МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «Програмна інженерія та   
інформаційні технології управління»

ЗВІТ

з лабораторної роботи № 4

з дисципліни

«ОСНОВИ ПРОГРАМУВАННЯ»

Виконала

Студентка групи КН-222В

Григор’єва Катерина Ігорівна

Перевірив

Асистент каф. ПІІТУ

Олексій КОНДРАТОВ

Харків 2023

**Тема:** Використання засобів Стандартної бібліотеки шаблонів в C++

**Мета:** Метою даної роботи є ознайомлення з основними засобами, які пропонує Стандартна бібліотека шаблонів (STL) в мові програмування C++. Досліджуючи STL, ми прагнемо зрозуміти переваги використання готових шаблонів і як вони можуть спростити процес розробки, підвищити швидкодію програми і забезпечити безпеку даних.

## **Завдання на лабораторну роботу**

### **1.1 Представлення й обробка даних про студентів з використанням засобів Стандартної бібліотеки шаблонів**

Виконати завдання 1.3 [другої лабораторної роботи](http://www.iwanoff.inf.ua/programming_2_ua/LabTraining02.html#Tasks) (Класи для представлення студента і групи).

У класі "Студент" для зберігання прізвища застосувати рядок типу std::string. Оцінки за останню сесію зберігати у векторі цілих чисел (std::vector<**int**>). Окрім функцій, перелічених в завданні 1.3 другої лабораторної роботи, слід реалізувати функції, які здійснюють:

* обчислення показника, за величиною якого здійснюється сортування відповідно до індивідуального завдання;
* перевірку умови, яка використовується для пошуку даних відповідно до індивідуального завдання.

У класі "Група студентів" розташувати вектор об'єктів класу, який представляє студента. Реалізувати функції, визначені в завданні 1.3 другої лабораторної роботи для цього класу. Для сортування масиву за ознакою, яка наведена в індивідуальному завданні скористатися алгоритмом sort(). Для пошуку даних про студентів, які відповідають умові, наведеній в індивідуальному завданні застосувати алгоритм for\_each().

Додатково розмістити об'єкти класу "Студент" в черзі з пріоритетом, з якої вилучати об'єкти за зменшенням середнього балу.

### **1.2 Вектор векторів для представлення двовимірного масиву**

Розв'язати завдання 1.4 [другої лабораторної роботи](http://www.iwanoff.inf.ua/programming_2_ua/LabTraining02.html#Tasks) (Клас для представлення двовимірного масиву), створивши клас, полем якого є вектор векторів Стандартної бібліотеки. Створений клас не вимагає наявності конструктора копіювання, деструктора і перевантаженої операції присвоєння, тому що не відбувається розміщення даних у динамічній пам'яті (за це відповідають вектори).

### **1.3 Підрахунок кількості повторень значень (додаткове завдання)**

Розробити програму, в якій зчитуються цілі значення і підраховується число повторень кожного значення, за винятком чисел, зазначених у списку. Підрахунок числа повторень числа реалізовувати за допомогою асоціативного масиву, список чисел для виключення реалізовувати за допомогою множини. Кожен раз, коли зустрілося число зі списку, виводити повідомлення.

2 Завдання

Перше завдання

Код до першого завдання

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <queue>

#include <queue>

using std::cout;

using std::endl;

using std::priority\_queue;

using std::string;

using std::vector;

const int MAX\_COUNT = 50; // Максимальна кількість студентів у группі

// Треба заздалегідь оголосити клас, щоб можна було створювати вказівник:

class Group;

// Клас для представлення міста

class Student

{

// Перевантажений оператор для виведення в потік

friend std::ostream &operator<<(std::ostream &out, const Student &student);

private:

unsigned int IDNumber; // Номер студентського посвідчення

string surname; // прізвище, яке виділено у дінамічній пам'яті

vector<int> myArray;

Group \*group = nullptr; // вказівник на групу

public:

// Конструктори:

Student() {}

Student(unsigned int IDNumber, const string surname, const vector<int> &myArray, Group \*group);

Student(const Student &student);

// Геттери:

unsigned int getIDNumber() const { return IDNumber; }

const string &getSurname() const { return surname; }

const vector<int> getArray() const

{

return myArray;

