Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

ОТЧЕТ

к лабораторной работе №4

**«Ассоциативные классы-контейнеры и классы-итераторы библиотеки STL»**

|  |  |
| --- | --- |
|  | |
|  |
|  |
|  |
|  |  | | |
|  | | |  | |  |
|  | | |  | |  |
| Студент гр. 324402 | | |  | | Цевелюк А.И. |
| Проверила | | |  | | Купрейчик А.С. |

Минск 2024

**1 Цель работы**

Цель работы:изучить ассоциативные классы-контейнеры и классы-итераторы стандартной библиотеки шаблонов C++ (STL). Овладеть навыками работы с различными типами ассоциативных контейнеров, такими как map, set, multimap, multiset, а также их неупорядоченными аналогами. Научиться применять итераторы для работы с элементами этих контейнеров. Реализовать базовые операции с контейнерами, такие как добавление, удаление, редактирование, поиск, сортировка и фильтрация данных. Закрепить навыки создания и работы с шаблонными классами STL, обеспечивающими эффективное хранение и доступ к данным по ключу. Освоить методы и приёмы работы с контейнерами, использующими ключи для быстрого доступа к данным, и изучить особенности реализации итераторов для этих контейнеров.

**2 Теоретические сведения по лабораторной работе**

Ассоциативные контейнеры в библиотеке STL предназначены для хранения элементов в структуре, обеспечивающей быстрый доступ по ключу. В отличие от последовательных контейнеров, где элементы хранятся в порядке их добавления, ассоциативные контейнеры используют ключи для упорядочивания или доступа к элементам. Основное преимущество этих контейнеров — высокая скорость поиска, вставки и удаления элементов. Существует несколько типов ассоциативных контейнеров:

- map — ассоциативный контейнер, представляющий собой отображение (ассоциативный массив), где каждому уникальному ключу сопоставляется одно значение. Ключи в map хранятся в отсортированном порядке.

- multimap — похож на map, но допускает несколько одинаковых ключей. Это полезно, когда требуется сохранить несколько значений для одного ключа.

- set — контейнер, хранящий уникальные ключи без ассоциированных значений. Все ключи хранятся в отсортированном порядке.

- multiset — похож на set, но допускает хранение дубликатов ключей.

- unordered\_map — неупорядоченная версия контейнера map, использующая хеш-функцию для доступа к элементам. Время поиска фиксированное, но элементы не упорядочены.

- unordered\_multimap — аналог unordered\_map, но допускает повторяющиеся ключи.

- unordered\_set — контейнер, хранящий уникальные ключи без упорядочивания, с использованием хеш-функции.

- unordered\_multiset — контейнер, похожий на unordered\_set, но допускающий дублирующиеся ключи.

Ассоциативные контейнеры поддерживают следующие операции:

- Добавление элемента — функция insert() добавляет элемент в контейнер. Для контейнеров map и set добавление элемента происходит за логарифмическое время.

- Удаление элемента — функция erase() удаляет элемент или диапазон элементов.

- Поиск элемента — функция find() возвращает итератор на элемент с указанным ключом или на конец контейнера, если ключ не найден.

- Получение диапазона — функции lower\_bound() и upper\_bound() возвращают итераторы на элементы, больше или равные (или строго больше) заданного ключа.

- Получение пар ключей — функция equal\_range() возвращает пару итераторов, указывающих на границы диапазона элементов с заданным ключом (актуально для мультимножеств).

Особенности контейнеров

Ассоциативные контейнеры map и set требуют уникальных ключей, тогда как их аналоги multimap и multiset допускают дублирование ключей.

Упорядоченные контейнеры (map, set, multimap, multiset) поддерживают хранение элементов в отсортированном порядке по ключу.

Неупорядоченные контейнеры (unordered\_map, unordered\_set, unordered\_multimap, unordered\_multiset) используют хеш-функции для организации своих элементов, что позволяет ускорить доступ.

