Funcții utile din librăria **algorithm**

Sau cum ne ușurăm viața

Ce este algorithm

- Librărie cu funcții ce operează cu vectori(și nu numai)
- Pentru utilizare trebuie inclusă librăria algorithm
- Exemplu: #include <algorithm>
- În exemplele ce urmează voi folosi variabilele de mai jos:
- int a[100], b[100],c[100], n, x, y, na, nb;

max_element/min_element

- Ce face: determină valoarea maximă dintr-un vector
- Exemplu: max_element(a+1,a+n+1) returnează poziția(adresa de memorie) pe care se afla elementul de valoare maximă din vectorul **a** de la poziția **1** la **n**.
- Exemplul 1: cout<<*max_element(a+1,a+n+1) afișează valoarea maximă din vectorul **a** de la poziția **1** la poziția **n.**
- Exemplul 2: i=max_element(a+1, a+n+1)-a, în **i** vom avea poziția maximului din vector(prima apariție)

find

- Ce face: determină prima poziție din vector unde se găsește un anumit element

- Exemplu: find(a+1,a+n+1, element) returnează poziția(adresa de memorie) pe care se afla element în vectorul **a** de la poziția **1** la **n**, dacă nu există în vector atunci returnează a+n+1.

search

- Ce face: caută un vector în alt vector și returnează prima poziție pe care l-a găsit.
- Exemplu: search(a+1, a+na+1,b+1,b+nb+1) returnează prima poziție(adresă) unde apare vectorul b în vectorul a. Dacă vectorul b nu este găsit în a, atunci se returnează a+na+1.

is_permutation

- Ce face: determină dacă un vector este o permutare a altui vector(dacă cei doi vectori conțin aceleași elemente)
- Exemplu:

  if(is_permutation(a+1, a+na+1, b+1))cout<<”E permutare”;
- Vectorul b trebuie să conțină același număr de elemente ca și a, altfel rezultatul e incert.
- Complexitate $O(n^2)$

is_sorted

- Ce face: verifică dacă secvența este ordonată crescător

- Exemplu:

```
if(is_sorted(a+1, a+na+1))cout<<”E ordonat”;
```

fill

- Ce face: pune o valoare în celulele unui vector
- Exemplu: fill(a+1, a+n+1, valoare) pune valoare în vectorul a de la poziția 1 până la poziția n.

copy

- Ce face: copiază un vector în alt vector
- Exemplu: copy(a+1, a+n+1, b+1) copiază elementele din vectorul a, de la poziția 1 până la poziția n, în vectorul b(de la poziția 1).
- Observații: zona de copiat NU trebuie să se suprapună cu zona în care se face copierea.

swap

- Ce face: interschimbă 2 valori între ele.

- Exemplu:

```
int a[100], b[100], x,y;
swap(x,y); //schimbă valoarea lui x cu a lui y
swap(a,b); // schimbă valorile celor 2 vectori între ei
```

swap_ranges

- Ce face: interschimbă 2 zone din vector/i.
- Exemplu: swap_ranges(a+1, a+na+1, b+1) interschimbă elementele din vectorul a de la poziția 1 până la poziția na, cu cele din vectorul b, începând de la poziția 1.
- Observație: zonele care se interschimbă NU trebuie să se suprapună.

reverse

- Ce face: inversează un vector (o secvență dintr-un vector)
- Exemplu: reverse(a+1, a+na+1) inversează elementele din vectorul a de la poziția 1 până la poziția na.

rotate

- Ce face: permută, la stânga, elementele unui vector

- Exemplu: rotate(a+1, a+x, a+na+1) mută elementele, începând de la poziția x, spre stânga, punând la final elementele de la poziția 1 la poziția x.

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
rotate(a+1, a+4, a+10);
//vectorul a devine {0,4,5,6,7,8,9,1,2,3}
```

sort

- Ce face: ordonează (crescător) un vector
- Exemplu: sort(a+1, a+na+1) ordonează crescător elementele din vectorul a de la poziția 1 la poziția na.
- Observație: funcția sort admite să primească drept parametru o funcție care să decidă criteriul de ordonare.

sort

```
#include <iostream>
#include <algorithm>
using namespace std;
typedef struct element {int a,b;};
element a[100];
int i,n;
int f(element x, element y)
{//sorteaza crescator dupa campul a
    return x.a<y.a;//returneaza 1 daca sunt corect ordonate
}
int main()
{
    cin>>n; for(i=1;i<=n;i++)cin>>a[i].a>>a[i].b;
    sort(a+1, a+n+1,f);
    for(i=1;i<=n;i++)cout<<a[i].a<<" "<<a[i].b<<"\n";
    return 0;
}
```

ostream std::cout

lower_bound/upper_bound

- Ce face: căutare binară(vectorul trebuie să fie sortat)

- lower_bound determină prima poziție din vector care conține o valoare NU este mai mică ca valoarea căutată
- upper_bound determină poziția primului element mai mare ca elementul căutat

- Exemplu:
- lower_bound(a+1, a+na+1, x) – returnează poziția primului element din vector care NU e mai mic ca x
- upper_bound(a+1, a+na+1, x) – returnează poziția primului element din vector care e mai mare ca x

lower_bound/upper_bound

```
#include <iostream>
#include <algorithm>
using namespace std;
int a[11]={0,1,2,3,3,4,4,4,4,5,6},n,*lb,*ub;
           //          5       9
int main()
{
    lb=lower_bound(a+1,a+11,4);
    ub=upper_bound(a+1,a+11,4);
    cout<<"Lower:"<<lb-a<<"\n";//5
    cout<<"Upper:"<<ub-a<<"\n";//9
    return 0;
}
```

merge

- Ce face: interclasează doi vectori ordonați, obținând un al treilea vector

- Exemplu: merge(a+1, a+na+1, b+1, b+nb+1, c) interclasează elementele vectorului a de la poziția 1 la na, cu elementele vectorului b de la poziția 1 la nb, rezultatul punându-se în c.

lexicographical_compare

- Ce face: compară lexicografic 2 vectori
- Exemplu: lexicographical_compare(a+1, a+na+1, b+1, b+nb+1) returnează true dacă primul vector este mai mic din punct de vedere lexicografic decât al doilea.

next_permutation/prev_permutation

- Ce fac: generează permutarea următoare/anterioară

- Exemplu:
- next_permutation(a+1,a+na+1) plecând de la valorile din vectorul a generează următoarea permutare crescător lexicografică și returnează true dacă a reușit.

- prev_permutation(a+1,a+na+1) plecând de la valorile din vectorul a generează permutarea anterioară crescător lexicografică și returnează true dacă a reușit.

next_permutation

```
#include <iostream>
#include <algorithm>
using namespace std;
int a[4]={0,2,1,5},i;
int main()
{
    do
    {
        for(i=1;i<=3;i++)cout<<a[i];
        cout<<endl;
    }while(next_permutation(a+1,a+4));
    return 0;
}
/*afiseaza
215
251
512
521
```

prev_permutation

```
#include <iostream>
#include <algorithm>
using namespace std;
int a[4]={0,2,1,5},i;
int main()
{
    do
    {
        for(i=1;i<=3;i++)cout<<a[i];
        cout<<endl;
    }while(prev_permutation(a+1,a+4));
    return 0;
}
/*afiseaza
215
152
125
*/
```