

# **CURS PREGATIRE ADMITERE UBB**

## **2023**

**-RECAPITULAREA MATERIEI DE CLASA A IX-A-**

# CE CONTINE ACEST CURS?

- IN CREEAREA ACESTUI CURS, AM CONDENSATE TOATE CAPITOLELE DE CARE AVETI NEVOIE PENTRU A OBTINE UN REZULTAT BUN LA ADMITERE
- PRIMELE 3 CURSURI VOR FI DE RECAPITULARE A MATERIEI DE LICEU, URMAND CA DUPA ACEEA SA VORBIM DESPRE COMPLEXITATI, RECURSIVITATE, PSEUDOCOD, SA REZOLVAM GRILE SI SA VEDEM CUM PUTEM MAXIMIZA NOTELE PE CARE LE VETI OBTINE

# CAPITOЛЕLE CLASEI A IX-A

## Elemente de baza ale limbajului

- Structuri de date
- Structuri liniare
- Structuri Alternative
- Structuri repetitive

## Algoritmi elementari

- Cifrele unui numar
- Divizibilitate
- Baze de numeratie
- Generari de siruri

# CAPITOLETE CLASEI A IX-A

## Vectori (Tablouri Unidimensionale)

- Inserari / Stergeri
- Sortari
- Interclasare
- Cautare Binara
- Frecvente si Secvente

## Matrici (Tablouri Bidimensionale)

- Matrici Oarecare
- Matrici Patratice

# RECAPITULAREA NOTIUNILOR DE BAZA

- VARIABILA – O LOCATIE DE MEMORIE ALOCATA UNUI SIMBOL (N, X) UNDE PUTEM STOCA DIFERITE INFORMATII
- STRUCTURA ALTERNATIVA – STRUCTURA DE DECIZIE (IF ... ELSE, SWITCH ... CASE)
- STRUCTURA REPETITIVA – LOOP-URILE (FOR, WHILE, DO ... WHILE)
- STRUCTURI LINIARE – EXPRESIILE CARE SUNT EVALUATE PE O SINGURA LINIE DE COD (DECLARARI, ATRIBUIRI ETC.)

# ELEMENTE DE BAZA ALE LIMBAJULUI

- ALATURAT SE POT VEDEA NISTE DECLARARI DE VARIABILE SI UTILIZAREA CELOR 2 STRUCTURE ALTERNATIVE DE BAZA DIN C++.
- DESIGUR, LA ADMITERE VA FI VORBA DESPRE PSEUDOCOD, INSA MOMENTAN PENTRU A INTELEGE TOTI ALGORITMII VOM MERGE PE C++

```
4 int main(){
5     int x; // declararea unei variabile
6     x = 10; // initializarea lui x
7     int y = 7; // declararea si initializarea unei variabile
8     if(y > 10)
9         cout << "Expresia evaluata a fost adevarata\n";
10    else cout << "Expresia evaluata a fost falsa\n";
11    switch(y){
12        case 10:
13            cout << "y este 10";
14            break;
15        case 9:
16            cout << "y este 9";
17            break;
18        default:
19            cout << "y nu este nici 10 si nici 9";
20    }
21    return 0;
22 }
```

# ELEMENTE DE BAZA ALE LIMBAJULUI

- ALATURAT VETI PUTEA VEDEA CELE 3 STRUCTURI REPETITIVE (EXISTA SI IN PSEUDOCOD).
- TOATE CELE 3 FAC ACELASI LUCRU, DAR SUNT SCRISE SUB CELE 3 FORME CUNOSCUTE:
  - STRUCTURA REPETITIVE CU TEST INITIAL SI NUMAR CUNOSCUT DE PASI
  - STRUCTURA REPETITIVE CU TEST INITIAL SI NUMAR NECUNOSCUT DE PASI
  - STRUCTURA REPETITIVE CU TEST FINAL SI NUMAR NECUNOSCUT DE PASI

```
4 int main(){
5
6     // Structura #1
7     for(int i = 1; i <= 5; ++i){
8         cout << i << ' ';
9     } // 1 2 3 4 5
10
11    // Structura #2
12    int i = 1;
13    while(i <= 5){
14        cout << i << ' ';
15        i++;
16    } // 1 2 3 4 5
17
18    // Structura #3
19    i = 1;
20    do{
21        cout << i << ' ';
22        i += 1;
23    }while(i <= 5); // 1 2 3 4 5
24
25    return 0;
26 }
```

