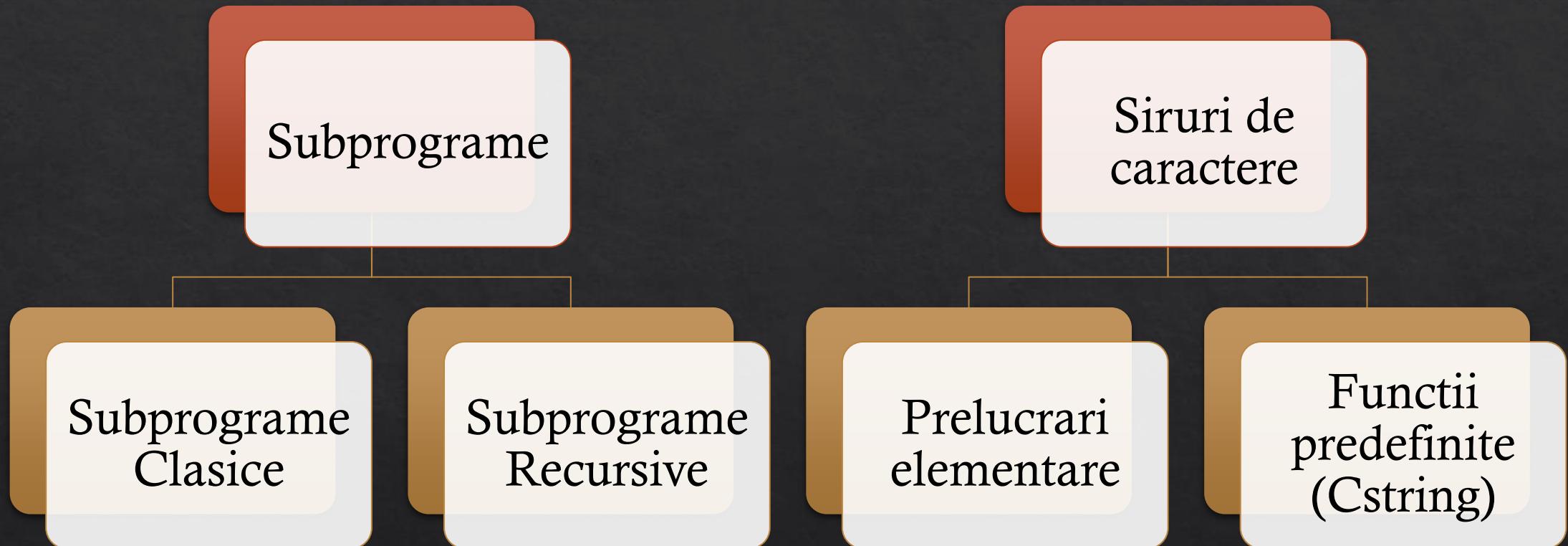


Recapitulare clasa a X-a

Curs Pregatire Admitere UBB 2022

Materia de clasa a X-a



Materie de clasa a X-a

Structuri
neomogene

Struct

Structuri Liniare

Stiva

Coadă

Matematica

Combinatorica

Geometrie

Observatii utile

- ❖ In ciuda faptului ca sirurile de caractere constituie un capitol extrem de important pentru examenul de Bacalaureat, imposibilitatea utilizarii functiilor predefinite pe CSTRING in C++ sau STRING.H din C din motive evidente (PSEUDOCOD), acest capitol nu este intalnit deloc in randul grilelor.
- ❖ Pentru a compensa acest gol, in randul grilelor vom vedea exercitii de matematica (Combinari, Aranjamente sau Permutari), dar si de geometrie.
- ❖ De asemenea, o buna cunoastere a problemelor cu stive si cozi reprezinta un ATUU.

Subprograme #1

- ❖ Scopul Subprogramelor este de a structura intr-un mod mai estetic codurile.
- ❖ Acestea sunt folosite nu doar pentru estetica, ci si pentru usurarea muncii informaticienilor, codul fiind mult mai usor de urmarit.
- ❖ Conventiile general valabile impun ca functiile sa faca un singur lucru. (Nu vom face mai multe lucruri in aceiasi Functie)
- ❖ Subprogramele sunt in general impartite in 2 mari categorii, cele care returneaza si cele care nu. Subprogramele care nu returneaza valori sunt in general folosite pentru a separa anumite portiuni de cod de altele pentru a putea usura intelegerea sursei.
- ❖ De asemenea, daca dorim modificarea unor valori, le putem transmite prin referinta.

