Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет инженерно-экономический

Кафедра экономической информатики

**Отчет к лабораторной работе №7**

**по теме**

**«АССОЦИАТИВНЫЕ КЛАССЫ-КОНТЕЙНЕРЫ БИБЛИОТЕКИ STL»**

|  |  |
| --- | --- |
| Выполнила: | Вдовенко Н. Д.  студент группы 272303 |
| Проверила: | Салапура М. Н.  ст. преподаватель |

Минск 2023

Общая постановка задачи

**Цель работы:** изучить ассоциативные классы-контейнеры библиотеки STL.

**Индивидуальное задание:**

Необходимо создать контейнеры, которые будут хранить объекты классов по предметной области «банковские сотрудники». Для контейнера реализовать добавление, удаление, редактирование, вывод содержимого контейнера на экран и в файл, поиск и сортировку элементов. Необходимо создать удобное пользовательское меню. Необходимо реализовать работу с контейнерами map, set, multimap, multiset, unordered\_map, unordered\_set, unordered\_multimap, unordered\_multiset.

**Краткие теоретические сведения:**

Контейнеры — это объекты, которые содержат в себе набор (совокупность) других объектов.

Ассоциативные контейнеры кардинально отличаются от последовательных: элементы в ассоциативном контейнере хранятся и предоставляются по ключу. Элементы последовательного контейнера хранятся и предоставляются последовательно, по их позиции в контейнере.

Ассоциативные контейнеры (associative container) обеспечивают быстрый поиск и предоставление элементов по ключу. Двумя первичными типами ассоциативных контейнеров являются map (карта) и set (набор). Элементами контейнера map являются пары ключ-значение (keyvalue pair): ключ выступает в роли индекса, а значение представляет собой хранимые в контейнере данные. Контейнер set содержит только ключи и предоставляет эффективные способы запроса на проверку наличия определенного ключа. Set можно использовать, например, для хранения слов, которые следует проигнорировать при обработке текста. Map можно использовать для создания словаря: слово будет ключом, а его определение — значением для этого ключа.

Библиотека STL предоставляет ассоциативные контейнеры, которые различаются по трем параметрам: − они являются набором (set) или картой (map); − они требуют уникальных ключей или допускает их совпадение; − они хранят элементы упорядочено или нет. В именах контейнеров, допускающих совпадение ключей, присутствует слово multi; имена контейнеров, не упорядочивающих хранимые ключи начинаются со слова unordered.

Типы ассоциативных контейнеров:

− map (ассоциативный массив, хранящий пары ключ-значение);

− set (контейнер, в котором ключ является значением);

− multimap (карта, допускающая совпадение ключей);

− multiset (набор, допускающий совпадение ключей);

− unordered\_map (карта, организованная по хеш-функции);

− unordered\_set (набор, организованный по хеш-функции);

− unordered\_multimap (хешированная карта; ключи могут повторяться);

− unordered\_multiset (хешированный набор; ключи могут повторяться).

**Диаграмма классов:**

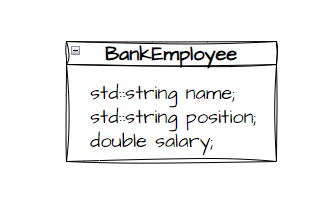


Рисунок 1 – Диаграмма классов

**Контрольные вопросы:**

1. Например, в языке Java имеется библиотека, аналогичная STL в C++, называемая Java Collections Framework. В C# существует пространство имен System.Collections, которое предоставляет структуры данных и алгоритмы, аналогичные функционалу STL. Кроме того, в Python есть различные встроенные структуры данных и библиотеки, такие как `collections` и `itertools`, которые обеспечивают сходный функционал STL (хотя концепция языка Python отличается от C++ и STL).
2. Шаблонный класс в языке программирования C++ представляет собой класс, который может работать с различными типами данных без необходимости создания отдельного класса для каждого типа. Шаблоны позволяют писать универсальный код.
3. STL предоставляет оптимизированные, проверенные временем структуры данных и алгоритмы, которые позволяют удобную и эффективную работу с данными. Вместо написания пользовательских функций и структур пользования стандартной библиотекой позволяет использовать готовые универсальные компоненты.
4. Итератор и указатель похожи в том, что оба предоставляют доступ к элементам коллекции. Однако итератор абстрагирует этот доступ от конкретной структуры данных, что позволяет использовать общие методы для перебора различных типов коллекций. Пример:

int\* ptr;

std::vector<int>::iterator it;

1. Метод `lower\_bound()` используется для поиска первого элемента, равного или большего, указанного значения в отсортированной последовательности. `upper\_bound()` находит первый элемент, больший указанного значения. Разница в том, что `lower\_bound()` возвращает позицию элемента в пределах последовательности, равного указанному значению, или итератор, указывающий на первый элемент, который больше указанного. `upper\_bound()` возвращает итератор, указывающий на первый элемент, который больше указанного значения в последовательности.
2. "Пара" в STL представляет структуру данных для хранения пары значений. Она широко используется для хранения двух значений различных типов вместе. Создание пары может быть сделано, например, с помощью конструктора `std::make\_pair`, либо напрямую передавая значения в конструктор `std::pair`.
3. `std::map` представляет отсортированное отображение, где каждый ключ связан со значением. `std::set` содержит отсортированный набор уникальных элементов.
4. `std::multimap` аналогичен `map`, но допускает хранение не уникальных ключей. `std::multiset` аналогичен `set`, но позволяет хранить не уникальные элементы.
5. `std::unordered\_map` и `std::unordered\_set` представляют аналоги `map` и `set`, но поддерживают хэшированные таблицы для реализации, что обеспечивает быстрое получение доступа к элементам.
6. std::unordered\_multimap` и `std::unordered\_multiset` представляют аналоги `multimap` и `multiset`, но также используют хэшированные таблицы и позволяют хранить не уникальные ключи или значения.

