

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Пензенский государственный университет»
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по курсу «Программирование»
на тему «База данных расписания занятий»

Выполнил:

студент группы 25BBB2

Тургенев И. В.

Приняла:

к.т.н., доцент Генералова А.А.

*Отмечено
21.05.2026
Г. Генералова*

Пенза 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

М.А. Митрохин

«11» февраля 2026г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Шурленеву Ивану Витальевичу Группа 25ВВВ2

Тема проекта: **Разработка программы сложной структуры методом нисходящего программирования.**

Исходные данные (технические требования) на проектирование
Программа составления расписаний: Занятия составлены на 2 недели
с учетом аудиторной нагрузки для 4 групп (численность каждой группы 20 чел.).
Пр. Файл содержит составленное программой расписание. Часы проведения
занятий заданы пользователем

Обязательные требования к программе:

1. Многомодульность.
2. Использование сложных типов данных – структур, списков, файлов.
3. Режим работы видеосистемы – текстовый / графический.
4. Устройство ввода информации – клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Наличие заставки.
7. Операционная система MS Windows.
8. Язык программирования – Си и Ассемблер.
9. Среда разработки ПО – Microsoft Visual Studio.

Объем работы по курсу

1. Расчетная часть

Разработка программы.

2. Графическая часть

Схема данных, схема ресурсов системы, схема работы системы, иерархическая структура программы, схема взаимодействия программы.

3. Экспериментальная часть

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания "11" февраля 2026 г.

Дата защиты проекта " " "

Руководитель В. Генералов А.А.

Задание получил "11" февраля 2026 г.

Студент Мурз Мурмелев

Содержание

Содержание	1
Введение	5
Декомпозиция задач и модульная структура программы.....	11
Выбор языка программирования, архитектура и структура программы, базы данных	15
Разработка функции на языке Assembler	29
Руководство пользователя	38
Отладка и возникшие трудности	41
Заключение.....	43
Список использованных источников.....	46
Приложение а. Листинг программы.....	49

Введение

Современные информационные системы обрабатывают колоссальные объемы данных. То, что раньше называли просто сведениями, фактами, данными или известиями, теперь мы обобщаем одним словом — «информация». Это понятие включает в себя всё, что зафиксировано в виде сообщений: от книг и газет до фотографий, фильмов, магнитных записей, каталогов, отчетов и писем.

Объемы данных растут экспоненциально, усложняя их поиск, хранение, обработку и передачу. Однако именно эти операции с информацией необходимы нам повсеместно.

Хранение информации — процесс, столь же древний, сколь и сама человеческая цивилизация. На заре истории эту функцию выполнял сам человек — его интеллект, его опыт, передававшиеся из поколения в поколение. Со временем носители информации менялись. Машинная обработка данных опирается на новые, электронные носители, где информация хранится в памяти машин. Эти хранящиеся в памяти машины сведения мы называем базами данных или информационными массивами, а для работы с ними используем информационно-поисковые системы.

База данных, по сути, — это структурированная организация данных, предназначенная для их хранения. Неразрывно с этим понятием термин «система управления базой данных» (СУБД). Это комплекс программных средств, который позволяет создавать структуры новых баз, наполнять их данными, редактировать содержимое и, конечно же, визуализировать полученную информацию.

Визуализация данных из базы означает отбор и представление информации согласно заданным критериям, её упорядочивание, форматирование и последующий вывод на устройства или передачу по каналам связи.

Актуальность темы нашей курсовой работы обусловлена несколькими факторами. Во-первых, потребность в автоматизации таких процессов, как

составление расписания, неуклонно растет. Учебные заведения развиваются, возникают новые форматы обучения (очное, заочное, дистанционное), увеличивается число индивидуальных образовательных траекторий. Во-вторых, разработка подобных систем предполагает использование структурированного подхода, который помогает управлять сложностью программы и обеспечивать ее надежность [3, 5, 6]. Метод структурного, или нисходящего, программирования как раз и предоставляет такой подход: он дает возможность разбить большую задачу на более мелкие, управляемые модули, каждый из которых выполняет свою подзадачу, а затем объединить их в единое целое [16, 19, 22]. Кроме того, составление расписания — классическая задача со сложной структурой взаимосвязей. Это делает ее идеальным объектом для демонстрации возможностей нисходящего подхода [18, 24].

Для достижения этой цели были поставлены следующие задачи:

- Изучить теоретические основы метода нисходящего программирования, проанализировать специфику систем управления расписанием и рассмотреть инструменты разработки, подходящие для создания сложных программных систем.
- Спроектировать архитектуру программы для составления расписания занятий, опираясь на принципы декомпозиции. Реализовать основные функциональные модули системы и провести тестирование разработанного решения.

Объектом исследования в данной работе является процесс разработки сложного программного обеспечения с применением метода нисходящего программирования.

Предметом исследования выступает сама программа для автоматизированного составления расписания занятий в учебном заведении. Она включает модули для ввода данных, формирования расписания и проверки на наличие конфликтов.

При выполнении этой курсовой работы применялись следующие методы:

- Анализ предметной области и функциональных требований к системе расписания [18, 24];
- Метод нисходящего программирования для поэтапного разбиения задачи на модули [3, 5, 16].

Все, что не так давно называли сведениями, фактами, данными, известиями, теперь называют одним общим словом «информация». Информация — это все фиксированные сообщения: книги, газеты, журналы, фотографии, фильмы, магнитные записи, записи на пластинках, каталоги, реклама, отчеты, письма и т. д.

Практическая значимость курсовой состоит в том, что разработанная программа может быть использована в реальных условиях учебных заведений для автоматизации процесса составления расписания, что позволит сократить временные затраты методистов, минимизировать количество ошибок и конфликтов, а также повысить эффективность использования аудиторного фонда и рабочего времени преподавателей. Кроме того, полученный опыт применения нисходящего подхода может быть использован при разработке других программных систем сложной структуры.

Структура курсовой включает введение, две главы, заключение, список использованных источников и приложения. В первой главе рассматриваются теоретические основы нисходящего программирования, анализируется предметная область систем расписания и описываются инструменты разработки. Во второй главе представлено проектирование архитектуры программы, описание реализации функциональных модулей и результаты тестирования. В заключении подводятся итоги выполненной курсовой и формулируются выводы о применимости метода нисходящего программирования к задачам подобного класса. Сущность и принципы нисходящего подхода в разработке ПО.

Нисходящее программирование представляет собой методологию разработки программного обеспечения, основанную на последовательной декомпозиции сложной задачи на более простые подзадачи. Суть подхода

заключается в том, что разработка начинается с самого верхнего уровня абстракции — с общей постановки задачи и главной функции программы, а затем постепенно спускается к деталям реализации, разбивая каждый модуль на более мелкие компоненты до тех пор, пока не будут получены элементарные операции, которые легко реализовать на выбранном языке программирования.

Исторически нисходящий подход возник в 1960-х годах как ответ на кризис программного обеспечения, когда проекты становились всё более сложными, а традиционные методы разработки приводили к неуправляемому коду, полному ошибок и практически не поддающемуся сопровождению. Пионеры структурного программирования, такие как Эдсгер Дейкстра и Никлаус Вирт, предложили систематический подход к построению программ, где каждый уровень абстракции решает свою чётко определённую задачу, а детали реализации скрыты от вышестоящих уровней [3, 5].

Основной принцип нисходящего программирования — это принцип «разделяй и властвуй» [2, 9]. Вместо того чтобы пытаться решить всю задачу сразу, программист разбивает её на несколько независимых или слабо связанных подзадач, каждая из которых проще исходной. Этот процесс повторяется рекурсивно: каждая подзадача, в свою очередь, разбивается на ещё более мелкие части, пока не будет достигнут уровень, на котором реализация становится тривиальной. В результате получается иерархическая структура программы, напоминающая дерево, где корень — это главная функция, а листья — элементарные операции.

