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:**

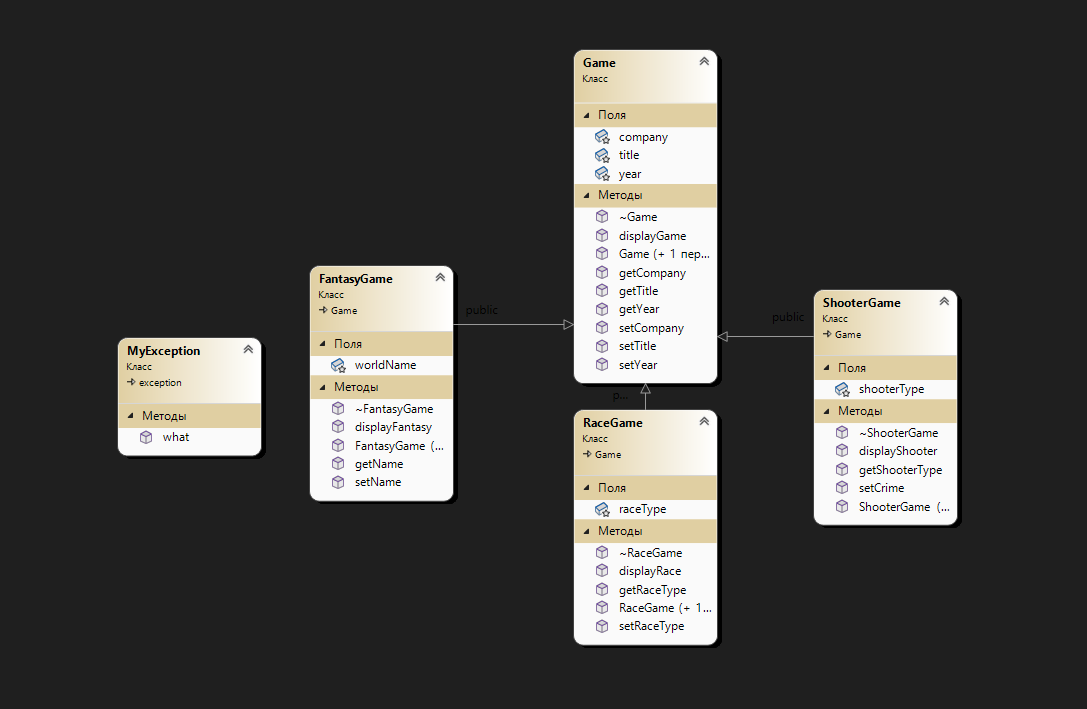


Рисунок 1 – Диаграмма классов

**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 <list>

#include <iostream>

#include <string>

#include <vector>

#include <array>

#include <windows.h>

#include <fstream>

#include <stdlib.h>

#include <conio.h>

#include <iomanip>

#include <Windows.h>

#define GAMESFILENAME "files\\games.txt"

using namespace std;

class Game {

protected:

string title;

string company;

int year;

public:

Game() {}

Game(string title, string company, int year)

{

this->title = title;

this->company = company;

this->year = year;

}

~Game() {}

string getTitle() const {

return title;

}

void setTitle(string newTitle) {

title = newTitle;

}

string getCompany() const {

return company;

}

void setCompany(string company) {

this->company = company;

}

int getYear() const {

return year;

}

void setYear(int newYear) {

year = newYear;

}

void displayGame()

{

cout << "\nНазвание: " << title << endl;

cout << "Издатель: " << company << endl;

cout << "Год: " << year << endl;

}

};

class RaceGame : public Game {

protected:

string raceType;

public:

RaceGame() : Game("", "", 0), raceType("") {}

RaceGame(string title, string company, int year, string raceType) :Game(title, company, year)

{

this->raceType = raceType;

}

string getRaceType() const {

return raceType;

}

void setRaceType(string raceType) {

this->raceType = raceType;

}

~RaceGame() {}

void displayRace()

{

cout << "\nИгры про гонки" << endl;

Game::displayGame();

cout << "Тип гонок: " << raceType << endl;

}

};

class ShooterGame : public Game {

protected:

string shooterType;

public:

ShooterGame() : Game("", "", 0), shooterType("") {}

ShooterGame(string title, string company, int year, string shooterType) :Game(title, company, year)

{

this->shooterType = shooterType;

}

string getShooterType() const {

return shooterType;

}

void setCrime(string newCrime) {

this->shooterType = shooterType;

}

~ShooterGame() {}

void displayShooter()

{

cout << "\nШутеры" << endl;

Game::displayGame();

cout << "Вид шутера: " << shooterType << endl;

}

};

class FantasyGame : public Game {

protected:

string worldName;

public:

FantasyGame() : Game("", "", 0), worldName("") {}

FantasyGame(string title, string author, int year, string worldName) :Game(title, company, year)

{

this->worldName = worldName;

}

string getName() const {

return worldName;

}

void setName(string worldName) {

worldName = worldName;

}

~FantasyGame() {}

void displayFantasy()

{

cout << "\nФантастика" << endl;

Game::displayGame();

cout << "Название мира: " << worldName << endl;

}

};

class MyException : public exception {

public:

const char\* what() const throw() {

return "Ошибка: Максимальное количество игр было достигнуто!";

}

};

class MyExceptionComputer : public exception

{

public:

const char\* what() const throw() {

return "Игр нет!";

}

};

void termFunc()

{

cout << "Сработала собственная функция завершения\n";

exit(-1);

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

const wchar\_t\* directoryPath = L"files";

if (!CreateDirectory(directoryPath, NULL)) {

if (GetLastError() == ERROR\_ALREADY\_EXISTS) {

std::wcout << L"Directory 'files' already exists.\n";

}

else {

std::cerr << "Error creating directory 'files'.\n";

}

}

else {

std::wcout << L"Directory 'files' created successfully.\n";

}

ofstream outfile(GAMESFILENAME);

if (outfile.is\_open()) {

outfile.close();

}

RaceGame\* race = NULL;

std::vector<RaceGame\*> raceGames;

int current\_race = 0;

ShooterGame\* shooter = NULL;

std::list<ShooterGame\*> shooterGames;

int current\_shooter = 0;

FantasyGame\* fantasy = NULL;

std::array<FantasyGame\*, 5> fantasyGames;

int current\_fantasy = 0;

int choice;

do {

try {

cout << "\nМеню выбора" << endl;

cout << "1 - Добавить игру" << endl;

cout << "2 - Просмотр игр" << endl;

cout << "3 - Поиск игры" << endl;

cout << "4 - Удалить игру" << endl;

cout << "5 - Изменить игру" << endl;

cout << "6 - Сортировать игры" << endl;

cout << "7 - Сохранить в файл" << endl;

cout << "0 - Выйти" << endl;

while (true) {

cout << "Выберите действие: ";

cin >> choice;

if (cin.good() && choice >= 0 && choice <= 7) {

break;

}

cin.clear();

cout << "Неправильный ввод данных\n";

cin.ignore(100, '\n');

}

switch (choice) {

case 1: {

system("cls");

int choice1;

do {

cout << "\nМеню выбора" << endl;

cout << "1 - Добавить гонку" << endl;

cout << "2 - Добавить шутер" << endl;

cout << "3 - Добавить фэнтези" << endl;

cout << "0 - Выйти" << endl;

while (true) {

cout << "Выберите действие: ";

cin >> choice1;

if (cin.good() && choice1 >= 0 && choice1 <= 3) {

break;

}

cin.clear();

cout << "Неправильный ввод данных\n";

cin.ignore(100, '\n');

}

switch (choice1) {

case 1: {

system("cls");

string title;

cout << "Введите название игры" << endl;

cin >> title;

string company;

cout << "Введите издателя игры" << endl;

cin >> company;

int year;

while (true) {

cout << "Введите год выпуска" << endl;

cin >> year;

if (cin.good() && year >= 1000 && year <= 2023) {

break;

}

cin.clear();

cout << "Неправильный ввод данных\n";

cin.ignore(100, '\n');