}

Group \*getGroup() const { return group; }

// Сеттери:

void setIDNumber(unsigned int number) { IDNumber = number; }

void setSurname(const string &surname)

{

if (!this->surname.empty()) // перевірка наявності попереднього значення прізвища

{

this->surname.clear(); // звільнення пам'яті, виділеної для попереднього значення

}

this->surname = surname; // присваивание нового значения фамилии

}

void setMyArray(const vector<int> &array)

{

myArray = array;

}

void setGroup(Group \*group) { this->group = group; }

// Перевантажена операція присвоєння

const Student &operator=(const Student &student);

double calculateIndicator() const

{

double sum = 0;

for (int i = 0; i < myArray.size(); i++)

{

sum += myArray[i];

}

return sum / myArray.size(); // Средняя оценка студента

}

};

// Конструктор з параметрами, реалізований через виклик сеттерів

Student::Student(unsigned int IDNumber, const string surname, const vector<int> &myArray, Group \*group)

{

setIDNumber(IDNumber);

setSurname(surname);

setMyArray(myArray);

setGroup(group);

}

// Конструктор копіювання

Student::Student(const Student &student)

{

IDNumber = student.IDNumber;

surname = student.surname;

myArray = student.myArray;

group = student.group;

}

class Group

{

// Перевантажений оператор для виведення в потік

friend std::ostream &operator<<(std::ostream &out, const Group &group)

{

out << "Group Index: " << group.groupIndex << "\n\n";

for (int i = 0; i < group.count; i++)

{

out << \*(group.student[i]) << endl;

}

out << endl;

return out;

}

private:

vector<Student \*> student;

unsigned int groupIndex; // індекс групи

int count = 0; // кількість вказівників у масиві

public:

// Конструктори:

Group() {}

Group(unsigned int groupIndex) { setGroupIndex(groupIndex); }

// Геттер

const char getGroupIndex() const { return groupIndex; }

// Сеттери:

void setGroupIndex(unsigned int Index) { groupIndex = Index; }

void setGroups(Student \*student[], int count);

// Перевантажена операція присвоєння

const Group &operator=(const Group &group);

// Вивід за ознакою

void printIf();

// перевантаження операції отримання елемента за індексом

Student &operator[](int index)

{

return \*student[index];

}

static bool sortByIndicator(const Student \*s1, const Student \*s2)

{

return s1->calculateIndicator() < s2->calculateIndicator();

}

void sortStudentsByIndicator()

{

sort(student.begin(), student.end(), sortByIndicator);

}

~Group()

{

for (const auto &student : student)

{

delete student;

}

}

};

// Отримаємо з параметру й заповнюємо масив студентів

void Group::setGroups(Student \*student[], int count)

{

this->count = count;

for (int i = 0; i < count; i++)

{

this->student.push\_back(student[i]);

this->student[i]->setGroup(this); // this позволяет получить доступ к текущему объекту класса Group.

// Затем вызывается метод setGroup() [который находитя у объекта класса Student],

// передавая ему указатель на текущий объект класса Group, чтобы установить связь между объектами Group и Students

}

}

std::ostream &operator<<(std::ostream &out, const Student &student)

{

out << "ID: " << student.IDNumber << "\n"; // out это поток

out << "Surname: " << student.surname << "\n";

out << "Array: ";

for (const auto &value : student.myArray)

{

out << value << " ";

}

out << "\n";

return out;

}

void Group::printIf()

{

for\_each(

student.begin(), student.end(), [this](const Student \*s) // this для доступа к переменной student

{

if ((s->getIDNumber()) % 2 != 0)

{

cout << \*s << endl;

} });

}

int main()

{

setlocale(LC\_ALL, "UKRAINIAN");

const int realCount = 4; // працюємо з чотирма студентами

Student \*student[realCount]; // створюємо масив вказівників на студентів

// заповнюємо масив

vector<int> tempArray = {100, 100, 100, 100, 100, 100, 100, 100};

student[0] = new Student(1421125, "Іванів", tempArray, nullptr);

student[1] = new Student(284942, "Кучма", tempArray, nullptr);

student[2] = new Student(54618, "Константінов", tempArray, nullptr);

student[3] = new Student(264753, "Григор'єва", tempArray, nullptr);

Group group = 211432;

group.setGroups(student, realCount);