Второй важный принцип — это принцип абстракции и сокрытия информации [4, 6]. На каждом уровне иерархии модуль предоставляет определённый интерфейс — набор функций или операций, которые могут быть использованы модулями верхнего уровня. При этом детали реализации скрыты: вышестоящий модуль не знает и не должен знать, как именно реализована та или иная операция, ему важен только результат. Это позволяет изменять внутреннюю реализацию модуля без влияния на остальную программу, что упрощает сопровождение и модификацию кода [10, 15].

Третий принцип — это принцип пошаговой детализации [16, 22]. Разработка программы ведётся итерационно: сначала создаётся общая схема решения задачи на высоком уровне абстракции, где используются ещё не реализованные функции и процедуры. Затем каждая из этих функций детализируется — для неё пишется алгоритм, который, в свою очередь, может использовать функции следующего уровня. Процесс продолжается до тех пор, пока все функции не будут выражены через базовые операции языка программирования. Такой подход позволяет сосредоточиться на логике решения задачи, не отвлекаясь на технические детали реализации на ранних этапах.

Четвертый принцип обеспечивает модульность и слабую связанность [21, 23]. Каждый модуль решает одну четко определенную задачу с минимальными зависимостями от других. Связи между модулями осуществляются через чётко определённые интерфейсы, что делает программу более понятной, упрощает тестирование отдельных компонентов и позволяет распределять работу между несколькими разработчиками.

Пятый принцип — это принцип отложенной реализации. При нисходящей разработке не обязательно сразу реализовывать все модули нижнего уровня. Можно создать заглушки (stubs) — упрощённые версии функций, которые возвращают фиксированные значения или выполняют минимальную функциональность. Это позволяет протестировать логику верхнего уровня и убедиться, что общая архитектура программы работает правильно, прежде чем погружаться в детали реализации каждого модуля.

Применение нисходящего подхода особенно эффективно при разработке программ сложной структуры, где количество взаимосвязей между компонентами велико, а требования к надёжности и сопровождаемости высоки. Метод позволяет контролировать сложность на каждом этапе разработки, обеспечивает прозрачность структуры программы и упрощает процесс отладки, поскольку ошибки локализуются в конкретных модулях, а не распространяются по всему коду.

Важно отметить, что нисходящее программирование не является универсальным решением для всех задач. В некоторых случаях более эффективным оказывается восходящий подход (bottom-up), когда сначала создаются базовые компоненты, а затем из них строится система более высокого уровня. На практике часто используется комбинированный подход: общая архитектура проектируется сверху вниз, а отдельные критически важные модули разрабатываются снизу вверх. Тем не менее, для задач с чётко определённой структурой и понятными требованиями, таких как система составления расписания занятий, нисходящий подход демонстрирует высокую эффективность и позволяет создавать качественное программное обеспечение (ПО) в разумные сроки.

Декомпозиция задач и модульная структура программы

Декомпозиция (разбиение) задач является центральным механизмом нисходящего программирования, позволяющим превратить сложную, трудно решаемую проблему в набор простых, понятных подзадач [2, 9, 22]. Процесс декомпозиции начинается с анализа исходной задачи и выявления её основных функциональных блоков. Каждый блок должен решать одну чётко определённую подзадачу и иметь понятный интерфейс взаимодействия с другими блоками. Качество декомпозиции напрямую влияет на качество итоговой программы: правильно выделенные модули легко тестировать, модифицировать и повторно использовать в других проектах [10, 15].

При разбиении задачи составления расписания занятий необходимо выделить основные функциональные области. Первая область — это управление данными: программа должна хранить информацию о группах студентов, преподавателях, дисциплинах, аудиториях и временных слотах. Вторая область — это формирование расписания: назначение конкретных занятий на определённое время в определённые аудитории с учётом всех ограничений. Третья область — это проверка корректности: выявление конфликтов, таких как одновременное использование одной аудитории разными группами или двойная нагрузка преподавателя. Четвёртая область — это пользовательский интерфейс: ввод данных, отображение расписания, редактирование и экспорт результатов.

Модульная структура программы строится на основе выделенных функциональных областей. Каждый модуль представляет собой логически завершённую единицу кода, которая инкапсулирует данные и операции над ними [4, 6]. Модуль имеет публичный интерфейс — набор функций, доступных извне, и приватную реализацию — внутренние структуры данных и вспомогательные функции, скрытые от других модулей. Такая организация обеспечивает принцип сокрытия информации: изменения внутри модуля не затрагивают остальную программу, если интерфейс остаётся неизменным.

Важным аспектом модульной структуры является определение зависимостей между модулями. В идеале модули должны быть слабо связаны и сильно связны внутри. Слабая связанность означает, что модули взаимодействуют друг с другом через минимальное количество интерфейсов и не зависят от внутренней реализации друг друга. Сильная связанность означает, что все элементы внутри модуля работают на решение одной задачи и логически связаны между собой. Нарушение этих принципов приводит к появлению спагетти-кода, где изменение одного модуля вызывает цепную реакцию изменений в других модулях.

Процесс декомпозиции можно представить в виде последовательности шагов, каждый из которых уточняет структуру программы. На первом шаге определяется главная функция программы и выделяются модули верхнего уровня. На втором шаге каждый модуль верхнего уровня разбивается на подмодули, которые решают более конкретные задачи. На третьем шаге подмодули детализируются до уровня отдельных функций и процедур. Процесс продолжается до тех пор, пока не будет достигнут уровень элементарных операций, которые можно непосредственно реализовать на языке программирования.

Для наглядного представления процесса декомпозиции задачи составления расписания занятий можно использовать блок-схему, которая показывает иерархию модулей и их взаимосвязи. Блок-схема демонстрирует, как главная задача разбивается на подзадачи, а те, в свою очередь, на ещё более мелкие компоненты (рисунок 1).

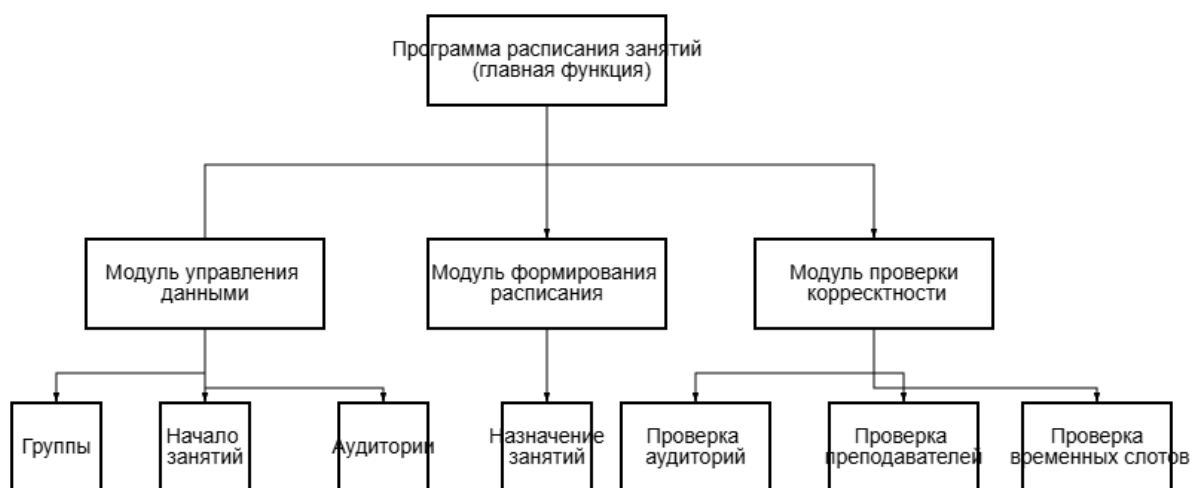


Рисунок 1 — Блок-схема «Иерархическая декомпозиция программы расписания занятий»

Представленная на рисунке 1 блок-схема наглядно демонстрирует трёхуровневую структуру программы. На верхнем уровне находится главная функция, которая координирует работу всех компонентов. На среднем уровне расположены три основных модуля, каждый из которых отвечает за свою функциональную область. На нижнем уровне находятся конкретные подмодули, реализующие элементарные операции. Такая структура позволяет легко ориентироваться в коде и понимать, какой модуль за что отвечает.

