumirea prajiturii.
- cantitate: cantitatea prajiturii.}
procedure adaugaPrajitura(var prajituri:SirPrajituri;id:integer;denumire:string;cantitate:double);
    var i:integer;
    begin
    inc(prajituri.nr);
    i:=prajituri.nr;
    prajituri.sir[i].id:=id;
    prajituri.sir[i].denumire:=denumire;
    prajituri.sir[i].cantitate:=cantitate;
end;

{Afiseaza un sir de prajituri.
Input:
- prajituri: sirul care sa se afiseze.}
procedure afisarePrajituri(var prajituri:SirPrajituri);
    var i:integer;
    begin
    for i:=1 to prajituri.nr do
        begin
            writeln(prajituri.sir[i].id,';',prajituri.sir[i].denumire,';',
               prajituri.sir[i].cantitate);
        end;
end;

{Afiseaza un sir de clienti.
Input:
- clienti: sirl care sa se afiseze.}
procedure afisareClienti(var clienti:SirClienti);
    var i:integer;
    begin
    for i:=1 to clienti.nr do
        begin
```

```
            writeln(clienti.sir[i].id,';',clienti.sir[i].nume,';',clienti.sir[↩
                i].adresa);
        end;
end;


{Determina prajiturile cu o cantitate mai mare decat un numar dat.
Input:
- prajituri: sirul de prajituri.
- cantitateMinima: numarul relativ la care se determina prajiturile cu o ↩
   cantitate minima.
Output:
- un sir de prajituri cu cantitate > cantitateMinima.}
function getPrajituriCuCantitateMaiMareDecat(prajituri:SirPrajituri;↩
   cantitateMinima:double):SirPrajituri;
    var i:integer;
    rezultat:SirPrajituri;
    begin
    rezultat.nr:=0;
    for i:=1 to prajituri.nr do
        begin
            if (prajituri.sir[i].cantitate > cantitateMinima) then
                begin
                inc(rezultat.nr);
                rezultat.sir[i]:=prajituri.sir[i];
                end;
        end;
    getPrajituriCuCantitateMaiMareDecat:=rezultat;
end;

{Determina o prajitura cu un id dat.
Input:
- prajituri: sirul de prajituri.
- id: id-ul dat.
Output:
- o prajitura cu id-ul id.}
function getPrajituraCuId(prajituri:SirPrajituri;id:integer):Prajitura;
    var i:integer;
    rezultat:Prajitura;
    begin
    for i:=1 to prajituri.nr do
        begin
            if (prajituri.sir[i].id = id) then
                rezultat:=prajituri.sir[i];
        end;
    getPrajituraCuId:=rezultat;
end;

{Determina un client cu un id dat.
Input:
- clienti: sirul de clienti.
- id: id-ul dat.
Output:
- un client cu id-ul id.}
function getClientCuId(clienti:SirClienti;id:integer):Client;
```

```
    var i:integer;
    rezultat:Client;
    begin
    for i:=1 to clienti.nr do
        begin
            if (clienti.sir[i].id = id) then
                rezultat:=clienti.sir[i];
        end;
    getClientCuId:=rezultat;
end;

{Adauga o comanda de prajitura.
Input;
- comenzi: sirul de comenzi.
- idComanda: id-ul comenzii.
- idClient: id-ul clientului (se presupune ca exista).
- idPrajitura: id-ul prajiturii (se presupune ca exista).
- an: anul comenzii.}
procedure adaugaComanda(var comenzi:SirComenzi;idComanda:integer;idClient:↩
    integer;idPrajitura:integer;an:integer);
    begin
    inc(comenzi.nr);
    comenzi.sir[comenzi.nr].idComanda:=idComanda;
    comenzi.sir[comenzi.nr].idClient:=idClient;
    comenzi.sir[comenzi.nr].idPrajitura:=idPrajitura;
    comenzi.sir[comenzi.nr].an:=an;
end;

{Afiseaza toate comenzile.
Input;
- comenzi: sirul comenzilor.
- clienti: sirul clientilor.
- prajituri: sirul prajiturilor.}
procedure afiseazaComenzi(comenzi:SirComenzi;clienti:SirClienti;prajituri:↩
    SirPrajituri);
    var i:integer;
    cmd:Comanda;
    p:Prajitura;
    cl:Client;
    begin
    for i:=1 to comenzi.nr do
        begin
            cmd:=comenzi.sir[i];
            p:=getPrajituraCuId(prajituri,cmd.idPrajitura);
            cl:=getClientCuId(clienti,cmd.idClient);
            writeln('Comanda #',cmd.idComanda,',din anul',cmd.an,',la clientul↩
                ',cl.nume,',cu adresa',cl.adresa,
                    ', pentru prajitura',p.denumire);
        end;
end;