}

string raceType;

cout << "Введите тип гонки" << endl;

cin >> raceType;

race = new RaceGame(title, company, year, raceType);

raceGames.push\_back(race);

current\_race++;

cout << "Игра успешно добавлена" << endl;

vector<RaceGame\*>::iterator i = raceGames.begin();

break;

}

case 2: {

system("cls");

string title;

cout << "Введите название игры" << endl;

cin >> title;

string company;

cout << "Введите издателя игры" << endl;

cin >> company;

int year;

while (true) {

cout << "Введите год выпуска" << endl;

cin >> year;

if (cin.good() && year >= 1000 && year <= 2023) {

break;

}

cin.clear();

cout << "Неправильный ввод данных\n";

cin.ignore(100, '\n');

}

string shooterType;

cout << "Введите вид шутера" << endl;

cin >> shooterType;

shooter = new ShooterGame(title, company, year, shooterType);

shooterGames.push\_back(shooter);

cout << "Игра успешно добавлена" << endl;

current\_shooter++;

break;

}

case 3: {

system("cls");

if (current\_fantasy < 6)

{

string title;

cout << "Введите название игры" << endl;

cin >> title;

string company;

cout << "Введите издателя игры" << endl;

cin >> company;

int year;

while (true) {

cout << "Введите год выпуска" << endl;

cin >> year;

if (cin.good() && year >= 1000 && year <= 2023) {

break;

}

cin.clear();

cout << "Неправильный ввод данных\n";

cin.ignore(100, '\n');

}

string worldName;

cout << "Введите название мира" << endl;

cin >> worldName;

fantasy = new FantasyGame(title, company, year, worldName);

fantasyGames[current\_fantasy] = fantasy;

cout << "Игра успешно добавлена" << endl;

current\_fantasy++;

}

cout << "\n";

break;

}

}

} while (choice1 != 0);

break;

}

case 2: {

system("cls");

if (current\_race > 0)

{

cout << "-------------------------------------------------------------------------------------\n";

cout << "| Гонки |\n";

cout << "-------------------------------------------------------------------------------------\n";

cout << '|' << setw(25) << left << "Название игры" << '|' << setw(20) << "Издатель " << '|' << setw(10) << "Год" << '|' << setw(25) << "Тип гонок" << '|' << endl;

cout << "-------------------------------------------------------------------------------------\n";

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

{

cout << '|' << setw(25) << left << raceGames[i]->getTitle() << '|' << setw(20)

<< raceGames[i]->getCompany() << '|' << setw(10) << raceGames[i]->getYear() << "|" << setw(25) << raceGames[i]->getRaceType() << "|" << endl;

cout << "-------------------------------------------------------------------------------------\n";

}

}

else

{

cout << "Отсутствуют гонки" << endl;

}

std::cout << "\n";

if (current\_shooter > 0)

{

cout << "-------------------------------------------------------------------------------------\n";

cout << "| Шутеры |\n";

cout << "-------------------------------------------------------------------------------------\n";

cout << '|' << setw(25) << left << "Название игры" << '|' << setw(20) << "Издатель " << '|' << setw(10) << "Год" << '|' << setw(25) << "Тип шутера" << '|' << endl;

cout << "-------------------------------------------------------------------------------------\n";

for (std::list<ShooterGame\*>::iterator p = shooterGames.begin(); p != shooterGames.end(); ++p)

{

cout << '|' << setw(25) << left << (\*p)->getTitle() << '|' << setw(20) << (\*p)->getCompany() << '|' << setw(10) << (\*p)->getYear() << "|" << setw(25) << (\*p)->getShooterType() << "|" << endl;

cout << "-------------------------------------------------------------------------------------\n";

}

}

else

{

cout << "Отсутсвуют шутеры" << endl;

}

std::cout << "\n";

if (current\_fantasy > 0)

{

cout << "-------------------------------------------------------------------------------------\n";

cout << "| Фэнтези |\n";

cout << "-------------------------------------------------------------------------------------\n";

cout << '|' << setw(25) << left << "Название игры" << '|' << setw(20) << "Издатель " << '|' << setw(10) << "Год" << '|' << setw(25) << " Название мира" << '|' << endl;

cout << "-------------------------------------------------------------------------------------\n";

for (int i = 0; i < current\_fantasy; i++)