Модуль управления данными разбивается на подмодули для работы с различными сущностями: группами студентов, преподавателями и аудиториями. Каждый подмодуль предоставляет операции добавления, удаления, изменения и поиска соответствующих объектов. Модуль формирования расписания содержит алгоритмы назначения занятий на временные слоты с учётом доступности ресурсов. Модуль проверки корректности включает подмодули для выявления различных типов конфликтов: пересечения по аудиториям, перегрузки преподавателей и нарушения временных ограничений. Преимущества такой модульной структуры очевидны. Во-первых, каждый модуль можно разрабатывать и тестировать независимо от других, что ускоряет процесс разработки и

упрощает поиск ошибок. Во-вторых, модули можно повторно использовать в других проектах: например, модуль управления данными о преподавателях может быть применён в системе учёта нагрузки или в электронном деканате. В-третьих, модульная структура облегчает сопровождение программы: если требуется изменить алгоритм проверки конфликтов, достаточно модифицировать соответствующий подмодуль, не затрагивая остальной код.

Кроме того, модульная структура позволяет распределять работу между несколькими разработчиками. Каждый программист может отвечать за свой модуль, и при условии соблюдения интерфейсов модули легко интегрируются в единую систему. Это особенно важно для крупных проектов, где один человек физически не может написать весь код в разумные сроки.

Следует отметить, что качество декомпозиции во многом зависит от опыта разработчика и глубины понимания предметной области. Неправильно выделенные модули могут привести к избыточной связанности, дублированию кода и сложностям при модификации программы. Поэтому на этапе проектирования важно тщательно продумать структуру модулей, их интерфейсы и зависимости, прежде чем приступать к реализации.

Выбор языка программирования, архитектура и структура программы, базы данных

Разработанная программа предназначена для автоматизации работы с расписанием занятий в университете. Основная задача — предоставить удобный инструмент для хранения, поиска и отображения информации о занятиях различных учебных групп. Программа позволяет быстро найти расписание конкретной группы или посмотреть, какие занятия проходят в определённый день недели. Это особенно актуально для студентов, преподавателей и сотрудников учебной части, которым постоянно приходится обращаться к расписанию.

В современных учебных заведениях расписание — это не просто список занятий, а сложная система, которая должна учитывать множество факторов: доступность аудиторий, загруженность преподавателей, распределение дисциплин по семестрам, соблюдение санитарных норм и требований учебного плана. Ручное составление и поиск информации в таком расписании занимает много времени и часто приводит к ошибкам. Автоматизация этого процесса позволяет значительно упростить работу всех участников образовательного процесса.

В проекте есть некоторый файл “расписание.csv”. В нем расположено все расписание занятий кафедры «Вычислительная техника». Формат csv необходим для представления табличных значений. Я выбрал именно его, потому что, в отличие от других форматов csv занимает минимальное пространство, что критично для больших наборов данных.

Программа написана на языке C в среде обработки Microsoft Visual Studio 2022 и реализована с применением метода нисходящего программирования. Выбор языка C обусловлен несколькими факторами. Во-первых, это классический язык системного программирования, который обеспечивает высокую производительность и эффективное использование ресурсов [8, 25]. Во-вторых, C позволяет работать на низком уровне абстракции, что даёт возможность интегрировать ассемблерный код для

демонстрации взаимодействия между высокоуровневыми и низкоуровневыми компонентами программы [12, 13, 17]. В-третьих, язык С широко используется в образовательных целях и является отличной базой для понимания принципов программирования.

Суть метода нисходящего программирования заключается в том, что сложная задача разбивается на более простые подзадачи, каждая из которых решается отдельным модулем или функцией [3, 5, 16]. Сначала была определена общая структура программы: что она должна делать на верхнем уровне. Затем каждая крупная задача была разбита на более мелкие операции, пока не были получены простые функции, которые легко реализовать. Такой подход позволил сосредоточиться на логике работы программы, не отвлекаясь на детали реализации на ранних этапах [22, 23].

Процесс разработки начался с определения главной цели: создать программу, которая может загружать расписание из файла, хранить его в памяти и предоставлять пользователю удобные средства для поиска нужной информации. Эта цель была декомпозирована на четыре основные подзадачи: организация хранения данных, загрузка данных из внешнего источника, реализация функций поиска и организация пользовательского интерфейса. Каждая подзадача, в свою очередь, была детализирована до уровня конкретных функций.

Структура программы построена по модульному принципу. На верхнем уровне находится главная функция `main`, которая координирует работу всех остальных компонентов. Она отвечает за инициализацию программы, загрузку данных из файла, отображение меню и обработку выбора пользователя. Функция `main` представляет собой управляющий центр программы, который вызывает другие функции в зависимости от действий пользователя.

Ниже располагаются функции, реализующие конкретные операции. Каждая функция решает одну чётко определённую задачу и может быть протестирована независимо от других. Это один из ключевых принципов

модульного программирования: каждый модуль должен иметь единственную ответственность и минимальные зависимости от других модулей.

Основные модули программы можно разделить на несколько функциональных категорий:

Первая категория — функции для работы с данными. Центральной функцией этой категории является `load_schedule_csv`, которая отвечает за чтение расписания из CSV-файла и заполнение массива структур. Эта функция выполняет критически важную задачу: она преобразует текстовые данные из файла в структурированный формат, с которым может работать программа. Кроме того, к этой категории относится вспомогательная функция `safe_copy`, которая обеспечивает безопасное копирование строк с проверкой границ массива, предотвращая переполнение буфера.

Вторая категория — функции поиска и фильтрации. Функция `find_by_group` позволяет найти все занятия конкретной учебной группы. Она принимает на вход название группы и выводит полное расписание этой группы с указанием дней недели, номеров пар, названий предметов и времени проведения занятий. Функция `find_by_day` работает аналогично, но осуществляет поиск по дню недели, показывая все занятия, которые проходят в указанный день. Эти функции реализуют основную бизнес-логику программы и являются наиболее востребованными с точки зрения пользователя.

Третья категория — функции вывода информации. Функция `print_all_schedule` выводит полное расписание всех групп в табличном виде с аккуратным выравниванием столбцов. Функция `show_menu` отображает главное меню программы с перечнем доступных действий. Эти функции отвечают за представление данных пользователю в удобной и понятной форме.

Четвёртая категория — вспомогательные функции на ассемблере. В программе реализованы две функции на языке ассемблера: `asm_strcmp` для сравнения строк и `asm_strcpy` для копирования строк. Эти функции демонстрируют возможность интеграции низкоуровневого кода в программу

на С и позволяют глубже понять, как работают базовые операции со строками на уровне процессора. Хотя в стандартной библиотеке С есть аналогичные функции, реализация собственных версий на ассемблере имеет образовательную ценность и показывает, как высокоуровневые конструкции транслируются в машинный код [1, 7].

Таблица 1 — «Модули и библиотека»

Название	Какую функцию выполняет	Значение функции
main.h	Объявляет глобальные переменных и функций	Представляет собой набор функций для: навигации, работы с базой, работы с записями, работы с файлом и многие другие
asm_strcmp.cpp	Сравнение строк на ассемблере	Для поиска по группе и дню недели
asm_strcpy.cpp	Копирования строк на ассемблере	Реализует функцию копирования строки через inline ассемблер, используется в safe_copу для безопасного копирования данных
print_all_schedule.cpp	Отображение всего расписания	Выводит всё расписание в виде таблицы, форматирует вывод с выравниванием колонок

Продолжение таблицы 1

show_menu.cpp	Отображение меню	Выводит меню с доступными действиями, простой модуль для отделения UI от логики
load_schedule_csv.cpp	Загрузка расписания из CSV файла	Читает CSV файл с расписанием, разбивает по полям и заполняет массив структур Lesson
trim_string.cpp	Утилиты для работы со строками	Удаляет пробелы, кавычки и переносы строк, безопасное копирование с проверкой размера буфера
find_by_group.cpp	Поиск по группе и дню недели	Для поиска по группе и неделе

Таблица 1 «Модули и библиотека» предоставляет пользователю понять, зачем нужен каждый модуль и библиотека в моем проекте.

