1. Які основні елементи включає Стандартна бібліотека C++?

*Стандартна бібліотека С++* являє собою колекцію класів і функцій для розв'язання складних і низькорівневих задач програмування. Бібліотека включає в себе такі компоненти:

* засоби для роботи з потоками введення / виведення;
* набір структурованих даних і алгоритмів, раніше відомих як Стандартна бібліотека шаблонів (Standard Template Library, STL);
* засоби локалізації (адаптації до національних мов);
* параметризований клас string;
* параметризований клас complex для подання комплексних величин;
* клас vallaray, оптимізований для обробки числових масивів;
* параметризований клас numeric\_limits і спеціалізації для кожного базового типу даних;
* засоби управління пам'яттю;
* велика підтримка національних наборів символів;
* засоби обробки винятків.

1. Що таке контейнер Стандартної бібліотеки?

До основних будівельних блоків, що надаються STL, можна віднести контейнери, ітератори й алгоритми. *Контейнер* - це клас, який зберігає колекцію інших об'єктів і включає базові функції для підтримки використання загальних алгоритмів. Стандартні контейнери не є похідними від деякого загального базового класу. Замість цього кожен контейнер забезпечує набір стандартних операцій зі стандартними іменами і сенсом.

1. Чим відрізняються послідовні й асоціативні контейнери?

Є такі основні групи контейнерів:

* послідовності - вектор (vector), список (list), дек (deque, черга з двома кінцями);
* адаптери послідовностей - стек (stack), черга (queue), черга з пріоритетом (priority\_queue);
* асоціативні контейнери - асоціативні масиви (map, multimap) і множини (set, multiset).

1. Що таке ітератор?

*Ітератор* - це об'єкт, який абстрагує поняття вказівника на елемент послідовності і дозволяє обходити елементи послідовності в певному напрямку. Кожен контейнерний клас в Стандартній бібліотеці С++ здатний згенерувати ітератор, який реалізує оптимальні механізми проходження елементів контейнера.

1. Які вимоги пред'являються до ітераторів Стандартної бібліотеки?

Об'єкт-ітератор повинен підтримувати такі операції:

* отримання поточного елементу (реалізовано операторами \* і ->);
* інкремент (реалізовано оператором ++);
* перевірка на рівність (реалізовано оператором ==);

1. У чому переваги вектора у порівнянні з масивом?

* За допомогою функції pop\_back() можна видалити останній елемент. Функції push\_back() і pop\_back() можуть бути застосовані до всіх послідовних контейнерів.
* Звертатися до окремих елементів можна за індексом, як до елементів масиву, наприклад:

for (int i = 0; і < a.size(); i++)

{

cіn >> a[i];

}

Можна також звертатися до елементів за допомогою функції at() (як для читання, так і для запису, оскільки ця функція повертає посилання):

* Ітератори

1. Які є способи створення вектора?

#include <vector>

**using** std::vector;

vector<**double**> a(n);

1. Як працювати з окремими елементами векторів?

Звертатися до окремих елементів можна за індексом, як до елементів масиву, наприклад:

**for** (**int** i = 0; і < a.size(); i++)

{

cіn >> a[i];

}

Можна також звертатися до елементів за допомогою функції at() (як для читання, так і для запису, оскільки ця функція повертає посилання):

1. Чим список відрізняється від масиву?

**Массив** (в некоторых [языках программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) также таблица, ряд, матрица) — [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом,

**Свя́зный спи́сок** — базовая динамическая структура данных в информатике, состоящая из [узлов](https://ru.wikipedia.org/wiki/%D0%A3%D0%B7%D0%B5%D0%BB_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), каждый из которых содержит как собственно [данные](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), так и одну или две [ссылки](https://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) («связки») на следующий и/или предыдущий узел списка.

Найбільш типові операції для роботи зі списком:

* insert(p, x) - додавання х перед p;
* insert(p, n, x) - додавання n копій х перед p;
* insert(p, first, last) - додавання перед p елементів послідовності, заданої ітераторами first і last;
* erase(p) - видалення елементу в позиції p;
* erase(first, last) - видалення послідовності, заданої ітераторами;
* clear() - видалення всіх елементів.

1. Як у Стандартній бібліотеці представлені рядки символів?

**typedef** basic\_string<**char**> string;

Шаблон basic\_string - це спеціалізований вектор, оптимізований для зберігання рядків

1. Що таке адаптер послідовності?