Итераторы — это объекты, которые обеспечивают доступ к элементам контейнеров. В STL они работают аналогично указателям и поддерживают такие операции, как разыменование, инкремент и декремент. Итераторы бывают нескольких типов, в зависимости от контейнера, к которому они применяются:

- Прямые итераторы (iterator) используются для прохода по элементам в прямом порядке.

- Константные итераторы (const\_iterator) предназначены для доступа к элементам контейнера, но не позволяют их изменять.

Обратные итераторы (reverse\_iterator) позволяют обходить контейнер в обратном порядке.

- Константные обратные итераторы (const\_reverse\_iterator) позволяют проходить элементы в обратном порядке, но только для чтения.

Основные операции с итераторами:

- begin() — возвращает итератор на первый элемент контейнера.

- end() — возвращает итератор, указывающий за последний элемент контейнера.

- rbegin() и rend() — возвращают итераторы для обхода контейнера в обратном порядке.

- find() — возвращает итератор на элемент с указанным ключом, если такой элемент существует.

Контейнер map использует пары ключ-значение, представленные объектами класса pair. Класс pair объединяет два значения (ключ и значение) в единый объект. Пары можно создавать либо напрямую, используя конструкторы, либо с помощью функции make\_pair()..

**3 Формулировка индивидуального задания**

Во всех вариантах индивидуальных заданий необходимо реализовать работу с контейнерами map, set, multimap, multiset, unordered\_map, unordered\_set, unordered\_multimap, unordered\_multiset. Уметь обосновать использование каждого типа контейнера.

В каждом индивидуальном задании необходимо создать контейнеры, которые будут хранить объекты классов по предметной области, указанной в таблице ниже (класс должен содержать функционал по предметной области (добавление, удаление, редактирование, поиск, фильтр данных)). Для контейнера реализовать добавление, удаление, редактирование, вывод содержимого контейнера на экран и в файл, поиск и сортировку элементов. Необходимо создать удобное пользовательское меню для тестирования всех реализованных функций.

**4 Код решения индивидуального задания**

// ReSharper disable CppClangTidyConcurrencyMtUnsafe

#include <iostream>

#include <map>

#include <set>

#include <unordered\_map>

#include <unordered\_set>

#include <string>

#include <fstream>

#include <algorithm>

#include <vector>

class employee\_lab4

{

int id\_;

std::string name\_;

std::string position\_;

std::string department\_;

int experience\_;

double salary\_;

public:

employee\_lab4() = default;

employee\_lab4(const int id, std::string name, std::string position, std::string department, const int experience, const double salary)

: id\_(id), name\_(std::move(name)), position\_(std::move(position)), department\_(std::move(department)), experience\_(experience), salary\_(salary) {}

int get\_id() const { return id\_; }

std::string get\_name() const { return name\_; }

std::string get\_position() const { return position\_; }

std::string get\_department() const { return department\_; }

int get\_experience() const { return experience\_; }

double get\_salary() const { return salary\_; }

void set\_name(const std::string& name) { name\_ = name; }

void set\_position(const std::string& position) { position\_ = position; }

void set\_department(const std::string& department) { department\_ = department; }

void set\_experience(const int experience) { experience\_ = experience; }

void set\_salary(const double salary) { salary\_ = salary; }

bool operator<(const employee\_lab4& other) const

{

return id\_ < other.id\_;

}

bool operator==(const employee\_lab4& other) const

{

return id\_ == other.id\_;

}

void display() const

{

std::cout << "ID: " << id\_ << ", Name: " << name\_

<< ", Position: " << position\_

<< ", Department: " << department\_

<< ", Experience: " << experience\_

<< " years, Salary: $" << salary\_ << '\n';

}

struct hash\_function

{

size\_t operator()(const employee\_lab4& emp) const

{

return std::hash<int>()(emp.get\_id());

}

};

};

employee\_lab4 createemployee\_lab4()

{

int id, experience;

std::string name, position, department;

double salary;

std::cout << "Enter Employee ID: "; std::cin >> id;

std::cout << "Enter Name: "; std::cin.ignore(); std::getline(std::cin, name);

std::cout << "Enter Position: "; std::getline(std::cin, position);

std::cout << "Enter Department: "; std::getline(std::cin, department);

std::cout << "Enter Experience (years): "; std::cin >> experience;

std::cout << "Enter Salary: "; std::cin >> salary;

return employee\_lab4{ id, name, position, department, experience, salary };

