Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

ОТЧЕТ

к лабораторной работе №3

**«Последовательные 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 и применить на практике последовательные контейнеры и классы-итераторы библиотеки STL языка C++. Основная задача состоит в создании программы, которая использует контейнеры list, vector и array для хранения объектов определенной предметной области. В рамках работы необходимо реализовать функционал добавления, удаления, редактирования, поиска и сортировки данных, а также вывести данные на экран и сохранить их в файл.

**2 Теоретические сведения по лабораторной работе**

STL (Standard Template Library) — это библиотека, включённая в стандарты языка C++ с 1998 года, которая предоставляет набор шаблонных классов и функций общего назначения. Эти классы реализуют различные структуры данных, а функции — алгоритмы для работы с ними. Основные компоненты библиотеки STL включают:

- Контейнеры — объекты, которые хранят наборы данных. Примеры: vector, list, array.

- Алгоритмы — функции, которые обрабатывают содержимое контейнеров (например, сортировка, поиск).

- Итераторы — объекты, позволяющие перебирать элементы контейнера, подобно указателям.

STL использует шаблоны (templates), что делает её универсальной и гибкой для работы с различными типами данных.

Последовательные контейнеры — это контейнеры, которые хранят элементы в строгом порядке их добавления. Эти контейнеры обеспечивают упорядоченный доступ к своим элементам, но не всегда позволяют быстрый произвольный доступ, в отличие от ассоциативных контейнеров. Основные последовательные контейнеры, используемые в работе:

1. vector — динамический массив:

- Позволяет хранить элементы, размер которых может меняться во время выполнения программы.

- Эффективен для последовательного доступа к элементам, а также поддерживает произвольный доступ по индексу.

- Увеличивает или уменьшает объём памяти при добавлении или удалении элементов.

- Часто используется для работы с большими наборами данных, которые изменяются во время выполнения программы.

2. list — двусвязный список:

- Эффективен для вставки и удаления элементов в середине или на краях контейнера.

- Позволяет перемещаться по элементам в обоих направлениях (вперёд и назад).

- Не поддерживает произвольный доступ по индексу, требует итерации для поиска элемента.

3. array — массив фиксированного размера:

- Представляет статический массив, размер которого задаётся во время компиляции.

- Эффективен для задач, где известно количество элементов заранее, и не требуется изменение размера.

- Обеспечивает быстрый доступ к элементам по индексу, но не может динамически расширяться или сжиматься.

Итераторы в STL выполняют роль указателей на элементы контейнера и позволяют циклически перебирать его содержимое. Существует несколько типов итераторов:

- Итераторы произвольного доступа (random access iterator) — позволяют перемещаться по элементам контейнера произвольно (например, в vector).

- Двунаправленные итераторы (bidirectional iterator) — позволяют перемещаться в обоих направлениях, но не поддерживают произвольный доступ (например, в list).

- Входные и выходные итераторы (input/output iterator) — обеспечивают доступ только для чтения или записи соответственно.

Использование итераторов делает возможным универсальную работу с контейнерами независимо от их внутренней структуры.

STL также предоставляет широкий набор алгоритмов, которые можно применять к элементам контейнеров с использованием итераторов. Эти алгоритмы включают сортировку, поиск, копирование и другие действия, упрощающие работу с данными в контейнерах.

**3 Формулировка индивидуального задания**

В каждом индивидуальном задании необходимо создать контейнеры list, vector, array, которые будут хранить объекты классов по предметной области, указанной в таблице (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 <vector>

#include <list>

#include <array>

#include <fstream>

#include <string>

class employee

{

int id\_;

std::string name\_;

std::string position\_;

int age\_;

public:

employee() = default;

employee(const int id, std::string name, std::string position, const int age)

: id\_(id), name\_(std::move(name)), position\_(std::move(position)), age\_(age) {}

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

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

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

int get\_age() const { return age\_; }

void display() const

{

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

<< ", Position: " << position\_ << ", Age: " << age\_ << '\n';

}

bool is\_manager() const

{

return position\_ == "Manager" || position\_ == "CEO";

