МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №6**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Робота з бібліотекою STL»

Варіант №5

**Виконав:**

студент гр. БС-71

Воробйов І.О.

**Перевірив:**

асистент каф. БМК

Рисін С.В.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2019

**🞏 Практичне заняття без зауважень**

**🞏 Практичне заняття має зауваження:**

**🞏 несвоєчасний захист**

**🞏 присутні зауваження до UML діаграми:**

**🞏 діаграма класу не відповідає коду**

**🞏 виконані не за стандартом:**

**🞏 атрибути**

**🞏 відношення**

**🞏 потужність**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 присутні зауваження до коду:**

**🞏 задача завдання вирішена хибно**

**🞏 код програми не компілюється**

**🞏 хибно задані специфікатори доступу**

**🞏 помилки у визначенні конструкторів / деструкторів**

**🞏 відсутні списки ініціалізації в конструкторах**

**🞏 константні методи**

**🞏 використано глобальні змінні**

**🞏 статичні змінні при роботі з масивами**

**🞏 оформлення коду**

**🞏 присутні зайві символи «{» та «}»**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 невірні відповіді на запитання:**

**🞏 №1 🞏 №2 🞏 №3 🞏 №4 🞏 №5**

**🞏 №6 🞏 №7 🞏 №8 🞏 №9 🞏 №10**

**🞏 маються інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Завдання:**

1. Ознайомитися з теоретичними відомостями для роботи зі стандартною бібліотекою STL за списком рекомендованої літератури.
2. Переробити розробленій в попередній лабораторній роботі шаблонний клас з використанням контейнерів STL.
3. Скласти і захистити звіт по роботі.

**UML діаграма**



**Код програми:**

**plural.h:**

#pragma once

#include <iostream>

#include <vector>

#include <iterator>

using namespace std;

template<typename T>

class plural

{

public:

vector<T> \*arr;

plural() {};

plural(int s)

{

arr = new vector<T>;

T tmp;

for (size\_t i = 0; i < s; i++)

{

cin >> tmp;

arr->push\_back(tmp);

}

}

bool is\_empty()

{

if (arr->empty())

return true;

else

return false;

}

int size()

{

return arr->size();

}

bool find(T data)

{

plural<T>\* tmp = new plural<T>;

tmp->arr = new vector<T>;

tmp->arr->push\_back(data);

for (size\_t i = 0; i < arr->size(); i++)

if (arr[i] == tmp->arr[0])

return true;

return false;

}

void add(T data)

{

if (!this->find(data))

{

arr->push\_back(data);

}

else

cout << "Elem inside." << endl;

}

void del(T data)

{

plural<T>\* tmp = new plural<T>;

tmp->arr = new vector<T>;

tmp->arr->push\_back(data);

for (size\_t i = 0; i < arr->size(); i++)

{

if (arr[i] == tmp->arr[0])

{

arr[i] = arr[arr->size()-1];

arr->pop\_back();

break;

}

}

}

template<typename T>

friend ostream& operator <<(ostream& cout, plural<T>& obj)

{

copy(obj.arr->begin(), obj.arr->end(), ostream\_iterator<int>(cout, " "));

return cout;

}

template<typename T>

friend plural<T> operator + (plural<T>& obj, plural<T>& obj1)

{

plural<T> tmp;

tmp.arr = new vector<T>;

tmp.arr->swap(\*obj.arr);

tmp.arr->swap(\*obj1.arr);

return tmp;

}

template<typename T>

friend plural<T> operator \* (plural<T>& obj, plural<T>& obj1)

{

plural<T> tmp;

int k = 0;

for (size\_t i = 0; i < obj.size(); i++)

{

for (size\_t j = 0; j < obj1.size(); j++)

{

if (obj.arr[i] == obj1.arr[j])

{

tmp.arr->push\_back(0);

tmp.arr[k]=obj.arr[i];

k++;

break;

}

}

}

return tmp;

}

};

**Source.cpp:**

#include "plural.h"

int main()

{

int n;

cout << "Input n: ";

cin >> n;

plural<int>\* arr\_list = new plural<int>(n);

int k = 1;

while (k != 0)

{

cout << "1-Check for empty." << endl;

cout << "2-Get size." << endl;

cout << "3-Find elem." << endl;

cout << "4-Add elem." << endl;

cout << "5-Del elem." << endl;

cout << "6-Union plural." << endl;

cout << "7-Intersection plural." << endl;

cout << "8-Print." << endl;

cout << "0-Exit" << endl;

cout << ">";

cin >> k;

switch (k)