Ниже, на рисунке 2 представлена схема данных.



Рисунок 3 — Блок схема программы «Source.cpp»

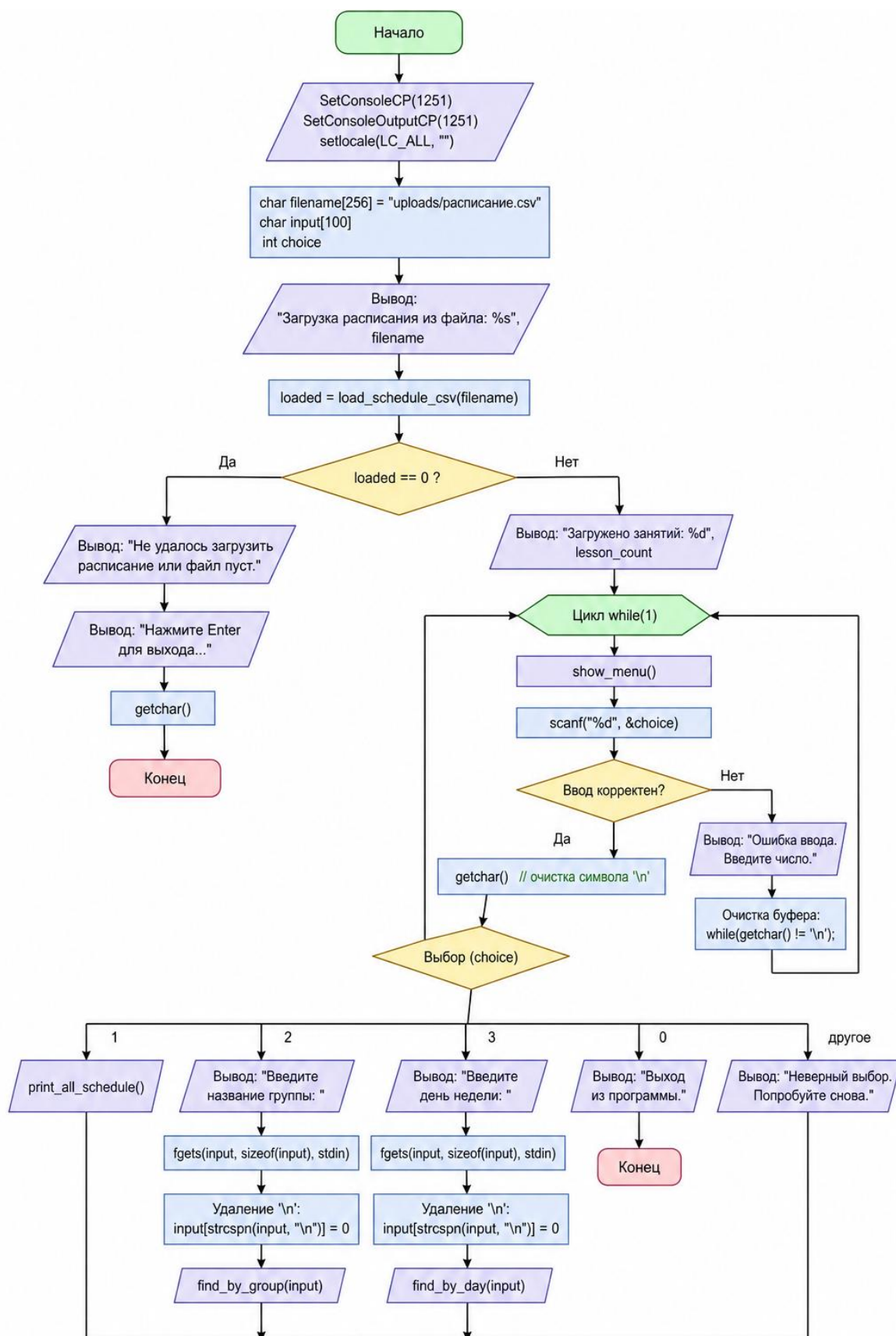


Рисунок 4 — Блок схема программы «main.h»

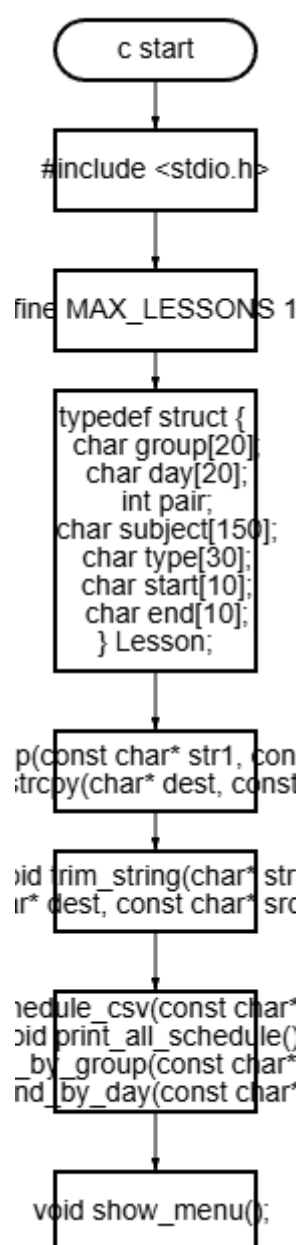


Рисунок 5 — Блок схема программы «asm_strcmp.cpp»

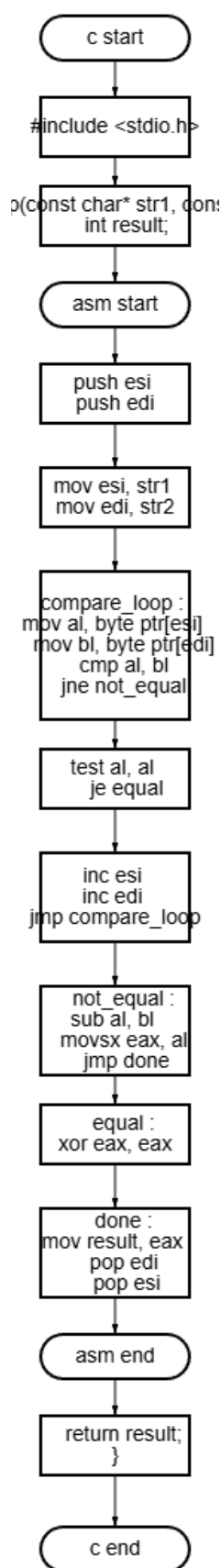


Рисунок 6 — Блок схема программы «asm_strcpy.c»

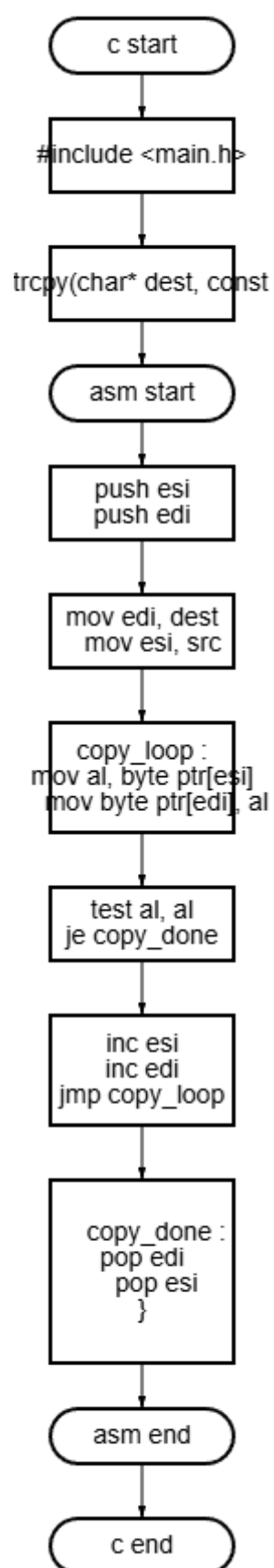


Рисунок 7 — Блок схема программы «find_by_group.cpp»

