МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Курсовая работа

по дисциплине: Объектно-ориентированное программирование

тема: «Календарь»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверил:
асс. Черников Сергей Викторович

Белгород 2024г.

Содержание

Цель работы: приобретение практических навыков создания приложений на языке C++.

Задание: Создать приложение реализующее календарь.

Постановка задачи:

Разработать консольное приложение на языке C++, которое позволяет пользователям просматривать календарь, добавлять, удалять и просматривать события, а также получать уведомления о событиях.

Основные требования

1. Добавление события - Пользователь может добавить новое событие через интерфейс, который взаимодействует с классом Calendar для добавления события в EventManager.
2. Удаление события - Пользователь может удалить существующее событие, используя интерфейс. Удаление происходит через методы Calendar и EventManager.
3. Просмотр событий на дату - Пользователь может просмотреть события, запланированные на конкретную дату. Это реализовано через методы Calendar и EventManager.
4. Просмотр всех событий - Пользователь может просмотреть все запланированные события. Этот функционал реализован через интерфейс, который вызывает методы Calendar и EventManager.
5. Обновление события - Пользователь может обновить существующее событие. Этот функционал также доступен через интерфейс, взаимодействующий с Calendar и EventManager.
6. Показ месячного календаря - Пользователь может просмотреть календарь на определенный месяц, который будет отображать все события на этот месяц.

7. Уведомления о событиях- Система уведомляет пользователя о событиях, которые должны произойти в ближайшее время. Это реализовано через класс Notification, который работает в отдельном потоке и взаимодействует с EventManager для проверки текущих событий.
8. Сохранение событий в файл - Пользователь может сохранить все события в файл. Этот функционал реализован в классе EventManager, который использует FileManager для выполнения операций ввода-вывода.
9. Загрузка событий из файла- Пользователь может загрузить события из файла. Это также реализовано в классе EventManager с использованием FileManager.
10. Отображение текущей даты и времени - Программа отображает текущую дату и время. Этот функционал реализован в классе Time.