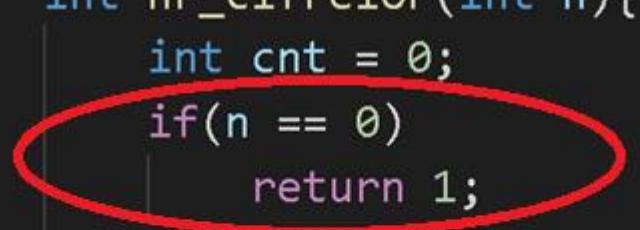
# ALGORITMI ELEMENTARI – CIFRELE UNUI NUMAR

- IN CADRUL CAPITOLULUI DE ALGORITMI ELEMENTARI, UNUL DINTRE CELE MAI IMPORTANTE SUBCAPITOЛЕ ESTE ACELA DE CIFRELE UNUI NUMAR.
- ACEST CAPITOL INSUMEAZA TOTI ALGORITMII CU AJUTORUL CARORA PRELUCRAM NUMERELE.
- DE REGULA, MAJORITATEA PROGRAMELOR CARE LUCREAZA CU DATE INTRODUSE DE USERI SE BAZEaza PE ALGORITMI DE PRELUCRAREA A NUMERELOR.

# CIFRELE UNUI NUMAR #1

```
4 int suma_cifrelor(int n){  
5     int sum = 0;  
6     while(n > 0){  
7         sum += (n % 10);  
8         n /= 10;  
9     }  
10    return sum;  
11 }
```

```
13 int nr_cifrelor(int n){  
14     int cnt = 0;  
15     if(n == 0)  
16         return 1;  
17     while(n > 0){  
18         cnt++;  
19         n /= 10;  
20     }  
21     return cnt;  
22 }
```



# CIFRELE UNUI NUMAR #2

```
4 int oglindit(int n){  
5     int ogl = 0;  
6     while(n > 0){  
7         ogl = ogl * 10 + (n % 10);  
8         n /= 10;  
9     }  
10    return ogl;  
11 }
```

```
13 // Se da un numar. Sa se returneze numarul  
14 // in urma eliminarii tuturor cifrelor impare  
15 int transformare(int n){  
16     int p = 1, nr_nou = 0;  
17     while(n > 0){  
18         int c = n % 10; // ultima cifra a lui n  
19         if(c % 2 == 0){  
20             nr_nou = nr_nou + c * p;  
21             p *= 10;  
22         }  
23         n /= 10;  
24     }  
25     return nr_nou;  
26 }
```

```
4 int transformare(int n){  
5     if(n == 0)  
6         return 0;  
7     if(n % 2 == 1)  
8         return transformare(n / 10);  
9     else return transformare(n / 10) * 10 + n % 10;  
10 }
```

Atentie la acest algoritm! Descrie UTILITATEA Recursivitatii!

# CIFRELE UNUI NUMAR #3

```
13 // Data de nastere din CNP
14 void data_de_nastere(long long cnp){
15     // cnp: 5012302436545
16     // 5 - sexul, 01 - anul, 23 - luna, 02 - ziua
17     cnp /= 1000000;
18     int zi = cnp % 100;
19     cnp /= 100;
20     int luna = cnp % 100;
21     cnp /= 100;
22     int an = cnp % 100;
23     if(an > 21)
24         an = 1900 + an;
25     else
26         an = 2000 + an;
27     if(an > 21)
28         cout << "Te-ai nascut in data de: " << zi << '.'
29         << luna << "." << an;
30     else
31         cout << "Te-ai nascut in data de: " << zi << ','
32         << luna << "." << an;
33 }
```

```
37     data_de_nastere(1731028060011);
38     // Te-ai nascut in data de 28.10.1973
```

Acesta este un exemplu practice de o un algoritm care verifica daca datele unui user au fost introduse corect. Din CNP, programul isi deduce datele de nastere si le compara cu cele introduse de user. Daca sunt bune, atunci userul este unul valid.

# ALGORITMI ELEMENTARI - DIVIZIBILITATE

- DESI PARE UN SUBCAPITOL MAI PUTIN IMPORTANT, VETI VEDEA CA IN ALGORITMICA, DIVIZIBILITATEA ESTE UNUL DINTRE CELE MAI COMPLEXE SI IMPORTANTE CAPITOLE.
- FIIND UNUL DINTRE CELE MAI STUFOASE, VOM SCRIE TOATE FUNCTIILE SI RECURSIV – RECURSIVITATEA FIIND UNUL DINTRE CAPITOLETELE PREFERATE ALE PROPUNATORILOR DE PROBLEME DE LA UBB
- ACEST CAPITOL CONTINUE: DESCOPUNEREA IN DIVIZORI, CMMDC, CMMMC, DESCOPUNERE IN FACTORI PRIMI ETC.