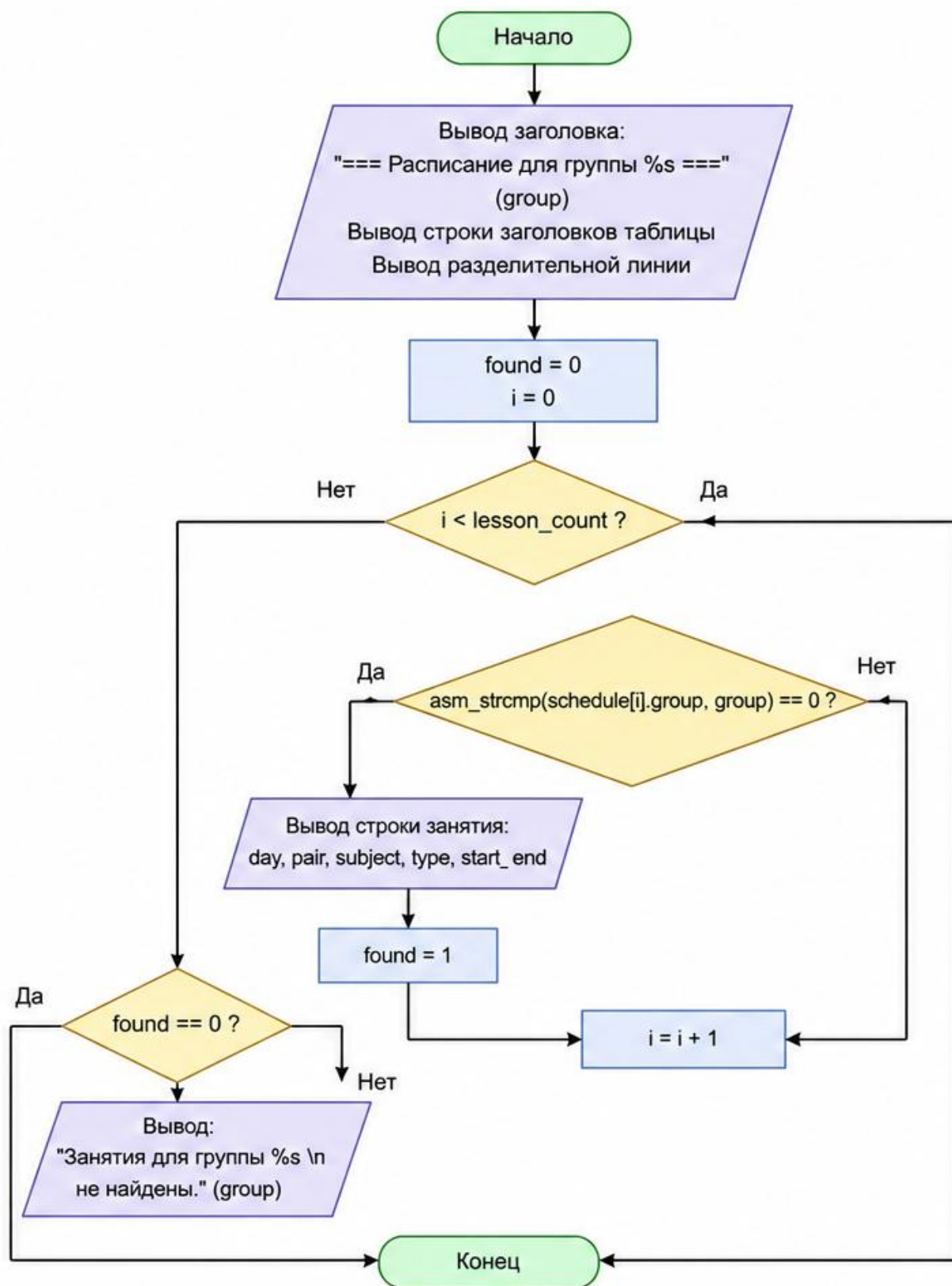


Рисунок 8 — Блок схема программы «load_schedule_csv.cpp»