Диаграммы

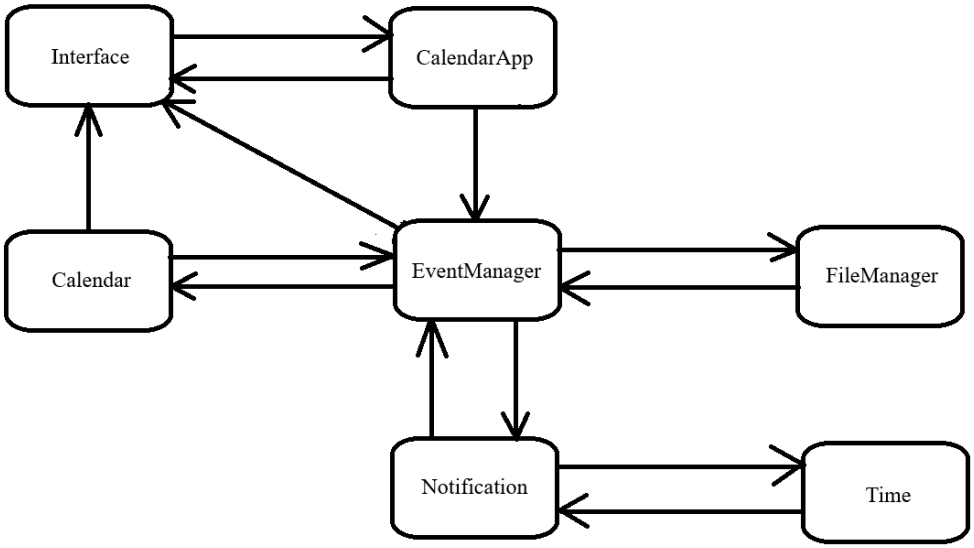

Рисунок 1 - Объектная декомпозиция

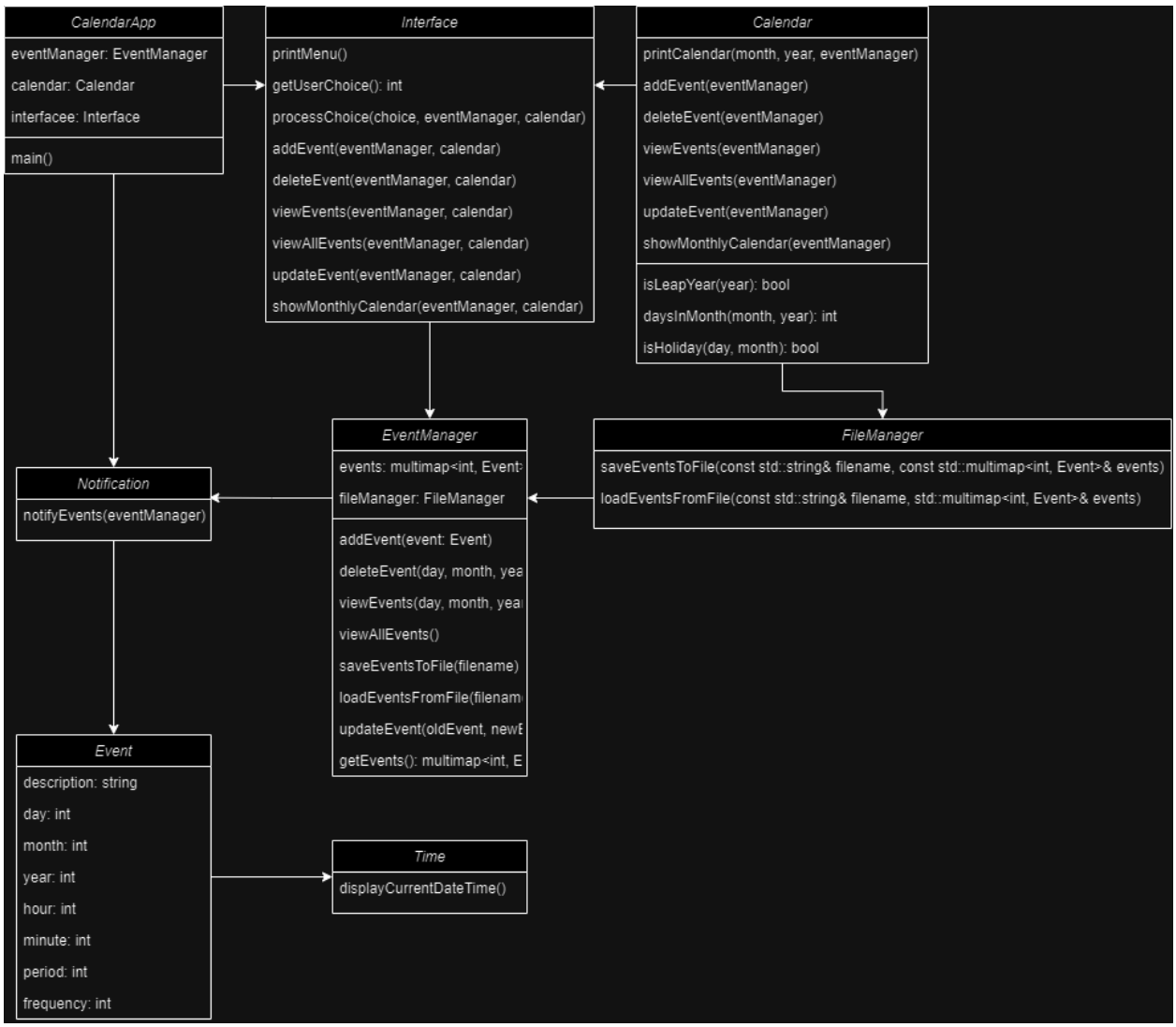

Рисунок 2 - Диаграмма классов

Паттерны проектирования

Singleton (EventManager) - обеспечивает наличие единственного экземпляра класса для управления событиями по всему приложению.

Observer (Notification) - работает в отдельном потоке и проверяет события в EventManager, уведомляя пользователя при наступлении события.

Command (Interface) - обрабатывает пользовательские команды и вызывает соответствующие методы для выполнения действий, таких как добавление, удаление и просмотр событий.

Facade (Interface) - упрощает взаимодействие пользователя с приложением, предоставляя методы для выполнения основных действий.

Multithreading (Notification) - обеспечивает выполнение фоновых задач, таких как уведомления, без блокировки основного потока приложения.

DAO (FileManager) - абстрагирует операции ввода-вывода, предоставляя методы для сохранения и загрузки событий.

Листинг программы:

```
#include "EventManager.h" // Для работы с событиями
#include "Calendar.h" // Для отображения календаря
#include "Notification.h" // Для уведомлений
#include "Time.h" // Для отображения текущего времени
#include "Interface.h" // Для работы с интерфейсом
#include <iostream>
#include <locale>
#include <thread>

int main() {
    SetConsoleOutputCP(CP_UTF8);

    EventManager eventManager;
    Calendar calendar;
    Interface interfacee;

    std::thread notifyThread(Notification::notifyEvents,
std::ref(eventManager)); // Запуск потока для уведомлений
    notifyThread.detach();

    while (true) {
        Time::displayCurrentDateTime(); // Отображение текущего времени
        interfacee.printMenu(); // Отображение меню
        int choice;
        std::cin >> choice;
        interfacee.processChoice(choice, eventManager, calendar); // Обработ-
ка выбора пользователя
        if (choice == 0) // Выход из цикла при выборе 0
            break;
    }

    return 0;
}
```

Листинг 1 – CalendarApp.cpp

```
#ifndef CALENDAR_H
#define CALENDAR_H

#include "EventManager.h"
#include <windows.h>
#include <iostream>
#include <cstdio>
#include <vector>
#include <utility>

class Calendar {
public:
    // Вывод календаря на указанный месяц и год с учетом событий
    void printCalendar(int month, int year, const EventManager& eventManager)
const;

    // Добавление события в календарь
    void addEvent(EventManager& eventManager);

    // Удаление события из календаря
    void deleteEvent(EventManager& eventManager);

    // Просмотр событий на указанную дату
    void viewEvents(EventManager& eventManager);
```

```
    // Просмотр всех событий
    void viewAllEvents(EventManager& eventManager);

    // Обновление события
    void updateEvent(EventManager& eventManager);

    // Отображение календаря на месяц с учетом событий
    void showMonthlyCalendar(EventManager& eventManager);

private:
    // Проверка, является ли указанный год високосным
    bool isLeapYear(int year) const;

    // Получение количества дней в указанном месяце и году
    int daysInMonth(int month, int year) const;

    // Проверка, является ли указанный день и месяц праздничным
    bool isHoliday(int day, int month) const;
};

#endif // CALENDAR_H
```