Functiile care returneaza

- ◊ Signatura functiei este prima linie din declararea functiei de mai jos.
Aceasta contine tipul de date a valorii pe care o returnam, numele functiei si variabilele pe care aceasta le primeste si urmeaza sa fie prelucrate

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

```
5 tip_de_data nume_functie(tip_de_data var1, ...){  
6     {corful_functiei}  
7     return valoare_finala;  
8 }
```

Functiile care returneaza

- ❖ Este important de precizat faptul ca in mod normal atunci cand trimitem un parametru unei functii, acesta nu se va modifica. De exemplu, alaturat, variabila n declarata in corpul functiei main nu va suferi modificari chiar daca n-ul utilizat in functia de suma cifrelor va deveni 0

```
5  int suma_cifrelor(int n){  
6      int sum = 0;  
7      while(n)  
8          sum += n % 10, n /= 10;  
9      return sum;  
10 }  
11  
12 int main(){  
13     int n;  
14     cin >> n;  
15     cout << suma_cifrelor(n);  
16     return 0;  
17 }
```

Transmiterea datelor prin referinta

- ◊ Transmiterea datelor din functie prin referinta (&) va duce la modificarea permanenta si a variabilelor mama de la locul unde a fost apelata functia.
- ◊ Vom explica acest lucru la capitolul de Aritmetica a Pointerilor.

```
5 int suma_cifrelor(int &n){  
6     int sum = 0;  
7     while(n)  
8         sum += n % 10, n /= 10;  
9     return sum;  
10 }  
11  
12 int main(){  
13     int n;  
14     cin >> n;  
15     cout << suma_cifrelor(n);  
16     return 0;  
17 }
```

Functiile fara return

- ❖ În cazul în care dorim pur și simplu să luăm o porțiune de cod și să o punem într-o funcție cu scopul de a ordona codul, nu vom vrea mereu să returneze ceva, doar să producă modificări asupra variabilelor. Acest lucru se poate face folosind funcții care nu returnează nimic (VOID).
- ❖ Pentru a declara o astfel de funcție, folosim tipul de date void și astfel, anunțăm compilatorul că nu vom returna nimic.
- ❖ De cele mai multe ori dorim să producем modificări asupra variabilelor noastre de intrare, motiv pentru care acestea vor fi transmise prin referință

Functiile fara return

- ❖ Alaturat avem doua functii, una care citeste un vector si cealalta ii afiseaza elementele. Intrucat variabila n se citeste, vrem sa-l modificam valoarea si va fi transmisa prin parametru.
- ❖ ATENTIE! Vectorii si matricile nu se transmit prin parametru intrucat valorile lor se modifica definitive in cadrul functiilor.

```
5 void citire(int a[], int &n){  
6     cin >> n;  
7     for(int i = 1; i <= n; ++i)  
8         cin >> a[i];  
9 }  
10  
11 void afisare(int a[], int n){  
12     for(int i = 1; i <= n; ++i)  
13         cout << a[i] << ' ';  
14 }
```

Functii fara return

- ❖ - Este important ca daca vrem sa apelam functii din alte functii, cele pe care le apelam sa fie situate deasupra de cea in care suntem.
- ❖ - Daca in momentul proiectarii nu se stie exact continutul tuturor functiilor, exista o modalitate utila de a declara aceste functii in asa fel incat sa nu fie o probleme cu zona in care a fost declarata.

```
1 #include <fstream>
2 #include <cstring>
3 using namespace std;
4 |
5 void afisare(int a[], int n);
6 void citire(int a[], int &n);
7
8 int main(){
9     int a[1001], n;
10    citire(a, n);
11    return 0;
12 }
13
14 void citire(int a[], int &n){
15     cin >> n;
16     for(int i = 1; i <= n; ++i)
17         cin >> a[i];
18     afisare(a, n);
19 }
20
21 void afisare(int a[], int n){
22     for(int i = 1; i <= n; ++i)
23         cout << a[i] << ' ';
24 }
```