{

case 1:

{

if (arr\_list->is\_empty())

cout << "Plural empty!" << endl;

else

cout << "Plural not empty!" << endl;

break;

}

case 2:

{

cout << "Numb of elem: " << arr\_list->size() << endl;

break;

}

case 3:

{

int value;

cout << "Input value:";

cin >> value;

if (arr\_list->find(value))

cout << "Elem inside." << endl;

else

cout << "Elem not find." << endl;

break;

}

case 4:

{

int value;

cout << "Input value:";

cin >> value;

arr\_list->add(value);

break;

}

case 5:

{

int value;

cout << "Input value:";

cin >> value;

arr\_list->del(value);

break;

}

case 6:

{

int n1;

cout << "Input n: ";

cin >> n1;

plural<int>\* arr1 = new plural<int>(n1);

plural<int> arr2 = \*arr\_list + \*arr1;

cout << arr2;

break;

}

case 7:

{

int n1;

cout << "Input n: ";

cin >> n1;

plural<int>\* arr1 = new plural<int>(n1);

plural<int> arr2 = \*arr\_list \* \*arr1;

cout << arr2;

break;

}

case 8:

{

cout << \*arr\_list;

}

}

system("pause");

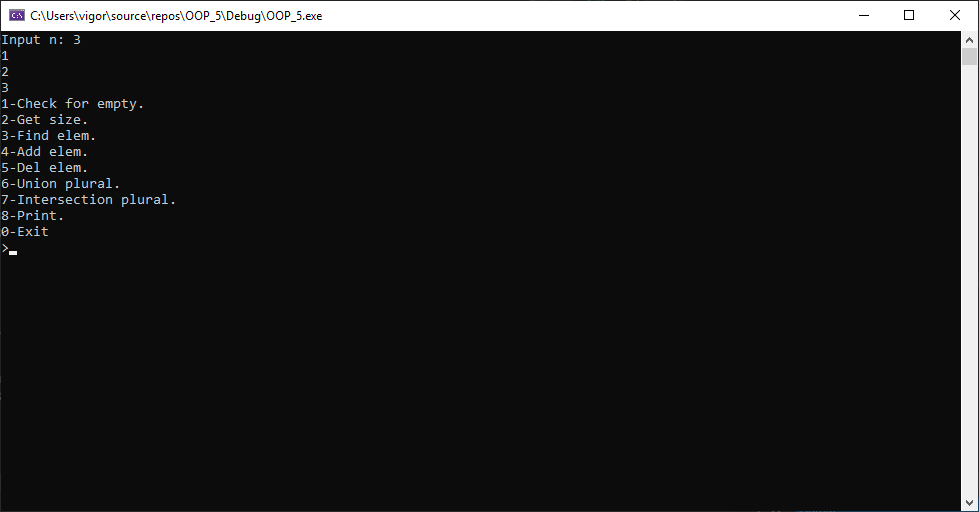
system("cls");

