Стандартна бібліотека шаблонів (STL)

Шаблон пари

```
pair <int, char> PAIR1 ;

PAIR1.first = 100;
PAIR1.second = 'G' ;

cout << PAIR1.first << " " ;
cout << PAIR1.second << endl ;

pair <string, unsigned> PAIR2 ("Мехмат",
65);

cout << PAIR2.first << " " ;
cout << PAIR2.second << endl ;

pair  g1;         //default
pair  g2(1, 'a');  //initialized,  different data
type
pair  g3(1, 10);   //initialized,  same data type
pair  g4(g3);    //copy of g3
```

```
template <class T, class U>
class pair {
    public:
      T first;
      U second;
       pair(): first(0), second(0){}
       pair(const T& x, const U & y):  first(x),
second(y){}
pair (const pair<U,V>& pr);
       pair& operator= (const pair& pr){
           return pair(pr->first, pr->second);
       }
****
};
template <class T1,class T2>
pair<T1,T2> make_pair (T1 x, T2 y){
    return ( pair<T1,T2>(x,y) );
}
template<class U, class V> pair (const
pair<U,V>& pr)
```

Використання стандарнтної пари

```
//CPP program to illustrate pair STL
#include <iostream>
#include <utility>
using namespace std;
int main(){
pair <int, char> PAIR1 ;
    pair <string, double> PAIR2 ("Студент", 4.23) ;
    pair <string, double> PAIR3 ;
    PAIR1.first = 100;
    PAIR1.second = 'G' ;
    PAIR3 = make_pair ("Мехмат кращій",4.56);
    cout << PAIR1.first << " " ;
    cout << PAIR1.second << endl ;
      cout << PAIR2.first << " " ;
    cout << PAIR2.second << endl ;
      cout << PAIR3.first << " " ;
    cout << PAIR3.second << endl
```

Оператори **(=, ==, !=, >=, <=) :** We can use operators with pairs as well.

(=) : Присвоює нову пару – конструктор копіювання

pair& operator= (const pair& pr);

Порівняння (==): Порівнює поля pair1 та pair2. Пари рівні якщо pair1.first рівне pair2.first та pair1.second рівне pair2.second.

Not equal (!=) operator with pair : Протилежний до рівності.

Логічні(>=, <=) оператори: Порівнює лише перше поле пари

```
pair<int, int>pair1 = make_pair(1, 12);
pair<int, int>pair2 = make_pair(9, 12);

cout << (pair1 == pair2) << endl;
cout << (pair1 != pair2) << endl;
cout << (pair1 >= pair2) << endl;
cout << (pair1 <= pair2) << endl;
cout << (pair1 > pair2) << endl;
cout << (pair1 < pair2) << endl;
```

```
0
1
0
1
0
1
```

swap

```
pair<char, int>pair1 = make_pair('A', 1);
   pair<char, int>pair2 = make_pair('B', 2);

   cout << "Before swapping:\n " ;
   cout << "Contents of pair1 = " << pair1.first << " " << pair1.second ;
   cout << "Contents of pair2 = " << pair2.first << " " << pair2.second ;
   pair1.swap(pair2);

   cout << "\nAfter swapping:\n ";
   cout << "Contents of pair1 = " << pair1.first << " " << pair1.second ;
   cout << "Contents of pair2 = " << pair2.first << " " << pair2.second ;
```

Стандартна бібліотека шаблонів C ++ (STL)

Стандартна бібліотека шаблонів (STL) - це набір шаблонів C ++ для створення загальних структурних даних і функцій, таких як списки, стеки, масиви і т.д.
STL має чотири компоненти:

Algorithms (алгоритми)
Containers (контейнери)
Functions (функціонали)
Iterators (ітератори)

Утіліти utility та клас string

Контейнери

Контейнери або класи контейнерів зберігають об'єкти і дані
Контейнери послідовності: реалізують структури даних, до яких можна звертатися послідовно:

- vector
- list
- deque
- arrays
- forward_list(C++11)

Container Adaptors (Контейнери адаптери) : інтерфейс для контейнерів баз даних.

- queue
- priority_queue
- stack

Associative Containers (асоціативні контейнери) : структури даних що зберігають сортоівані структури даних що дозволяють швидкий пошук (O(log n) складність).

- set
- multiset
- map
- multimap

Stack in C++ STL

empty() – Чи порожній стек – Time Complexity : O(1)
size() – розмір стеку– Time Complexity : O(1)
top() – верхній елемент – Time Complexity : O(1)
push(g) – додає елемент ‘g’ на верхівку стеку – Time Complexity : O(1)
pop() – видаляє верхівку – Time Complexity : O(1)

```
#include <iostream>
#include <stack>
using namespace std;

void showstack(stack <int> s)
{
    while (!s.empty())
    {
        cout << '\t' << s.top();
        s.pop();
    }
    cout << '\n';
}
```

```
int main () {
    stack <int> s;
    s.push(10);
    s.push(30);
    s.push(20);
    s.push(5);
    s.push(1);
      cout << "The stack is : ";
 showstack(s); // The stack is :     1    5    20    30    10
      cout << "\ns.size() : " << s.size(); .//5
    cout << "\ns.top() : " << s.top();  //1
    cout << "\ns.pop() : "; //
    s.pop();
    showstack(s); //s.pop() :     5    20    30    10
}
```

Queue in Standard Template Library (STL)

empty() – Returns whether the queue is empty.
size() – Returns the size of the queue.
queue::swap() : Exchange the contents of two queues but the queues must be of same type, although sizes may differ.
emplace() : Insert a new element into the queue container, the new element is added to the end of the queue.
front() function returns a reference to the first element of the queue
back() –function returns a reference to the last element of the queue.
push(g) function adds the element 'g' at the end of the queue.
pop() function deletes the first element of the queue

```
#include <iostream>
#include <queue>
using namespace std;
void showq(queue <int> gq){
    queue <int> g = gq;
    while (!g.empty())     {
        cout << '\t' << g.front();
        g.pop();
    }
    cout << '\n';
}
```

```
int main () {
    priority_queue <int> gquiz;
    gquiz.push(10);     gquiz.push(30);     gquiz.push(20);
    gquiz.push(5);     gquiz.push(1);
     cout << "The priority queue gquiz is : ";
    showpq(gquiz);
      cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.top() : " << gquiz.top();
    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showpq(gquiz);
}
```