}

};

void add\_employee(std::vector<employee>& employees, const employee& emp)

{

employees.push\_back(emp);

}

void display\_employees(const std::vector<employee>& employees)

{

for (const auto& emp : employees)

emp.display();

}

void add\_older\_employee(std::list<employee>& senior\_employees, const employee& emp, const int age\_threshold)

{

if (emp.get\_age() > age\_threshold)

senior\_employees.push\_back(emp);

}

void display\_employees(const std::list<employee>& employees)

{

for (const auto& emp : employees)

emp.display();

}

template <std::size\_t N>

void add\_manager(std::array<employee, N>& managers, const employee& emp, std::size\_t& current\_manager\_count)

{

if (emp.is\_manager() && current\_manager\_count < N)

managers[current\_manager\_count++] = emp;

}

template <std::size\_t N>

void display\_employees(const std::array<employee, N>& employees, const std::size\_t current\_manager\_count)

{

for (std::size\_t i = 0; i < current\_manager\_count; ++i)

employees[i].display();

}

template <typename Container>

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

{

std::ofstream file(filename);

for (const auto& emp : employees)

file << emp.get\_id() << " " << emp.get\_name() << " "

<< emp.get\_position() << " " << emp.get\_age() << std::endl;

file.close();

}

void end()

{

system("pause");

system("cls");

}

int main()

{

std::vector<employee> employees;

std::list<employee> senior\_employees;

std::array<employee, 2> managers;

std::size\_t current\_manager\_count = 0;

// ReSharper disable once CppTooWideScope

constexpr int senior\_age\_threshold = 50;

int choice = 0;

do

{

if (choice) end();

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

std::cout << "1. Add employee to company\n";

std::cout << "2. Display all employees\n";

std::cout << "3. Display senior employees (older than " << senior\_age\_threshold << ")\n";

std::cout << "4. Display managers\n";

std::cout << "5. Save all employees to file\n";

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

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

std::cin >> choice;

switch (choice)

{

case 1:

{

int id, age;

std::string name, position;

std::cout << "Enter 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 Age: ";

std::cin >> age;

employee emp(id, name, position, age);

add\_employee(employees, emp);

add\_older\_employee(senior\_employees, emp, senior\_age\_threshold);

add\_manager(managers, emp, current\_manager\_count);

break;

}

case 2:

std::cout << "All employees:\n";

display\_employees(employees);

break;

case 3:

std::cout << "Senior employees:\n";

display\_employees(senior\_employees);

break;

case 4:

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

display\_employees(managers, current\_manager\_count);

break;

case 5:

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

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

break;

case 0:

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

break;

default:

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

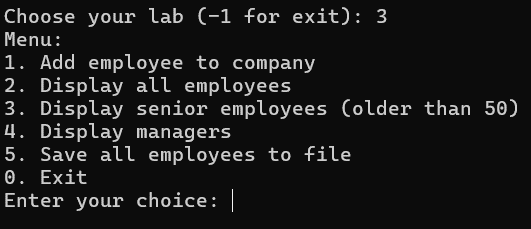
}

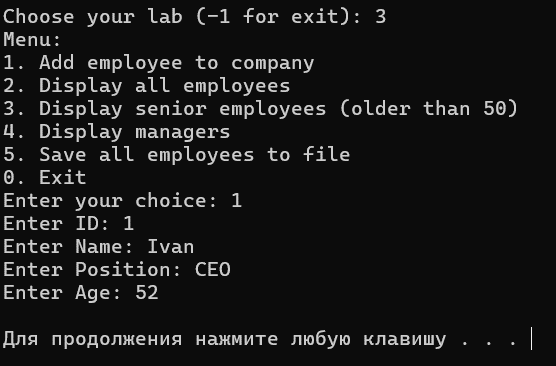
} while (choice != 0);