}

template<typename Container>

void display\_all(const Container& container)

{

for (const auto& entry : container)

entry.display();

}

template<typename Container>

void display\_all\_with\_key(const Container& container)

{

for (const auto& entry : container)

entry.second.display();

}

template<typename Container>

void save\_to\_file(const std::string& filename, const Container& container)

{

std::ofstream file(filename);

if (file.is\_open())

{

for (const auto& entry : container)

file << entry.get\_id() << "," << entry.get\_name() << "," << entry.get\_position() << ","

<< entry.get\_department() << "," << entry.get\_experience() << "," << entry.get\_salary() << '\n';

std::cout << "Data saved to file: " << filename << '\n';

}

else

std::cerr << "Error opening file!\n";

}

template<typename Container>

void save\_to\_file\_with\_key(const std::string& filename, const Container& container)

{

std::ofstream file(filename);

if (file.is\_open())

{

for (const auto& entry : container)

file << entry.second.get\_id() << "," << entry.second.get\_name() << "," << entry.second.get\_position() << ","

<< entry.second.get\_department() << "," << entry.second.get\_experience() << "," << entry.second.get\_salary() << '\n';

std::cout << "Data saved to file: " << filename << '\n';

}

else

std::cerr << "Error opening file!\n";

}

template<typename Container>

void find\_employee\_by\_id(const Container& container, int id)

{

auto it = container.find(id);

if (it != container.end())

it->second.display();

else

std::cout << "Employee with ID " << id << " not found.\n";

}

template<typename Container>

void find\_employee\_in\_unordered(const Container& container, int id)

{

for (const auto& entry : container)

{

if (entry.get\_id() == id)

{

entry.display();

return;

}

}

std::cout << "Employee with ID " << id << " not found.\n";

}

template<typename Container>

void edit\_employee(Container& container, int id)

{

auto it = container.find(id);

if (it != container.end())

{

std::string name, position, department;

int experience;

double salary;

std::cout << "Enter new Name: "; std::cin.ignore(); std::getline(std::cin, name);

std::cout << "Enter new Position: "; std::getline(std::cin, position);

std::cout << "Enter new Department: "; std::getline(std::cin, department);

std::cout << "Enter new Experience (years): "; std::cin >> experience;

std::cout << "Enter new Salary: "; std::cin >> salary;

it->second.set\_name(name);

it->second.set\_position(position);

it->second.set\_department(department);

it->second.set\_experience(experience);

it->second.set\_salary(salary);

std::cout << "Employee updated.\n";

}

else

std::cout << "Employee not found.\n";

}

template<typename Container>

void delete\_employee(Container& container, int id)

{

auto it = container.find(id);

if (it != container.end())

{

container.erase(it);

std::cout << "Employee deleted.\n";

}

else

std::cout << "Employee not found.\n";

}

template<typename Container>

void delete\_employee\_in\_unordered(Container& container, int id)

{

for (auto it = container.begin(); it != container.end(); ++it)

{

if (it->get\_id() == id)

{

container.erase(it);

std::cout << "Employee deleted.\n";

return;

}

}

std::cout << "Employee not found.\n";

}

template<typename Container>

void sort\_by\_salary(Container& container)

{

std::vector<employee\_lab4> employees(container.begin(), container.end());

std::sort(employees.begin(), employees.end(), [](const employee\_lab4& a, const employee\_lab4& b)

{

return a.get\_salary() < b.get\_salary();

});

for (const auto& emp : employees)

emp.display();

}

void end()

{

system("pause");

system("cls");

}

int main()