{

cout << '|' << setw(25) << left << fantasyGames[i]->getTitle() << '|' << setw(20) << fantasyGames[i]->getCompany() << '|' << setw(10) << fantasyGames[i]->getYear() << "|" << setw(25) << fantasyGames[i]->getName() << "|" << endl;

cout << "-------------------------------------------------------------------------------------\n";

cout << "\n";

}

}

else

{

cout << "Отсутсвует фэнтези" << endl;

}

cout << "\n";

break;

}

case 3: {

system("cls");

int choice1;

do {

cout << "\nМеню выбора" << endl;

cout << "1 - Поиск гонок" << endl;

cout << "2 - Поиск шутера" << endl;

cout << "3 - Поиск фэнтези" << endl;

cout << "0 - Выйти" << endl;

while (true) {

cout << "Выберите действие: ";

cin >> choice1;

if (cin.good() && choice1 >= 0 && choice1 <= 3) {

break;

}

cin.clear();

cout << "Неправильный ввод данных\n";

cin.ignore(100, '\n');

}

switch (choice1) {

case 1: {

system("cls");

if (current\_race > 0)

{

string title;

cout << "Введите название игры:" << endl;

cin >> title;

for (std::vector<RaceGame\*>::iterator i = raceGames.begin(); i != raceGames.end(); i++)

{

if ((\*i)->getTitle() == title)

{

cout << "Найденная игра\n" << endl;

(\*i)->displayRace();

}

else { cout << "Нет такой игры"; }

}

}

else

{

cout << "Нет игр" << endl;

}

cout << "\n";

break;

}

case 2: {

system("cls");

if (current\_shooter > 0)

{

string title;

cout << "Введите название игры:" << endl;

cin >> title;

for (std::list<ShooterGame\*>::iterator i = shooterGames.begin(); i != shooterGames.end(); i++)

{

if ((\*i)->getTitle() == title)

{

cout << "Найденная игра\n" << endl;

(\*i)->displayShooter();

}

else { cout << "Нет такой игры"; }

}

}

else

{

cout << "Нет игр" << endl;

}

cout << "\n";

break;

}

case 3: {

system("cls");

if (current\_fantasy > 0)

{

string title;

cout << "Введите название игры:" << endl;

cin >> title;

for (std::array<FantasyGame\*, 5>::iterator i = fantasyGames.begin(); i != fantasyGames.end(); i++)

{

if ((\*i)->getTitle() == title)

{

cout << "Найденная игра\n" << endl;

(\*i)->displayFantasy();

}

else { cout << "Нет такой игры"; }

}

}

else

{

cout << "Нет игр" << endl;

}

cout << "\n";

break;

}

}

} while (choice1 != 0);

break;

}

case 4: {

system("cls");

int choice1;

do {

cout << "\nМеню выбора" << endl;

cout << "1 - Удалить гонку" << endl;

cout << "2 - Удалить шутер" << endl;

cout << "3 - Удалить фэнтези" << endl;

cout << "0 - Выйти" << endl;

while (true) {

cout << "Выберите действие: ";

cin >> choice1;

if (cin.good() && choice1 >= 0 && choice1 <= 3) {

break;

}

cin.clear();

cout << "Неправильный ввод данных\n";

cin.ignore(100, '\n');

}

switch (choice1) {

case 1: {

system("cls");

if (current\_race > 0)

{

std::vector<RaceGame\*>::const\_iterator p = raceGames.cbegin();

while (p != raceGames.cend())

{

(\*p)->displayRace();

p++;

}

cout << "\n";

string name\_poi;

cout << "Введите название игры, которую хотите удалить" << endl;

cin >> name\_poi;

int x = 0;

int flag = 0;

for (std::vector<RaceGame\*>::iterator i = raceGames.begin(); i != raceGames.end(); i++)

{

if ((\*i)->getTitle() == name\_poi)

{

flag = 1;

raceGames.erase(raceGames.begin() + x);

cout << "Игра удалена" << endl;

current\_race;

break;

}

x++;

}

if (flag == 0)

{

cout << "Такой игры нет" << endl;