Priority Queue in STL

empty() function returns whether the queue is empty.
size() function returns the size of the queue.
top()– Returns a reference to the top most element of the queue
push(g) function adds the element ‘g’ at the end of the queue.
pop() function deletes the first element of the queue.
swap() This function is used to swap the contents of one priority queue with another priority queue of same type and size.
emplace() This function is used to insert a new element into the priority queue container, the new element is
priority_queue value_type in C++ STL– Represents

```
#include <iostream>
#include <queue>
using namespace std;
void showpq(priority_queue <int> gq)
{
    priority_queue <int> g = gq;
    while (!g.empty())     {
        cout << '\t' << g.top();
        g.pop();
    }
    cout << '\n';
}
```

```
int main() {
    queue <int> gquiz;
    gquiz.push(10);     gquiz.push(20);    gquiz.push(30);
    cout << "The queue gquiz is : ";
    showq(gquiz); //    30    20    10    5    1
    cout << "\ngquiz.size() : " << gquiz.size(); //5
    cout << "\ngquiz.front() : " << gquiz.front(); //30
    cout << "\ngquiz.back() : " << gquiz.back(); //1
    cout << "\ngquiz.pop() : "; //30
    gquiz.pop();
    showq(gquiz); // 20    10    5  1
}
```

Deque

Двонаправлені черги являють собою контейнери послідовності з функцією розширення і стиснення на обох кінцях.
insert() : вставляє в дек нові елементи та повертає ітератор на початок встановлюваних елементів.
max_size() : максимальний розмір контейнера.
assign() присвоює нові зачення контейнеру.
resize() змінює розмір деку.
push_front() вставляє елементи на початок.
push_back() вставляє елемент в кінець.
pop_front() ,pop_back() : **pop_front()** видаляє елемент зпочатку. **pop_back()** видаляє елемент з кінця.
front() повертає посилання на перший елемент. **back()** повертає вказівник на останній елемент.
clear() видаляє всі елементи деку. **erase()** видаляє елементи всередині вказаного діапазону.
empty() перевіряє чи порожній дек. **size()** повертає розмір деку.
operator= присвоює новий дек. **operator[]** повретає елемент заданої позиції.
at() повертає елмент на даній позиції. **swap()** замінює значення деків.

Приелад використання деку

```
#include <iostream>
#include <deque>
using namespace std;
void showdq(deque <int> g){
    deque <int> :: iterator it;
    for (it = g.begin(); it != g.end(); ++it)
        cout << '\t' << *it;
    cout << '\n';
}

int main()
{
    deque <int> gquiz;
    gquiz.push_back(10);
    gquiz.push_front(20);
    gquiz.push_back(30);
    gquiz.push_front(15);
    cout << "The deque gquiz is : ";
    showdq(gquiz);
    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.max_size() : " <<
gquiz.max_size();

    cout << "\ngquiz.at(2) : " << gquiz.at(2);
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop_front() : ";
    gquiz.pop_front();
    showdq(gquiz);

    cout << "\ngquiz.pop_back() : ";
    gquiz.pop_back();
    showdq(gquiz);

    return 0;
}
```

Клас array

Operations on array :-
1. at() :- доступ до елементу
2**. operator[]** :- доступ до елементу C-style arrays.

```
// C++ code to demonstrate working of array,  // to() and get()
#include<iostream>
#include<array> // for array, at()
using namespace std;
int main() {
    array<int,6> ar = {1, 2, 3, 4, 5, 6}; // Initializing the array elements
    cout << "The array elemets are (using at()) : "; // Printing array elements using at()
    for ( int i=0; i<6; i++)
    cout << ar.at(i) << " ";
    cout << endl;
      // Printing array elements using operator[]
    cout << "The array elements are (using operator[]) : ";
    for ( int i=0; i<6; i++)
    cout << ar[i] << " ";
    cout << endl;
```

4. front() :- перший елемент.
5. back() :- останній елемент

```
// C++ code to demonstrate working of
// front() and back()
#include<iostream>
#include<array> // for front() and back()
using namespace std;
int main() {
    // Initializing the array elements
    array<int,6> ar = {1, 2, 3, 4, 5, 6};

    // Printing first element of array
    cout << "First element of array is : ";
    cout << ar.front() << endl;

    // Printing last element of array
    cout<< "Last element of array is : ";
    cout << ar.back() << endl;
```

6. size() – розмір масиву
7. max_size() :- максимальна кількість елементів даного масиву The size() and max_size() return the same value.

```
// C++ code to demonstrate working of
// size() and max_size()
#include<iostream>
#include<array> // for size() and max_size()
using namespace std;
int main() {
    // Initializing the array elements
    array<int,6> ar = {1, 2, 3, 4, 5, 6};

    // Printing number of array elements
    cout << "The number of array elements is : ";
    cout << ar.size() << endl;

    // Printing maximum elements array can hold
    cout << "Maximum elements array can hold is : ";
    cout << ar.max_size() << endl;
```

8. swap() :- The swap() swaps all elements of one array with other.

```
#include<iostream>
#include<array> // for swap() and array
using namespace std;
int main() {
    array<int,6> ar = {1, 2, 3, 4, 5, 6}; // Initializing 1st array
    array<int,6> ar1 = {7, 8, 9, 10, 11, 12};  // Initializing 2nd array
    // Printing 1st and 2nd array before swapping
    cout << "The first array elements before swapping are : ";
    for (int i=0; i<6; i++) {     cout << ar[i] << " "; }
    cout << endl;
    cout << "The second array elements before swapping are : ";
    for (int i=0; i<6; i++) {    cout << ar1[i] << " "; }
    cout << endl;
    // Swapping ar1 values with ar
    ar.swap(ar1);
    // Printing 1st and 2nd array after swapping
```

9. empty() :- чи порожній масив.
10. fill() :- Заповнює масив певним елементом чи значеннями функції

```
#include<iostream>
#include<array> // for fill() and empty()
using namespace std;
int main() {
    array<int,6> ar;     // Declaring 1st array
    array<int,0> ar1;    // Declaring 2nd array
    // Checking size of array if it is empty
    ar1.empty()? cout << "Array empty":
        cout << "Array not empty";
    cout << endl;
    // Filling array with 0
    ar.fill(0);
    // Displaying array after filling
    cout << "Array after filling operation is : ";
    for ( int i=0; i<6; i++)
        cout << ar[i] << " ";
```