# DIVIZIBILITATE #1

```
4 void afisare_divizori(int n){  
5     for(int i = 1; i * i <= n; ++i)  
6         if(n % i == 0 && i * i < n)  
7             cout << i << ' ' << n / i << ' ';  
8         else if(i * i == n)  
9             cout << i; // pentru a nu afisa acelasi div de 2 ori  
10    }  
11  
12 void afisare_divizori_rec(int n, int d = 1){ // d = 1  
13     if(d * d > n)  
14         return ;  
15     if(d * d == n)  
16         cout << d, afisare_divizori_rec(n, d+1);  
17     else{  
18         if(n % d == 0)  
19             cout << d << ' ' << n / d << ' ';  
20         afisare_divizori_rec(n, d+1);  
21     }  
22 }
```

\* Sarim peste nr\_div si sum\_div pt ca se fac identice functiile

# DIVIZIBILITATE #2 (CMMDC)

```
4 // CMMDC
5 int cmmdc_scaderi(int a, int b){
6     while(a != b)
7         if(a > b)
8             a -= b;
9         else b -= a;
10    return a; // sau b (a = b)
11 }
12
13 int cmmdc_impartiri(int a, int b){
14     while(b > 0){
15         int r = a % b;
16         a = b;
17         b = r;
18     }
19     return a;
20 }
```

```
4 // CMMDC
5 int cmmdc_scaderi_rec(int a, int b){
6     if(a == b)
7         return a;
8     if(a < b)
9         return cmmdc_scaderi_rec(a, b - a);
10    else return cmmdc_scaderi_rec(a - b, b);
11 }
12
13 int cmmdc_impartiri_rec(int a, int b){
14     if(b == 0)
15         return a;
16     return cmmdc_scaderi_rec(b, a % b);
17 }
18
```

\* Varianta recursiva a celor din stanga

# DIVIZIBILITATE #3 (CMMMC)

CEL MAI MIC MULTIPLU COMUN A 2  
NUMERE SE CALCULEAZA DUPA  
FORMULA:  $A * B / \text{CMMDC}(A, B)$

```
1 #include <iostream>
2 using namespace std;
3
4 int cmmdc(int a, int b){
5     if(b == 0)
6         return a;
7     return cmmdc(b, a % b);
8 }
9
10 int main(){
11     int a, b;
12     cin >> a >> b;
13     cout << a / cmmdc(a, b) * b;
14     return 0;
15 }
```

## DIVIZIBILITATE #4 (VERIFICARE PRIM)

- EXISTA MULTIPLE METODE DE A VERIFICA PRIMALITATEA UNUI NUMAR. UN NR E PRIM DACA ARE NUMARUL DE DIVIZORI 2. ALATURAT AVETI 2 METODE DE VERIFICARE A PRIMALITATII.

```
4  bool is_prim(int n){  
5      for(int d = 2; d * d <= n; ++d)  
6          if(n % d == 0)  
7              return false;  
8      return true;  
9  }  
10  
11 bool is_prim(int n){  
12     if(n == 1 || n == 0)  
13         return false;  
14     if(n % 2 == 0 && n != 2)  
15         return false;  
16     for(int d = 3; d * d <= n; d += 2)  
17         if(n % d == 0)  
18             return false;  
19     return true;  
20 }
```

# DIVIZIBILITATE #5 (FACTORIZARE)

- ALGORITMUL ALATURAT AFISEAZA DESCOMPUNEREA IN FACTORI PRIMI A UNUI NUMAR.

EX: N = 120

2 ^ 3

3 ^ 1

5 ^ 1

```
4 void factorizare(int n){  
5     int d = 2;  
6     while(n > 1){  
7         int p = 0;  
8         while(n % d == 0){  
9             p += 1;  
10            n /= d;  
11        }  
12        if(p)  
13            cout << d << '^' << p << '\n';  
14        d++;  
15        if(d * d > n)  
16            d = n;  
17    }  
18 }
```

# DIVIZIBILITATE #6

## (NR DIV DIN FACTORIZARE)

- PENTRU UN NUMAR  $N = 2^{P_1} * 3^{P_2} * 5^{P_3}$
- FORMULA MATEMATICA NE ARATA CA NUMARUL DE DIVIZORI AI NUMARULUI N ESTE EGAL CU  $(P_1 + 1) * (P_2 + 1) * (P_3 + 1)$

```
4  int nr_div(int n){  
5      int d = 2;  
6      int nrdiv = 1;  
7      while(n > 1){  
8          int p = 0;  
9          while(n % d == 0){  
10              n /= d;  
11              p++;  
12          }  
13          if(p)  
14              nrdiv *= (p + 1);  
15          d++;  
16          if(d * d > n)  
17              d = n;  
18      }  
19      return nrdiv;  
20  }
```

