Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет инженерно-экономический

Кафедра экономической информатики

**Отчет к лабораторной работе №6**

**по теме**

**«ПОСЛЕДОВАТЕЛЬНЫЕ КЛАССЫ-КОНТЕЙНЕРЫ И КЛАССЫ-ИТЕРАТОРЫ БИБЛИОТЕКИ STL»**

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

Минск 2023

Общая постановка задачи

**Цель работы:** изучить последовательные классы-контейнеры и классы-итераторы библиотеки STL.

**Индивидуальное задание:**

Необходимо создать контейнеры list, vector, array, которые будут хранить объекты классов по предметной области «компьютерные игры». Для контейнера реализовать добавление, удаление, редактирование, вывод содержимого контейнера на экран и в файл, поиск и сортировку элементов. Необходимо создать удобное пользовательское меню.

**Краткие теоретические сведения:**

Основой библиотеки STL являются три основных элемента: контейнеры, алгоритмы, итераторы. Они работают совместно друг с другом, предоставляя тем самым готовые решения для различных задач.

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

Последовательные контейнеры: − array (массив фиксированного размера) 5 − deque (двунаправленная, двунаправленная очередь) − forward\_list (односвязный список) − list (двусвязный линейный список) − vector (динамический массив, массив переменного размера) − string (контейнер, содержащий символы)

Векторы представляют собой динамические массивы. Класс vector поддерживает динамический массив, который при необходимости может изменять (увеличивать или уменьшать) свой размер. Размер статического массива фиксируется во время компиляции. И хотя это самый эффективный способ реализации массивов, он в то же время является и самым ограничивающим, поскольку размер массива нельзя изменять во время выполнения программы.

Список является контейнером с двунаправленным последовательным доступом к элементам. Класс list поддерживает функционирование двунаправленного линейного списка. В отличие от вектора, в котором реализована поддержка произвольного доступа, список позволяет получать к своим элементам только последовательно.

Двунаправленность списка означает, что доступ к его элементам возможен в двух направлениях: от начала к концу и от конца к началу. Шаблонная спецификация класса list выглядит следующим образом.

Контейнер array представляет аналог массива. Он также имеет фиксированный размер. Для создания объекта array в угловых скобках после названия типа необходимо передать его тип и размер. По умолчанию все элементы контейнера имеют неопределенные значения. Чтобы инициализировать контейнер определенными значениями, можно использовать инициализатор (в фигурных скобках передать значения элементам контейнера). Фиксированный размер накладывает ограничение на инициализацию: количество передаваемых контейнеру элементов не должно превышать его размер.

**Диаграмма классов:**

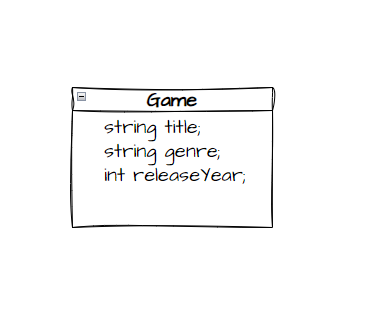


Рисунок 1 – Диаграмма классов

**Контрольные вопросы:**

1. Контейнеры — это объекты, которые содержат в себе набор (совокупность) других объектов.
2. Итераторы действуют подобно указателям. Они позволяют циклически опрашивать содержимое контейнера практически так же, как это делается с помощью указателя при циклическом опросе элементов массива.
3. Алгоритмы представляют собой функции, которые обрабатывают содержимое контейнеров. Их возможности включают средства инициализации, сортировки, поиска, преобразования содержимого контейнеров. Многие алгоритмы работают с заданным диапазоном элементов контейнера.
4. В STL существует несколько видов итераторов: “input iterators”, “output iterators”, “orward iterators”, “bidirectional iterators` и “random access iterators”. Каждый из них предоставляет различный набор функций в зависимости от своих характеристик.
5. Предикат (predicate) – функция, возвращающая в качестве результата значение ИСТИНА/ЛОЖЬ. Некоторые алгоритмы и контейнеры используют специальный тип функции, называемый предикатом. Существует два варианта предикатов: унарный и бинарный. Унарный предикат принимает один аргумент, а бинарный — два. Эти функции возвращают значения ИСТИНА/ЛОЖЬ, но точные условия, которые заставят их вернуть истинное или ложное значение, определяются программистом.
6. Вектор (vector) представляет собой динамический массив, который обеспечивает быстрый доступ к элементам и возможность динамического изменения размера контейнера.
7. Список (list) представляет собой двусвязный список, обеспечивающий эффективную вставку и удаление элементов в середине контейнера.
8. Массив (array) представляет собой контейнер фиксированного размера, предоставляющий доступ к элементам по индексу с помощью проверки выхода за границы. Размер массива определяется на этапе компиляции.
9. Основное различие между ними заключается в способе хранения и доступа к элементам. Вектор использует динамический массив, список - двусвязный список, а массив - фиксированный по размеру.