{

std::map<int, employee\_lab4> employee\_map;

std::set<employee\_lab4> employee\_set;

std::multimap<std::string, employee\_lab4> employee\_multi\_map;

std::multiset<employee\_lab4> employee\_multi\_set;

std::unordered\_map<int, employee\_lab4> employee\_unordered\_map;

std::unordered\_set<employee\_lab4, employee\_lab4::hash\_function> employee\_unordered\_set;

std::unordered\_multimap<std::string, employee\_lab4> employee\_unordered\_multi\_map;

std::unordered\_multiset<employee\_lab4, employee\_lab4::hash\_function> employee\_unordered\_multi\_set;

int choice = 0;

do

{

if (choice) end();

std::cout << "\nMenu:\n";

std::cout << "1. Add Employee\n";

std::cout << "2. Display All Employees\n";

std::cout << "3. Save to File\n";

std::cout << "4. Find Employee by ID\n";

std::cout << "5. Edit Employee by ID\n";

std::cout << "6. Delete Employee by ID\n";

std::cout << "7. Sort Employees by Salary\n";

std::cout << "0. Exit\n";

std::cout << "Enter choice: ";

std::cin >> choice;

switch (choice)

{

case 1:

{

employee\_lab4 emp = createemployee\_lab4();

employee\_map[emp.get\_id()] = emp;

employee\_set.insert(emp);

employee\_multi\_map.emplace(emp.get\_department(), emp);

employee\_multi\_set.insert(emp);

employee\_unordered\_map[emp.get\_id()] = emp;

employee\_unordered\_set.insert(emp);

employee\_unordered\_multi\_map.emplace(emp.get\_department(), emp);

employee\_unordered\_multi\_set.insert(emp);

break;

}

case 2:

std::cout << "\nemployee\_map:\n";

display\_all\_with\_key(employee\_map);

std::cout << "\nemployee\_set:\n";

display\_all(employee\_set);

std::cout << "\nemployee\_multi\_map:\n";

display\_all\_with\_key(employee\_multi\_map);

std::cout << "\nemployee\_multi\_set:\n";

display\_all(employee\_multi\_set);

std::cout << "\nemployee\_unordered\_map:\n";

display\_all\_with\_key(employee\_unordered\_map);

std::cout << "\nemployee\_unordered\_set:\n";

display\_all(employee\_unordered\_set);

std::cout << "\nemployee\_unordered\_multi\_map:\n";

display\_all\_with\_key(employee\_unordered\_multi\_map);

std::cout << "\nemployee\_unordered\_multi\_set:\n";

display\_all(employee\_unordered\_multi\_set);

break;

case 3:

save\_to\_file\_with\_key("employees\_map.txt", employee\_map);

save\_to\_file("employees\_set.txt", employee\_set);

save\_to\_file\_with\_key("employees\_multi\_map.txt", employee\_multi\_map);

save\_to\_file("employees\_multi\_set.txt", employee\_multi\_set);

save\_to\_file\_with\_key("employees\_unordered\_map.txt", employee\_unordered\_map);

save\_to\_file("employees\_unordered\_set.txt", employee\_unordered\_set);

save\_to\_file\_with\_key("employees\_unordered\_multi\_map.txt", employee\_unordered\_multi\_map);

save\_to\_file("employees\_unordered\_multi\_set.txt", employee\_unordered\_multi\_set);

break;

case 4:

{

int id;

std::cout << "Enter Employee ID: "; std::cin >> id;

find\_employee\_by\_id(employee\_map, id);

find\_employee\_in\_unordered(employee\_unordered\_set, id);

break;

}

case 5:

{

int id;

std::cout << "Enter Employee ID to edit: "; std::cin >> id;

edit\_employee(employee\_map, id);

break;

}

case 6:

{

int id;

std::cout << "Enter Employee ID to delete: "; std::cin >> id;

delete\_employee(employee\_map, id);

delete\_employee\_in\_unordered(employee\_unordered\_set, id);

break;

}

case 7:

sort\_by\_salary(employee\_set);

break;

case 0:

std::cout << "Have a nice day!\n";

break;

default:

std::cout << "Invalid choice. Try again.\n";