# DIVIZIBILITATE EXTRA – FUNCTII RECURSIVE

```
4 void factorizare(int n, int d){  
5     if(n <= 1)  
6         return ;  
7     int p = 0;  
8     while(n % d == 0)  
9         n /= d, p++;  
10    if(p)  
11        cout << d << '^' << p << '\n';  
12    if(d * d > n)  
13        factorizare(n, n);  
14    else factorizare(n, d + 1);  
15 }
```

```
4 void factorizare(int n, int d, int p){  
5     if(n <= 1)  
6         return ;  
7     if(n % d == 0){  
8         n /= d, p ++;  
9         if(n % d)  
10            cout << d << '^' << p << '\n';  
11         factorizare(n, d, p);  
12     }  
13     else{  
14         if(d * d > n)  
15             factorizare(n, n, 0);  
16         else factorizare(n, d + 1, 0);  
17     }  
18 }
```

# BAZE DE NUMERATIE

- REPREZENTAREA NUMERELO  
R POATE FI FACUTA IN MULTIPLE BAZE  
DE NUMERATIE, BAZA DE  
NUMERATIE FOLOSITA DE OAMENI  
FIIND 10.
- CALCULATORUL FOLOSESTE BAZELE  
2 SI 16 IN FUNCTIE DE SITUATIE,  
ASTFEL, VOM VEDEA 2 FUNCTII  
CARE FAC CONVERSII DIN BAZA 10  
IN BAZA 2 SI BAZA 16.

```
4 void baza2(int n){  
5     int cifre[1001], indc = 0;  
6     while(n){  
7         cifre[++indc] = n % 2;  
8         n /= 2;  
9     }  
10    for(int i = indc; i >= 1; --i)  
11        cout << cifre[i];  
12    }  
13  
14 void baza16(int n){  
15     char cifre16[]={'0', '1', '2', '3', '4', '5', '6', '7',  
16     | | | | '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};  
17     int cifre[1001], indc = 0;  
18     while(n){  
19         cifre[++indc] = n % 16;  
20         n /= 16;  
21     }  
22     for(int i = indc; i >= 1; --i)  
23         cout << cifre16[cifre[i]];  
24 }
```

# BAZE DE NUMERATIE - RECURSIVE

```
11 void baza16(int n){  
12     char cifre16[]={'0', '1', '2', '3', '4', '5', '6', '7',  
13     | | | | | | | | '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};  
14     if(n == 0)  
15         return ;  
16     baza16(n / 16);  
17     cout << cifre16[n % 16];  
18 }
```

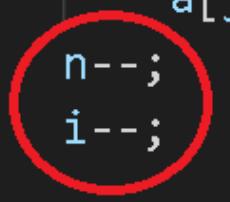
```
4 void baza2(int n){  
5     if(n == 0)  
6         return ;  
7     baza2(n / 2);  
8     cout << n % 2;  
9 }
```

# VECTORI (TABLOURI UNIDIMENSIONALE)

- CEA MAI SIMPLA STRUCTURA DE DATE DIN INFORMATICA O REPREZINTA VECTORII. ACESTIA NE AJUTA ATUNCI CAND VREM SA MEMORAM UN NUMAR NECUNOSCUT SAU FOARTE MARE DE VARIABILE.
- DATELE SE TIN ORDONAT DE LA STANGA LA DREAPTA AVAND INDICI INTRE 1 SI N SAU INTRE 0 SI N-1.
- PE VECTORI POT AVEA LOC MAI MULTE OPERATII: STERGERI, INSERARI, SORTARI, CAUTARI, INTERCLASARI
- EXISTA MAI MULTE TIPURI DE VECTORI: CLASICI, CARACTERISTICI, DE FRECVENTA

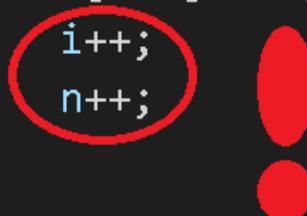
# STERGERE DE ELEMENT DIN VECTOR

```
5 void stergere(int &n, int a[], int poz){  
6     for(int i = poz + 1; i <= n; ++i)  
7         a[i-1] = a[i];  
8     n--;  
9 }  
10  
11 void stergere_pare(int &n, int a[]){  
12     for(int i = 1; i <= n; ++i)  
13         if(a[i] % 2 == 0){  
14             for(int j = i + 1; j <= n; ++j)  
15                 a[j - 1] = a[j];  
16             n--;  
17             i--;  
18 }  
19 }  
20
```