Вектор

- vector::begin() and vector::end()
- vector rbegin() and rend()
- vector::cbegin() and vector::cend()
- vector::crend() and vector::crbegin()
- vector::assign()
- vector::at()
- vector::back()
- vector::capacity()
- vector::clear()
- vector::push_back()
- vector::pop_back()
- vector::empty()
- vector::erase()

- vector::size()
- vector::swap()
- vector::reserve()
- vector::resize()
- vector::shrink_to_fit()
- vector::operator=
- vector::operator[]
- vector::front()
- vector::data()
- vector::emplace_back()
- vector::emplace()
- vector::max_size()
- vector::insert()

Модіфікатори (Modifiers):

assign() – присвоюємо значення у векторі

push_back() – додає елемент у кінець вектору

pop_back() – видаляє елемент з кінця вектору

insert() – додає елемент(елементи) у вказану позицію

erase() – видаляє елементи з вектору .

swap() – Міняє значення векторів з одного в інший. Розміри векторів можуть відрізнятись.

clear() – видаляє елементи з вектору.

emplace() – Розширює вектор вставляючи нові елементи на дану позицію

emplace_back() – Додає нові елементи в кінець вектору

```
// C++ program to illustrate the
// Modifiers in vector
#include <bits/stdc++.h>
#include <vector>
using namespace std;
int main() {
    // Assign vector
    vector<int> v;

    // fill the array with 10 five times
    v.assign(5, 10);

    cout << "The vector elements are: ";
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " ";

    // inserts 15 to the last position
    v.push_back(15);
    int n = v.size();
    cout << "\nThe last element is: " << v[n -
1];

    // removes last element
    v.pop_back();

    // prints the vector
    cout << "\nThe vector elements are: ";
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " ";

    // inserts 5 at the beginning
    v.insert(v.begin(), 5);

    cout << "\nThe first element is: " << v[0];

    // removes the first element
    v.erase(v.begin());

    cout << "\nThe first element is: " << v[0];
```

```
    // inserts at the beginning
    v.emplace(v.begin(), 5);
    cout << "\nThe first element is: " << v[0];

    // Inserts 20 at the end
    v.emplace_back(20);
    n = v.size();
    cout << "\nThe last element is: " << v[n - 1];
}
```

```
The vector elements are: 10 10 10 10 10
The last element is: 15
The vector elements are: 10 10 10 10 10
The first element is: 5
The first element is: 10
The first element is: 5
The last element is: 20
```

Обсяг (Capacity)

size() – кількість елементів вектора.
max_size() – максимальна кількість елементів вектору.
capacity() – розмір алокатору, виділенного під цій контейнер .
resize() – Змінює розмір вектору.
empty() – Повертає true, якщо контейнер порожній.
shrink_to_fit() – Зменшує розмір контейнеру видаляючи неініціалізовані елементи.
reserve() –виділяє пам'ять для зберігання рівно n елементів.

```
vector<int> g1;

    for (int i = 1; i <= 5; i++)
        g1.push_back(i);

    cout << "Size : " << g1.size();
    cout << "\nCapacity : " << g1.capacity();
    cout << "\nMax_Size : " << g1.max_size();

    // resizes the vector size to 4
    g1.resize(4);

    // prints the vector size after resize()
    cout << "\nSize : " << g1.size();

    // checks if the vector is empty or not
    if (g1.empty() == false)
        cout << "\nVector is not empty";
    else
        cout << "\nVector is empty";

    // Shrinks the vector
        g1.shrink_to_fit();
        cout << "\nVector elements are: ";
        for (auto it = g1.begin(); it != g1.end();
    it++)
            cout << *it << " ";
     }
```

```
Size : 5
Capacity : 8
Max_Size : 461168601842738790 3
Size : 4
Vector is not empty
Vector elements are: 1 2 3 4
```

Доступ до елементів:

reference operator [g] – Перевантажений оператор для доступу до ‘g’-го елементу вектору

at(g) – Повертає посидання на ‘g’-ий елемент вектору

front() – повертає посилання на перший елемент контейнеру

back() – повертає посилання на останній елемент контейнеру

data() – повертає вказівник на місце де зберігаються дані.

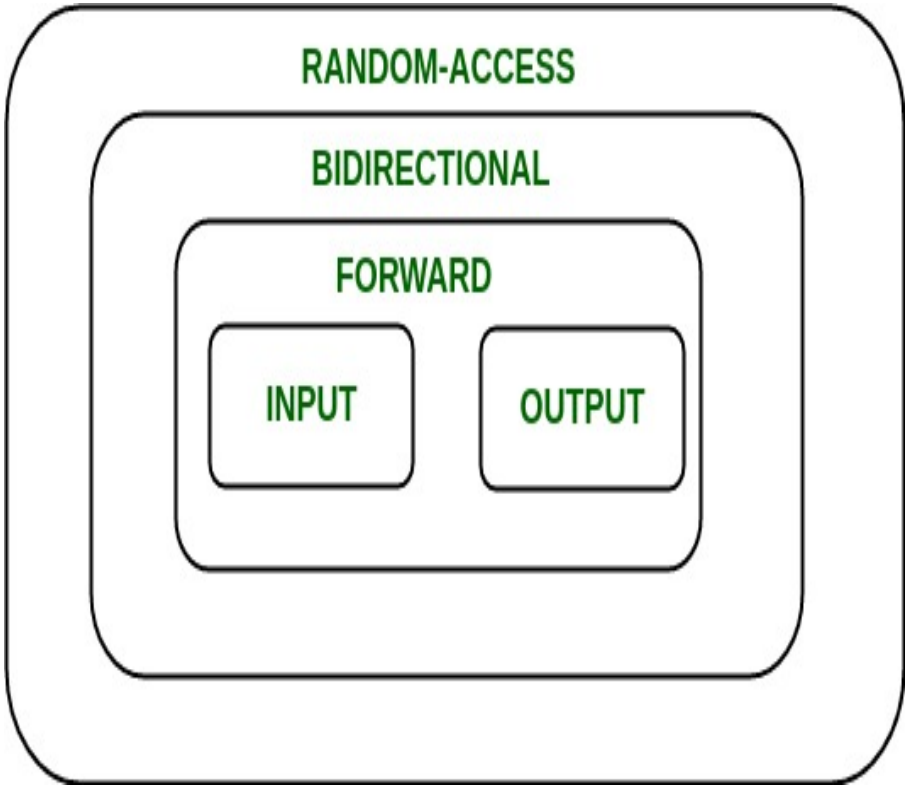
```
vector<int> g1;
  for (int i = 1; i <= 10; i++)   g1.push_back(i * 10);

cout << "\nReference operator [g] : g1[2] = " << g1[2];
  cout << "\nat : g1.at(4) = " << g1.at(4);
  cout << "\nfront() : g1.front() = " << g1.front();
  cout << "\nback() : g1.back() = " << g1.back();
  // pointer to the first element
int* pos = g1.data();
  cout << "\nThe first element is " << *pos;
```