Листинг 2 – Calendar.h

```
#include "Calendar.h"
#include "EventManager.h"
#include <chrono>
#include <windows.h>
#include <iostream>
#include <vector>
#include <string>

// Добавление события в календарь
void Calendar::addEvent(EventManager &eventManager) {
    Event event;
    std::cout << "Введите описание события: ";
    std::cin.ignore();
    std::getline(std::cin, event.description);
    std::cout << "Введите день: ";
    std::cin >> event.day;
    std::cout << "Введите месяц: ";
    std::cin >> event.month;
    std::cout << "Введите год: ";
    std::cin >> event.year;
    std::cout << "Введите час: ";
    std::cin >> event.hour;
    std::cout << "Введите минуты: ";
    std::cin >> event.minute;
    std::cout << "Введите период (0 для нет): ";
    std::cin >> event.period;
    std::cout << "Введите частоту (0 для нет): ";
    std::cin >> event.frequency;

    // Проверка, является ли день праздничным
    if (isHoliday(event.day, event.month)) {
        std::cout << "Этот день является праздничным!" << std::endl;
    }

    eventManager.addEvent(event);
}

// Удаление события из календаря
```

```
void Calendar::deleteEvent(EventManager &eventManager) {
    int day, month, year, hour, minute;
    std::string description;
    std::cout << "Введите день события: ";
    std::cin >> day;
    std::cout << "Введите месяц события: ";
    std::cin >> month;
    std::cout << "Введите год события: ";
    std::cin >> year;
    std::cout << "Введите час события: ";
    std::cin >> hour;
    std::cout << "Введите минуты события: ";
    std::cin >> minute;
    std::cout << "Введите описание события: ";
    std::cin.ignore();
    std::getline(std::cin, description);

    eventManager.deleteEvent(day, month, year, hour, minute, description);
}

// Просмотр событий на указанную дату
void Calendar::viewEvents(EventManager &eventManager) {
    int day, month, year;
    std::cout << "Введите день: ";
    std::cin >> day;
    std::cout << "Введите месяц: ";
    std::cin >> month;
    std::cout << "Введите год: ";
    std::cin >> year;
    eventManager.viewEvents(day, month, year);
}

// Просмотр всех событий
void Calendar::viewAllEvents(EventManager &eventManager) {
    eventManager.viewAllEvents();
}

// Обновление события
void Calendar::updateEvent(EventManager &eventManager) {
    Event oldEvent, newEvent;
    std::cout << "Введите данные текущего события для обновления:" <<
std::endl;
    std::cout << "Описание: ";
    std::cin.ignore();
    std::getline(std::cin, oldEvent.description);
    std::cout << "День: ";
    std::cin >> oldEvent.day;
    std::cout << "Месяц: ";
    std::cin >> oldEvent.month;
    std::cout << "Год: ";
    std::cin >> oldEvent.year;
    std::cout << "Час: ";
    std::cin >> oldEvent.hour;
    std::cout << "Минуты: ";
    std::cin >> oldEvent.minute;
    std::cout << "Период: ";
    std::cin >> oldEvent.period;
    std::cout << "Частота: ";
    std::cin >> oldEvent.frequency;

    std::cout << "Введите новые данные события:" << std::endl;
    std::cout << "Описание: ";
    std::cin.ignore();
    std::getline(std::cin, newEvent.description);
```

```
    std::cout << "День: ";
    std::cin >> newEvent.day;
    std::cout << "Месяц: ";
    std::cin >> newEvent.month;
    std::cout << "Год: ";
    std::cin >> newEvent.year;
    std::cout << "Час: ";
    std::cin >> newEvent.hour;
    std::cout << "Минуты: ";
    std::cin >> newEvent.minute;
    std::cout << "Период: ";
    std::cin >> newEvent.period;
    std::cout << "Частота: ";
    std::cin >> newEvent.frequency;

    eventManager.updateEvent(oldEvent, newEvent);
}

// Отображение календаря на месяц
void Calendar::showMonthlyCalendar(EventManager &eventManager) {
    int month, year;
    std::cout << "Введите месяц: ";
    std::cin >> month;
    std::cout << "Введите год: ";
    std::cin >> year;
    printCalendar(month, year, eventManager);
}

// Проверка, является ли год високосным
bool Calendar::isLeapYear(int year) const {
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

// Получение количества дней в месяце
int Calendar::daysInMonth(int month, int year) const {
    switch (month) {
        case 2:
            return isLeapYear(year) ? 29 : 28;
        case 4:
        case 6:
        case 9:
        case 11:
            return 30;
        default:
            return 31;
    }
}

// Отображение календаря на месяц
void Calendar::printCalendar(int month, int year, const EventManager
&eventManager) const {
    std::cout << "   Календарь для " << month << "/" << year << std::endl;
    std::cout << "---------------------------" << std::endl;
    std::cout << "  Пн  Вт  Ср  Чт  Пт  Сб  Вс" << std::endl;

    // Определение дня недели, с которого начинается месяц
    int startDay = 1; // Пусть 1 января 1 года - понедельник
    for (int i = 1; i < year; ++i) {
        startDay += isLeapYear(i) ? 366 : 365;
    }
    for (int i = 1; i < month; ++i) {
        startDay += daysInMonth(i, year);
    }
    startDay %= 7;
```

```
    startDay = (startDay + 6) % 7;

    // Вывод пустых ячеек для первой недели
    for (int i = 0; i < startDay; ++i) {
        std::cout << "    ";
    }

    // Установка кодировки для корректного вывода кириллицы в консоли Windows
    SetConsoleOutputCP(1251);

    int totalDays = daysInMonth(month, year);
    for (int day = 1; day <= totalDays; ++day) {
        printf("%4d", day);
        auto range = eventManager.getEvents().equal_range(day);
        for (auto it = range.first; it != range.second; ++it) {
            const Event &event = it->second;
            if (event.month == month && event.year == year) {
                std::cout << "\n    - " << event.description << " (" <<
event.hour << ":" << event.minute << ")";
            }
        }
        startDay++;
        if (startDay > 6) {
            startDay = 0;
            std::cout << std::endl;
        }
    }

    // Восстановление кодировки по умолчанию
    SetConsoleOutputCP(CP_UTF8);
    std::cout << std::endl;
}

// Проверка, является ли указанный день и месяц праздничным
bool Calendar::isHoliday(int day, int month) const {
    const std::vector<std::pair<int, int>> holidays = {
            {1,  1},
            {7,  1},
            {23, 2},
            {8,  3},
            {1,  5},
            {9,  5},
            {12, 6},
            {4,  11}
    };
    return std::find(holidays.begin(), holidays.end(), std::make_pair(day,
month)) != holidays.end();
}
```

Листинг 3 – Calendar.cpp

```
#ifndef EVENT_H
#define EVENT_H

#include <string>

// Структура представляющая событие
struct Event {
    std::string description; // Описание события
    int day, month, year; // День, месяц и год события
```

```
    int hour, minute; // Часы и минуты события
    int period, frequency; // Период и частота повторения события

    // Перегрузка оператора '<' для сравнения событий по времени
    bool operator<(const Event& other) const;

    // Перегрузка оператора '==' для сравнения событий на равенство
    bool operator==(const Event& other) const;
};

#endif // EVENT_H
```

Листинг 4 – Event.h

```
#include "Event.h"
#include <tuple>

// Перегрузка оператора '<' для сравнения событий
bool Event::operator<(const Event& other) const {
    return std::tie(year, month, day, hour, minute) <
           std::tie(other.year, other.month, other.day, other.hour, oth-
er.minute);
}

// Перегрузка оператора '==' для сравнения событий
bool Event::operator==(const Event& other) const {
    return std::tie(description, day, month, year, hour, minute, period, fre-
quency) ==
           std::tie(other.description, other.day, other.month, other.year,
other.hour, other.minute, other.period, other.frequency);
}
```

Листинг 5 – Event.cpp

```
#ifndef EVENTMANAGER_H
#define EVENTMANAGER_H

#include <map>
#include <string>
#include "Event.h"
#include "FileManager.h"

// Класс для управления событиями
class EventManager {
public:
    // Добавление события в менеджер событий
    void addEvent(const Event& event);

    // Удаление события из менеджера событий
    void deleteEvent(int day, int month, int year, int hour, int minute,
const std::string& description);

    // Просмотр событий на указанную дату
    void viewEvents(int day, int month, int year) const;

    // Просмотр всех событий
    void viewAllEvents() const;

    // Сохранение событий в файл
    void saveEventsToFile(const std::string& filename) const;

    // Загрузка событий из файла
    void loadEventsFromFile(const std::string& filename);
```

```
    // Обновление события в менеджере событий
    void updateEvent(const Event& oldEvent, const Event& newEvent);

    // Получение всех событий
    const std::multimap<int, Event>& getEvents() const;

private:
    std::multimap<int, Event> events_; // Мультимап для хранения событий с
ключом по дню
    FileManager fileManager_; // Менеджер файлов для сохранения и загрузки
событий
};

#endif // EVENTMANAGER_H
```