- STERGEA ELEMENTELOR DINTR-UN VECTOR SE FACE MUTAND FIECARE ELEMENT DIN VECTOR DE DUPA POZITIA PE CARE O STERGEM CATE O POZITIE IN SPATE

```
5 void inserare(int &n, int a[], int poz, int val){  
6     for(int i = n; i >= poz; --i)  
7         a[i + 1] = a[i];  
8     a[poz] = val;  
9     n++;  
10 }  
11  
12 void inserare_pare_jumatare(int &n, int a[]){  
13     for(int i = 1; i <= n; ++i)  
14         if(a[i] % 2 == 0){  
15             for(int j = n; j >= i; --j)  
16                 a[j+1] = a[j];  
17             a[i+1] = a[i] / 2;  
18             i++;  
19             n++;  
20         }  
21 }
```



## INSERARE DE ELEMENTE IN VECTOR

- INSERAREA DE NOI ELEMENTE IN VECTOR SE FACE MUTAND CU O POZITIE LA DREAPTA A TUTUROR ELEMENTELOR DE LA LOCUL UNDE DORIM SA FACEM INSERAREA

# SORTARILE VECTORILOR

- ATENTIE FOARTE MARE LA ACEST CAPITOL!
- IN SUBIECTELE DE ADMITERE ESTE PLIN DE METODE DE SORTARE, ELEVII FIND INTREBATI INCLUSIVE DESPRE NUMELE METODEI RESPECTIVE
- IN URMATOARELE SLIDE-URI VOM VEDEA TOATE METODELE DE SORTARE NECESARE PENTRU ADMITERE SCRISE ATAT RECURSIV CAT SI ITERATIVE
- ESTE IMPORTANT SA INTELEGETI CONCEPTELE IN ASA FEL INCAT SA PUTETI RECUNOASTE O ASEMENEA METODA ATUNCI CAND SE IMPUNE

# SELECTION SORT (ITERATIV)

```
28 void selection_sort_long(int a[], int n){ // sortare descrescatoare - O(n^2)
29     for(int i = 1; i < n; ++i){
30         int imax = i;
31         for(int j = i + 1; j <= n; ++j)
32             if(a[j] > a[imax])
33                 imax = j;
34         swap(a[i], a[imax]);
35     }
36 }
```

```
47 void selection_sort_short(int a[], int n){ // sortare crescatoare - O(n^2)
48     for(int i = 1; i < n; ++i)
49         for(int j = i + 1; j <= n; ++j)
50             if(a[j] < a[i])
51                 swap(a[j], a[i]);
52 }
```

# SELECTION SORT (RECURSIV)

```
38 void selection_sort_rec(int a[], int n){ // sortare crescatoare recursiva- O(n^2)
39     if(n == 1)
40         return ;
41     for(int i = n - 1; i >= 1; --i)
42         if(a[i] > a[n])
43             swap(a[i], a[n]);
44     selection_sort_rec(a, n-1);
45 }
```

# INSERTION SORT (ITERATIV)

```
54 void insertion_sort_long(int a[], int n){ // sortare crescatoare - O(n^2)
55     for(int i = 1; i <= n; ++i){
56         int ind = i;
57         for(int j = i - 1; j >= 1; --j)
58             if(a[j] > a[i])
59                 ind = j;
60         int cpy = a[i];
61         for(int j = i - 1; j >= ind; --j)
62             a[j + 1] = a[j];
63         a[ind] = cpy;
64     }
65 }
```

# INSERTION SORT (ITERATIV + RECURSIV)

```
67 void insertion_sort_short(int a[], int n){ // sortare descrescatoare - O(n ^ 2)
68     for(int i = 1; i <= n; ++i){
69         int j = i;
70         while(a[j] > a[j-1] && j >= 2)
71             swap(a[j], a[j-1]), j--;
72     }
73 }
74
75 void insertion_sort_rec(int a[], int n, int i){ // sortare crescatoare recursiva - O(n ^ 2)
76     if(i > n)
77         return ;
78     int j = i;
79     while(a[j] < a[j-1] && j >= 1)
80         swap(a[j], a[j-1]), j--;
81     insertion_sort_rec(a, n, i+1);
82 }
```

# BUBBLE SORT (ITERATIV)

```
5 void bubble_sort(int a[], int n){ // sortare crescatoare - O(n^2)
6     bool sortat = false;
7     while(!sortat){
8         sortat = true;
9         for(int i = 1; i < n; ++i)
10            if(a[i] > a[i+1]){
11                swap(a[i], a[i+1]);
12                sortat = false;
13            }
14    }
15 }
```