// Создаем очередь с приоритетом для хранения студентов

priority\_queue<Student \*, vector<Student \*>, decltype(&Group::sortByIndicator)> studentPriorityQueue(&group.sortByIndicator);

// Добавляем студентов в очередь с приоритетом

for (int i = 0; i < realCount; i++)

{

studentPriorityQueue.push(student[i]);

}

// Выводим данные группы перед сортировкой

cout << "Display the group data: \n"

<< group << "\n\n";

// Выводим информацию о студенте по индексу

cout << "Display information about the student by index: \n"

<< \*student[0] << "\n\n";

// Сортируем студентов в группе по среднему баллу

group.sortStudentsByIndicator();

// Выводим данные группы после сортировки

cout << "Display all data after sorting\n"

<< group << "\n\n";

// Выводим студентов с нечетными номерами студенческого билета

cout << "Remove students with odd student ID numbers:"

<< "\n\n";

group.printIf();

// Выводим студентов из очереди с приоритетом в порядке убывания среднего балла

cout << "Display all data from priority queue:"

<< "\n\n";

while (!studentPriorityQueue.empty())

{

cout << \*studentPriorityQueue.top() << endl;

studentPriorityQueue.pop();

}

return 0;

}

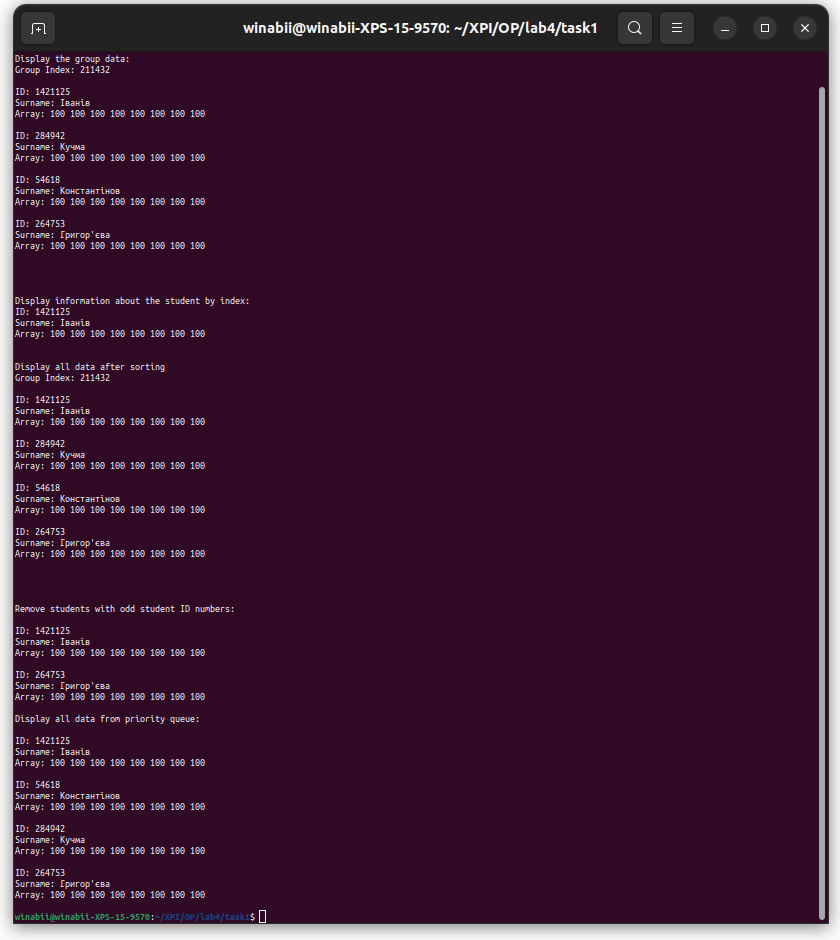


Рис 1 – Приклад роботи першої програми

Друге завдання

Код до другого завдання

#include <iostream>

#include <math.h>

#include <vector>

#include <algorithm>

using std::cin;

using std::cout;

using std::endl;

using std::for\_each;

using std::istream;

using std::ostream;

using std::transform;

using std::vector;

class UnsignedIntArray

{

friend ostream &operator<<(ostream &out, const UnsignedIntArray &a);

friend istream &operator>>(istream &in, UnsignedIntArray &a);

friend vector<vector<int>> operator+(const UnsignedIntArray &a, const UnsignedIntArray &b);

friend vector<vector<int>> operator-(const UnsignedIntArray &a, const UnsignedIntArray &b);

friend vector<vector<int>> operator\*(const UnsignedIntArray &a, const UnsignedIntArray &b);

private:

vector<vector<int>> data;

int rows = 0;

int cols = 0;

public:

class OutOfBounds

{

int index;

public:

OutOfBounds(int i) : index(i) {}

int getIndex() const { return index; }

};

UnsignedIntArray() {}

UnsignedIntArray(int r, int c)

{

if (r <= 0)

{

throw OutOfBounds(r);

}

if (c <= 0)

{

throw OutOfBounds(c);

}

rows = r;

cols = c;

data.resize(rows, vector<int>(cols, 0));

}

UnsignedIntArray(const UnsignedIntArray &arr);

vector<int> &operator[](int index)

{

if (index < 0 || index >= rows)

{

throw OutOfBounds(index);

}

return data[index];

}

int getRows() const { return rows; }

int getCols() const { return cols; }

};

UnsignedIntArray::UnsignedIntArray(const UnsignedIntArray &arr)

{

rows = arr.rows;

cols = arr.cols;

data = arr.data;

}

vector<vector<int>> operator+(const UnsignedIntArray &a, const UnsignedIntArray &b)

{

if (a.getRows() != b.getRows() || a.getCols() != b.getCols())

{

throw std::invalid\_argument("Matrices must have the same size");

}

int n = a.getRows();

vector<vector<int>> result(n);

for (int i = 0; i < n; i++)

{

result[i].resize(a.getCols());

transform(a.data[i].begin(), a.data[i].end(), b.data[i].begin(), result[i].begin(), std::plus<int>());

}

return result;

}

vector<vector<int>> operator-(const UnsignedIntArray &a, const UnsignedIntArray &b)

{

if (a.getRows() != b.getRows() || a.getCols() != b.getCols())

{

throw std::invalid\_argument("Matrices must have the same size");

}

int n = a.getRows();

vector<vector<int>> result(n);

for (int i = 0; i < n; i++)

{

result[i].resize(a.getCols());

transform(a.data[i].begin(), a.data[i].end(), b.data[i].begin(), result[i].begin(), std::minus<int>());

}

return result;

}

vector<vector<int>> operator\*(const UnsignedIntArray &a, const UnsignedIntArray &b)

{

if (a.getCols() != b.getRows())

{

throw std::invalid\_argument("Matrix dimensions are not compatible for multiplication");

}

int m = a.getRows();

int n = a.getCols();

int p = b.getCols();

vector<vector<int>> result(m, vector<int>(p, 0));

for (int i = 0; i < m; i++)

{

for (int j = 0; j < p; j++)

{

for (int k = 0; k < n; k++)

{

result[i][j] += a.data[i][k] \* b.data[k][j];

}

}

}

return result;

}

ostream &operator<<(ostream &out, const UnsignedIntArray &a)

{

out << "\nArray:" << endl;

for\_each(a.data.begin(), a.data.end(), [&](const vector<int> &row)

{

for\_each(row.begin(), row.end(), [&](int element) { out << element << " "; });

out << endl; });

return out;

}

istream &operator>>(istream &in, UnsignedIntArray &a)

{

cout << "Please enter the elements of the array, row by row:" << endl;

for (int i = 0; i < a.getRows(); i++)

{

cout << "Row " << i + 1 << ": ";

for (int j = 0; j < a.getCols(); j++)

{

in >> a.data[i][j];

cout << a.data[i][j] << " ";

}

cout << endl;

}

return in;

}

void getLog10(UnsignedIntArray &a)

{

for (int i = 0; i < a.getRows(); i++)

{

for (int j = 0; j < a.getCols(); j++)

{

if (a[i][j] > 0)

{

a[i][j] = log10(a[i][j]);

}

}

}

}

int main()

{

UnsignedIntArray a(2, 2);

cin >> a;

cout << "Matrix a:" << a << endl;

UnsignedIntArray b(2, 2);

cin >> b;

cout << "Matrix b:" << b << endl;

try

{

vector<vector<int>> cplus = a + b;

cout << "Matrix a + b:" << endl;

for\_each(cplus.begin(), cplus.end(), [&](const vector<int> &row)