Введення до ітераторів у C ++

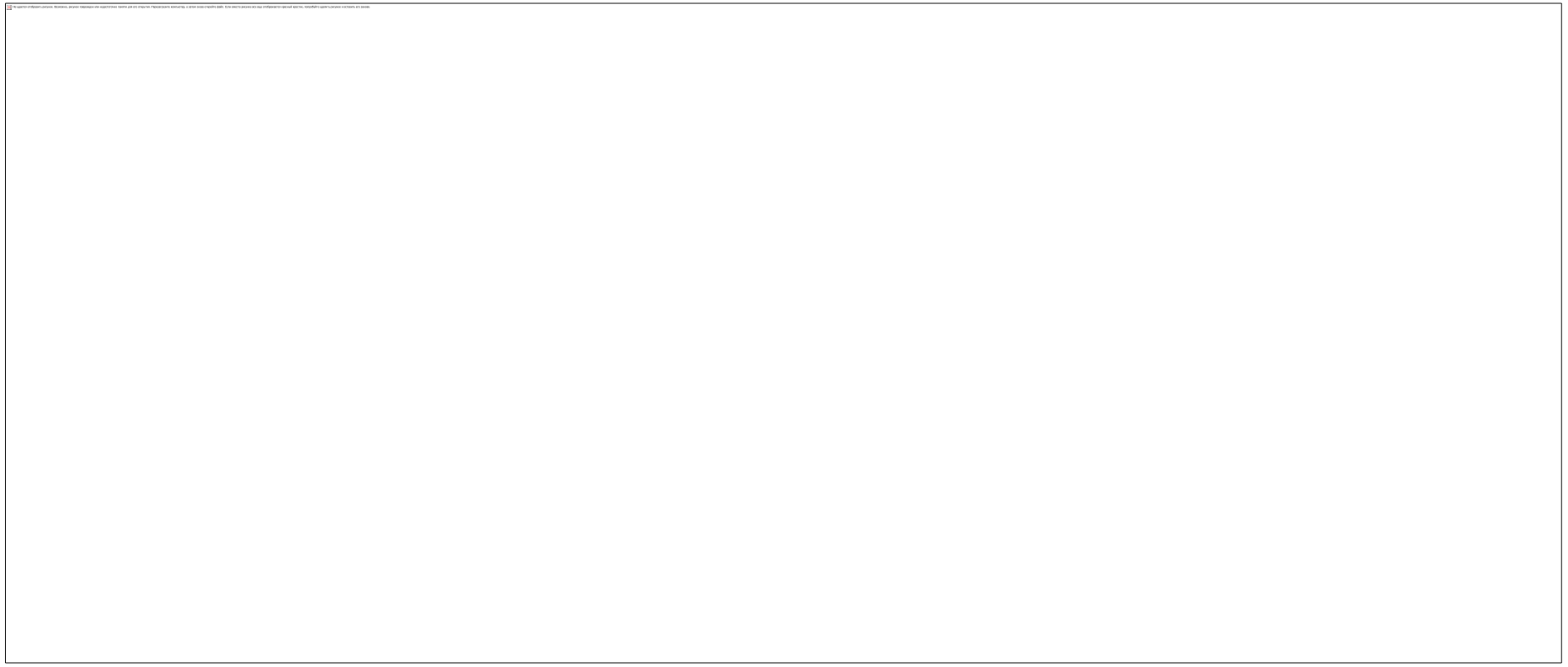
Ітератор - це об'єкт (як покажчик), який вказує на елемент всередині контейнера. Ми можемо використовувати ітератори для переміщення по вмісту контейнера. Їх можна візуалізувати як щось подібне до вказівника, який вказує на певне місце, і ми можемо отримати доступ до вмісту з цього конкретного місця.

Ітератори відіграють важливу роль у підключенні алгоритму з контейнерами разом з маніпуляціями даними, що зберігаються всередині контейнерів. Найбільш очевидною формою ітератора є покажчик. Покажчик може вказувати на елементи в масиві і може перебирати їх за допомогою оператора інкременту (++). Але не всі ітератори мають подібної функціональності, як у покажчиків.