# BUBBLE SORT (RECURSIV)

```
17 void bubble_sort_rec(int a[], int n){ // sortare crescatoare recursiva - O(n^2)
18     bool sortat = true;
19     for(int i = 1; i < n; ++i)
20         if(a[i] > a[i+1]){
21             swap(a[i], a[i+1]);
22             sortat = false;
23         }
24     if(!sortat)
25         bubble_sort_rec(a, n);
26 }
```

# INTERCLASAREA

- PENTRU 2 VECTORI SORTATI, PUTEM FOLOSI UN ALGORITM LINIAR CARE SA IL DETERMINE PE AL 3-LEA SORTAT CARE CONTINE ELEMENTELE DIN CELI 2 VECTORI INITIALI. ACEST ALGORITM ESTE INTERCLASAREA.
- PRINCIPIOUL ESTE RELATIV SIMPLU. URMARIM IN AMBII VECTORI CATE UN ELEMENT SI LA FIECARE PAS COMPARAM CELE 2 ELEMENTE PENTRU A VEDEA CARE SE POTRIVESTE CEL MAI BINE IN CEL DE AL 3-LEA VECTOR SI IL COPIEM ACOLO.
- INTERCLASAREA SE IMPLEMENTEAZA IN GENERAL PE SIRURI ORDONATE CRESCATOR, DAR PROCESUL E SIMILAR SI PENTRU VECTORII DESCRESATORI

# INTERCLASAREA (VARIANTA CLASICA)

```
5 void interclasare(int n, int a[], int m, int b[], int& indc, int c[]){
6     indc = 0;
7     int inda = 1, indb = 1;
8     while(inda <= n && indb <= m)
9         if(a[inda] <= b[indb])
10            c[++indc] = a[inda++];
11        else c[++indc] = b[indb++];
12     while(inda <= n)
13         c[++indc] = a[inda++];
14     while(indb <= m)
15         c[++indc] = b[indb++];
16 }
```

```
6 void mergeSort(int st, int dr){  
7     if(st == dr)  
8         return ;  
9     else{  
10         int mij = (st + dr) / 2;  
11         mergeSort(st, mij);  
12         mergeSort(mij + 1, dr);  
13         int inda = st, indb = mij + 1, indc = 0;  
14         while(inda <= mij && indb <= dr)  
15             if(a[inda] <= a[indb])  
16                 c[++indc] = a[inda++];  
17             else c[++indc] = a[indb++];  
18         while(inda <= mij)  
19             c[++indc] = a[inda++];  
20         while(indb <= dr)  
21             c[++indc] = a[indb++];  
22         for(int i = 1; i <= indc; ++i)  
23             a[st + i - 1] = c[i];  
24     }  
25 }
```

# MERGE SORT (DIVIDE ET IMPERA)

- MERGE SORT ESTE O METODA DE SORTARE CARE SE BAZEaza PE DIVIDE ET IMPERA SI INTERCLASARE SI REUSESTE PERFORMANTA DE A SORTA UN SIR CRESCATOR IN  $O(N \cdot \log_2 N)$  – O COMPLEXITATE EXTREM DE BUNA PENTRU SORTARE

# CAUTAREA BINARA

- CAUTAREA BINARA ESTE UN PROCEDEU DE CAUTARE DE VALORI PE UN SIR ORDONAT CRESCATOR CARE REUSESTE PERFORMANTA DE A RASPUNDE LA ORICE QUERY IN COMPLEXITATE  $O(\log_2 n)$
- PRINCIPIUL ESTE DESTUL DE SIMPLU, VECTORUL FIIND ORDONAT LA FIECARE PAS ISI INJUMATATESTE INTERVALUL DE CAUTARE, COMPARAND ELEMENTUL DIN MIJLOC A FIECARUI INTERVAL CU VALOAREA PE CARE O CAUTA.
- IN ACEST MOD, ALGORITMUL REUSESTE SA FIE CEL MAI RAPID ALGORITM DE CAUTARE POSIBIL SI SA ISI PRIMEASCA SI NOTORIETATEA PE CARE O ARE IN INFORMATICA.

# CAUTARE BINARA CLASICA

```
5  bool CB(int n, int a[], int val){  
6      int st = 1, dr = n;  
7      while(st <= dr){  
8          int mij = (st + dr) / 2;  
9          if(a[mij] == val)  
10             return true;  
11          else if(a[mij] <= val)  
12              st = mij + 1;  
13          else dr = mij - 1;  
14      }  
15      return false;  
16 }
```

# CAUTARE BINARA RECURSIVA (DIVIDE ET IMPERA)

```
6  bool cb(int a[], int st, int dr, int val){  
7      if(st == dr)  
8          |    return a[st] == val;  
9      int mij = (st + dr) / 2;  
10     if(a[mij] == val)  
11         |    return true;  
12     if(a[mij] < val)  
13         |    return cb(a, mij + 1, dr, val);  
14     return cb(a, st, mij - 1, val);  
15 }
```