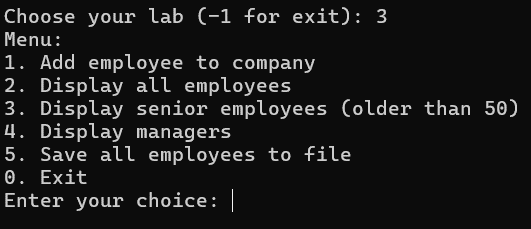
}

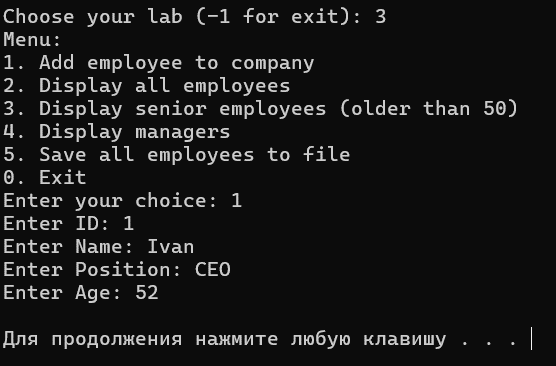
} while (choice != 0);

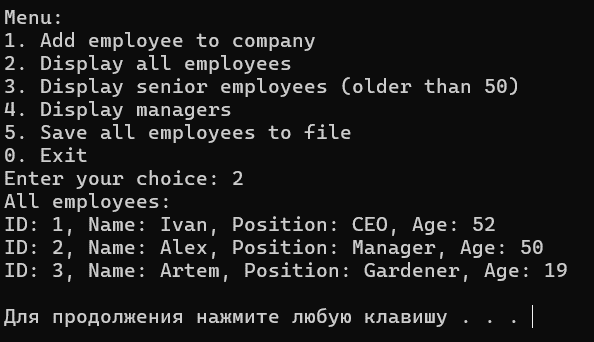
return 0;

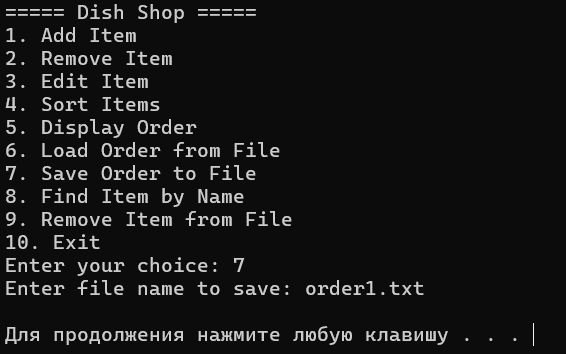
}

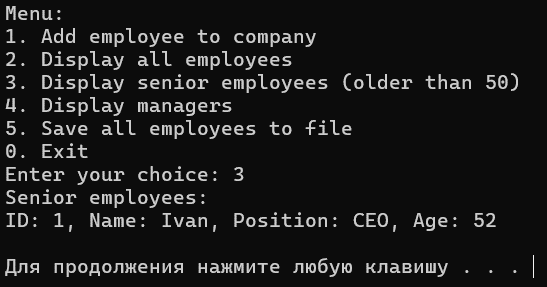
**5 Скриншоты выполнения индивидуального задания**