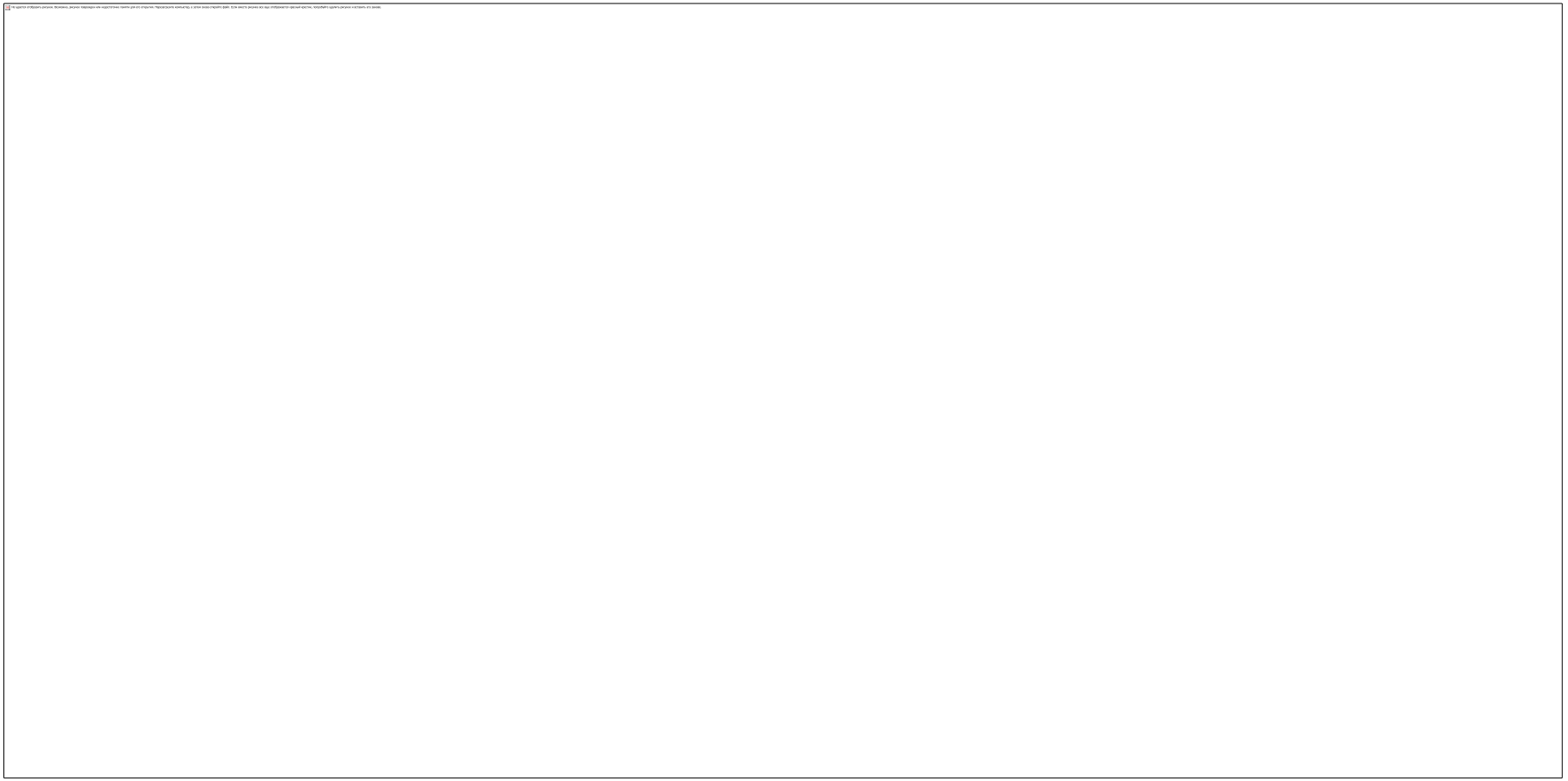


1.Ітератори вводу **(Input iterators**):
2. Ітератори виводу(**Output Iterators**):
3. Однонаправлений ітератор (**Forward Iterator**):
4. Двонаправлені ітератори(**Bidirectional Iterators**):
5. Ітератори прямого доступу(**Random-Access Iterators**):

Властивості ітераторів



Контейнери та їх ітератори



Операція * повертає елемент , що стоїть в поточній позиції. Якщо цей елемент має члени, то за допомогою операції -> можна одержати доступ до них безпосередньо з ітератора.

Операція ++ переміщає ітератор уперед на наступний елемент. Більшість ітераторів також дозволяють повернення до попереднього елементу за допомогою операції --.

Операції == і != повертають результат перевірки, чи представляють два ітератори ту саму позицію .

Операція = присвоює ітератор (позицію елемента, на яку він посилається).

```
// .iterator, begin() and end()
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main() {
    vector<int> ar = { 1, 2, 3, 4, 5 };

    // Обявили iterator до vector
    vector<int>::iterator ptr;

    // Виведення за допомогою begin() та end()
    cout << "The vector elements are : ";
    for (ptr = ar.begin(); ptr < ar.end(); ptr++)
        cout << *ptr << " ";

    return 0;
}
```

3. advance() :- Функція **increment the iterator position** вказану кількість разів.

```
// C++ code to demonstrate the working of  // advance()
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main() {
    vector<int> ar = { 1, 2, 3, 4, 5 };
    // iterator на vector
    vector<int>::iterator ptr = ar.begin();
    // Using advance() to increment iterator position points to 4
    advance(ptr, 3);

    // виведення
    cout << "The position of iterator after advancing is : ";
    cout << *ptr << " ";
    return 0;
}
```

4. next() :- Повертає новий **iterator** що стає на наступну позицію після руху **вперед** на вказану у аргументах позицію.
5. prev() :- Повертає новий **iterator** що стає на наступну позицію після руху **назад** на вказану у аргументах позицію.

```
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main() {
    vector<int> ar = { 1, 2, 3, 4, 5 };
   // iterators to a vector
    vector<int>::iterator ptr = ar.begin();
    vector<int>::iterator ftr = ar.end();
    // next() пересуває ітератор на  4
    auto it = next(ptr, 3);
    // Using prev() to return new iterator
    // points to 3
    auto it1 = prev(ftr, 3);
  }
    cout << "The position of new
iterator using next() is : ";
    cout << *it << " ";
    cout << endl;

    cout << "The position of new
iterator using prev()  is : ";
    cout << *it1 << " ";
    cout << endl;

    return 0;
```

6. inserter() :- Вставляє елементи в будь-яку позицію контейнеру.
Має 2 аргументи**, container та iterator на потрібну позицію**.

```
// C++ code to demonstrate the working of // inserter()
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main() {
    vector<int> ar = { 1, 2, 3, 4, 5 };
    vector<int> ar1 = {10, 20, 30};
    // декларація ітератору
    vector<int>::iterator ptr = ar.begin();
    // просування ітератору
    advance(ptr, 3);
    // копіювання елементів
    // вставляє ar1 після 3-ої позиції в ar
    copy(ar1.begin(), ar1.end(), inserter(ar,ptr));
    cout << "The new vector after
inserting elements is : ";
    for (int &x : ar)
        cout << x << " ";

    return 0;
}
```

Приклад використання ітератору

```
// stl/list1old.cpp
#include <list>
#include <iostream>
using namespace std;
int main(){
list<char> coll; // список символів
// додаємо елементи від 'a'  до 'z'
for (char c='a'; c<='z'; ++c) {
coll.push_back(c);
}
//  виводимо на друк всі елементи :
// - обходимо всі елементи
list<char>::const_iterator pos;
for (pos = coll.begin(); pos != coll.end(); ++pos) {
cout << *pos << ' ';
}
cout << endl;
}
```

```
/*Знову після створення списку і заповнення його символами від 'a' до 'z', ми виводимо на екран всі елементи. Однак замість діапазонного циклу for:*/
for (auto elem : coll) { // C++11
cout << elem << ' ';
}
/*Тепер всі елементи виводяться в звичайному циклі за допомогою ітераторів, що обходить елементи контейнера:*/
list<char>::const_iterator pos;
for (pos = coll.begin(); pos != coll.end(); ++pos)
{
cout << *pos << ' ';
}

/*Ітератор pos з'являється безпосередньо перед циклом. Він має тип ітератора для доступу до константних елементів контейнерного класу.*/
list<char>::const_iterator pos;
```