# VECTORI CARACTERISTICI

- VECTORII CARACTERISTICI SUNT FOLOSITI PENTRU A MARCA DACA ANUMITE VALORI AU APARUT SAU NU. INTERESANT ESTE MODUL LOR DE A SE FOLOSI, VALORILE FIIND STOCATE CA INDEXI SI STAREA LOR FIIND REPREZENTATA DE VALOAREA PE CARE O AU.
- ASTFEL, NU PUTEM AVEA VALORI MARI STOCATE, MAXIM 1-2 MILIOANE.
- DE EXEMPLU, PENTRU A STOCA DATE DESPRE O VALOAREA, VOM PROCEDA IN FELUL URMATOR:

`INT X = 10;`

`F[X] = 1;`

# SECVENTE PE VECTORI

- O SECVENTA DE ELEMENTE SEMNIFICA O INSIRUIRE DE MAI MULTE ELEMENTE. ASTFEL, DACA AVEM UN SIR, SECVENTA VA SEMNIFICA ORICE SET DE VALORI CONSECUTIVE IN SIR.
- NU VOM SCRIE ALGORITMI NICI DESPRE SECVENTE PENTRU NU ESTE NIMIC IESIT DIN COMUN SI NICI NU SUNT INTALNITI IN SUBIECTELE DE BAC, DAR E IMPORTANT DE SPECIFICAT CA ACEST CAPITOL EXISTA SI SE OCUPA DE ANUMITE CALCULARE PE VECTORI
- IN MARE, DACA STAPANITI DESPR EVECTOR TOT CE A FOST PANA AICI, CU SIGURANTA EXAMENUL DE ADMITERE VA MERGE MULT, MULT MAI BINE

# MATRICI (TABLOURI BIDIMENSIONALE)

- MATRICILE SUNT TABLOURI BIDIMENSIONALE CARE IN FAPT SUNT UN FEL DE VECTORI DE VECTORI. IN C++ NU PUTEM TINE MINTE UN VECTOR CA FIIND UN ELEMENT, MOTIV PENTRU CARE AVEM NEVOIE DE MATRICI SI NU PUTEM FACE VECTOR[VECTOR].
- EXISTA 2 TIPURI DE MATRICI: PATRATICE SI OARECARE.
- LA ACEST CAPITOL, CEL MAI MULT VOM DISCUTA DESPRE MATRICILE PATRATICE, ELE FIIND CELE CARE AU CATEVA PROPRIETATI.