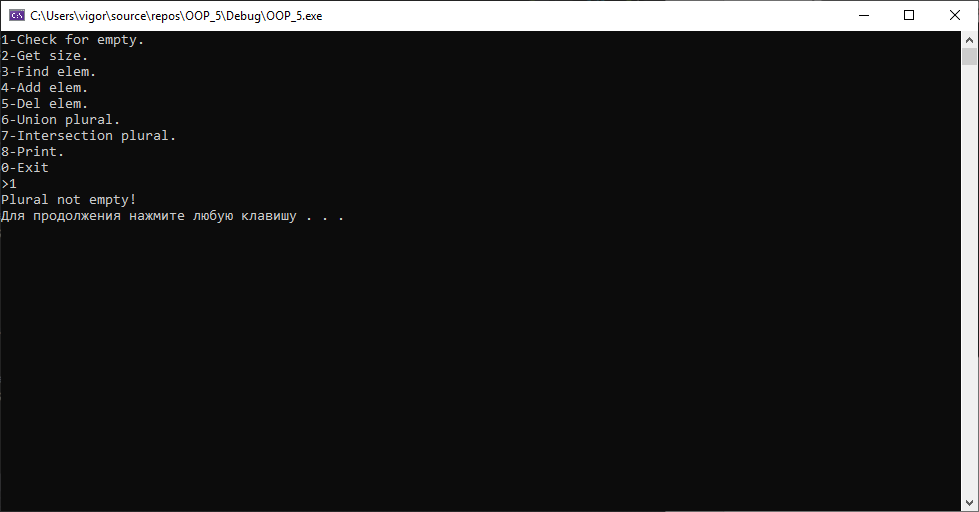
}

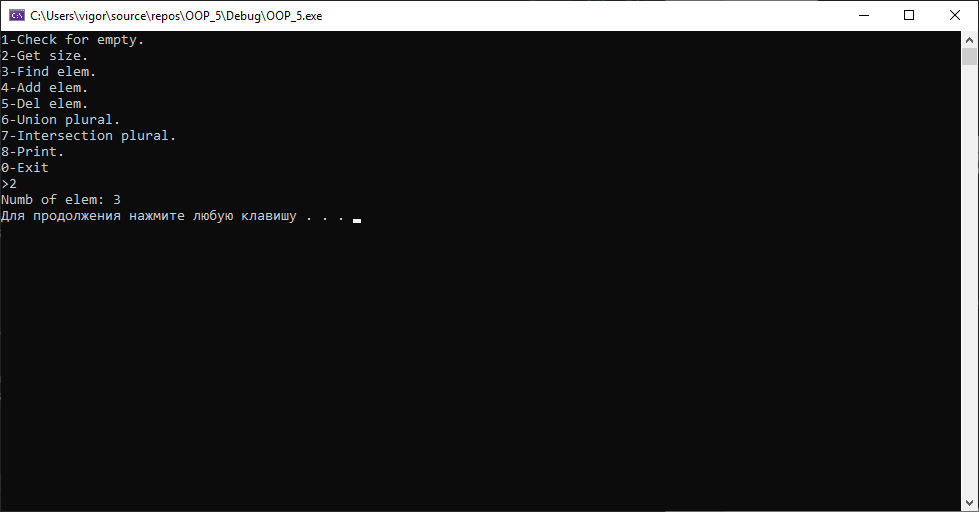
return 0;