Functii recursive

- ❖ O functie care se auto-apeleaza se numeste o functie recursiva. O functie recursiva are mereu o conditie de oprire si un auto-apel in ea.
- ❖ In general, structurile repetitive pot fi foarte usor inlocuite cu o functie recursive.
- ❖ De asemenea, atunci cand stim o formula de recursie (Ex: $a[n] = a[n-1] * 2$) – putem aplica foarte usor o astfel de functie.
- ❖ De regula, aceste functii sunt folosite pentru a usura eforturile de programare ale informaticianului.
- ❖ Exista 2 categorii de functii recursive:
 - ❖ Divide Et Impera
 - ❖ Clasice

Transformarea in Functie Recursiva

- ❖ Alaturat vedem o functie iterative care calculeaza suma cifrelor unui numar, iar mai jos se poate vedea transformarea functiei in una recursiva.
- ❖ Se poate observa faptul ca in cea de-a 2 functie nu avem o structura repetitiva, acest lucru fiind simulat de recursie.
- ❖ De asemenea, in cadrul functiilor recursive observam prezenta conditiei de oprire.

```
5  int suma_cifrelor(int n){  
6      int sum = 0;  
7      while(n){  
8          sum += n % 10;  
9          n /= 10;  
10     }  
11     return sum;  
12 }  
13  
14 int suma_cifrelor(int n){  
15     if(n == 0)  
16         return 0;  
17     return n % 10 + suma_cifrelor(n / 10);  
18 }
```

Exemple de recursivitati

```
6 int factorial(int n){  
7     if(n <= 1)  
8         return 1;  
9     else return n * factorial(n - 1);  
10}
```

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

```
6 int fibonacci(int n){  
7     if(n == 1 || n == 2)  
8         return 1;  
9     return fibonacci(n-1) + fibonacci(n-2);  
10}
```

```
6 int sum_div(int n, int d){  
7     if(n % d == 0 && d * d < n)  
8         return n / d + d + sum_div(n, d+1);  
9     if(d * d == n)  
10        return d;  
11    if(d < n)  
12        return sum_div(n, d + 1);  
13    return 0;  
14 }
```

Exemple de recursivitati

```
6 int sum_vector(int a[], int st, int dr){  
7     if(st > dr)  
8         return 0;  
9     return a[st] + sum_vector(a, st+1, dr);  
10 }
```

```
6 int sum_cif(int n){  
7     if(n == 0)  
8         return 0;  
9     return n % 10 + sum_cif(n / 10);  
10 }
```

```
5 int sum_vec(int a[], int n){  
6     if(n == 0)  
7         return 0;  
8     return a[n] + sum_vec(a, n-1);  
9 }
```

Probleme cu recursivitati

- ◊ Ce face functia alaturata?
- ◊ Rezultatul ridicarii la putere a lui a^b
- ◊ Care e complexitatea?
- ◊ $\log_2 b$
- ◊ De ce in practica functia alaturata nu ar adduce niciun plus valoare (complexitate timp) programului nostru?
- ◊ Pentru ca o ridicare la putere pe care ar putea face diferența complexitatea ieșe din memorie
- ◊ Cum am putea face aceasta functie sa fie relevanta?
- ◊ Calcula MODULUL!

```
5 int a_la_b(int a, int b){  
6     if(b == 0)  
7         return 1;  
8     int pow = a_la_b(a, b/2);  
9     if(b % 2 == 0)  
10        return pow * pow;  
11    return pow * pow * a;  
12 }  
13  
14 int main(){  
15     cout << a_la_b(3, 5);  
16     return 0;  
17 }
```

Cerință

Se dau **2** numere naturale reprezentând scorul în timpul actual. Să se determine în câte moduri se poate ajunge de la **0-0** la acel scor.