# MATRICILE PATRATICE ( $N * N$ ELEMENTE)

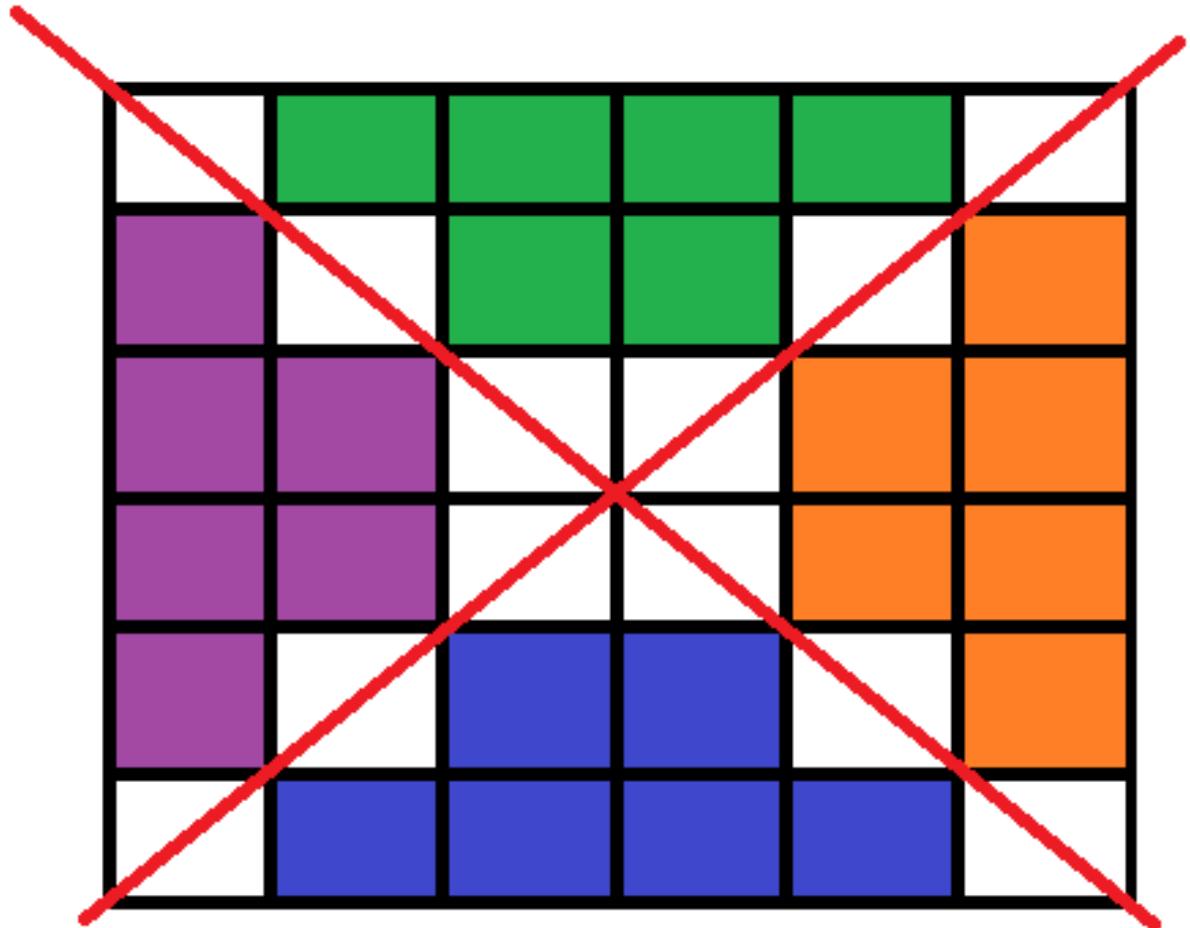
- O MATRICE ESTE PATRATICA DACA ACEASTA CONTINE EXACT LA FEL DE MULTE LINII CA SI COLOANE.
- SPRE DEOSEBIRE DE MATRICILE CLASICE, MATRICILE PATRATICE AU MAI MULTE PROPRIETATI:
  - AU O DIAGONALA PRINCIPALA SI UNA SECUNDARA
  - AU 4 ZONE DELIMITATE DE CELE 2 DIAGONAL
  - AU ELEMENTE OGLINDITE FATA DE DIAGONAL

# DIAGONALELE

- EXISTA 2 TIPURI DE IMPLEMENTARI ALE MATRICILOR: DE LA 0 SAU DE LA 1
- DIN MOMENT CE UBB UTILIZEAZA INDEXAREA DE LA 1, ASA VOM FACE SI NOI
- DIAGONALA PRINCIPALA:  $A[i][i]$  – INDICE LINIE = INDICE COLOANAL
- DIAGONA SECUNDARA:  $A[i][N - i + 1]$  – INDICE LINIE =  $N - \text{INDICE LINIE} + 1$
- CONDITIA DE VERIFICARE A UNUI ELEMENT DACA E PE DIAGON SECUNDARA:
  - $A[i][j]$  – PE DIAGON SECUNDARA  $\Leftrightarrow i = N - j + 1$

# **ZONELE UNUI MATRICI PATRATICE**

- ZONA VERDE (1) – ZONA DE DEASUPRA DIAGON PRINCIP, DEASUPRA DIAGON SEC
  - ZONA PORTOCALIE (2) – ZONA DE DEASUPRA DIAGON PRINCIP, SUB DIAGON SEC
  - ZONA ALBASTRA (3) – ZONA DE SUB DIAGON PRINCIP, SUB DIAGON SEC
  - ZONA MOV (4) – ZONA DE SUB DIAGON PRINCIP, DEASUPRA DIAGON SEC



# IDENTIFICAREA ZONELOR

- ÎNAINTE DE A IDENTIFICA ZONELE, ESTE NECESSAR SA IDENTIFICAM POZITIONAREA FATA DE DIAGON PRINCIP / DIAGON SEC
- $A[I][J]$ :
  - DEASUPRA DIAGON PRINCIP DACA:  $I < J$
  - SUB DIAGON PRINCIP DACA:  $I > J$
  - DEASUPRA DIAGON SEC DACA:  $I < N - J + 1$
  - SUB DIAGON SEC DACA:  $I > N - J + 1$

# IDENTIFICAREA ZONELOR

```
5  int zona(int i, int j, int n){  
6      if(i < j && i < n - j + 1)  
7          return 1;  
8      if(i < j && i > n - j + 1)  
9          return 2;  
10     if(i > j && i > n - j + 1)  
11         return 3;  
12     if(i > j && i < n - j + 1)  
13         return 4;  
14 }
```

VA MULTUMESC PENTRU ATENTIE!