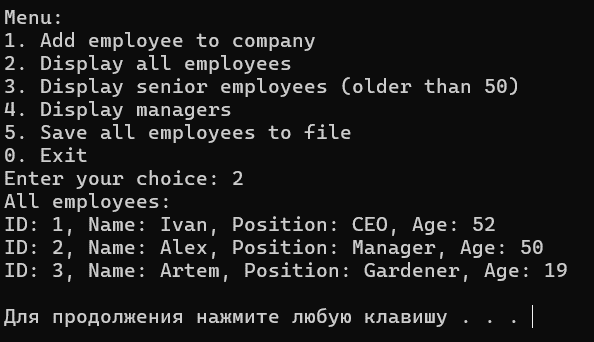
return 0;

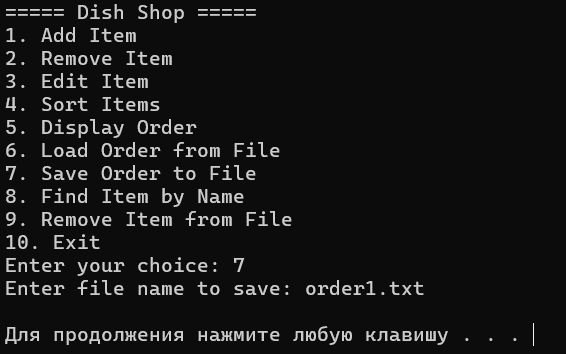
}

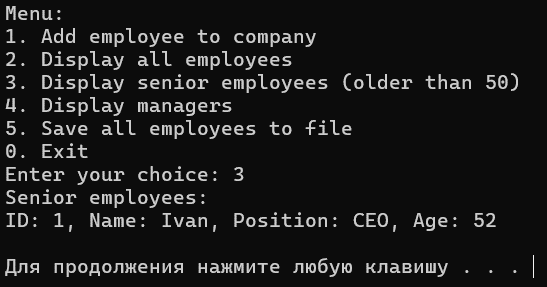
**5 Скриншоты выполнения индивидуального задания**

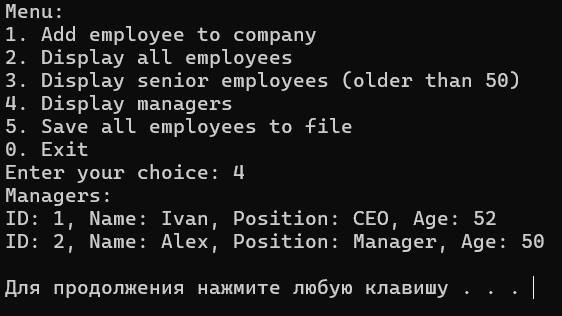


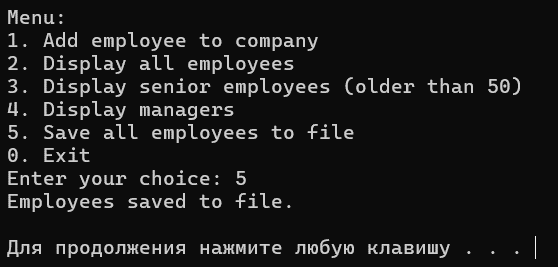


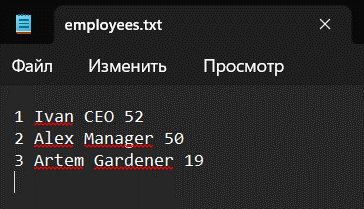


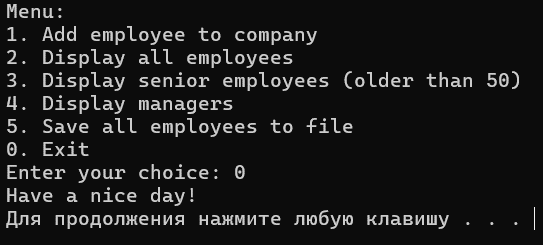












**6 Ответы на контрольные вопросы**

1. Что такое контейнер в библиотеке STL?

Контейнер в STL — это объект, который хранит коллекцию элементов (объектов) и предоставляет методы для работы с этими элементами. Контейнеры могут быть последовательными (например, vector, list, array), ассоциативными (например, map, set) или адаптерами контейнеров (например, stack, queue).