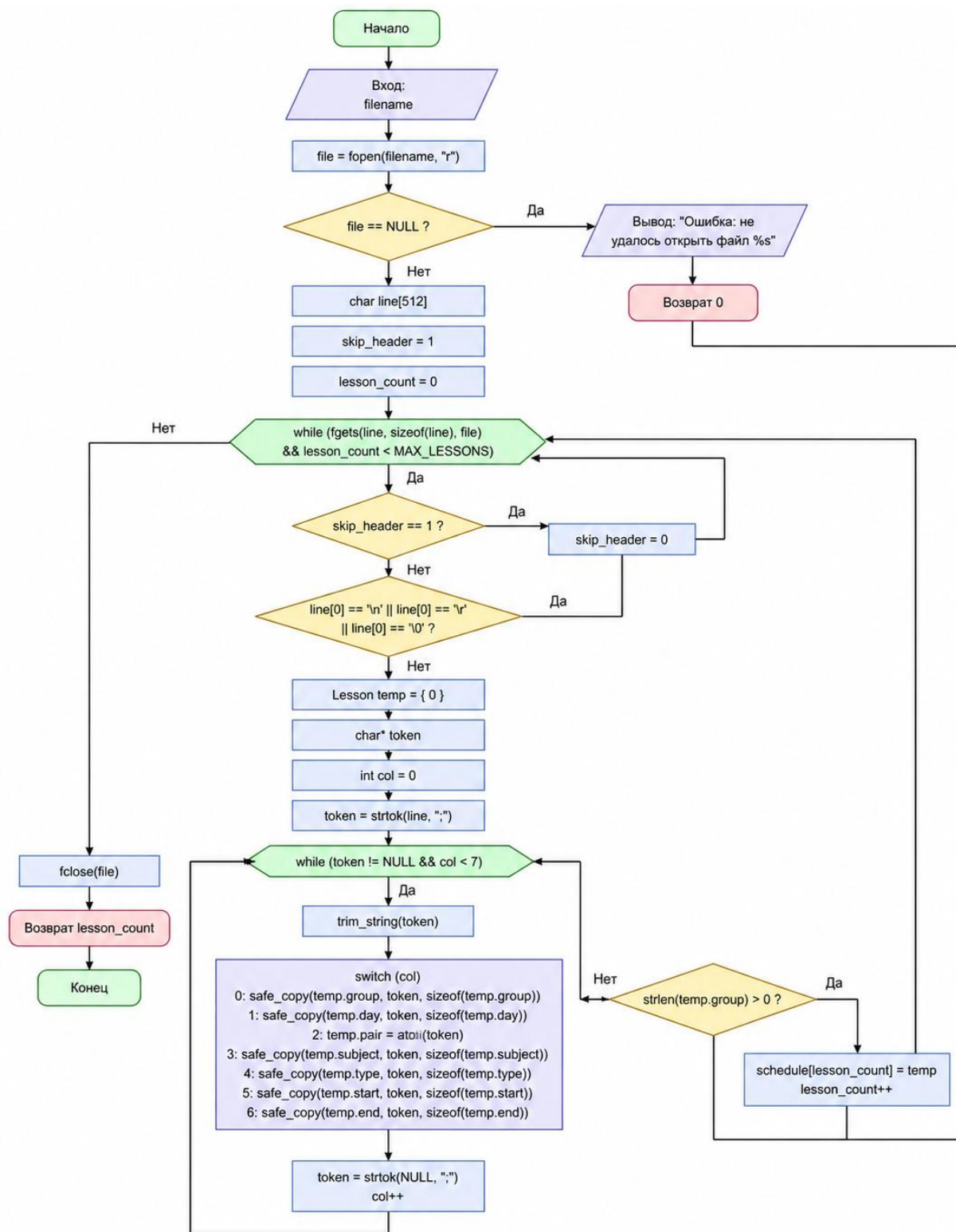


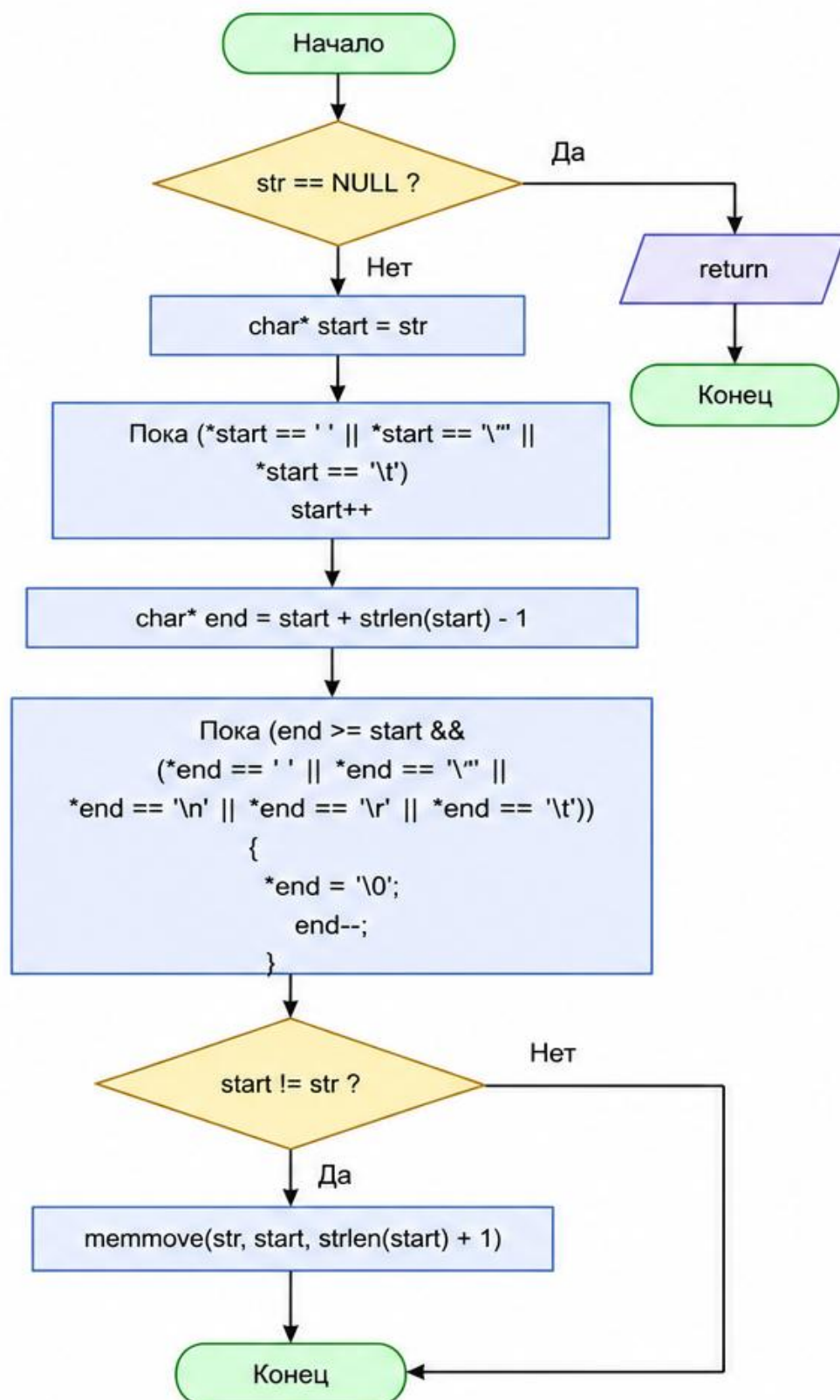
Рисунок 9 —Блок схема программы «show_menu.c»



Рисунок 11 — Блок схема программы «print_all_schedule.cpp»



Рисунок 12 — Блок схема программы «trim_string.cpp»



Разработка функции на языке Assembler

В процессе разработки программы для работы с расписанием занятий передо мной встала задача реализовать несколько базовых операций со строками, которые используются практически в каждой функции программы. Речь идёт о сравнении строк и их копировании. Эти операции выполняются очень часто: при поиске занятий по группе нужно сравнивать название группы из файла с тем, что ввёл пользователь, при поиске по дню недели — сравнивать дни, при загрузке данных из файла — копировать строки в структуры [8, 25]. Конечно, в стандартной библиотеке языка C есть готовые функции `strcmp` и `strcpy`, которые прекрасно справляются с этими задачами, но в рамках курсовой работы было интересно реализовать эти функции самостоятельно на языке ассемблера, чтобы лучше понять, как они работают на низком уровне и как происходит взаимодействие между высокоуровневым кодом на C и машинными командами процессора [12, 13, 14, 20].

Встроенный ассемблер Visual C++ позволяет интегрировать низкоуровневый код напрямую в системном программировании, когда нужно получить максимальный контроль над выполнением операций или оптимизировать критичные участки кода. В моём случае основная цель была не столько в оптимизации, сколько в образовательных целях: я хотел разобраться, как устроены базовые строковые операции на уровне регистров процессора, как передаются параметры, как происходит работа с памятью и как возвращается результат. Кроме того, это позволило продемонстрировать возможность интеграции ассемблерного кода в программу на C, что является важным навыком для понимания архитектуры компьютера и принципов работы компилятора.

Я реализовал две функции на ассемблере: `asm_strcmp` для сравнения строк и `asm_strcpy` для копирования строк. Начну с описания функции сравнения строк, так как она используется чаще всего и её логика немного сложнее.

Функция сравнения строк `asm_strcmp`

Функция `asm_strcmp` принимает на вход два указателя на строки и возвращает целое число: ноль, если строки идентичны, отрицательное число, если первая строка меньше второй в лексикографическом порядке, и положительное число, если первая строка больше. Это стандартное поведение функции `strcmp` из библиотеки `C`, и я постарался воспроизвести его максимально точно.

Прототип функции выглядит следующим образом:

```
int asm_strcmp(const char* str1, const char* str2);
```

Функция объявлена как обычная функция на языке `C`, но её тело содержит блок встроенного ассемблера, который обозначается ключевым словом `__asm`. Это синтаксис компилятора `Microsoft Visual C++`, который позволяет вставлять ассемблерный код прямо внутри функции на `C`.

Передача параметров в функцию происходит через стек в соответствии с соглашением о вызовах `cdecl`, которое используется по умолчанию в программах на `C`. Это означает, что параметры помещаются в стек справа налево, и вызывающая функция отвечает за очистку стека после возврата из функции. Внутри ассемблерного блока я могу обращаться к параметрам по их именам `str1` и `str2`, и компилятор автоматически подставит правильные адреса из стека.

Вот полный код функции с подробными комментариями:

```
int asm_strcmp(const char* str1, const char* str2) {
    int result;
    __asm {
        push esi
        push edi
        mov esi, str1
        mov edi, str2
        compare_loop:
        mov al, byte ptr[esi]
        mov bl, byte ptr[edi]
        cmp al, bl
        jne not_equal
    }
```

```

        test al, al
        je equal
        inc esi
        inc edi
        jmp compare_loop
not_equal:
        sub al, bl
        movsx eax, al
        jmp done
equal:
        xor eax, eax
done:
        mov result, eax
        pop edi
        pop esi
    }
    return result;
}

```

Разберём эту функцию построчно, чтобы понять, как она работает.

Первым делом я сохраняю значения регистров `esi` и `edi` в стек с помощью команды `push`. Это необходимо, потому что эти регистры могут использоваться в вызывающей функции, и, если я их изменю, это может привести к непредсказуемым ошибкам.

Соглашение о вызовах требует, чтобы вызываемая функция сохраняла значения определённых регистров, и `esi` и `edi` как раз входят в их число. В конце функции я восстанавливаю их значения с помощью команды `pop`.

Затем я загружаю адреса строк в регистры:

```

mov esi, str1
mov edi, str2

```

Регистр `esi` (Source Index) будет указывать на текущий символ первой строки, а регистр `edi` (Destination Index) — на текущий символ второй строки. Эти регистры традиционно используются для работы со строками в

ассемблере, хотя можно было бы использовать и другие регистры общего назначения.