}

std::vector<RaceGame\*>::iterator i = raceGames.begin();

}

else

{

cout << "Отсутствуют гонки" << endl;

}

cout << "\n";

break;

}

case 2: {

system("cls");

if (current\_shooter > 0)

{

std::list<ShooterGame\*>::const\_iterator p = shooterGames.cbegin();

while (p != shooterGames.cend())

{

(\*p)->displayShooter();

p++;

}

cout << "\n";

string name\_poi;

cout << "Введите название игры, которую вы хотите удалить" << endl;

cin >> name\_poi;

int x = 0;

int flag = 0;

for (std::list<ShooterGame\*>::iterator i = shooterGames.begin(); i != shooterGames.end(); i++)

{

if ((\*i)->getTitle() == name\_poi)

{

flag = 1;

shooterGames.erase(i);

cout << "Игра удалена" << endl;

current\_shooter--;

break;

}

x++;

}

if (flag == 0)

{

cout << "Такой игры нет" << endl;

}

}

else

{

cout << "Отсутствуют игры" << endl;

}

cout << "\n";

break;

}

case 3: {

system("cls");

if (current\_fantasy > 0)

{

for (int i = 0; i < current\_fantasy; i++)

{

fantasyGames[i]->displayFantasy();

}

cout << "\n";

string name\_poi;

cout << "Введите название игры, который хотите удалить" << endl;

cin >> name\_poi;

int x = 0;

for (int i = 0; i < current\_fantasy; i)

{

if (fantasyGames[i]->getTitle() == name\_poi)

{

for (int j = i; j < current\_fantasy; j)

{

fantasyGames[j] = fantasyGames[j + 1];

}

cout << "Игра удалена" << endl;

current\_fantasy--;

x;

break;

}

}

if (x == 0)

{

cout << "Такой игры нет" << endl;

}

}

else

{

cout << "Отсутствуют игры" << endl;

}

cout << "\n";

break;

}

}

} while (choice1 != 0);

break;

}

case 5: {

system("cls");

int choice1;

do {

cout << "\nМеню выбора" << endl;

cout << "1 - Изменить название гонки" << endl;

cout << "2 - Изменить название шутера" << endl;

cout << "3 - Изменить название фэнтези" << endl;

cout << "0 - Выйти" << endl;

while (true) {

cout << "Выберите действие: ";

cin >> choice1;

if (cin.good() && choice1 >= 0 && choice1 <= 3) {

break;

}

cin.clear();

cout << "Неправильный ввод данных\n";

cin.ignore(100, '\n');

}

switch (choice1) {

case 1: {

system("cls");

if (current\_race > 0)

{

std::vector<RaceGame\*>::const\_iterator p = raceGames.cbegin();

while (p != raceGames.cend())

{

(\*p)->displayRace();

p++;

}

string name\_poi;

cout << "Введите название, которое хотите изменить:" << endl;

cin >> name\_poi;

int poi = 0;

for (std::vector<RaceGame\*>::iterator i = raceGames.begin(); i != raceGames.end(); i++)

{

if ((\*i)->getTitle() == name\_poi)

{

poi = 1;

std::string name\_new;

cout << "Введите новое название игры:" << endl;

cin >> name\_new;

(\*i)->setTitle(name\_new);

cout << "Название изменено" << endl;

break;

}

}

if (poi == 0)

{

cout << "Такой игры нет" << endl;

}

std::vector<RaceGame\*>::iterator i = raceGames.begin();

}

else

{

cout << "Отсутствуют игры" << endl;

}

cout << "\n";

break;

}

case 2: {

system("cls");

if (current\_shooter > 0)

{

std::list<ShooterGame\*>::const\_iterator it = shooterGames.cbegin();

while (it != shooterGames.cend())

{

(\*it)->displayShooter();

it++;

}

cout << "\n";

string name\_poi;

cout << "Введите название игры, которое хотите изменить:" << endl;

cin >> name\_poi;

int poi = 0;

for (std::list<ShooterGame\*>::iterator i = shooterGames.begin(); i != shooterGames.end(); i++)

{

if ((\*i)->getTitle() == name\_poi)