2. Что такое итератор в библиотеке STL?

Итератор в STL — это объект, который действует как указатель на элемент контейнера. Он позволяет перебирать элементы контейнера и манипулировать ими, независимо от структуры самого контейнера. Итераторы можно инкрементировать, декрементировать и разыменовывать для доступа к элементам.

3. Что такое алгоритм в библиотеке STL?

Алгоритм в STL — это шаблонная функция, которая выполняет стандартные операции над элементами контейнеров, такие как сортировка, поиск, копирование, удаление и другие. Алгоритмы могут работать с диапазонами элементов контейнеров с использованием итераторов.

4. Какие виды итераторов существуют в библиотеке STL?

В STL существует несколько типов итераторов:

- Итератор произвольного доступа (random access iterator) — поддерживает произвольный доступ к элементам (например, в vector и array).

- Двунаправленный итератор (bidirectional iterator) — позволяет перемещаться в обоих направлениях (например, в list).

- Однонаправленный итератор (forward iterator) — позволяет перемещаться только вперед.

- Входной итератор (input iterator) — предназначен для чтения данных из контейнера.

- Выходной итератор (output iterator) — используется для записи данных в контейнер.

5. Что представляет собой предикат в библиотеке STL?

Предикат в STL — это функция или функциональный объект, который возвращает булевое значение (истина или ложь). Предикаты используются в алгоритмах для проверки условий (например, для фильтрации элементов). Существуют унарные предикаты (с одним аргументом) и бинарные предикаты (с двумя аргументами).

6. Что собой представляет контейнер вектор (vector) в STL C++?

vector — это динамический массив, который может изменять свой размер во время выполнения программы. Он обеспечивает произвольный доступ к элементам по индексу и поддерживает операции вставки и удаления элементов в конце. vector автоматически управляет выделением и освобождением памяти.

7. Что собой представляет контейнер список (list) в STL C++?

list — это двусвязный список, который позволяет эффективно вставлять и удалять элементы в любом месте контейнера. В отличие от vector, list не поддерживает произвольный доступ по индексу, доступ к элементам осуществляется только через итераторы.

8. Что собой представляет контейнер массив (array) в STL C++?

array — это контейнер фиксированного размера, который представляет собой обобщённый аналог статического массива в C++. Размер массива задаётся при его создании и не может изменяться. array обеспечивает быстрый доступ к элементам по индексу.

9. В чем различие между контейнерами list, vector, array?

- vector — динамический массив, который может изменять размер и предоставляет быстрый произвольный доступ к элементам по индексу. Хорошо подходит для работы с динамическими данными.

- list — двусвязный список, в котором элементы связаны ссылками друг на друга. Поддерживает эффективные операции вставки и удаления в любом месте контейнера, но не обеспечивает произвольный доступ по индексу.

- array — массив фиксированного размера, который предоставляет быстрый доступ к элементам, но не может изменять свой размер после создания.

10. Приведите пример кодя для создания контейнеров list, array, vector.

#include <iostream>

#include <vector>

#include <list>

#include <array>

int main()

{

// Создание vector для хранения целых чисел

std::vector<int> vec = {1, 2, 3, 4, 5};

// Создание list для хранения строк

std::list<std::string> lst = {"Hello", "STL", "World"};

// Создание array для хранения трёх целых чисел

std::array<int, 3> arr = {10, 20, 30};

// Вывод элементов vector

for (const auto& v : vec)

std::cout << v << " ";

std::cout << std::endl;

// Вывод элементов list

for (const auto& l : lst)

std::cout << l << " ";

std::cout << std::endl;

// Вывод элементов array

for (const auto& a : arr)

std::cout << a << " ";

std::cout << std::endl;

return 0;

}

**7 Выводы по лабораторной работе**

Вывод: я научился применять на практике последовательные контейнеры и классы-итераторы библиотеки STL языка C++. Я создал программу, которая использует контейнеры list, vector и array для хранения объектов определенной предметной области. Я реализовал функционал добавления, удаления, редактирования, поиска и сортировки данных, а также вывода данных на экран и сохранения их в файл.