Класи stack (стек), queue (черга) та priority\_queue (черга з пріоритетом) визначено не як окремі контейнери, а як адаптери базових контейнерів. Адаптери контейнерів надають обмежений інтерфейс до контейнера. Зокрема, вони не надають ітераторів. Стандартні контейнери зберігаються як елементи даних у класах-адаптерах в розділі **private**.

1. Чим черга відрізняється від стеку?
2. Що таке асоціативний масив?

*Асоціативний масив* (map, його іноді називають словником) зберігає пари значень. Маючи одне зі значень (ключ), можна отримати доступ до іншого (значення). Асоціативний масив можна представляти як масив, в якому індекс не обов'язково є цілим.

1. Чим відрізняється map від multimap?

Вони відрізняються тим, що в контейнері mapключі повинні бути унікальними, а в контейнері multimap - ні (ключі можуть повторюватися).

1. Де застосовують асоціативні масиви?

*Множину* можна розглядати як варіант асоціативного масиву, в якому присутні тільки ключі

Ассоциативные массивы, называемые также хэш-массивами или просто хэшами, позволяют задавать и хранить значения, с каждым из которых ассоциируется некоторая строка, называемая ключом. В дальнейшем обращение к сохраненному значению осуществляется с помощью его ключа. Хэш-массивы позволяют легко реализовывать динамические структуры данных (списки, деревья и т. д.), а также функциональность простейших систем управления базами данных. Концептуально хэш-массивы представляют собой неупорядоченное множество ассоциированных пар ключ/значение, что отражено в семантике их конструкторов.

1. Чим множина відрізняється від інших контейнерів?

Для множин не реалізована операція індексації

1. Які є стандартні класи для представлення множини?

Звичайна множина (тип set) може містити тільки унікальні елементи. Спеціальний тип multiset допускає повторення елементів множини. Для роботи з обома множинами слід підключати заголовний файл <set>. Під час визначення змінних типу множини необхідно вказувати тільки один тип - тип елементу множини, наприклад:

set<string> words;

1. Що таке алгоритм Стандартної бібліотеки?

*Алгоритм* (algorithm) - це шаблонна функція, оголошена в просторі імен std і в заголовному файлі <algorithm>, яка працює з елементами довільних послідовностей, заданих ітераторами і визначає обчислювальну процедуру.

1. Які є групи алгоритмів?

Існує кілька груп алгоритмів:

* алгоритми, які не модифікують послідовності;
* алгоритми, що модифікують послідовність;
* алгоритми сортування;
* алгоритми для роботи з множинами;
* алгоритми для роботи з динамічною пам'яттю (heap);
* алгоритми знаходження максимумів і мінімумів;
* алгоритми перестановок.

1. Як алгоритм пов'язаний з типом контейнеру, до якого він застосовується?

вони можуть працювати зі створеними користувачем структурами даних, якщо ці структури даних мають типи ітераторів, що задовольняють припущеннями в алгоритмах

1. Що таке функціональний об'єкт?

*Функціональний об'єкт* (функтор, functor) є екземпляром деякого класу, в якому за допомогою функції-елементу перевантажена операція "круглі дужки" (виклик функції)

1. Що таке функція-предикат? Які є стандартні функції-предикати?

**Предика́т** ([лат.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *praedicatum* — заявленное, упомянутое, сказанное)

*Предикат* - це функція (або функціональний об'єкт), що повертає значення типу **bool**

До цих предикатів відносяться унарний предикат logical\_not (який реалізує !) і бінарні предикати:

* equal\_to (==)
* not\_equal\_to (!=)
* greater (>)
* less (<)
* greater\_equal (>=)
* less\_equal (<=)
* logical\_and (&&)
* logical\_or (||).

Кожен такий предикат є шаблоном, які приймає як параметр тип значення, для якого здійснюється операція. У наведеному нижче прикладі використовується предикат logical\_not() в алгоритмі transform(). Цей алгоритм приймає дві послідовності - вхідну (визначену двома ітераторами) і результуючу (визначену одним ітератором):

vector<**int**> a = { 0, 1, 0, 2, 0, 3 };

vector<**int**> b(6); // результуюча послідовність

transform(a.begin(), a.end(), b.begin(), logical\_not<**int**>());

// b: 1 0 1 0 1 0

1. Що таке адаптери функцій?