У кожнім контейнері з'являються два типи ітераторів.
1. ***Iератор контейнер ::iterator*** переміщається по елементах у режимі читання/запису.
2. ***Iератор контейнер ::const_iterator*** переміщається по елементах тільки в режимі читання.
Наприклад, у класі list визначення можуть мати наступний вид:

```
namespace std {
template <typename T>
class list {
public:
typedef ... iterator;
typedef ... const_iterator;
...
};
}
```

Точний тип **iterator** і **const_iterator** визначається реалізацією.
У циклі for ітератор pos ініціалізується позицією першого елемента:
pos = coll.begin() ma coll.cbegin()
Цикл продовжується, поки ітератор pos не досягне кінця контейнера:
pos != coll.end() ma coll.cend()

Ітератор вводу

```
// C++ program to demonstrate output iterator
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int>v1 = {1, 2, 3, 4, 5};
    vector<int>::iterator i1;

    for (i1=v1.begin();i1!=v1.end();++i1)
    {
        // Зміна значень по ітератору
        *i1 = 1;
    }
    // результат1 1 1 1 1
    return 0;
}
```

Ітератор довільного доступу

```
// C++ program to demonstrate Random-access iterator
#include<iostream>
#include<vector>
using namespace std;
int main() {
    vector<int>v1 = {1, 2, 3, 4, 5};
    vector<int>::iterator i1;
    vector<int>::iterator i2;

    // i1 на початок
    i1 = v1.begin();
    // i2 на кінець
    i2 = v1.end();

    // порівняння
    if ( i1 < i2)     {
        cout << "Yes";
    }

    // арифметичні операції
    int count = i2 - i1;
    cout << "\ncount = " << count;

    int i;
      // доступ[ ]
    for(i=0;i<v1.size();++i)     {
        cout << v1[i] << " ";
    }
}
```

Множина Set

```
//  set.cpp: Дві множини
  include <iostream>
#include <set>
using namespace  std;
int set1()
{    set<int, less<int> > S, T;
S.insert(10);  S.insert(20);  S.insert(30); S.insert (10);
T.insert (20);  T.insert (30);  T.insert (10);

if (S == T) cout << "Equal sets, containing
for (set<int,  less<int> >::iterator i = T.begin();
       i != T.end(); i++) {
       cout << *i << " ";
 }        // Результат: Equal sets, containing: 10 20 30
cout << endl;
return 0;
}
```

```
// multiset.cpp:  Дві множини з дублікатами// мільтимножина
#include <iostream>
#include <set>
using namespace  std;
int multiset1()
{   multiset<int, less<int> > S, T;
S.insert(10); S.insert(20); S.insert(30); S.insert(10);
T.insert(20); T.insert(30); T.insert(10);

if (S == T) cout << "Equal multisets: \n";
else cout << "Unequal multisets:\n";

cout << "S:  ";
copy (S.begin() , S.end(),
   ostream_iterator<int>(cout, " "));          // Вивод:
cout << endl; cout << "T:  ";                  //  Unequal multisets:
copy (T.begin() , T.end(),                     // S: 10 10 20 30
   ostream_iterator<int>(cout, " "));  //       T: 10 20 30
cout << endl;
return 0;
}
```

Множини

```
int set_algorithm()  {

const int N = 5;
string s1[]= {"Bill", "Jessica", "Ben", "Mary",  Monica"};
string s2[N] = {"Sju","Monica","John","Bill","Sju"};
typedef set<string> SetS;
SetS A(s1,  s1 + N);
SetS B(s2,  s2 + N);
print(A); print(B);
SetS prod, sum; // множини для результату
set_intersection (A.begin(), A.end(), B.begin(), B.end(),
                  inserter(prod, prod.begin()));
print(prod);
set_union (A.begin(), A.end(), B.begin(),B.end(),
inserter (sum, sum.begin()));
print(sum);
if (includes (A.begin(), A.end(), prod.begin(),   prod.end()))
cout << "Yes" << endl;
else cout <<"No" << endl;
return 0;
}
```

```
// Результат:
 Ben  Bill  Jessica  Mary
Monica // Множина A

// Множина B
 // Перетин set_intersection ->
prod
 Bill  Monica
 // Обєднання set_union  ->
sum
 Ben  Bill  Jessica John  Mary
Monica  Sju          //
Включення includes можини
prod в множину A
 Yes
```

```
// mapl.cpp:
  include <iostream>
#include <string>
#include <map>
using namespace  std;
// Створення функтору для порівняння
class  compare2 {
public:
bool operator()(const char *s, const char *t) const{
      return strcmp (s, t) <  0;
}
};
```

```
int map1()
{   map<char*, long, compare2> D;
D["Johnson, J."]   =  12345;
D["Smith, P."]     =  54321;
D["Shaw, A."]      =  99999;
D["Atherton, K."]  =  11111;
char GivenName [30] ;
cout  <<  "Enter a name:  ";
cin.get(GivenName,  30);
if  (D.find (GivenName) != D.end())
     cout << "The number is " << D[GivenName];
else cout  <<  "Not  found.";
cout  <<  endl;
return  0;
}
```

```
typedef multimap<char*, long, compare3> mmtype;
int multimap1()
{   mmtype D;
D.insert(mmtype::value_type("Johnson, J.",  12345));
D.insert(mmtype::value_type("Smith, P.",  54321));
D.insert(mmtype::value_type("Johnson, J.",  10000));
cout << "There are " << D.size() << " elements. \n";
return  0;
}
```