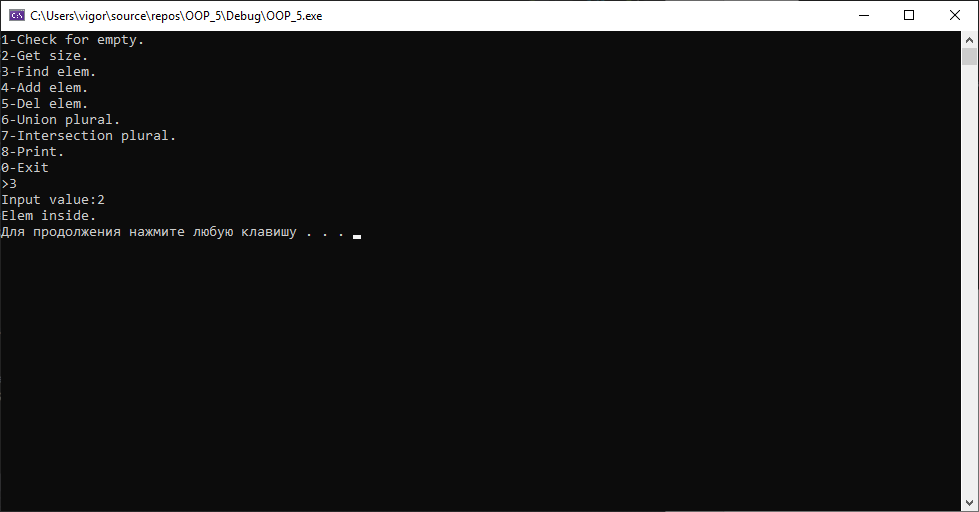
}

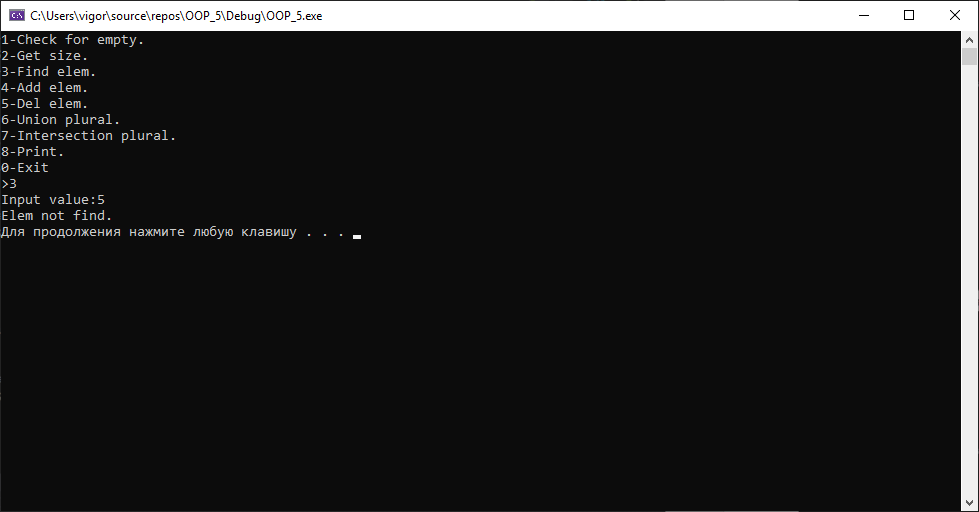
**Результат роботи програми:**

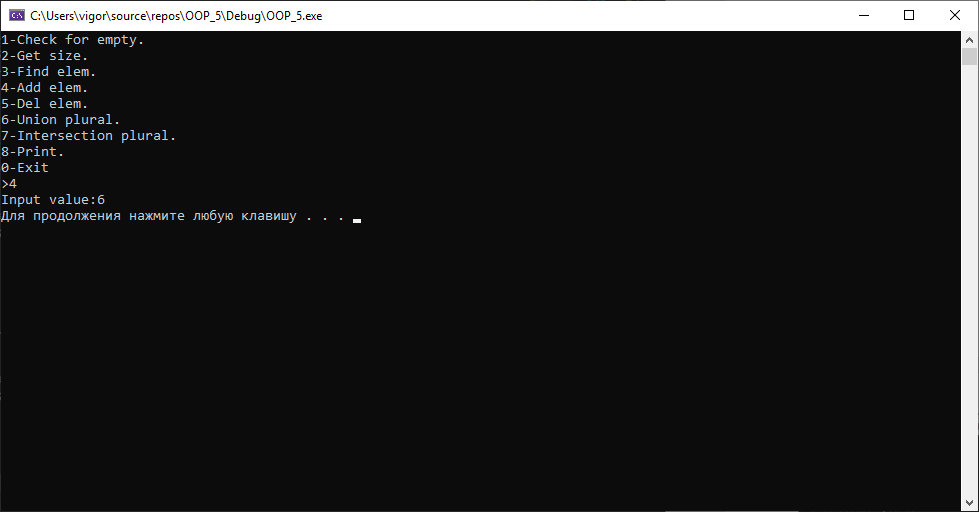


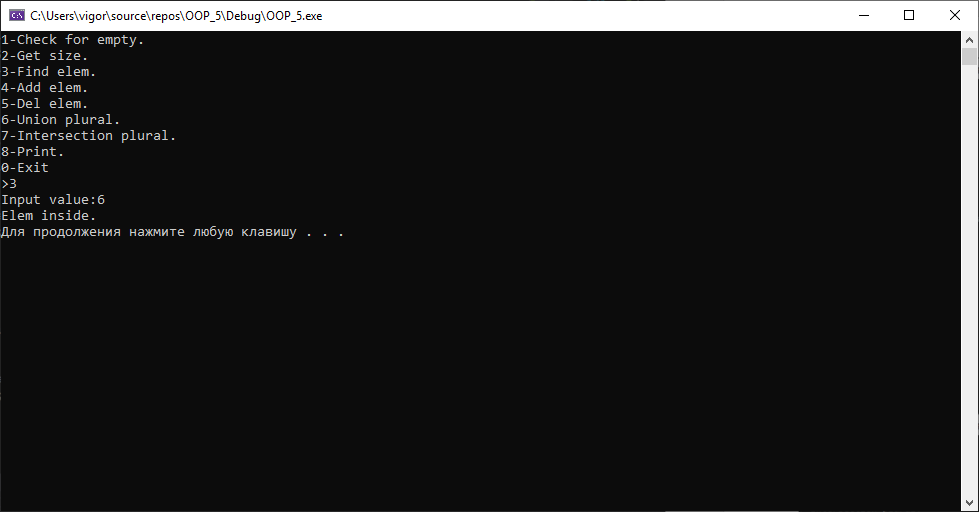


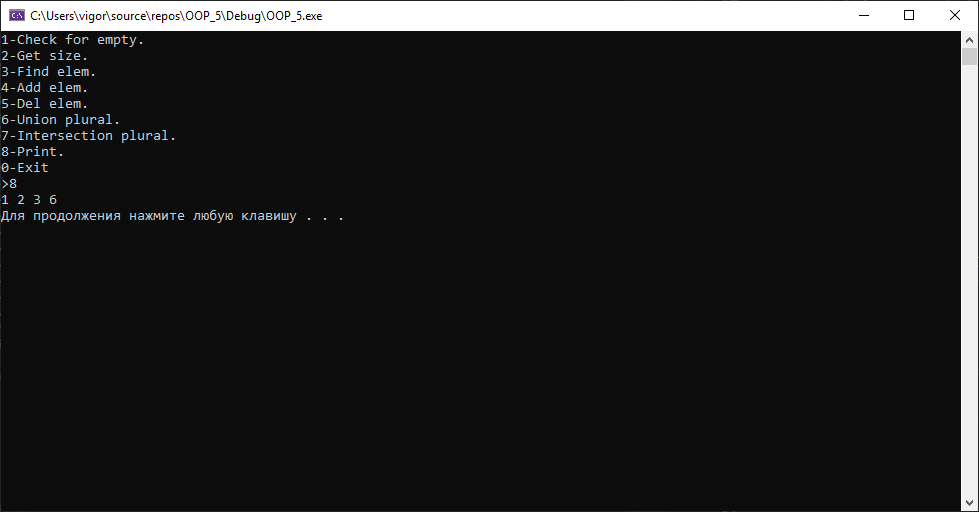


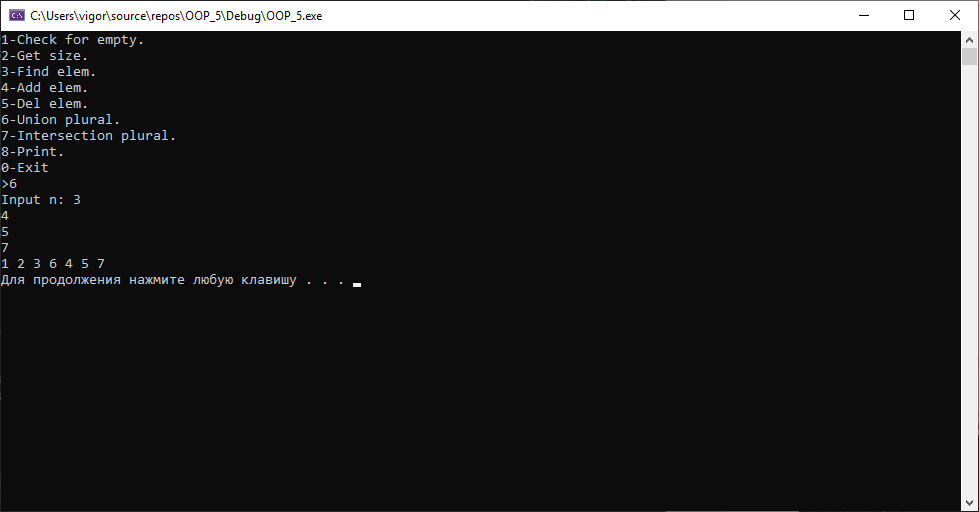












**Контрольні запитання**

1. **Опишіть організацію контейнера STL.**

STL організована навколо трьох основних елементів: контейнерів, алгоритмів і ітераторів. Алгоритми застосовуються до контейнерів через ітератори.

Контейнери містять набір однотипних елементів.

Контейнеры библиотеки STL можно разделить на четыре категории: последовательные, ассоциативные, контейнеры-адаптеры и псевдоконтейнеры. В контейнерах для хранения элементов используется семантика передачи объектов по значению. Другими словами, при добавлении контейнер получает копию элемента.

1. **Які класи належать до категорії контейнерів?**

Контейнерні класи - це універсальні шаблонні класи, призначені для зберігання елементів заданого типу в областях пам'яті. Існує кілька різних типів контейнерів. Наприклад, клас vector (динамічний масив), deque( двостороння черга), list (двозв'язний список).