{

string name\_new;

poi = 1;

cout << "Введите новое название" << endl;

cin >> name\_new;

(\*i)->setTitle(name\_new);

cout << "Название изменено" << endl;

break;

}

}

if (poi == 0)

{

cout << "Такой игры нет" << endl;

}

}

else

{

cout << "Отсутствуют игры" << endl;

}

cout << "\n";

break;

}

case 3: {

system("cls");

if (current\_fantasy > 0)

{

for (int i = 0; i < current\_fantasy; i++)

{

fantasyGames[i]->displayFantasy();

}

cout << "\n";

string name\_poi;

cout << "Введите название, которое хотите изменить:" << endl;

cin >> name\_poi;

int poi = 0;

for (int i = 0; i < current\_fantasy; i++)

{

if (fantasyGames[i]->getTitle() == name\_poi)

{

string name\_new;

poi = 1;

cout << "Введите новое название:" << endl;

cin >> name\_new;

fantasyGames[i]->setTitle(name\_new);

cout << "Название изменено" << endl;

break;

}

}

if (poi == 0)

{

cout << "Такой игры нет" << endl;

}

}

else

{

cout << "Отсутствуют игры" << endl;

}

cout << "\n";

break;

}

}

} while (choice1 != 0);

break;

}

case 6: {

system("cls");

int choice1;

do {

cout << "\nМеню выбора" << endl;

cout << "1 - Отсортировать гонки по названию" << endl;

cout << "2 - Отсортировать шутеры по году выпуска" << endl;

cout << "3 - Отсортировать фэнтези по названию мира" << endl;

cout << "0 - Выйти" << endl;

while (true) {

cout << "Выберите действие: ";

cin >> choice1;

if (cin.good() && choice1 >= 0 && choice1 <= 3) {

break;

}

cin.clear();

cout << "Неправильный ввод данных\n";

cin.ignore(100, '\n');

}

switch (choice1) {

case 1: {

system("cls");

if (current\_race > 0)

{

std::list<std::string> sort;

for (std::vector<RaceGame\*>::iterator it = raceGames.begin(); it != raceGames.end(); it++)

{

sort.push\_back((\*it)->getTitle());

}

sort.sort();

cout << "Отсортированный список:" << endl;

for (std::list<std::string>::iterator it = sort.begin(); it != sort.end(); it++)

{

for (std::vector<RaceGame\*>::iterator i = raceGames.begin(); i != raceGames.end(); i++)

{

if ((\*i)->getTitle() == \*it)

{

(\*i)->displayRace();

}

}

}

}

else

{

cout << "Отсутствуют игры" << endl;

}

cout << "\n";

break;

}

case 2: {

system("cls");

if (current\_shooter > 0)

{

std::list<int> sort;

for (std::list<ShooterGame\*>::iterator it = shooterGames.begin(); it != shooterGames.end(); it++)

{

sort.push\_back((\*it)->getYear());

}

sort.sort();

cout << "Отсортированный список:" << endl;

for (std::list<int>::iterator it = sort.begin(); it != sort.end(); it++)

{

for (std::list<ShooterGame\*>::iterator i = shooterGames.begin(); i != shooterGames.end(); i++)

{

if ((\*i)->getYear() == \*it)

{

(\*i)->displayShooter();

}

}

}

}

else

{

cout << "Отсутствуют шутеры" << endl;

}

cout << "\n";

break;

}

case 3: {

system("cls");

if (current\_fantasy > 0)

{

std::string\* temporary = new std::string[current\_fantasy];

for (int i = 0; i < current\_fantasy; i++) {

temporary[i] = fantasyGames[i]->getName();

}

for (int i = 0; i < current\_fantasy; i++) {

for (int j = current\_fantasy - 1; j > i; j--) {

if (temporary[j - 1] > temporary[j]) {

swap(temporary[j - 1], temporary[j]);

}

}

}

for (int i = 0; i < current\_fantasy; i++) {

for (int j = 0; j < current\_fantasy; j++)

{

if (temporary[i] == fantasyGames[j]->getName())

{

fantasyGames[j]->displayFantasy();