**Листинг кода:**

#include <iostream>

#include <map>

#include <set>

#include <unordered\_map>

#include <unordered\_set>

#include <fstream>

#include <algorithm>

#include <Windows.h>

class BankEmployee {

public:

std::string name;

std::string position;

double salary;

BankEmployee(const std::string& n, const std::string& pos, double sal) : name(n), position(pos), salary(sal) {}

void displayInfo() const {

std::cout << "Name: " << name << "\tPosition: " << position << "\tSalary: " << salary << std::endl;

}

size\_t hash() const {

return std::hash<std::string>()(name) ^ std::hash<std::string>()(position) ^ std::hash<double>()(salary);

}

bool operator<(const BankEmployee& other) const {

return name < other.name;

}

bool operator==(const BankEmployee& other) const {

return name == other.name && position == other.position && salary == other.salary;

}

};

namespace std {

template <>

struct hash<BankEmployee> {

size\_t operator()(const BankEmployee& employee) const {

return employee.hash();

}

};

}

template<typename Container>

void displayContainer(const Container& container) {

for (const auto& item : container) {

item.second.displayInfo();

}

}

template<typename Container>

void saveToFile(const Container& container, const std::string& filename) {

std::ofstream file(filename);

for (const auto& item : container) {

file << item.first << "," << item.second.name << "," << item.second.position << "," << item.second.salary << "\n";

}

file.close();

}

template<typename Container>

void addEmployee(Container& container) {

int id;

std::string name, position;

double salary;

std::cout << "Enter employee ID: ";

std::cin >> id;

std::cout << "Enter employee name: ";

std::cin >> name;

std::cout << "Enter employee position: ";

std::cin >> position;

std::cout << "Enter employee salary: ";

std::cin >> salary;

container.emplace(id, BankEmployee(name, position, salary));

std::cout << "Employee added successfully!\n";

}

template<typename Container>

void deleteEmployee(Container& container) {

int id;

std::cout << "Enter the ID of the employee to delete: ";

std::cin >> id;

auto it = container.find(id);

if (it != container.end()) {

container.erase(it);

std::cout << "Employee deleted successfully!\n";

}

else {

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

}

}

template<typename Container>

void searchEmployee(const Container& container) {

int id;

std::cout << "Enter the ID of the employee to search: ";

std::cin >> id;

auto it = container.find(id);

if (it != container.end()) {

it->second.displayInfo();

}

else {

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

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::map<int, BankEmployee> employeeMap;

std::set<BankEmployee> employeeSet;

std::multimap<int, BankEmployee> employeeMultimap;

std::multiset<BankEmployee> employeeMultiset;

std::unordered\_map<int, BankEmployee> employeeUnorderedMap;

std::unordered\_set<BankEmployee> employeeUnorderedSet;

std::unordered\_multimap<int, BankEmployee> employeeUnorderedMultimap;

std::unordered\_multiset<BankEmployee> employeeUnorderedMultiset;

int choice;

do {

std::cout << "\n=== Bank Employee Management System ===\n";

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

std::cout << "2. Delete Employee\n";

std::cout << "3. Display Employees\n";

std::cout << "4. Save to File\n";

std::cout << "5. Search Employee\n";

std::cout << "6. Exit\n";

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

std::cin >> choice;

switch (choice) {

case 1:

addEmployee(employeeMap);

break;

case 2:

deleteEmployee(employeeMap);

break;

case 3:

displayContainer(employeeMap);

break;

case 4:

saveToFile(employeeMap, "employees.txt");

std::cout << "Employees saved to file successfully!\n";

break;

case 5:

searchEmployee(employeeMap);

break;

case 6:

std::cout << "Exiting program. Goodbye!\n";

break;

default:

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

}

} while (choice != 6);

return 0;

}

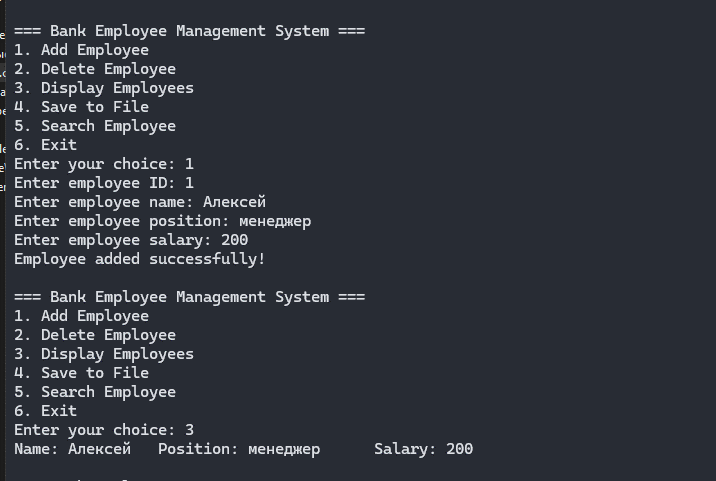


Рисунок 2 – Ввод информации

**Вывод**: в ходе выполнения данной лабораторной работы были изучены и применены основные принципы работы с ассоциативными контейнерами.