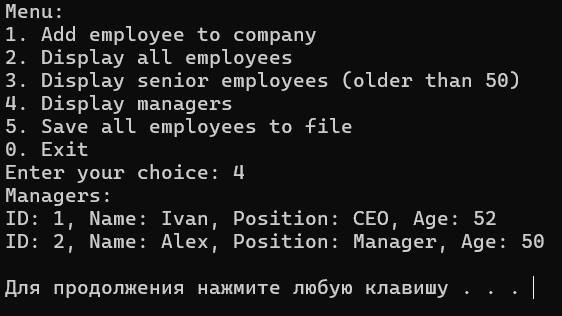


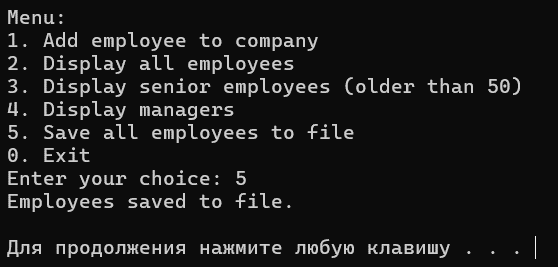


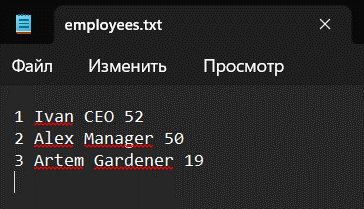


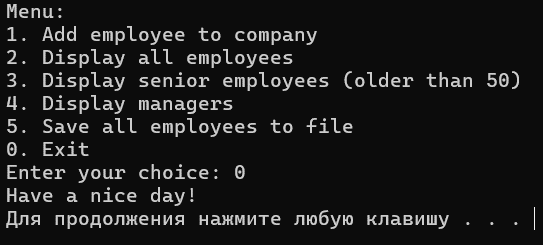












**6 Ответы на контрольные вопросы**

1. Назовите аналоги библиотеки STL в других языках программирования. Назовите 4 основные составляющие библиотеки STL?

Аналогами библиотеки STL в других языках программирования являются:

- Java: Коллекции (например, HashMap, TreeMap, HashSet, TreeSet).

- Python: Структуры данных, такие как dict, set, OrderedDict, frozenset.

- C#: Коллекции (например, Dictionary, HashSet, SortedSet).

- JavaScript: Объекты и карты (например, Map, Set).

Четыре основные составляющие библиотеки STL:

- Алгоритмы — набор универсальных функций, которые работают с контейнерами (например, sort(), find(), copy()).

- Контейнеры — структуры данных для хранения элементов (например, vector, list, map, set).

- Итераторы — объекты, которые используются для доступа к элементам контейнеров.

- Функторы (функциональные объекты) — объекты, которые могут вести себя как функции (например, для сравнения - элементов).

2. Что такое шаблонный класс?

Шаблонный класс — это класс, который параметризован одним или несколькими типами данных. Это позволяет создавать обобщённые классы, которые могут работать с любыми типами данных, обеспечивая высокую степень повторного использования кода. Например, класс map<Key, Value> может работать как с типами int, float, так и с пользовательскими типами, такими как структуры или объекты.

Пример шаблонного класса:

template <typename T>

class Box {

T value;

public:

Box(T val) : value(val) {}

T getValue() { return value; }

};

3. Зачем необходима STL, когда мы можем выполнять все операции, используя пользовательские структуры данных и функции для их обработки?

STL предоставляет готовые, отлаженные, высокоэффективные и стандартные структуры данных и алгоритмы, что позволяет:

- Сэкономить время на разработку и тестирование собственных реализаций.

- Использовать оптимизированные решения для типичных задач (поиск, сортировка и т.д.).

- Гарантировать переносимость и совместимость кода благодаря стандартным интерфейсам и поведению.

- Повысить читаемость и поддерживаемость кода за счет использования стандартных решений.

- STL помогает сосредоточиться на логике программы, а не на реализации низкоуровневых структур данных.

4. Каковы сходства и различия между итератором и указателем? Приведите пример объявления указателя и итератора.

Сходства:

- Итераторы и указатели позволяют перебирать элементы контейнера.

- Они поддерживают операции разыменования (\*) для получения значения и перемещения (++/--).

Различия:

- Итераторы могут работать с контейнерами, для которых указатели не применимы (например, списки).

- Указатели являются низкоуровневыми объектами, ссылающимися на физическую память, тогда как итераторы работают на уровне логики контейнера и могут быть более безопасными в использовании.

- Итераторы имеют более широкий функционал, адаптированный для различных контейнеров STL.

Пример объявления указателя:

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

int\* ptr = arr; // указатель на первый элемент массива

Пример объявления итератора:

std::vector<int> vec = {1, 2, 3};

std::vector<int>::iterator it = vec.begin(); // итератор на первый элемент вектора

5. Для чего используются методы lower\_bound() и upper\_bound()? В чем между ними различия?

lower\_bound(key) возвращает итератор на первый элемент, который не меньше заданного ключа key (то есть >= key).

upper\_bound(key) возвращает итератор на первый элемент, который больше заданного ключа key (то есть > key).