Далее начинается основной цикл сравнения, который помечен меткой `compare_loop`. Метки в ассемблере используются для обозначения точек, к которым можно переходить с помощью команд условного и безусловного перехода.

```
compare_loop:
    mov al, byte ptr[esi]
    mov bl, byte ptr[edi]
```

Здесь я загружаю по одному байту (символу) из каждой строки в младшие части регистров `eax` и `ebx`. Регистр `al` — это младший байт регистра `eax`, а `bl` — младший байт регистра `ebx`. Конструкция `byte ptr[esi]` означает, что мы читаем один байт из памяти по адресу, который хранится в регистре `esi`.

После того как символы загружены, я сравниваю их с помощью команды `cmp`:

```
cmp al, bl
jne not_equal
```

Команда `cmp` выполняет вычитание `al - bl`, но не сохраняет результат, а только устанавливает флаги процессора в зависимости от результата операции. Команда `jne` (Jump if not equal) проверяет флаг нуля: если он не установлен, то есть символы не равны, происходит переход на метку `not_equal`. Если символы равны, выполнение продолжается дальше.

Следующая проверка определяет, не достигли ли мы конца строки:

```
test al, al
je equal
```

Команда `test al, al` выполняет побитовую операцию AND между регистром `al` и самим собой. Это стандартный способ проверить, равен ли регистр нулю. Если `al` содержит нулевой символ (конец строки в C обозначается символом с кодом 0), то результат операции будет нулевым, и флаг нуля установится. Команда `je` (Jump if equal) проверяет этот флаг и

переходит на метку `equal`, если он установлен. Это означает, что мы дошли до конца обеих строк одновременно, и они полностью совпадают.

Если символы равны, но это ещё не конец строки, мы переходим к следующей паре символов:

```
inc esi
inc edi
jmp compare_loop
```

Команда `inc` увеличивает значение регистра на единицу, то есть мы сдвигаем указатели на следующий символ в каждой строке. Затем команда `jmp` выполняет безусловный переход обратно на начало цикла `compare_loop`, и процесс повторяется.

Если на каком-то шаге символы оказались не равны, выполнение переходит на метку `not_equal`:

```
not_equal:
sub al, bl
movsx eax, al
jmp done
```

Здесь я вычисляю разность между символами с помощью команды `sub al, bl`. Результат остаётся в регистре `al`. Затем команда `movsx eax, al` выполняет знаковое расширение байта из `al` в полный 32-битный регистр `eax`. Это необходимо, потому что функция должна вернуть целое число типа `int`, которое занимает 4 байта. Команда `movsx` (move with sign extension) копирует значение из младшего байта в старшие байты, сохраняя знак числа. Если символ в `al` был отрицательным (что может произойти при вычитании), старшие биты `eax` заполнятся единицами, иначе — нулями.

Если строки оказались равны, выполнение переходит на метку `equal`:

```
equal:
xor eax, eax
```

Команда `xor eax, eax` выполняет операцию исключающего ИЛИ регистра с самим собой, что всегда даёт ноль. Это стандартный и очень

быстрый способ обнулить регистр в ассемблере. Таким образом, в `eax` записывается ноль, что означает, что строки идентичны.

Наконец, на метке `done` происходит завершение работы функции:

```
done:
    mov result, eax
    pop edi
    pop esi
```

Значение из регистра `eax` копируется в локальную переменную `result`, которая объявлена в начале функции. Затем восстанавливаются сохранённые значения регистров `edi` и `esi` из стека в обратном порядке (последним сохранённым был `edi`, поэтому он восстанавливается первым). После выхода из блока `__asm` функция возвращает значение переменной `result` с помощью обычного оператора `return` языка C.

Функция копирования строк `asm_strcpy`

Вторая функция, которую я реализовал на ассемблере, — это `asm_strcpy`, которая копирует содержимое одной строки в другую. Прототип функции:

```
void asm_strcpy(char* dest, const char* src);
```

Функция принимает два параметра: указатель на строку назначения `dest` и указатель на исходную строку `src`. Она копирует все символы из `src` в `dest`, включая завершающий нулевой символ.

Вот полный код функции:

```
void asm_strcpy(char* dest, const char* src) {
    __asm {
        push esi
        push edi
        mov edi, dest
        mov esi, src
    copy_loop:
        mov al, byte ptr[esi]
        mov byte ptr[edi], al
        test al, al
        je copy_done
```

```

        inc esi
        inc edi
        jmp copy_loop
copy_done:
        pop edi
        pop esi
    }
}

```

Логика этой функции очень похожа на `asm_strcpy`, но проще, так как здесь не нужно сравнивать символы, а только копировать их.

Сначала я сохраняю регистры `esi` и `edi` в стек, затем загружаю адреса строк:

```

mov edi, dest
mov esi, src

```

На этот раз `edi` указывает на строку назначения, а `esi` — на исходную строку. Это соответствует традиционному использованию этих регистров в строковых операциях: `esi` для источника (`source`), `edi` для назначения (`destination`).

Основной цикл копирования начинается с метки `copy_loop`:

```

copy_loop:
    mov al, byte ptr[esi]
    mov byte ptr[edi], al

```

Первая команда загружает один байт из исходной строки в регистр `al`, вторая команда записывает этот байт в строку назначения. Это и есть операция копирования одного символа.

Затем проверяется, не был ли скопирован нулевой символ:

```

test al, al
je copy_done

```

Если `al` содержит ноль, значит мы скопировали завершающий символ строки, и можно завершать работу функции, переходя на метку `copy_done`.

Если символ ненулевой, продолжаем копирование:

```

inc esi

```

```
inc edi
jmp copy_loop
```

Увеличиваем оба указателя на единицу, чтобы перейти к следующему символу, и возвращаемся в начало цикла.

На метке `copy_done` восстанавливаются сохранённые регистры:

```
copy_done:
pop edi
pop esi
```

Функция не возвращает никакого значения, так как она объявлена с типом `void`. Результат её работы — изменённая строка `dest`, которая теперь содержит копию строки `src`.

Использование ассемблерных функций в программе

Обе эти функции активно используются в программе для работы с расписанием. Функция `asm_strcmp` вызывается в функции поиска `find_by_group` и `find_by_day` для сравнения названий групп и дней недели:

```
if (asm_strcmp(schedule[i].group, group) == 0) {
    ...
}
```

Функция `asm_strcpy` используется внутри вспомогательной функции `safe_copy`, которая обеспечивает безопасное копирование строк с проверкой размера буфера. Сначала строка копируется во временный буфер с помощью стандартной функции `strcpy`, которая ограничивает количество копируемых символов, а затем из временного буфера в конечное место назначения с помощью `asm_strcpy`:

```
void safe_copy(char* dest, const char* src, size_t dest_size)
{
    if (dest_size == 0) return;
    size_t src_len = strlen(src);
    if (src_len >= dest_size) {
        src_len = dest_size - 1;
    }
    char temp[512];
```

```
    strncpy(temp, src, src_len);  
    temp[src_len] = '\\0';  
    asm_strcpy(dest, temp);  
}
```

Такой подход позволяет избежать переполнения буфера, которое может привести к серьёзным ошибкам и уязвимостям в программе.