#include <iostream>

#include <vector>

#include <list>

#include <array>

using namespace std;

int main() {

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

list<int> lst = {6, 7, 8, 9, 10};

array<int, 5> arr = {11, 12, 13, 14, 15};

return 0;

}

**Листинг кода:**

#include <iostream>

#include <list>

#include <vector>

#include <array>

#include <fstream>

#include <algorithm>

class Game {

public:

std::string title;

std::string genre;

int releaseYear;

Game(const std::string& t, const std::string& g, int year) : title(t), genre(g), releaseYear(year) {}

Game() : releaseYear(0) {}

void displayInfo() const {

std::cout << "Title: " << title << "\tGenre: " << genre << "\tRelease Year: " << releaseYear << std::endl;

}

bool operator<(const Game& other) const {

return title < other.title;

}

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

return title == other.title;

}

};

template<typename Container>

void displayContainer(const Container& container) {

for (const auto& item : container) {

item.displayInfo();

}

}

template<typename Container>

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

std::ofstream file(filename);

for (const auto& item : container) {

file << item.title << "," << item.genre << "," << item.releaseYear << "\n";

}

file.close();

}

template<typename Container>

void addGame(Container& container) {

std::string title, genre;

int releaseYear;

std::cout << "Enter game title: ";

std::cin >> title;

std::cout << "Enter game genre: ";

std::cin >> genre;

std::cout << "Enter release year: ";

std::cin >> releaseYear;

container.emplace\_back(title, genre, releaseYear);

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

}

template<typename Container>

void deleteGame(Container& container) {

std::string title;

std::cout << "Enter the title of the game to delete: ";

std::cin >> title;

auto it = std::find\_if(container.begin(), container.end(), [&title](const Game& game) {

return game.title == title;

});

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

container.erase(it);

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

}

else {

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

}

}

template<typename Container>

void searchGame(const Container& container) {

std::string title;

std::cout << "Enter the title of the game to search: ";

std::cin >> title;

auto it = std::find\_if(container.begin(), container.end(), [&title](const Game& game) {

return game.title == title;

});

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

it->displayInfo();

}

else {

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

}

}

template<typename Container>

void sortGames(Container& container) {

container.sort([](const Game& a, const Game& b) {

return a.title < b.title;

});

std::cout << "Games sorted successfully!\n";

}

int main() {

std::list<Game> gameList;

std::vector<Game> gameVector;

std::array<Game, 10> gameArray;

int choice;

do {

std::cout << "\n=== Game Management System ===\n";

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

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

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

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

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

std::cout << "6. Sort Games\n";

std::cout << "7. Exit\n";

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

std::cin >> choice;

switch (choice) {

case 1:

addGame(gameList);

break;

case 2:

deleteGame(gameList);

break;

case 3:

displayContainer(gameList);

break;

case 4:

saveToFile(gameList, "games.txt");

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

break;

case 5:

searchGame(gameList);

break;

case 6:

sortGames(gameList);

break;

case 7:

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

break;

default:

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

}

} while (choice != 7);

return 0;

}

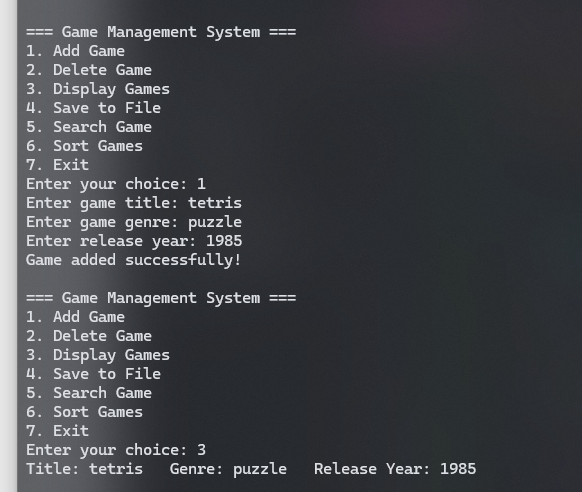


Рисунок 2 – Ввод информации

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