1. **Які вимоги до типів, які можна використовувати з контейнерами STL?**

Оскільки контейнер – це шаблонних клас, то він може зберігати об’єкти будь-якого типу(int, char, користувацькі і т.д). Контейнері часто требуют открітій констуктора копирования, оператора присвоемия и деструктора.

1. **Яка різниця між контейнерами та адаптерами контейнерів?**

До складу STL входять три адаптера контейнерів - stack, queue іpriority\_queue. Контейнер-адаптер - це різновид послідовного або асоціативного контейнера, який обмежує інтерфейс. Адаптери не підтримують роботу з ітераторами. Це відрізняє їх від інших контейнерів.

1. **Що таке ітератори, яких типів вони бувають та в чому їх принципова різниця?**

Ітератор - це об'єкт, який може виконувати ітерацію елементів в контейнері STL і надавати доступ до окремих елементів.

Типи:

*Вхідні*(забезпечують доступ для читання в одному напрямку): operator ++, operator \*, operator->, конструктор копії, operator =, operator ==, operator! =

*Вихідні*(забезпечують доступ для запису в одному напрямку, їх не можна

порівнювати на рівність): operator ++, operator \*, конструктор копії

*Однонаправлені*(забезпечують доступ для читання і записи в одному напрямку, їх можна порівнювати на рівність): operator ++, operator \*, operator->, конструктор копії, конструктор за замовчуванням, operator =, operator ==, operator! =

*Двонаправлені*(підтримують всі функції, описанні для односпрямованих ітераторів,

дозволяють переходити до попереднього елементу): operator ++, operator--, operator \*, operator->, конструктор копії, конструктор за замовчуванням, operator =, operator ==, operator! =

*Довільного доступу*: оperator ++, operator--, operator \*, operator->, конструктор копії, конструктор за замовчуванням, operator =, operator ==, operator! =, operator +, operator-, operator + =,

operator- =, operator <, operator>, operator <=, operator> =, operator []

Однонаправлені ітератори задовольняють всім вимогам ітераторів введення і виведення і можуть використовуватися щоразу, коли визначається той чи інший вид. Двонаправлені ітератори задовольняють всім вимогам послідовних ітераторів і можуть вживатися щоразу, коли визначається послідовний ітератор. Ітератори довільного доступу задовольняють всім вимогам двонапрямлених ітераторів і можуть вживатися щоразу, коли визначається двонаправлений ітератор.

1. **Які оператори можна використовувати над усіма типами літераторів(ітераторів) ?**

operator ++, operator \*.

1. **Що таке ітератор довільного доступу? Які операції він дозволяє використовувати?**

Ітератори довільного доступу еквівалентні вказівникам: підтримують арифметику покажчиків, синтаксис індексації масивів і всі форми порівняння.

1. **Які вимоги до ітераторів з боку алгоритмів?**

Алгоритми можуть працювати з користувацькими структурами даних, коли ці структури даних мають типи ітераторів. Деяким алгоритмам достатньо оператора считування(наприклад, for\_each()), а іншим треба считующи(наприклад ,ostream\_iterator),inserter, front\_inserter, back\_inserter, begin(),end() ), комусь достатьньо послідовних(наприклад, replace()), а іним треба ітератор довільного доступу(наприклад,sort()).

1. **Що таке стандартні алгоритми? Наведіть приклад.**

Це алгоритми, реалізація яких не залежить від типу контейнера, тому одна заснована на шаблонах реалізація може працювати з усіма контейнерами.

Приклад: алгоритми пошуку(find(), search(), count()), сортування та впорядкування(sort(), merge(), partition()), видалення(unique(), remove()), чисельні алгоритми(accumulate(), partial\_sum()), генерації і зміни послідовності (generate(), copy(), transform()), порівняння (equal(), min(), max() ).

1. **Які алгоритми вимагають впорядкованості?**

Наприклад модифікуючі алгоритми unique (It p, It q), unique (It p, It q, Pr pred) Видаляють однакові елементи, йдуть поспіль, залишаючи тільки по одному елементу для кожного значення. Елементи послідовності повинні бути відсортовані.

bool binary\_search(It p, It q, const T &x)

Возвращает true, если в упорядоченной последовательности есть элемент, значение которого равно x, false в противном случае.