Листинг 6 – EventManager.h

```
#include "EventManager.h"

// Добавление события в менеджер событий
void EventManager::addEvent(const Event& event) {
    events_.insert({event.day, event});
}

// Удаление события из менеджера событий
void EventManager::deleteEvent(int day, int month, int year, int hour, int
minute, const std::string& description) {
    auto range = events_.equal_range(day);
    for (auto it = range.first; it != range.second; ++it) {
        const Event& event = it->second;
        if (event.day == day && event.month == month && event.year == year &&
            event.hour == hour && event.minute == minute && event.description
== description) {
            events_.erase(it);
            std::cout << "Мероприятие успешно удалено." << std::endl;
            return;
        }
    }
    std::cout << "Событие не найдено." << std::endl;
}

// Просмотр событий на указанную дату
void EventManager::viewEvents(int day, int month, int year) const {
    std::cout << "Мероприятия для " << day << "/" << month << "/" << year <<
":" << std::endl;
    auto range = events_.equal_range(day);
    for (auto it = range.first; it != range.second; ++it) {
        const Event& event = it->second;
        if (event.month == month && event.year == year) {
            std::cout << "- " << event.description << " (" << event.hour <<
":" << event.minute << ")" << std::endl;
        }
    }
}

// Просмотр всех событий
void EventManager::viewAllEvents() const {
    std::cout << "Все события отсортированы по дате и времени:" << std::endl;
    for (const auto& pair : events_) {
        const Event& event = pair.second;
        std::cout << event.day << "/" << event.month << "/" << event.year <<
" _ "
                  << event.hour << "/" << event.minute << " - " <<
```

```
event.description << std::endl;
    }
}

// Сохранение событий в файл
void EventManager::saveEventsToFile(const std::string& filename) const {
    fileManager_.saveEventsToFile(filename, events_);
}

// Загрузка событий из файла
void EventManager::loadEventsFromFile(const std::string& filename) {
    fileManager_.loadEventsFromFile(filename, events_);
}

// Обновление события в менеджере событий
void EventManager::updateEvent(const Event& oldEvent, const Event& newEvent)
{
    auto range = events_.equal_range(oldEvent.day);
    for (auto it = range.first; it != range.second; ++it) {
        const Event& event = it->second;
        if (event == oldEvent) {
            events_.erase(it);
            events_.insert({newEvent.day, newEvent});
            return;
        }
    }
}

// Получение всех событий
const std::multimap<int, Event>& EventManager::getEvents() const {
    return events_;
}
```

Листинг 7 – EventManager.cpp

```
#ifndef CALENDARCONSOL_FILEMANAGER_H
#define CALENDARCONSOL_FILEMANAGER_H

#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <map>
#include "Event.h"

// Класс для работы с файлами, сохранения и загрузки событий
class FileManager {
public:
    // Сохранение событий в файл
    void saveEventsToFile(const std::string& filename, const
std::multimap<int, Event>& events) const;

    // Загрузка событий из файла
    void loadEventsFromFile(const std::string& filename, std::multimap<int,
Event>& events) const;
};

#endif //CALENDARCONSOL_FILEMANAGER_H
```

Листинг 8 – FileManager.h

```
#include "FileManager.h"

// Сохранение событий в файл
```

```
void FileManager::saveEventsToFile(const std::string& filename, const
std::multimap<int, Event>& events) const {
    std::ofstream outFile(filename);
    if (!outFile) {
        std::cerr << "Ошибка открытия файла для записи: " << filename <<
std::endl;
        return;
    }

    // Запись событий в файл
    for (const auto& pair : events) {
        const Event& event = pair.second;
        outFile << event.day << "/" << event.month << "/" << event.year << "
- "
                << event.hour << "/" << event.minute << " - " <<
event.description << " - "
                << event.period << " - " << event.frequency << std::endl;
    }

    std::cout << "События сохранены в " << filename << std::endl;
}

// Загрузка событий из файла
void FileManager::loadEventsFromFile(const std::string& filename,
std::multimap<int, Event>& events) const {
    std::ifstream inFile(filename);
    if (!inFile) {
        std::cerr << "Ошибка открытия файла для чтения: " << filename <<
std::endl;
        return;
    }

    std::string line;
    while (std::getline(inFile, line)) {
        std::istringstream iss(line);
        int day, month, year, hour, minute, period, frequency;
        char sep1, sep2, sep3, sep4, sep5, sep6, sep7;
        std::string description;
        // Чтение данных из файла и создание события
        if (iss >> day >> sep1 >> month >> sep2 >> year >> sep3 >> hour >>
sep4 >> minute >> sep5 >> period >> sep6 >> frequency >> sep7) {
            std::getline(iss, description);
            description = description.substr(1); // Удаление ведущего пробела
            events.insert({day, Event{description, day, month, year, hour,
minute, period, frequency}});
        }
    }

    std::cout << "События загружены из " << filename << std::endl;
}
```

Листинг 9 – FileManager.cpp

```
#ifndef CALENDARCONSOL_INTERFACE_H
#define CALENDARCONSOL_INTERFACE_H

#include "EventManager.h"
#include "Calendar.h"
#include <iostream>

// Класс для взаимодействия с пользователем через консольный интерфейс
class Interface {
public:
    // Вывод меню на экран
```

```
    static void printMenu();

    // Получение выбора пользователя
    static int getUserChoice();

    // Обработка выбора пользователя
    static void processChoice(int choice, EventManager& eventManager, Calen-
dar& calendar);

    // Добавление события через интерфейс
    static void addEvent(EventManager& eventManager, Calendar& calendar);

    // Удаление события через интерфейс
    static void deleteEvent(EventManager& eventManager, Calendar& calendar);

    // Просмотр событий на указанную дату через интерфейс
    static void viewEvents(EventManager& eventManager, Calendar& calendar);

    // Просмотр всех событий через интерфейс
    static void viewAllEvents(EventManager& eventManager, Calendar& calen-
dar);

    // Обновление события через интерфейс
    static void updateEvent(EventManager& eventManager, Calendar& calendar);

    // Показ календаря на месяц через интерфейс
    static void showMonthlyCalendar(EventManager& eventManager, Calendar&
calendar);
};

#endif //CALENDARCONSOL_INTERFACE_H
```