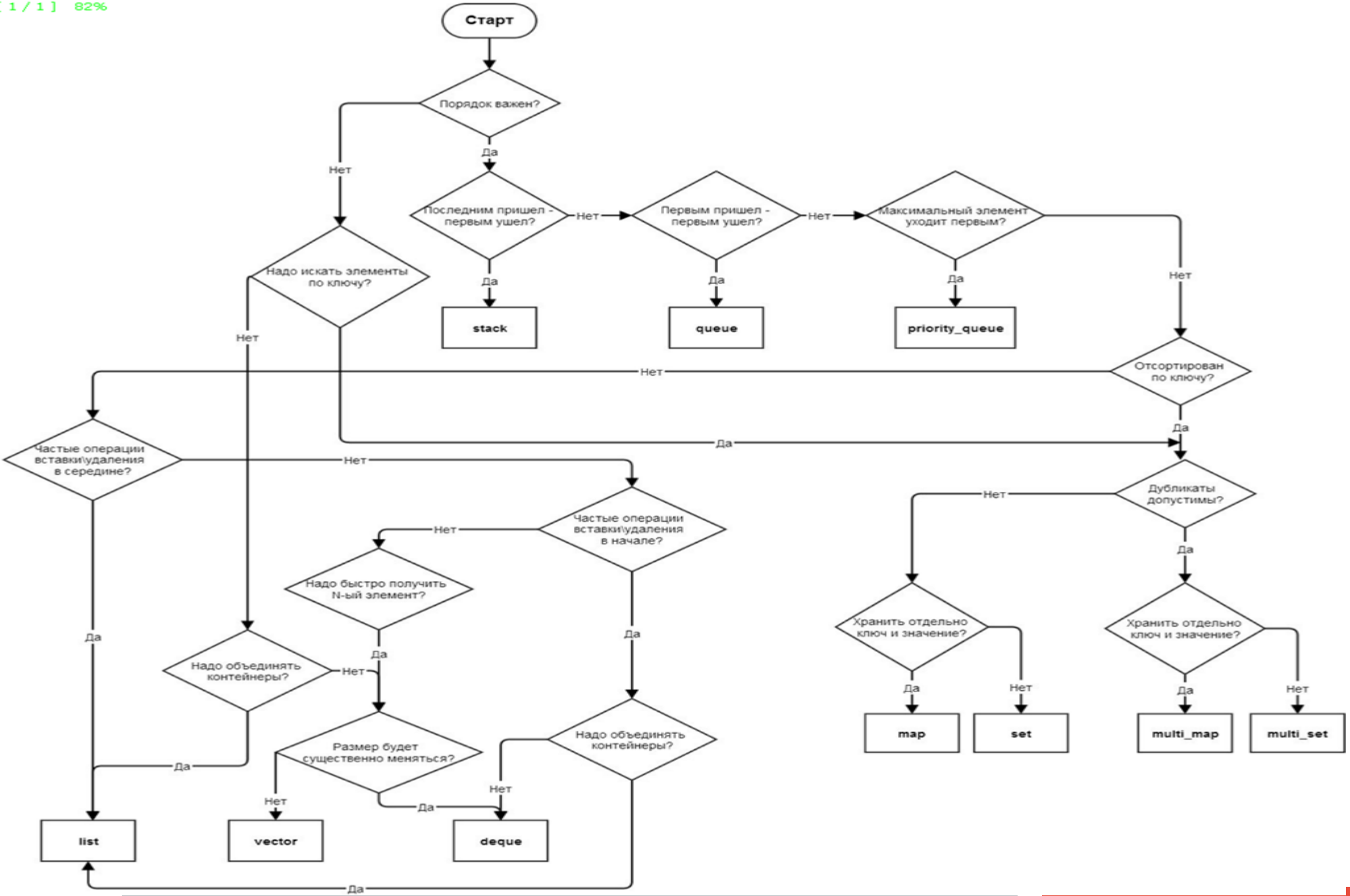
Програма виводить:

```
There are 3 elements.
```

```
//D["Johnson, J."] = 12345; - некорктно для мільтимножини
D.insert (mmtype::value_type ("Johnson, J.", 12345));
/*де mmtype - це
multimap<char*,  long,  compare3> */
```

Приклад

```
char punct[6] = {'.', ',', '?', '!', ':', ';'};
set <char> punctuation(punct, punct + 6);
ifstream in("prose.txt");
if (!in)  { cerr << "File not found\n"; exit(1);}
map<string, int> first << setw(4) << right << it->secwordCount;
string s;
while (in >> s)
{  int n = s.size();
   if (punctuation.count(s[n - 1]))
         s.erase(n - 1, n);
   ++wordCount[s];
}
ofstream out("freq_map.txt");
map<string, int>::const_iterator it =   wordCount.begin();
for (it; it  != wordCount.end(); ++it)
         out << setw(20) << left << it->ond << endl;
cout <<"Rezalt in file freq_map.txt" << endl;
return 0;
```


Старт
Порядок важен?
Да
Нет
Последним пришел - первым ушел?
Нет
Первым пришел - первым ушел?
Нет
Максимальный элемент уходит первым?
Нет
Да
stack
queue
priority_queue
Надо искать элементы по ключу?
Нет
Да
Отсортирован по ключу?
Нет
Да
Частые операции вставки\удаления в середине?
Нет
Да
Дубликаты допустимы?
Нет
Да
Частые операции вставки\удаления в начале?
Нет
Да
Надо быстро получить N-ый элемент?
Да
Надо объединять контейнеры?
Нет
Да
Размер будет существенно меняться?
Нет
Да
Надо объединять контейнеры?
Нет
Да
Хранить отдельно ключ и значение?
Да
Нет
Хранить отдельно ключ и значение?
Да
Нет
map
set
multi_map
multi_set
list
vector
deque
Да

Функтор

MyFunctor(10); < == > MyFunctor.operator()(10);

```
 #include <bits/stdc++.h>
using namespace std;

// Функтор
class increment {
private:
    int num;
public:
    increment(int n) : num(n) {  }

    // оператор інкременту на дане число
    int operator () (int arr_num) const {
        return num + arr_num;
    }
};
```

Алгоритми: сортування

sort(startaddress, endaddress) // sort() сортує інтервал [startaddress,endaddress)
startaddress: адреса початку інтервалу array
endaddress: адреса кінця інтервалу array.

```
#include <iostream>
#include <algorithm>
using namespace std;
void show(int a[]) {   for(int i = 0; i < 10; ++i) cout << a[i] << " "; }

int main() {
    int a[10]= {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
    cout << "\n The array before sorting is : ";
    show(a);

    sort(a, a+10); //   sort(a.begin(), a.end()); // sort(a.begin(), a.begin()+10);
    cout << "\n\n The array after sorting is : ";
    show(a);
  }
```