Основные унарные и бинарные функциональные объекты, необходимые для сравнения, уже включены в STL и используются с добавлением хедера **functional**. Все они являются шаблонными классами и требуют определения необходимых операторов в типе данных, с которым работают. Примеры: std::greater<>, std::less<>.

 Иногда, нам может потребоваться реализовать, например, подсчет элементов исходя из нескольких условий, т.е. необходимо скомбинировать результаты с определенными значениями или другими функциями. В таких случаях применяют функциональные адаптеры. Необходимо отметить, что и сами адаптеры могут служить частью вычислений других адаптеров, за счет чего достигается гибкость вычислений. Основные адаптеры: bind1st, bind2nd, not1, not2.  
  
Пример: подсчитать количество элементов, больших (>), чем два.

int main()   
{   
    **const** std::size\_t N = 3;   
    int A[N] = {3, 2, 5};   
    std::cout << std::count\_if(A, A + N, std::bind2nd(std::greater<int>(), 2));   
}

Данный код выведет 2. Все верно, лишь элементы 3 и 5 превосходят 2.

1. Які нові можливості надає стандарт C++11?

Дванадцятого серпня 2011 року було затверджено новий офіційний стандарт мови C++. Нова стандартизована версія мови включає багато цікавих можливостей:

* Списки ініціалізації
* Автоматичне визначення типів
* Цикл for, побудований на діапазоні
* Лямбда-функції і вирази
* Альтернативний синтаксис функцій
* Можливість виклику конструкторів з інших конструкторів у списку ініціалізації
* Явне перевизначення віртуальних функцій (модифікатор **override** після заголовку перевизначеної функції)
* Константа для нульового вказівника **nullptr**
* Строго типізовані перечислення
* Локальні і безіменні типи як аргументи шаблонів
* Символи і рядки в Unicode
* "Сирі" рядки (Raw string literals)
* Створення можливості реалізації прибирання сміття

Є також додаткові синтаксичні нюанси і відмінності у внутрішній організації ядра мови.

1. Що таке список ініціалізації?

Традиційно списки ініціалізації використовують для ініціалізації масивів і структур

Розширення можливостей списків ініціалізації пов'язане з шаблонним класом std::initializer\_list. Об'єкти відповідного типу є неявними константами. Найчастіше цей тип використовують для опису параметру функції, зокрема, конструктору. Подібні конструктори додані до існуючих класів-колекцій. Наприклад:

vector<**int**> a({ 1, 2 });

vector<**int**> b = { 3, 4 };

1. Що таке автоматичне визначення типів?

Механізм автоматичного визначення типів дозволяє компілятору створювати локальні змінні, тип яких залежить від контексту. Для опису таких змінних застосовують ключове слово **auto**. Такі змінні обов'язково повинні бути ініціалізовані. Тип змінної компілятор визначає відповідно до типу виразу, яким ця змінна ініціалізується. Наприклад:

**auto** i = 10; // ціла змінна

**auto** s = "Hello"; // вказівник на символ

1. Що таке цикл for, побудований на діапазоні?

Для перебору елементів масиву (або іншої колекції) без застосування індексу нова версія C++ пропонує альтернативну конструкцію циклу **for** ("range-based for"):

**for** (тип\_елементу &змінна: масив\_або\_колекція)

тіло\_циклу

Всередині тіла циклу можна застосовувати змінну як поточний елемент масиву (колекції). Не можна використовувати індекс або обійти частину масиву. Наприклад, так можна вивести на екран усі елементи вектора:

vector<**int**> v = { 1, 2 };

**for** (**int** &k : v)

{

cout << k;

}

1. Які є способи використання лямбда-виразів?

Стандарт мови C++11 надає спеціальний синтаксис для анонімних функціональних об'єктів - так звані *лямбда-вирази* (lambda expressions)

Можна створити змінну, яка може бути використана для виклику:

auto f = [](int a, int b) { return a \* b; };

cout << f(2, 2); // 4

Лямбда-вирази можуть бути використані всюди, де потрібен вказівник на функцію. Наприклад:

int a[] = { 1, 2, 3 };

for\_each(a, a + 3, [](int x) { cout << x << endl; }); // виведення елементів

Можна також створити посилання на змінну. Так можна реалізувати приклад, пов'язаний зі зберіганням елементів вектора в текстовому файлі за допомогою алгоритму for\_each():

vector<**int**> a = { 1, 2, 3, 4 };

ofstream out("result.txt");

for\_each(a.begin(), a.end(), [&out](**const** **int**& i) { out << i << endl; });