Probleme cu recursivitate

- ❖ La aceasta problema am putea implementa o solutie simpla care calculeaza numarul de moduri de a ajunge la scorul $I - J$ ca fiind numarul de moduri de a ajunge la scorul $I-1 - J$ plus numarul de moduri de a ajunge la scorul $I - J-1$.
- ❖ Recursia se opreste cand I sau J sunt 0 -> $\text{nr_mod} = 1$

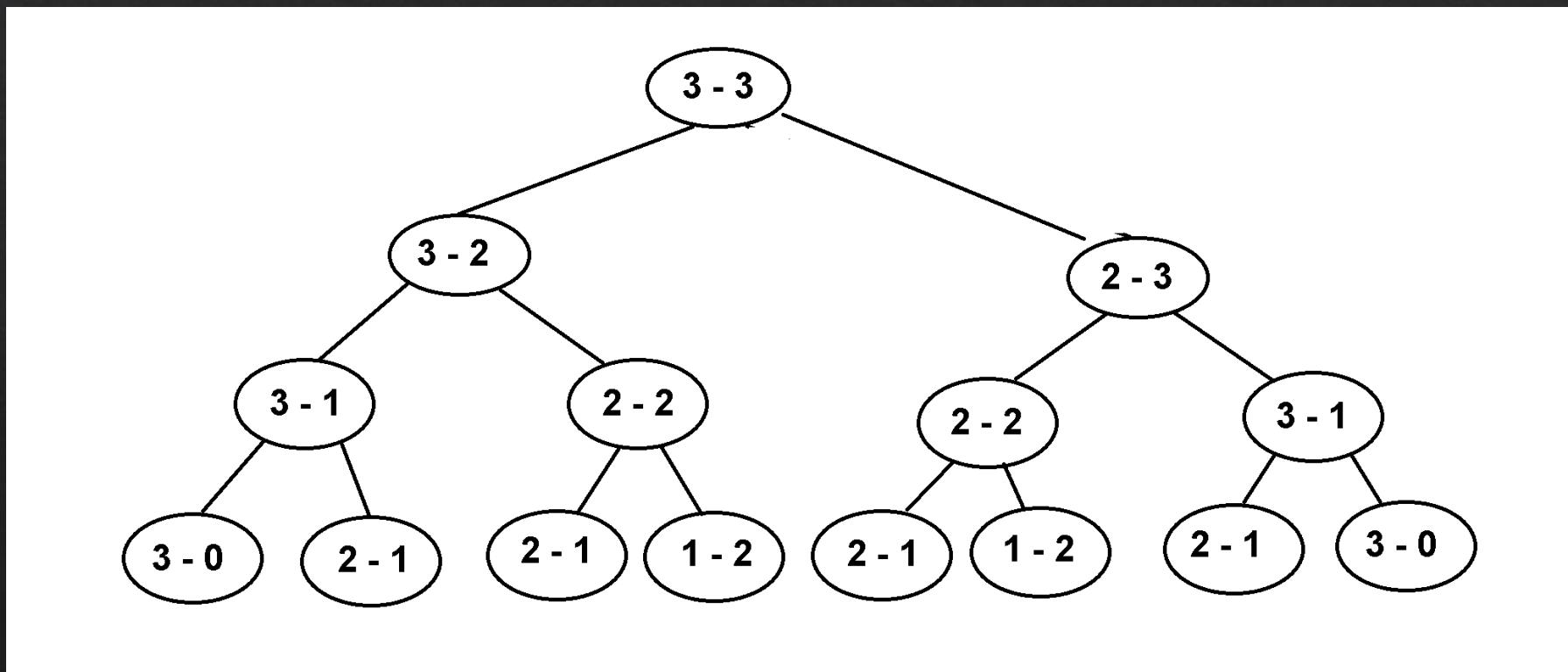
Probleme cu recursivitate

- ◊ La aceasta problema, lucrul care ne pune in dificultate este faptul ca functia recursiva va face foarte multi pasi. Am simulat numarul de pasi pentru 10-10 si rezultatul este infiorator! 369511
- ◊ Aces numar foarte mare este dat de faptul ca programul nostru va repeta apele in adancime identice.
- ◊ Exista un procedeu pentru a Evita acest lucru numit procedeul de MEMOIZARE.

```
1 #include <iostream>
2 using namespace std;
3
4 /*
5 (3 - 3) - (3 - 2) + 1 gol, (2 - 3) + 1 gol
6 */
7
8 int scor(int x, int y){
9     if(x == 0 || y == 0)
10        return 1;
11    return scor(x-1, y) + scor(x, y-1);
12 }
13
14 int main(){
15     int x, y;
16     cin >> x >> y;
17     cout << scor(x, y);
18     return 0;
19 }
```