{

for\_each(row.begin(), row.end(), [&](int element) { cout << element << " "; });

cout << endl; });

vector<vector<int>> cminus = a - b;

cout << "Matrix a - b:" << endl;

for\_each(cminus.begin(), cminus.end(), [&](const vector<int> &row)

{

for\_each(row.begin(), row.end(), [&](int element) { cout << element << " "; });

cout << endl; });

vector<vector<int>> cmult = a \* b;

cout << "Matrix a \* b:" << endl;

for\_each(cmult.begin(), cmult.end(), [&](const vector<int> &row)

{

for\_each(row.begin(), row.end(), [&](int element) { cout << element << " "; });

cout << endl; });

}

catch (const std::invalid\_argument &e)

{

cout << "Error: " << e.what() << endl;

}

getLog10(a);

cout << "Matrix a after applying log10: " << a << endl;

return 0;

}

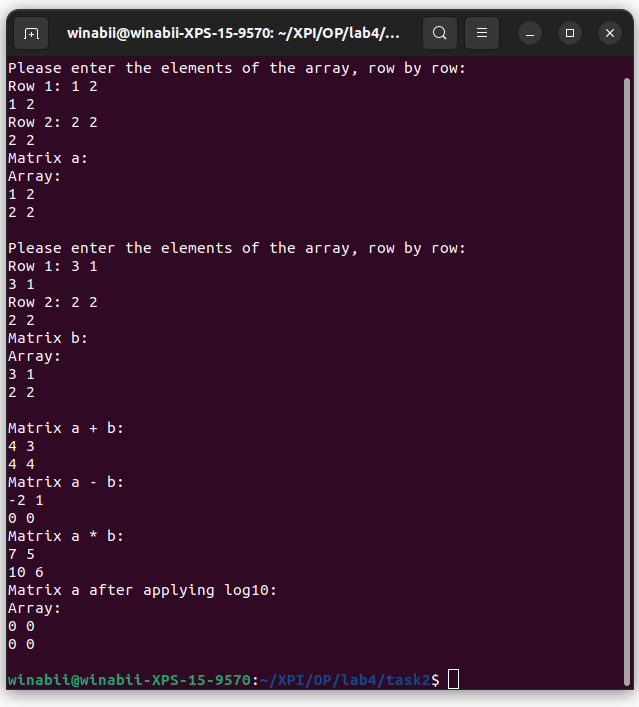


Рис 2 – Приклад роботи другої програми

Третє завдання

Код до третього завдання

#include <iostream>

#include <fstream>

#include <map>

#include <set>

using std::cout;

using std::endl;

using std::ifstream;

using std::map;

using std::set;

int main()

{

map<int, int> m;

set<int> list = {1, 2, 3, 4};

int i;

{

ifstream in("values.txt");

while (in >> i)

{

if (list.count(i) != 0) // Исправлено: проверяем наличие i в множестве list

{

cout << i << " - this number from the list" << endl;

// Исправлено: устанавливаем значение 1, так как это первое вхождение числа i

}

else

{

m[i]++; // Исправлено: увеличиваем счетчик для числа i

}

}

}

map<int, int>::iterator mi;

for (mi = m.begin(); mi != m.end(); mi++)

{

cout << mi->first << " " << mi->second << endl;

}

return 0;

