Standard Template Library

Structura STL

- Structura STL cuprinde trei elemente principale: *containerele*,
- *iteratorii* și *algoritmii*.

- *Containerele* sunt diferite tipuri de obiecte care conțin alte obiecte, de tipuri predefinite sau definite de programator.

- *Iteratorii* sunt obiecte care se comportă asemănător pointerilor şi care sunt utilizați pentru a accesa elementele unui container.

- *Algoritmii* furnizează funcționalități de acces şi prelucrare asupra elementelor containerelor.

Containerele

- Secvențiale: list, vector, deque
- Asociative: set, multiset, map, multimap
- Adaptive: stack, queue, priority_queue

Containere secvențiale

- Containerele secvențiale sunt colecții liniare și ordonate de date în care accesul se face pe baza poziției elementului în cadrul containerului.
- Cele trei clase predefinite de tip container secvențial sunt:
  - vector
  - list
  - deque
- Ordinea în care se găsesc elementele în cadrul acestor containere este ordinea în care ele sunt adăugate.
- De reținut că, în cazul șabloanelor list şi deque elementele pot fi adăugate pe la ambele capete.

Containere secvențiale - vector

- Pentru a putea fi folosit trebuie inclus <vector>

- Declarare:
  - vector <tip> identificator;
  - vector <tip> identificator(număr_elemente)
  - vector <tip> identificator(număr_elemente, valoare_inițială)

Containere secvențiale - vector

- Metode

Nume	Ce face
begin()	returnează un iterator la începutul vectorului
end()	returnează un iterator la sfârșitul vectorului
size()	returnează numărul de elemente din vector
empty()	returnează true dacă vectorul e gol
front()	returnează o referință la primul element din vector
back()	returnează o referință la ultimul element din vector
push_back()	adaugă un element la sfârșitul vectorului (acesta crește!)
pop_back()	Șterge ultimul element
insert()	Inserează un vector în alt vector la poziția specificată
erase()	șterge un element sau mai multe
clear()	șterge toate elementele din vector

Containere secvențiale - vector

```
#include <iostream>
#include <vector>
using namespace std;
vector <int> a;
vector <int>b(5);
vector<int>::iterator it;
int i;
int main()
{
    for(i=1;i<=4;i++)a.push_back(i);
    for(i=0;i<b.size();i++)b[i]=i*10;
    b.front()=5+b.back();
    it=b.begin();
    b.insert(it+2,a.begin(),a.end());
    for(i=0;i<b.size();i++)cout<<b[i]<<" ";
    return 0;
}
```

Containere secvențiale - deque

- Pentru a putea fi folosit trebuie inclus <deque>

- Declarare:
  - deque <tip> identificator;
  - deque <tip> identificator(număr_elemente)
  - deque <tip> identificator(număr_elemente, valoare_inițială)

Containere secvențiale - deque

- Metode
- Identice cu cele din vector
- Apar în plus: push_front, pop_front

Containere secvențiale - deque

```
#include <iostream>
#include <deque>
#include <algorithm>
using namespace std;
deque <int> a(5); deque <int> b(4, 7);
int main()
{ for(auto i: a) cout << i << " "; // afiseaza toate elementele din a
  cout << endl; for(int i=2; i<5; i++) b.push_back(i);
  for(int i=8; i<12; i++) b.push_front(i);
  for(auto i: b) cout << i << " ";// afiseaza 11 10 9 8 7 7 7 7 2 3 4
  cout << endl; reverse(b.begin(), b.end());
  while(!b.empty())
   { cout << b.front() <<" ";
     b.pop_front();} // afiseaza 4 3 2 7 7 7 7 8 9 10 11
  cout << "\n" <<b.size();
  return 0;
}
```

Container asociativ - map

- Pentru a putea fi folosit trebuie inclus <map>
- Declarare:
- map <tip_cheie, tip_valoare_asociata> nume_map;

- Inserarea se poate face doar ca pereche cheie - valoare_asociată

Container asociativ - map

```
#include <iostream>
#include <map>
using namespace std;
map <string, int> a;
int n, i, x; string s;
int main()
{ cin >> n;
  for(i=1; i<=n; i++) { cin >> s >> x; a[s]+=x; }
  for(auto i: a) // parcurge si afiseaza toate perechile cheie-valoare
    cout << i.first << " " << i.second << endl;
  cout << a.size(); // afiseaza numarul de elemente
  return 0;
}
```

Container asociativ - map

```
#include <bits/stdc++.h>
using namespace std;
map <vector<int>, int> a; ///utilizare ca “indice” un vector
vector<int>z, k;
int i, j, n;
int main()
{   cin >> n; for(i=1; i<=n; i++)z.push_back(i);
    do { a[z]++; }while(next_permutation(z.begin(), z.end()));
    for(i=1; i<=n;i++)
        if(prev_permutation(z.begin(),z.end()))a[z]++;
    while(prev_permutation(z.begin(),z.end()));
    for(auto i:a)
        {   k=i.first;
            for(auto j:k)cout << j << ' `;
            cout << '='<<i.second <<endl;
        }
return 0; }
```

Container asociativ - set

- Se folosește pentru a memora un set de elemente unice (mulțime de elemente)
- Declarare

set <tip> nume_variabila;

Container asociativ - set

- Metode

Nume	Ce face
insert	Inserează un element în mulțime
erase	Șterge elementul din mulțime
clear	Golește mulțimea
size	Returnează numărul de elemente
count	Numără de câte ori apare un element – valoarea returnată poate fi 0 sau 1

```cpp
#include <iostream>
#include <set>
using namespace std;
set <int> s;
int x;
int main()
{
  cin >> x;
  while(x) { s.insert(x); cin >> x; }
  cout << s.size();
  return 0;
}
```

Bitset

- bitset nu face parte din STL, dar poate fi util când avem puțină memorie și trebuie să notăm prezența/absența unor elemente. Este practic un vector de biți.
- Declarare:

  bitset <nr_elemente> nume_variabila;
- Se pot face inițializări la declarare:

  bitset <32>a (241) // pune în a valoare 241 în binar

  bitset <32> b(“101”) // pune în b 101

bitset

- Metode

Nume	Ce face
set()	Se poate folosi pentru a pune toți biții pe 1
reset()	Se poate folosi pentru a pune toți biții pe 0
flip()	Biții care erau 1 devin 0 și cei care erau 0 devin 1
count()	Numără câți biți au valoarea 1
to_string()	Convertește biții din bitset în string
to_ulong()	Convertește în **long**, dacă bitset-ul are prea multe cifre se va genera eroare
to_ullong()	Convertește în **long long**, dacă bitset-ul are prea multe cifre se va genera eroare

bitset

```
#include <iostream>
#include <bitset>
using namespace std;
bitset <10> a(241); // 241 = 11110001 in binar
bitset <10> b("1001"); // 1001 = 9 in zecimal
int x;
int main()
{ cout << a.to_string(); //afiseaza 0011110001
  x = b.to_ulong(); cout << x; // afiseaza 9
  b = 45; // pune in b valoare 45 in binar
  cout << b.to_string(); // afiseaza 0000101101
  return 0;
}
```