Листинг 10 – Interface.h

```
#include "Interface.h"

// Добавление события через интерфейс
void Interface::addEvent(EventManager& eventManager, Calendar& calendar) {
    calendar.addEvent(eventManager);
}

// Удаление события через интерфейс
void Interface::deleteEvent(EventManager& eventManager, Calendar& calendar) {
    calendar.deleteEvent(eventManager);
}

// Просмотр событий на указанную дату через интерфейс
void Interface::viewEvents(EventManager& eventManager, Calendar& calendar) {
    calendar.viewEvents(eventManager);
}

// Просмотр всех событий через интерфейс
void Interface::viewAllEvents(EventManager& eventManager, Calendar& calendar)
{
    calendar.viewAllEvents(eventManager);
}

// Обновление события через интерфейс
void Interface::updateEvent(EventManager& eventManager, Calendar& calendar) {
    calendar.updateEvent(eventManager);
}

// Показ календаря на месяц через интерфейс
void Interface::showMonthlyCalendar(EventManager& eventManager, Calendar&
```

```
calendar) {
    calendar.showMonthlyCalendar(eventManager);
}

// Вывод меню на экран
void Interface::printMenu() {
    std::cout << "Меню:" << std::endl;
    std::cout << "1. Добавить событие" << std::endl;
    std::cout << "2. Удалить событие" << std::endl;
    std::cout << "3. Просмотреть события на дату" << std::endl;
    std::cout << "4. Просмотреть все события" << std::endl;
    std::cout << "5. Сохранить события в файл" << std::endl;
    std::cout << "6. Загрузить события из файла" << std::endl;
    std::cout << "7. Обновить событие" << std::endl;
    std::cout << "8. Показать календарь на месяц" << std::endl;
    std::cout << "0. Выйти" << std::endl;
    std::cout << "Введите ваш выбор: ";
}

// Получение выбора пользователя
int Interface::getUserChoice() {
    int choice;
    std::cin >> choice;
    return choice;
}

// Обработка выбора пользователя
void Interface::processChoice(int choice, EventManager& eventManager,
Calendar& calendar) {
    switch (choice) {
        case 1: // Добавить событие
            calendar.addEvent(eventManager);
            break;
        case 2: // Удалить событие
            calendar.deleteEvent(eventManager);
            break;
        case 3: // Просмотреть события на дату
            calendar.viewEvents(eventManager);
            break;
        case 4: // Просмотреть все события
            calendar.viewAllEvents(eventManager);
            break;
        case 5: // Сохранить события в файл
            eventManager.saveEventsToFile("events.txt");
            break;
        case 6: // Загрузить события из файла
            eventManager.loadEventsFromFile("events.txt");
            break;
        case 7: // Обновить событие
            calendar.updateEvent(eventManager);
            break;
        case 8: // Показать календарь на месяц
            calendar.showMonthlyCalendar(eventManager);
            break;
        case 0: // Выйти
            std::cout << "Выход из программы..." << std::endl;
            break;
        default:
            std::cout << "Неправильный выбор. Попробуйте еще раз." <<
std::endl;
    }
}
```

Листинг 11 – Interface.cpp

```
#ifndef CALENDARCONSOL_NOTIFICATION_H
#define CALENDARCONSOL_NOTIFICATION_H

#include "EventManager.h"

// Класс для отправки уведомлений о событиях
class Notification {
public:
    static void notifyEvents(EventManager& eventManager); // Объявление функ-
ции для отправки уведомлений
};

#endif //CALENDARCONSOL_NOTIFICATION_H
```

Листинг 12 – Notification.h

```
#include "Notification.h"
#include <chrono>
#include <ctime>
#include <windows.h>
#include <ShObjIdl.h>
#include <thread>

// Функция для отправки уведомлений о событиях
void Notification::notifyEvents(EventManager& eventManager) {
    CoInitialize(NULL); // Инициализация COM

    while (true) {
        auto now = std::chrono::system_clock::now();
        std::time_t now_c = std::chrono::system_clock::to_time_t(now);
        std::tm* now_tm = std::localtime(&now_c);

        // Получение текущей даты и времени
        int currentDay = now_tm->tm_mday;
        int currentMonth = now_tm->tm_mon + 1;
        int currentYear = now_tm->tm_year + 1900;
        int currentHour = now_tm->tm_hour;
        int currentMinute = now_tm->tm_min;

        // Проверка наличия событий в текущий момент времени
        auto range = eventManager.getEvents().equal_range(currentDay);
        for (auto it = range.first; it != range.second; ++it) {
            const Event& event = it->second;
            // Если событие соответствует текущему времени
            if (event.month == currentMonth && event.year == currentYear &&
                event.hour == currentHour && event.minute == currentMinute) {
                // Преобразование описания события в тип wstring для исполь-
зования в уведомлении
                std::wstring title(event.description.begin(),
event.description.end());
                std::wstring message(event.description.begin(),
event.description.end());

                // Создание объекта уведомления и отображение всплывающего
сообщения
                IUserNotification *un = NULL;
                if (CoCreateInstance(CLSID_UserNotification, 0, CLSCTX_ALL,
IID_IUserNotification, (void**)&un) == S_OK) {
                    un->SetBalloonInfo(L"Напоминание", message.c_str(), NI-
IF_INFO);
                    un->Show(NULL, 0);
                    un->Release();
                }
```

```
            }
        }

        // Пауза перед следующей проверкой
        std::this_thread::sleep_for(std::chrono::minutes(1));
    }

    CoUninitialize(); // Завершение работы с COM
}
```

Листинг 13 – Notification.cpp

```
#ifndef CALENDARCONSOL_TIME_H
#define CALENDARCONSOL_TIME_H

#include <iostream>
#include <iomanip>
#include <chrono>
#include <ctime>

// Класс для работы с временем
class Time {
public:
    static void displayCurrentDateTime(); // Объявление функции для отображе-
ния текущей даты и времени
};

#endif //CALENDARCONSOL_TIME_H
```

Листинг 14 – Time.h

```
#include "Time.h"

// Функция для отображения текущей даты и времени
void Time::displayCurrentDateTime() {
    auto now = std::chrono::system_clock::now();
    std::time_t now_c = std::chrono::system_clock::to_time_t(now);
    std::tm* now_tm = std::localtime(&now_c);

    // Вывод текущей даты и времени на экран
    std::cout << "Текущие дата и время: "
              << std::put_time(now_tm, "%Y-%m-%d %H:%M:%S") << std::endl;
}
```

Листинг 15 – Time.cpp

Описание программы

Программа — это консольное приложение для управления событиями календаря. Она позволяет пользователям добавлять, удалять, обновлять и просматривать события, а также сохранять их в файл и загружать из файла. Дополнительно, программа уведомляет пользователей о предстоящих событиях.