Probleme cu recursivitati

- ❖ Uitati arborele de apeluri pentru scorul 3-3:



Probleme cu recursivitati

- ◆ Observam ca repetam multeapeluri asa ca pentru a evita acest lucru si a nu lua limite de timp si de memorie cand calculam numarul de posibilitati, vom memora toateapeluri deja calculate, astfel cand vrem a 2-a oara sa apelam pentru un scor deja calculat sa nu fim nevoiti sa facem acest lucru.
- ◆ Acest procedeu de memorare aapelurilor se numeste memoizare, iar pentru a le memora vom tine minte o matrice in care $\text{scor}[i][j]$ – reprezinta numarul de moduri de a ajunge la scorul $i-j$.
- ◆ Numarul de auto-apeluri pentru 10-10 acum este doar 120, in comparatie cu 360000 cate erau fara Memoizare.

```
1 #include <iostream>
2 using namespace std;
3
4 long long scor[1001][1001];
5
6 int scor_rec(int x, int y){
7     if(x == 0 || y == 0)
8         return 1;
9     if(scor[x-1][y] == 0)
10        scor[x-1][y] = scor_rec(x-1, y);
11     if(scor[x][y-1] == 0)
12        scor[x][y-1] = scor_rec(x, y-1);
13     return scor[x-1][y] + scor[x][y-1];
14 }
15
16 int main(){
17     int x, y;
18     cin >> x >> y;
19     cout << scor_rec(x, y);
20     return 0;
21 }
```

Probleme rezolvate

- ◆ O alta metoda de a rezolva aceasta problema este utilizant Programare Dinamica. Aceasta metoda este putin, putin mai buna decat cea prin memoizare, dar diferențele sunt insesizabile, pentru 10-10 vorbim de o diferență de maximum 20 de pasi.

```
1 #include <iostream>
2 #include <queue>
3 using namespace std;
4
5 long long m[1001][1001];
6
7 long long fotbal(int a, int b){
8     if(a == 0 || b == 0)
9         return 1;
10    for(int i = 0; i <= b; ++i)
11        m[0][i] = 1;
12    for(int i = 0; i <= a; ++i)
13        m[i][0] = 1;
14    for(int i = 1; i <= a; ++i)
15        for(int j = 1; j <= b; ++j)
16            m[i][j] = m[i-1][j] + m[i][j-1];
17    return m[a][b];
18 }
19
20 int main()
21 {
22     int a, b;
23     cin >> a >> b;
24     cout << fotbal(a, b);
25     return 0;
26 }
```

Divide Et Impera

- ❖ Divide et Impera este un concept filozofic antic care are in vedere un principiu destul de simplu: divide si stapaneste.
- ❖ In informatica, acest lucru va fi folosit sub o forma usor diferita. Acest algoritm se foloseste cel mai mult pentru a determina proprietati ale vectorilor si ale matricilor.
- ❖ Ideea este destul de simpla, pentru a stii proprietati ale vectorilor, vom cauta secential pe acesta si ii vom determina starea.
 - ❖ Spre exemplu: Daca ne intereseaza sa stim daca un vector are toate elementele pare, este sufficient pentru a verifica daca ambele jumatati ale acestuia au toate elementele pare. Divizarea sirului continua pana secenta are un singur element

Exemple de Divide Et Impera I

```
6  int sum_vector(int a[], int st, int dr){  
7      if(st == dr)  
8          return a[st];  
9      int mij = (st + dr) / 2;  
10     return sum_vector(a, st, mij) + sum_vector(a, mij + 1, dr);  
11 }
```

```
6  bool toate_pare(int a[], int st, int dr){  
7      if(st == dr)  
8          return a[st] % 2 == 0;  
9      int mij = (st + dr) / 2;  
10     return toate_pare(a, st, mij) && toate_pare(a, mij + 1, dr);  
11 }
```

Cautare binara cu 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 }
```

Sortare cu DIVIDE ET IMPERA

- ❖ Înainte de a vedea metoda de sortare propriu-zisa este important de precizat ca Divide et Impera sta la baza sortarilor în timp $n * \log_2 n$ – cele mai rapide sortari.
- ❖ În aceasta lectie vom vorbi despre metoda MERGE sort
- ❖ Ideea de la baza acestei metode este sortarea celor 2 jumătăți de secvențe și interclasarea lor. Astfel, având în vedere că lungimile secvențelor se înjumatătesc constant, numărul de pași în adâncime pe care îi va face recursivitatea este maxim $\log_2 n$, iar toate interclasările vor duce la complexitatea finală n (dimensiunea sirului întreg)