Бінарний пошук

binary_search(startaddress, endaddress, valuetofind)
startaddress: початок.
endaddress: кінець.
valuetofind: мета пошуку.

```
int main() {
    int a[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0, 23, 3 };
    int asize = sizeof(a) / sizeof(a[0]);
    cout << "\n The array is : ";     show(a, asize);
    sort(a, a + asize);
    show(a, asize);
    if (binary_search(a, a + 10, 2))
            cout << "\nElement found in the array";
    else
            cout << "\nElement not found in the array";
    if (binary_search(a, a + 10, 10))
        cout << "\nElement found in the array";
    else
        cout << "\nElement not found in the array";
```

Non-Manipulating Algorithms

sort(first_iterator, last_iterator) – сортує контейнер.
reverse(first_iterator, last_iterator) – обертає контейнер.
***max_element (first_iterator, last_iterator)** – максимальний елемент.
***min_element (first_iterator, last_iterator)** –мінімальний елемент.
accumulate(first_iterator, last_iterator, initial value of sum) – сумує всі елементи контейнеру
count(first_iterator, last_iterator,x) – кількість x в контейнері.
find(first_iterator, last_iterator, x) – останнє входження **x**
binary_search(first_iterator, last_iterator, x) – бінарний пошук.
lower_bound(first_iterator, last_iterator, x) – повертає ітератор на перший елент в ітервалі [first,last) що має значення не менше ‘x’.
upper_bound(first_iterator, last_iterator, x) – повертає ітератор на перший елент в ітервалі [first,last) що має значеннея більше ‘x’.

```
// A C++ program to demonstrate working of sort(), reverse()
#include <algorithm>
#include <iostream>
#include <vector>
int main() {
    int arr[] = {10, 20, 5, 23 ,42 , 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);
    cout << "Vector is: ";
    for (int i=0; i<n; i++)         cout << vect[i] << " ";
    // Сортуємо по зростанню
    sort(vect.begin(), vect.end());
    cout << "\nVector after sorting is: ";
    for (int i=0; i<n; i++)
       cout << vect[i] << " ";

    // реверсія контейнеру
    reverse(vect.begin(), vect.end());
```

```
    cout << "\nVector after reversing is: ";
    for (int i=0; i<6; i++)
        cout << vect[i] << " ";

    cout << "\nMaximum element of vector is: ";
    cout << *max_element(vect.begin(), vect.end());

    cout << "\nMinimum element of vector is: ";
    cout << *min_element(vect.begin(), vect.end());

    // сумуємо починаючи з 0
    cout << "\nThe summation of vector elements is: ";
    cout << accumulate(vect.begin(), vect.end(), 0);
}
```

```
  include <algorithm>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {

    int arr[] = {10, 20, 5, 23 ,42, 20, 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Occurrences of 20 in vector : ";
    // Кількість 20-ток від початку до кінця
    cout << count(vect.begin(), vect.end(), 20);

    // find() повертає ітератор на кінець, якщо не знайшло елемент
find(vect.begin(), vect.end(),5) != vect.end()?
                cout<< "\nElement found«:    cout << "\nElement not found";
```

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int main() {
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);
    // Сортуємо масив для lower_bound() та upper_bound()
    sort(vect.begin(), vect.end());
    // Перше входження 20 , auto == vector<int>::iterator
    auto q = lower_bound(vect.begin(), vect.end(), 20);
    // Останнє входження 20
    auto p = upper_bound(vect.begin(), vect.end(), 20);
    cout << "The lower bound is at position: ";
    cout << q-vect.begin() << endl;

    cout << "The upper bound is at position: ";
    cout << p-vect.begin() << endl;
```

Some Manipulating Algorithms

arr.erase(position to be deleted) – видаляє вказані елементи вектору.
arr.erase(unique(arr.begin(),arr.end()),arr.end()) – видаляє повторні включення елементів
next_permutation(first_iterator, last_iterator) – This modified the vector to its next permutation.
prev_permutation(first_iterator, last_iterator) – This modified the vector to its previous permutation.
distance(first_iterator,desired_position) – відстань від першого ітератору до даної позиції. Зручна для знаходження індексів.

```
int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
int n = sizeof(arr)/sizeof(arr[0]);
vector<int> vect(arr, arr+n);

cout << "Vector is :";
for (int i=0; i<6; i++)
    cout << vect[i]<<" ";

// видаляємо другий елемент
vect.erase(vect.begin()+1);

cout << "\nVector after erasing the element: ";
for (int i=0; i<5; i++)
    cout << vect[i] << " ";
```

```
    // сортуємо весь вектор
    sort(vect.begin(), vect.end());

    cout << "\nVector before removing duplicate
occurrences: ";
    for (int i=0; i<5; i++)
        cout << vect[i] << " ";

    // видаляємо повтори
    vect.erase(unique(vect.begin(),vect.end()),v
ect.end());

    cout << "\nVector after deleting duplicates: ";
    for (int i=0; i< vect.size(); i++)
        cout << vect[i] << " ";
```

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    // повертає відстань до максимального елементу - 9
    cout << "Distance between first to max element: ";
    cout << distance(vect.begin(),
                max_element(vect.begin(), vect.end()));
```

```
// Обчислення суми елементів послідовності
#include <iostream>
#include <numeric>   //друга бібліотека алгоритмів
using namespace std;
int accum1_massiv()
{   const int N = 8;
int a[N] = {4, 12, 3, 6, 10, 7, 8, 5}, sum = 0;
sum = accumulate(a, a+N, sum);
cout << "Sum of all elements:  " << sum << endl;
cout << "1000 + a[2] + a[3] + a[4]  = “
 << accumulate(a+2, a+5, 1000) << endl;
return 0;
}
Sum of all elements: 55
1000 + a[2] + a[3] + a[4] = 1019
```

Функціонал з алгоритмом

```
 /* Добуток
     Шаблон multiplies<int>() аналогичний до шаблону greater<int>().
Можна його використати для обичслення добутку по всй
послідовності:
*/
#include <iostream>
#include <numeric>
#include <algorithm>
#include <functional> // Функціонали в STL
using namespace std;
int accum2_massiv(){
   const int N = 4;
int a[N] = {2, 10, 5, 3}, prod =  1;

prod = accumulate(a, a+N, prod, multiplies<int>());

cout << "Product of all elements: " << prod << endl;

}
   // Результат 300 (=1x2x10x5x3).
```