В основе программы лежит несколько классов, каждый из которых отвечает за свою часть функциональности. Класс EventManager управляет событиями, храня их в структуре данных и предоставляя методы для работы с ними. Класс Calendar взаимодействует с EventManager, позволяя пользователям добавлять, удалять и просматривать события, а также отображать календарь на месяц.

Интерфейс программы представлен классом Interface, который отображает меню и обрабатывает ввод пользователя, вызывая соответствующие методы в Calendar и EventManager. Когда пользователь выбирает, например, добавить событие, Interface вызывает метод addEvent у Calendar, который, в свою очередь, вызывает метод addEvent у EventManager для сохранения события.

Для уведомлений о предстоящих событиях используется класс Notification, который работает в отдельном потоке. Он постоянно проверяет текущее время и дату и уведомляет пользователя, если наступает время какого-либо события. Уведомления отображаются с помощью системных баллонов.

Класс Time отвечает за отображение текущей даты и времени, что полезно для пользователя в меню программы. Класс FileManager обеспечивает сохранение и загрузку событий из файла, позволяя пользователям сохранять своё расписание и восстанавливать его при необходимости.

Когда программа запускается, создаются объекты EventManager,

Calendar и Interface. Запускается поток для уведомлений, и программа входит в основной цикл, где отображается текущее время и меню. Пользователь может выбрать различные действия, такие как добавление, удаление или просмотр событий. Выбор пользователя обрабатывается Interface, который вызывает соответствующие методы других классов.

Программа позволяет сохранять созданные события в файл и загружать их из файла. Пользователь может выбрать опцию для сохранения всех событий в текстовый файл, что позволяет сохранить их для будущего использования. Также можно загрузить события из файла, восстанавливая ранее сохранённое расписание.

Программа работает до тех пор, пока пользователь не выберет опцию выхода, после чего основной цикл завершается и программа закрывается.

Результаты работы программы:

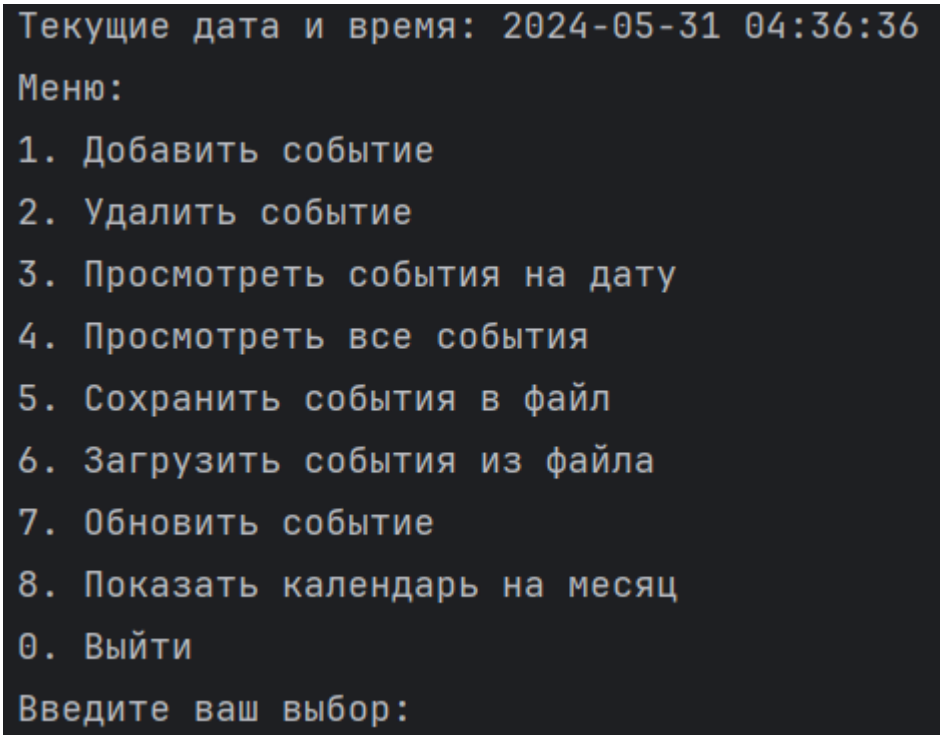

Рисунок 3 – Работа программы

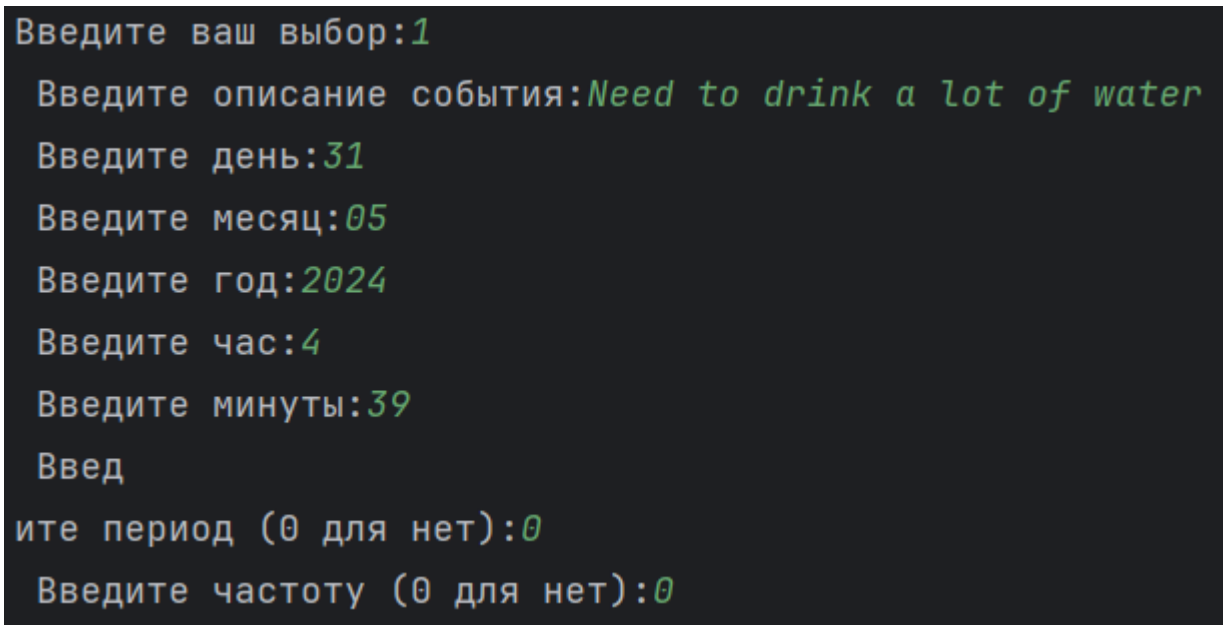

Рисунок 4 – Работа программы

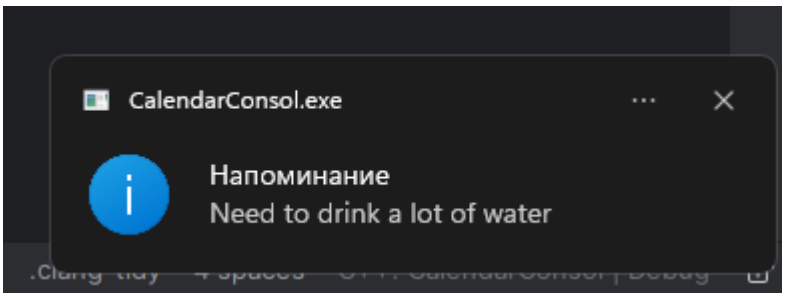

Рисунок 5 – Работа программы

```
 Текущие дата и время: 2024-05-31 04:38:04
Меню:
1. Добавить событие
2. Удалить событие
3. Просмотреть события на дату
4. Просмотреть все события
5. Сохранить события в файл
6. Загрузить события из файла
7. Обновить событие
8. Показать календарь на месяц
0. Выйти
Введите ваш выбор:3
 Введите день:31
 Введите месяц:05
 Введите год:2024
 Мероприятия для 31/5/2024:
- Need to drink a lot of water (4:39)
```

Рисунок 6 – Работа программы

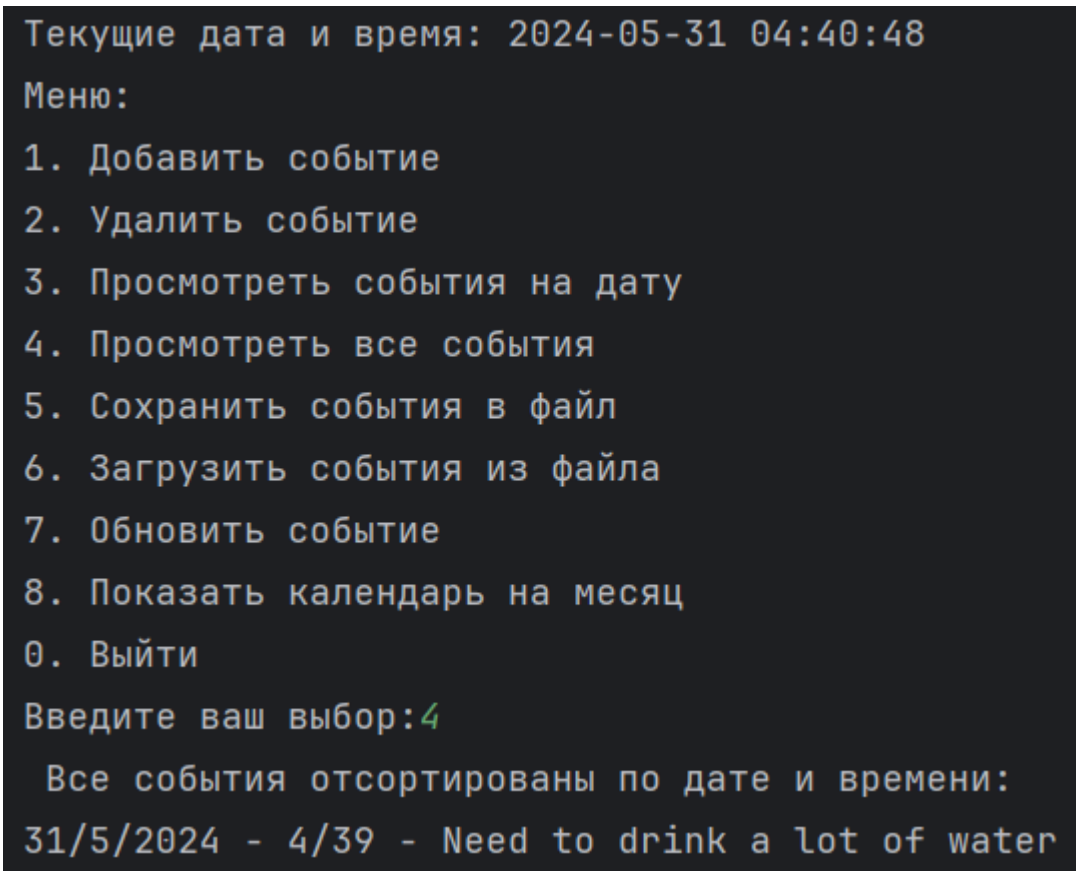

Рисунок 7 – Работа программы

```
3. Просмотреть события на дату
4. Просмотреть все события
5. Сохранить события в файл
6. Загрузить события из файла
7. Обновить событие
8. Показать календарь на месяц
0. Выйти
Введите ваш выбор:5
 События сохранены в events.txt
Текущие дата и время: 2024-05-31 04:41:53
Меню:
1. Добавить событие
2. Удалить событие
3. Просмотреть события на дату
4. Просмотреть все события
5. Сохранить события в файл
6. Загрузить события из файла
7. Обновить событие
8. Показать календарь на месяц
0. Выйти
Введите ваш выбор:6
 События загружены из events.txt
Текущие дата и время: 2024-05-31 04:42:03
Меню:
```

Рисунок 8 – Работа программы

```
7. Обновить событие
8. Показать календарь на месяц
0. Выйти
Введите ваш выбор:7
 Введите данные текущего события для обновления:
Описание:Need to drink a lot of water
 День:31
 Месяц:05
 Год:2024
 Час:4
 Минуты:39
 Период:00
 Частота:0
 Введите новые данные события:
Описание:Tomato juice is delicious
 День:8
 Месяц:06
 Год:2024
 Час:12
 Минуты:00
 Период:1
 Частота:1
```