}

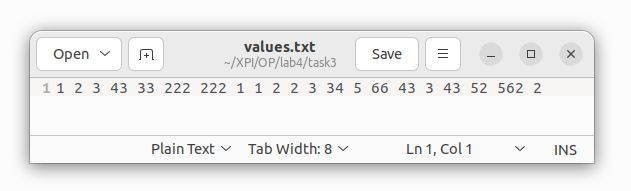


Рис 3 – Файл values.txt

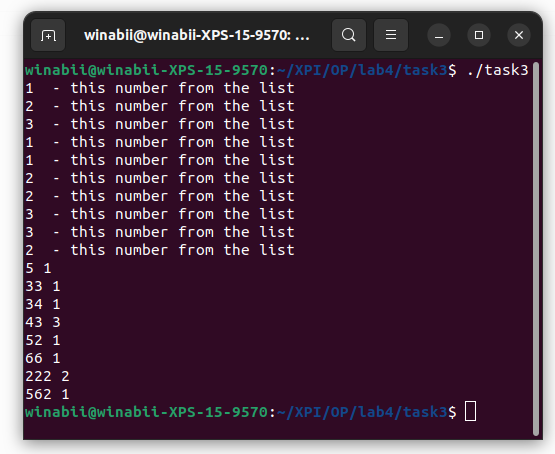


Рис 4 – Приклад роботи третьої програми

**GitHub з завданнями:** https://github.com/winabi/OP/tree/main/lab3

**Контрольні запитання**

1. Які основні елементи включає Стандартна бібліотека C++?
2. Що таке контейнер Стандартної бібліотеки?
3. Чим відрізняються послідовні й асоціативні контейнери?
4. Що таке ітератор?
5. Які вимоги пред'являються до ітераторів Стандартної бібліотеки?
6. У чому переваги вектора у порівнянні з масивом?
7. Які є способи створення вектора?
8. Як працювати з окремими елементами векторів?
9. Чим список відрізняється від масиву?
10. Як у Стандартній бібліотеці представлені рядки символів?
11. Що таке адаптер послідовності?
12. Чим черга відрізняється від стека?
13. Що таке асоціативний масив?
14. Чим відрізняється map від multimap?
15. Де застосовують асоціативні масиви?
16. Чим множина відрізняється від інших контейнерів?
17. Які є стандартні класи для представлення множини?
18. Що таке алгоритм Стандартної бібліотеки?
19. Які є групи алгоритмів?
20. Як алгоритм пов'язаний з типом контейнера, до якого він застосовується?
21. Що таке функціональний об'єкт?
22. Що таке функція-предикат? Які є стандартні функції-предикати?
23. Що таке адаптери функцій?