```
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 }
```

Probleme cu Divide Et Impera

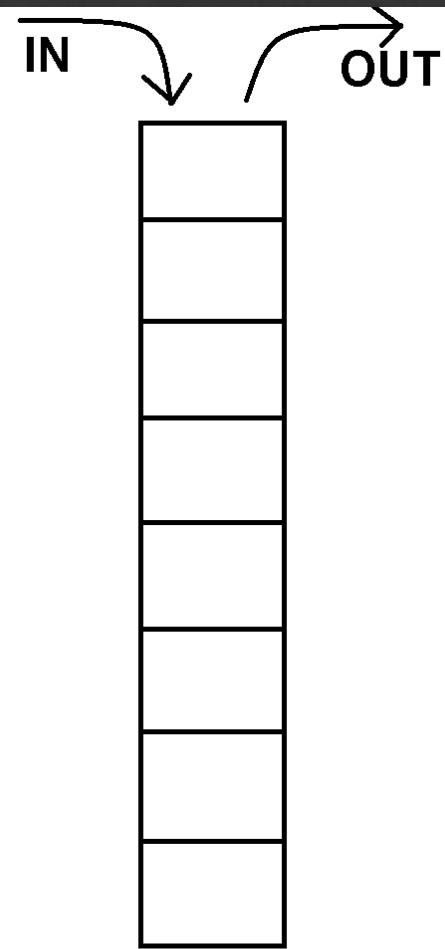
Cerință

Se dă un vector cu n elemente numere naturale. Folosind metoda **Divide et Impera** să se verifice dacă are elementele ordonate crescător.

```
4  bool verif_ord(int a[], int st, int dr){  
5      if(st == dr)  
6          return true;  
7      int mij = (st + dr) / 2;  
8      return verif_ord(a, st, mij) && verif_ord(a, mij + 1, dr) && a[mij] <= a[mij + 1];  
9  }
```

Stiva

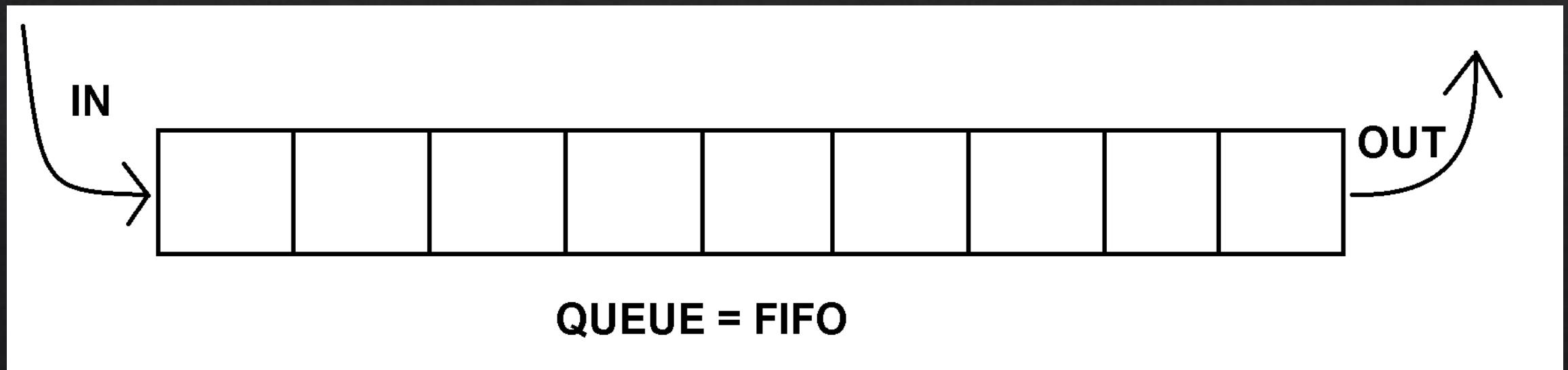
- ❖ Stiva este o structura de date care respecta regula ca primul element introdus sa fie ultimul extras.
- ❖ FILO = First In Last Out
- ❖ Mereu ultimul element introdus este primul scos.



STACK = FILO

Coada

- ❖ Coada este o structura de date care respecta regula ca primul element introdus sa fie primul extras.
- ❖ FIFO = First In First Out
- ❖ Mereu primul element introdus este primul extras
- ❖ Functioneaza la o coada reala: Primul venit -> Primul servit



Functiile unei stive

- ◆ Pe o stiva există o serie de operații pe care le putem face: Push, Pop, Top și Empty