Рисунок 9 – Работа программы

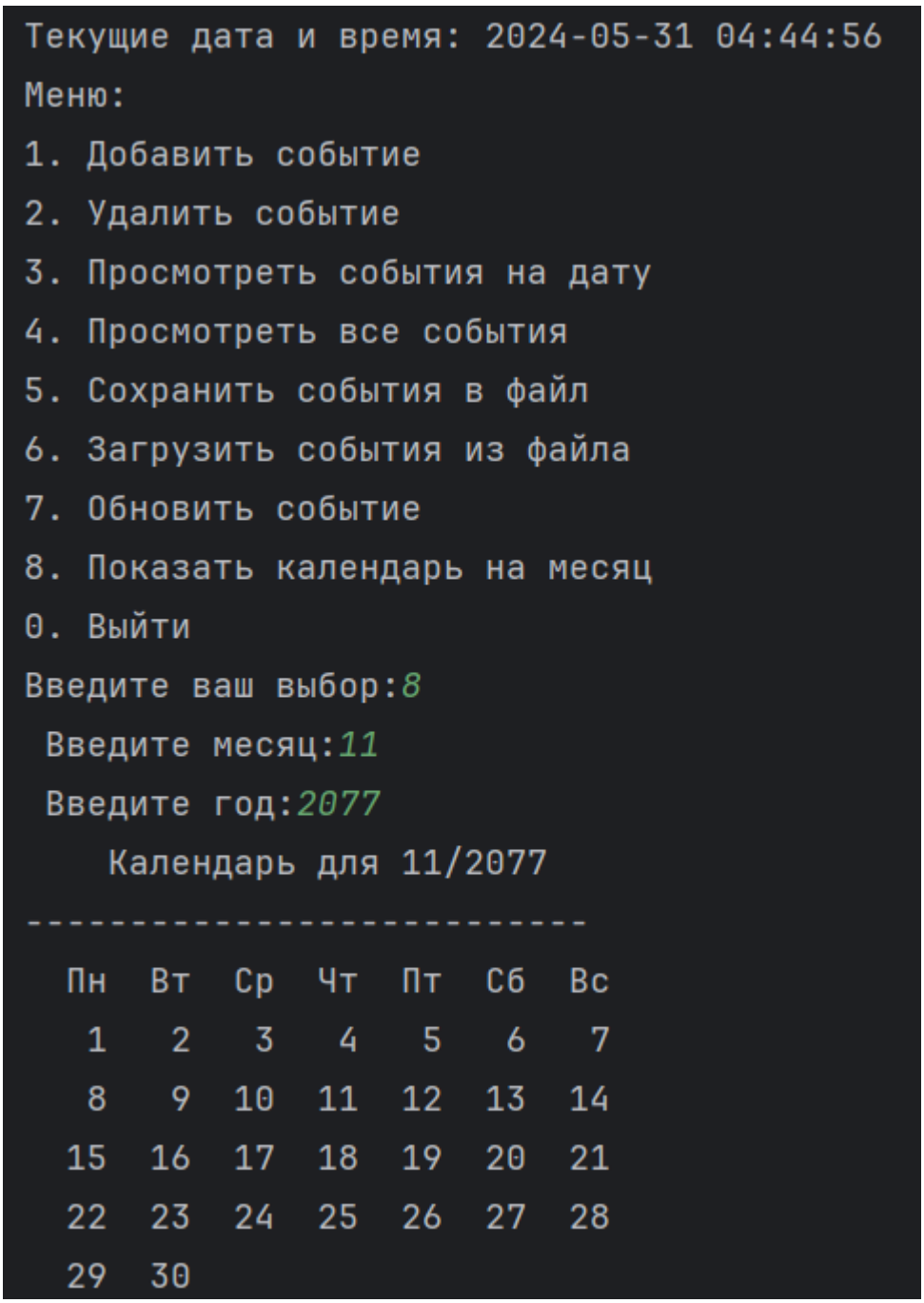

Рисунок 10 – Работа программы

Заключение

В ходе выполнения данной курсовой работы было разработано консольное приложение-календарь, предоставляющее пользователям удобные инструменты для управления своими событиями. Приложение позволяет добавлять новые события, удалять уже существующие, просматривать список событий на определенную дату, а также сохранять и загружать данные о событиях из файла. Дополнительно, приложение оснащено функцией уведомлений, которая информирует пользователя о предстоящих событиях.

Разработка данного приложения была связана с рядом технических и методологических аспектов, которые позволили закрепить и расширить мои знания и навыки в области программирования на языке C++. В частности, работа с датами и временем, пользовательским вводом и выводом, а также с файловой системой, дали ценный опыт в области разработки приложений, взаимодействующих с пользователем и хранящих данные.

Кроме того, проектирование и реализация приложения позволили применить принципы объектно-ориентированного программирования (ООП) и шаблоны проектирования, такие как паттерн " Наблюдатель", "Одиночка" и другие, что способствует повышению эффективности и читаемости кода, а также обеспечивает его модульность и масштабируемость.

Выполнение данной курсовой работы не только расширило мой практический опыт в области программирования на C++, но и предоставило ценные знания и навыки, которые могут быть применены в будущих проектах и задачах разработки программного обеспечения.

Список литературы

1. Керниган Б., Ричи Д. Язык программирования Си. — 2-е изд., доп. — М.: Вильямс, 2015. — 288 с. ISBN 978-5-8459-1260-8.
2. Страуструп Б. Программирование: Принципы и практика с использованием C++. — 2-е изд. — М.: Вильямс, 2016. — 1104 с. ISBN 978-5-8459-1807-5.
3. Страуструп Б. Язык программирования C++. Специальное издание. — М.: БХВ-Петербург, 2017. — 1376 с. ISBN 978-5-9775-5030-5.
4. Мейерс С. Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ. — 3-е изд. — М.: ДМК Пресс, 2015. — 320 с. ISBN 978-5-94074-644-2.
5. Саттер Г. Исключения: Рекомендации по разработке на C++. — М.: ДМК Пресс, 2014. — 464 с. ISBN 978-5-94074-682-4.
6. Грасс М. Стандартная библиотека C++. Специальное издание. — М.: БХВ-Петербург, 2016. — 960 с. ISBN 978-5-9775-5034-3.
7. Прата С. Язык программирования C++. Лекции и упражнения. — 6-е изд. — СПб.: Питер, 2018. — 960 с. ISBN 978-5-4461-0706-5.
8. Шилдт Г. Полный справочник по C++. — 4-е изд. — М.: Вильямс, 2014. — 1088 с. ISBN 978-5-8459-1850-1.
9. ISO/IEC JTC1/SC22/WG21. Международный стандарт ISO/IEC 14882:2017(E). Язык программирования C++. — 2017.
10. Cplusplus.com. Документация по стандартной библиотеке C++. [Электронный ресурс]. URL: http://www.cplusplus.com/reference/ (дата обращения: 29.05.2024).
11. cppreference.com. C++ Reference. [Электронный ресурс]. URL: https://en.cppreference.com/w/ (дата обращения: 29.05.2024).