**Відповіді**

1. Основні елементи, що включає Стандартна бібліотека C++, включають контейнери, алгоритми, ітератори, функції обробки рядків, потоки вводу/виводу, функції для роботи з файлами та інші корисні класи та функції.
2. Контейнер Стандартної бібліотеки - це структура даних, яка забезпечує зберігання та організацію даних в пам'яті. Він може бути використаний для зберігання колекцій об'єктів і надає різні методи доступу та операції над елементами.
3. Послідовні контейнери зберігають елементи в послідовному порядку і дозволяють повторення елементів. Асоціативні контейнери зберігають елементи у відсортованому порядку і не допускають повторень елементів.
4. Ітератор - це об'єкт, який дозволяє ітерувати (перебирати) по контейнеру та отримувати доступ до його елементів. Ітератори можуть бути використані для читання та зміни елементів контейнера.
5. До ітераторів Стандартної бібліотеки ставляться певні вимоги. Вони повинні підтримувати оператори зсуву, доступу до значень, порівняння, а також оператори присвоєння та копіювання.
6. Переваги вектора у порівнянні з масивом полягають у можливості динамічного змінення розміру, автоматичному керуванні пам'яттю, зручному доступі до елементів та широких можливостях зберігання та маніпулювання даними.
7. Створення вектора може здійснюватись за допомогою конструктора за замовчуванням, конструктора з кількістю елементів та початковим значенням, або шляхом копіювання.
8. Для роботи з окремими елементами векторів використовуються індекси або ітератори. За допомогою індексів можна звертатися до елементів за їхніми позиціями в векторі. Наприклад, вектор "myVector" з доступом до елемента з індексом "i" буде мати синтаксис "myVector[i]". Крім того, ітератори дозволяють перебирати елементи вектора за допомогою циклу і виконувати різні операції з ними.
9. Список відрізняється від масиву тим, що він є динамічно змінюваним контейнером. Масив має фіксований розмір, визначений при його створенні, тоді як список може динамічно змінюватись, додавати або видаляти елементи у будь-якій частині списку.
10. У Стандартній бібліотеці C++ рядки символів представлені класом "std::string". Він надає багато функцій та методів для роботи з рядками, включаючи зчитування, запис, об'єднання, порівняння та модифікацію рядків.
11. Адаптер послідовності - це клас або функція, які змінюють інтерфейс або поведінку існуючого контейнера або послідовності. Наприклад, "std::stack" є адаптером послідовності, який забезпечує роботу зі стеком на базі іншого контейнера, наприклад, вектора чи списку.
12. Черга і стек - це два різних типи контейнерів. У черзі елементи видаляються з одного кінця і додаються в інший (FIFO - першим вийшов, першим вийшов), тоді як у стеку елементи додаються та видаляються з одного й того самого кінця (LIFO - останнім вийшов, першим вийшов).
13. Асоціативний масив (associative array) - це контейнер, що забезпечує зв'язку між ключами та значеннями. Кожен ключ унікальний і використовується для доступу до відповідного значення. В Стандартній бібліотеці C++ асоціативні масиви представлені класами "std::map" і "std::multimap".
14. Відмінності між "std::map" і "std::multimap" полягають у тому, що "std::map" допускає лише унікальні ключі, тобто кожен ключ мапується в єдине значення, тоді як "std::multimap" дозволяє повторювані ключі, тобто один ключ може мати декілька значень.
15. Асоціативні масиви застосовуються там, де необхідно зберігати пари ключ-значення та швидко отримувати доступ до значення за допомогою ключа. Вони корисні для реалізації різних структур даних, таких як словники, індексовані таблиці, кеші, бази даних та багато інших випадків.
16. Множина - це контейнер, що зберігає унікальні значення без залежності від їх порядку. У множини немає повторюваних елементів, і вони автоматично сортуються за певним порядком (за замовчуванням - від наїменшого до найбільшого). Множини використовуються для зберігання та роботи з унікальними значеннями та для виконання операцій над ними, таких як перетин, об'єднання та різниця.
17. Стандартні класи для представлення множини в Стандартній бібліотеці C++ включають "std::set" та "std::unordered\_set". "std::set" зберігає унікальні значення в відсортованому порядку, тоді як "std::unordered\_set" зберігає унікальні значення без впорядкування.
18. Алгоритм Стандартної бібліотеки - це функція, яка виконує певну операцію на послідовності елементів. Вона може застосовуватись до різних типів контейнерів та послідовностей даних.
19. Групи алгоритмів включають алгоритми сортування, пошуку, маніпулювання послідовностями, роботи зі стеками та чергами, операції над множинами, обробку рядків, числові алгоритми та інші.
20. Алгоритм пов'язаний з типом контейнера, до якого він застосовується, оскільки кожен алгоритм може працювати з певними операціями, які контейнер надає. Наприклад, алгоритм сортування може вимагати можливості порівнювання елементів контейнера, тому він може бути застосований тільки до контейнерів, які підтримують таку операцію.
21. Функціональний об'єкт (function object) - це об'єкт, який може бути викликаний як функція. Він може мати оператор виклику (operator()) перевантажений, що дозволяє використовувати його як функцію.
22. Функція-предикат - це функція, яка повертає булеве значення (true або false) залежно від заданої умови. Стандартні функції-предикати включають "std::less", "std::greater", "std::equal\_to" та інші, які використовуються для порівняння значень.
23. Адаптери функцій (function adapters) - це функціональні об'єкти або функції, які модифікують поведінку інших функцій або функціональних об'єктів. Вони дозволяють змінювати інтерфейс або впливати на спосіб виконання функцій.

**Висновок:** Після успішного виконання лабораторної роботи, в якій було використано засоби Стандартної бібліотеки шаблонів (STL) в мові програмування C++, отримано цінний практичний досвід. Використання STL дозволило ефективно використовувати готові шаблони, спрощуючи процес розробки і забезпечуючи високу швидкодію програм.

Під час роботи було з'ясовано, що STL надає широкий набір контейнерів, алгоритмів і ітераторів, що дозволяють зручно працювати зі зберіганням і обробкою даних. Використання готових компонентів STL полегшує розробку програм і сприяє підвищенню продуктивності, оскільки зменшує необхідність у написанні власних алгоритмів та контейнерів.

Крім того, використання шаблонів у C++ дозволяє створювати універсальні функції та класи, які можна повторно використовувати в різних програмних проектах. Це сприяє зменшенню повторення коду і забезпечує більшу гнучкість та легкість змін у програмі.

В цілому, використання засобів STL та шаблонів виявилося важливим для покращення ефективності та продуктивності програмного процесу. Отриманий досвід дозволить успішно застосовувати ці знання у подальших програмних проектах і прискорить їх розробку.