{Adauga date initiale.
Input;
- comenzi: sirul comenzilor.
- clienti: sirul clientilor.
```

```
- prajituri: sirul prajiturilor.}
procedure populareCuDate(var clienti:SirClienti;var prajituri:SirPrajituri;var↩
    comenzi:SirComenzi);
    begin
    adaugaClient(clienti, 1, 'Vlad', 'adrVlad');
    adaugaClient(clienti, 2, 'Adriana', 'adrAdriana');
    adaugaClient(clienti, 3, 'Maria', 'adrMaria');

    adaugaPrajitura(prajituri, 1, 'ecler', 10);
    adaugaPrajitura(prajituri, 2, 'rulada', 20);
    adaugaPrajitura(prajituri, 3, 'lamaita', 30);
    adaugaPrajitura(prajituri, 4, 'nucsoara', 40);
    adaugaPrajitura(prajituri, 5, 'dobos', 50);
    adaugaPrajitura(prajituri, 6, 'cheesecake', 60);

    adaugaComanda(comenzi, 1, 1, 1, 2017);
    adaugaComanda(comenzi, 2, 1, 2, 2017);
    adaugaComanda(comenzi, 3, 1, 3, 2016);
    adaugaComanda(comenzi, 4, 2, 5, 2017);
    adaugaComanda(comenzi, 5, 2, 6, 2017);
    adaugaComanda(comenzi, 6, 2, 4, 2016);
    adaugaComanda(comenzi, 7, 2, 3, 2017);
    adaugaComanda(comenzi, 8, 3, 1, 2017);
    adaugaComanda(comenzi, 9, 3, 3, 2016);
    adaugaComanda(comenzi, 10, 3, 4, 2017);
    adaugaComanda(comenzi, 11, 3, 5, 2017);
    adaugaComanda(comenzi, 12, 3, 6, 2016);
end;

{Afiseaza meniul cu optiunile utilizatorului.}
procedure afisareMeniu();
    begin
    writeln('1. Adauga client');
    writeln('2. Adauga prajitura');
    writeln('3. Afisare client.');
    writeln('4. Afisare prajituri.');
    writeln('5. Raport prajituri cu o anumita cantitate minima.');
    writeln('6. Adauga comanda.');
    writeln('7. Afisare comenzi.');
    writeln('0. Iesire');
end;

{Porneste interfata cu utilizatorul.
Input:
- comenzi: sirul comenzilor.
- clienti: sirul clientilor.
- prajituri: sirul prajiturilor.}
procedure rulareInterfata(var clienti:SirClienti;var prajituri:SirPrajituri;↩
   var comenzi:SirComenzi);
    var id,idClient,idPrajitura,an,optiune:integer;
    var denumire,nume,adresa:string;
    var cantitate:double;
    var aux:SirPrajituri;
    begin
    while (true) do
```

```
begin
afisareMeniu();
readln(optiune);
case optiune of
1:
begin
    write('Dati id-ul:');
    readln(id);
    write('Dati numele:');
    readln(nume);
    write('Dati adresa:');
    readln(adresa);
    adaugaClient(clienti, id, nume, adresa);
    break;
  end;
2:
begin
    write('Dati id-ul:');
    readln(id);
    write('Dati denumirea:');
    readln(denumire);
    write('Dati cantitate:');
    readln(cantitate);
    adaugaPrajitura(prajituri, id, denumire, cantitate);
    break;
   end;
3:
begin
    afisareClienti(clienti);
    break;
    end;
4:
begin
    afisarePrajituri(prajituri);
    break;
    end;
5:
begin
    writeln('Dati cantitatea minima:');
    readln(cantitate);
    aux:=getPrajituriCuCantitateMaiMareDecat(prajituri, cantitate);
    afisarePrajituri(aux);
    break;
    end;
6:
begin
    write('Dati id-ul comenzii:');
    readln(id);
    write('Dati id-ul clientului:');
    readln(idClient);
    write('Dati id-ul prajiturii:');
    readln(idPrajitura);
    write('Dati anul:');
    readln(an);
    adaugaComanda(comenzi, id, idClient, idPrajitura, an);
```

```
            break;
            end;
        7:
        begin
            afiseazaComenzi(comenzi, clienti, prajituri);
            break;
            end;
        0:
            break;
        else
            writeln('Comanda invalida!');
        end;
    end;
end;

var clienti:SirClienti;
var prajituri:SirPrajituri;
var comenzi:SirComenzi;

begin
    clienti.nr:=0;
    prajituri.nr:=0;
    comenzi.nr:=0;
    populareCuDate(clienti, prajituri, comenzi);
    rulareInterfata(clienti, prajituri, comenzi);
end.
```