Различия:

- lower\_bound() может вернуть элемент, равный key, если такой существует.

- upper\_bound() всегда возвращает элемент строго больше key.

Пример использования:

std::set<int> s = {1, 3, 5, 7};

auto it1 = s.lower\_bound(5); // it1 указывает на 5

auto it2 = s.upper\_bound(5); // it2 указывает на 7

6. Что собой представляет «пара» в библиотеке STL? Опишите различные методы создания «пары» для вставки её в контейнер STL.

Пара (pair) — это шаблонный класс STL, который объединяет два значения разных типов в одну структуру. Элемент first хранит первое значение (обычно ключ), а элемент second — второе (значение, ассоциированное с ключом).

Создание пары:

- Явное создание через конструктор:

std::pair<int, std::string> p(1, "apple");

- Использование функции make\_pair(), которая автоматически определяет типы:

auto p = std::make\_pair(1, "apple");

- Пример вставки пары в контейнер map:

std::map<int, std::string> m;

m.insert(std::make\_pair(1, "apple"));

7. Кратко опишите свойства, особенности и различия контейнеров map, set.

map — хранит пары "ключ-значение". Ключи уникальны, и элементы автоматически сортируются по ключам. Позволяет быстрый поиск, добавление и удаление по ключу.

set — хранит только уникальные ключи, без значений. Элементы автоматически сортируются. Можно использовать для проверки наличия элемента, например, чтобы игнорировать повторяющиеся данные.

Различия:

- В map каждому ключу сопоставляется значение, а в set хранятся только ключи.

- Операции доступа к элементам в map происходят по ключу, а в set элементы сами по себе являются ключами.

8.Кратко опишите свойства, особенности и различия контейнеров multimap, multiset.

multimap — это версия map, которая позволяет нескольким элементам иметь один и тот же ключ.

multiset — это версия set, которая допускает хранение дублирующихся элементов.

Различия:

- В multimap можно хранить несколько значений для одного и того же ключа.

- В multiset можно хранить несколько одинаковых ключей.

9. Кратко опишите свойства, особенности и различия контейнеров unordered\_map, unordered\_set.

unordered\_map — контейнер, который хранит пары "ключ-значение", но без упорядочивания. Для быстрого доступа к элементам используется хеш-функция.

unordered\_set — контейнер, который хранит уникальные элементы (ключи), но также без упорядочивания. Использует хеш-функцию для организации данных.

Различия:

- В unordered\_map каждый ключ ассоциирован со значением, а в unordered\_set хранятся только ключи.

- В обоих контейнерах элементы не упорядочены, а доступны через хеш-функцию, что обеспечивает быструю вставку и поиск.

10. Кратко опишите свойства, особенности и различия контейнеров unordered\_multimap, unordered\_multiset.

unordered\_multimap — версия unordered\_map, допускающая несколько элементов с одинаковыми ключами. Использует хеш-функцию для доступа.

unordered\_multiset — версия unordered\_set, которая позволяет хранить дубликаты ключей, также используя хеш-функцию.

Различия:

- Оба контейнера не упорядочены и используют хеширование, но в unordered\_multimap можно хранить несколько значений для одного ключа, а в unordered\_multiset можно хранить несколько одинаковых ключей.

**7 Выводы по лабораторной работе**

Вывод: я изучил ассоциативные классы-контейнеры и классы-итераторы стандартной библиотеки шаблонов C++ (STL). Я овладел навыками работы с различными типами ассоциативных контейнеров, такими как map, set, multimap, multiset, а также их неупорядоченными аналогами. Научился применять итераторы для работы с элементами этих контейнеров. Реализовал базовые операции с контейнерами, такие как добавление, удаление, редактирование, поиск, сортировка и фильтрация данных. Закрепил навыки создания и работы с шаблонными классами STL, обеспечивающими эффективное хранение и доступ к данным по ключу. Освоил методы и приёмы работы с контейнерами, использующими ключи для быстрого доступа к данным, и изучил особенности реализации итераторов для этих контейнеров.