```
5  int S[1001];
6  int st = 1, dr = 0;
7
8  bool empty(){
9      return st > dr;
10 }
11
12 void pop(){
13     if (!empty())
14         dr--;
15 }
16
17 int top(){
18     if (!empty())
19         return S[dr];
20 }
21
22 void push(int val){
23     S[++dr] = val;
24 }
```

Functiile unei cozi

- ❖ Pe o coada există o serie de operații pe care le putem face: Push, Pop, Front și Empty

```
5  int Q[5001], st = 1, dr = 0;
6
7  bool empty(){
8  |   return st > dr;
9  }
10
11 void push(int val){
12 |   Q[++dr] = val;
13 }
14
15 void front(){
16 |   if (!empty())
17 |   |   cout << Q[st] << "\n";
18 }
19
20 void pop(){
21 |   if (!empty())
22 |   |   st++;
23 }
```

Problema Coada

Cerință

Să se scrie un program care gestionează o coadă de numere întregi. Inițial coada este vidă. Programul va citi de la tastatură o listă de operații, care pot fi:

- `push X` – adaugă valoarea întreagă `X` în coadă;
- `pop` – elimină elementul din coadă;
- `front` – afișează elementul de la începutul cozii.

Programul va realiza asupra cozii operațiile citite, în ordine. Afisările se fac pe ecran, câte o valoare pe linie.

```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 int Q[5001], st = 1, dr = 0;
6
7 bool empty(){
8     return st > dr;
9 }
10
11 void push(int val){
12     Q[++dr] = val;
13 }
14
15 void front(){
16     if (!empty())
17         cout << Q[st] << "\n";
18 }
19
20 void pop(){
21     if (!empty())
22         st++;
23 }
```

Rezolvare:

```
26 int main()
27 {
28     int n;
29     cin >> n;
30     char s[5];
31     for (int i = 1; i <= n; ++i)
32     {
33         cin >> s;
34         if (strcmp("push", s) == 0)
35         {
36             int x;
37             cin >> x;
38             push(x);
39         }
40         else if (strcmp("pop", s) == 0)
41             pop();
42         else
43             front();
44     }
45     return 0;
46 }
```

Problema Stiva

Cerință

Să se scrie un program care gestionează o stivă de numere întregi. Inițial stiva este vidă. Programul va citi de la tastatură o listă de operații, care pot fi:

- `push X` – adaugă valoarea întreagă `X` pe stivă;
- `pop` – elimină elementul din vârful stivei;
- `top` – afișează elementul din vârful stivei.

Programul va realiza asupra stivei operațiile citite, în ordine. Afisările se fac pe ecran, câte o valoare pe linie.

```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 int S[1001];
6 int st = 1, dr = 0;
7
8 bool empty(){
9     return st > dr;
10 }
11
12 void pop(){
13     if (!empty())
14         dr--;
15 }
16
17 int top(){
18     if (!empty())
19         return S[dr];
20 }
21
22 void push(int val){
23     S[++dr] = val;
24 }
```

Rezolvare:

```
26 int main(){
27     int n;
28     cin >> n;
29     char s[5];
30     for (int i = 1, x; i <= n; ++i){
31         cin >> s;
32         if (strcmp(s, "push") == 0){
33             cin >> x;
34             push(x);
35         }
36         else if (strcmp(s, "pop") == 0)
37             pop();
38         else
39             cout << top() << "\n";
40     }
41     return 0;
42 }
```

- ❖ Aceasta prezentare a fost conceputa de Somesan Paul-Ioan
- ❖ Este proprietatea Academiei-Zecelainfo
- ❖ VA RUGAM NU DISTRIBUITI ACESTE MATERIALE!

VA MULTUMESC!