}

}

}

}

else

{

cout << "Отсутствуют игры" << endl;

}

cout << "\n";

break;

}

}

} while (choice1 != 0);

break;

}

case 7: {

system("cls");

std::ofstream File;

File.open(GAMESFILENAME, std::ios::binary);

if (File.is\_open())

{

File << "-------------------------------------------------------------------------------------\n";

File << "| Гонки |\n";

File << "-------------------------------------------------------------------------------------\n";

File << '|' << setw(25) << left << "Название игры" << '|' << setw(20) << "Издатель " << '|' << setw(10) << "Год" << '|' << setw(25) << "Тип гонок" << '|' << endl;

File << "-------------------------------------------------------------------------------------\n";

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

{

File << '|' << setw(25) << left << raceGames[i]->getTitle() << '|' << setw(20)

<< raceGames[i]->getCompany() << '|' << setw(10) << raceGames[i]->getYear() << "|" << setw(25) << raceGames[i]->getRaceType() << "|" << endl;

File << "-------------------------------------------------------------------------------------\n";

}

File << "-------------------------------------------------------------------------------------\n";

File << "| Шутеры |\n";

File << "-------------------------------------------------------------------------------------\n";

File << '|' << setw(25) << left << "Название игры" << '|' << setw(20) << "Издатель " << '|' << setw(10) << "Год" << '|' << setw(25) << "Тип шутера" << '|' << endl;

File << "-------------------------------------------------------------------------------------\n";

for (std::list<ShooterGame\*>::iterator p = shooterGames.begin(); p != shooterGames.end(); ++p)

{

File << '|' << setw(25) << left << (\*p)->getTitle() << '|' << setw(20) << (\*p)->getCompany() << '|' << setw(10) << (\*p)->getYear() << "|" << setw(25) << (\*p)->getShooterType() << "|" << endl;

File << "-------------------------------------------------------------------------------------\n";

}

File << "-------------------------------------------------------------------------------------\n";

File << "| Фэнтези |\n";

File << "-------------------------------------------------------------------------------------\n";

File << '|' << setw(25) << left << "Название игры" << '|' << setw(20) << "Издатель " << '|' << setw(10) << "Год" << '|' << setw(25) << " Название мира" << '|' << endl;

File << "-------------------------------------------------------------------------------------\n";

for (int i = 0; i < current\_fantasy; i++)

{

File << '|' << setw(25) << left << fantasyGames[i]->getTitle() << '|' << setw(20) << fantasyGames[i]->getCompany() << '|' << setw(10) << fantasyGames[i]->getYear() << "|" << setw(25) << fantasyGames[i]->getName() << "|" << endl;

File << "-------------------------------------------------------------------------------------\n";

File << "\n";

}

std::cout << "Данные успешно записаны в файл\n";

}

else std::cout << "Ошибка открытия файла\n";

File.close();

break;

}

}

}

catch (invalid\_argument& e) {

cout << e.what() << endl;

}

catch (MyException& e) {

cout << e.what() << endl;

}

catch (MyExceptionComputer& e) {

cout << e.what() << endl;

}

} while (choice != 0);

return 0;

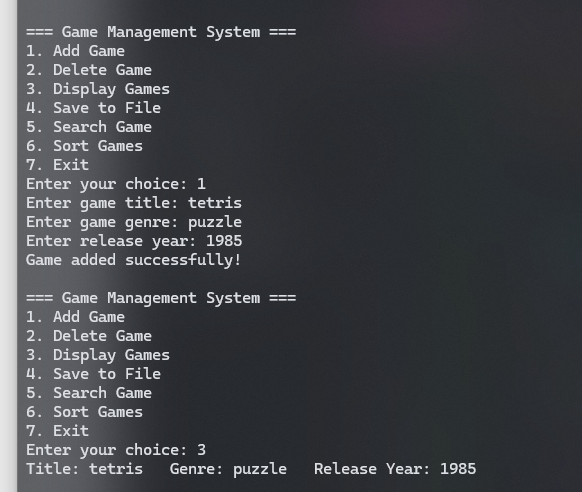
} 

Рисунок 2 – Ввод информации

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