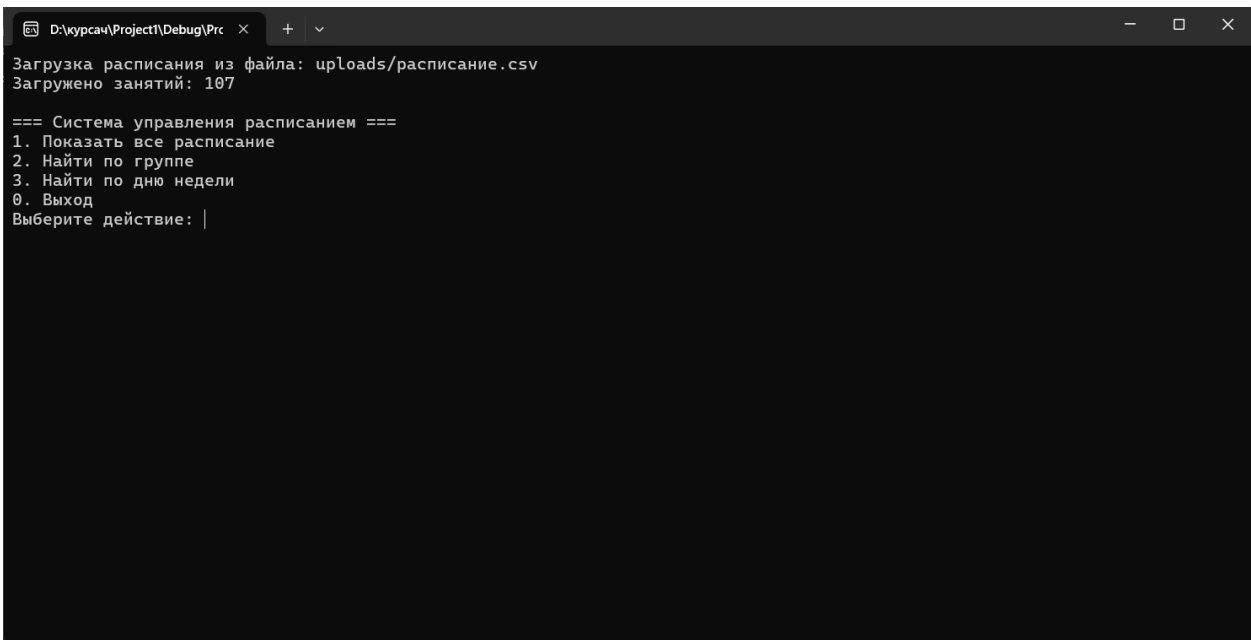
Руководство пользователя

Проект Project1.sln предназначен для хранения информации о расписании занятий в университете. Программа имеет понятный интерфейс и поддерживает такие операции как:

- Сколько занятий загружено в Базу данных;
- Показать все расписание сразу;
- Показать расписание определенной группы;
- Показать расписание групп по определенному дню недели;
- Выйти из программы.

После запуска программы мы увидим окошко с некоторыми возможными действиями, которые доступны пользователю.

Рисунок 13 — Консольное окно с возможными действиями



```
D:\курсач\Project1\Debug\Prс  x  +  v
Загрузка расписания из файла: uploads/расписание.csv
Загружено занятий: 107

=== Система управления расписанием ===
1. Показать все расписание
2. Найти по группе
3. Найти по дню недели
0. Выход
Выберите действие: |
```

В открывшемся меню выберем пункт «Показать все расписание»

Рисунок 14 и 15 — Консольное окно с пунктом «Показать все расписание»

```

D:\курсач\Project\Debug\Prc x + v
Загрузка расписания из файла: uploads/расписание.csv
Загружено занятий: 107

=== Система управления расписанием ===
1. Показать все расписание
2. Найти по группе
3. Найти по дню недели
0. Выход
Выберите действие: 1

=== Полное расписание ===
Группа День Пара Предмет Тип Начало Конец
-----
25VVV1 MON 2 gym - 9:50 11:25
25VVV1 MON 3 physics lecture 11:40 13:15
25VVV1 MON 4 logic foundations/curatorial hour lecture/lesson 13:45 15:20
25VVV1 TUES 4 logic foundations laboratory 13:45 15:20
25VVV1 TUES 5 maths practical 15:35 17:10
25VVV1 TUES 6 information technology laboratory 17:25 19:00
25VVV1 WEDNES 4 physics laboratory 13:45 15:20
25VVV1 WEDNES 5 programming lecture 15:35 17:10
25VVV1 WEDNES 6 programming laboratory 17:25 19:00
25VVV1 THURS 3 russian language practical 11:40 13:15
25VVV1 THURS 4 legal studies/maths practical/practical 13:45 15:20
25VVV1 THURS 5 maths lecture 15:35 17:10
25VVV1 THURS 6 nothing/history nothing/practical 17:25 19:00
25VVV1 FRI 2 gym - 9:50 11:25
25VVV1 FRI 3 history lecture 11:40 13:15
25VVV1 FRI 4 information networks/foreign language laboratory/laboratory 13:45 15:20
25VVV1 SATUR 3 nothing/russian language nothing/lecture 11:40 13:15

```

```

D:\курсач\Project\Debug\Prc x + v
25VOA1 FRI 3 maths lecture 11:40 13:15
25VOA1 FRI 4 technology lecture 13:45 15:20
25VOA1 SATUR 1 nothing/technology nothing/laboratory 8:00 9:35
25VOA1 SATUR 2 technology laboratory 9:50 11:25
25VOA1 SATUR 3 technology practical 11:40 13:15
25VOA2 MON 2 gym - 9:50 11:25
25VOA2 MON 3 physics/nothing lecture/nothing 11:40 13:15
25VOA2 MON 4 nothing/curatorial hour nothing/lesson 13:45 15:20
25VOA2 TUES 5 programming laboratory 15:35 17:10
25VOA2 TUES 6 programming laboratory 17:25 19:00
25VOA2 WEDNES 3 nothing/history nothing/practical 11:40 13:15
25VOA2 WEDNES 4 history lecture 13:45 15:20
25VOA2 WEDNES 5 physics laboratory 15:35 17:10
25VOA2 THURS 3 nothing/legal support nothing/lecture 11:40 13:15
25VOA2 THURS 4 programming lecture 13:45 15:20
25VOA2 THURS 5 foreign language laboratory 15:35 17:10
25VOA2 THURS 6 legal support practical 17:25 19:00
25VOA2 FRI 2 gym - 9:50 11:25
25VOA2 FRI 3 maths lecture 11:40 13:15
25VOA2 FRI 4 technology lecture 13:45 15:20
25VOA2 SATUR 2 maths practical 9:50 11:25
25VOA2 SATUR 3 technology laboratory 11:40 13:15
25VOA2 SATUR 4 technology/nothing laboratory/nothing 13:45 15:20

=== Система управления расписанием ===
1. Показать все расписание
2. Найти по группе
3. Найти по дню недели
0. Выход
Выберите действие: |

```

Теперь выберем пункт «Найти по группе» и введем любую группу кафедры «Вычислительная техника». Нам выдаст все расписание именно этой группы на все 2 недели

Рисунок 15 — Консольное окно с пунктом «Найти по группе»

```

D:\курсач\Project\Debug\Prc x + v
Выберите действие: 2
Введите название группы: 25VVV2

=== Расписание для группы 25VVV2 ===

```

День	Пара	Предмет	Тип	Начало	Конец
MON	2	gym	-	9:50	11:25
MON	3	physics	lecture	11:40	13:15
MON	4	logic foundations/curatorial hour	lecture/lesson	13:45	15:20
TUES	3	logic foundations	laboratory	11:40	13:15
TUES	4	maths	practical	13:45	15:20
TUES	5	information technology	laboratory	15:35	17:10
WEDNES	3	physics	laboratory	11:40	13:15
WEDNES	4	programming	laboratory	13:45	15:20
WEDNES	5	programming	lecture	15:35	17:10
THURS	3	foreign language/information networks	practical/laboratory	11:40	13:15
THURS	4	maths/legal studies	practical/practical	13:45	15:20
THURS	5	maths	lecture	15:35	17:10
FRI	2	gym	-	9:50	11:25
FRI	3	history	lecture	11:40	13:15
FRI	4	history/russian language	practical/practical	13:45	15:20
SATUR	3	nothing/russian language	nothing/lecture	11:40	13:15
SATUR	4	nothing/information networks	nothing/lecture	13:45	15:20

```

=== Система управления расписанием ===
1. Показать все расписание
2. Найти по группе
3. Найти по дню недели
0. Выход
Выберите действие: |

```

Далее нажмем на клавиатуре на цифру 3 и выберем любой день недели. Получим на экране все занятия в этот день

Рисунок 16 — Консольное окно с пунктом «Найти по дню недели»

```

D:\курсач\Project\Debug\Prc x + v
Введите день недели: MON

=== Расписание на MON ===

```

Группа	Пара	Предмет	Тип	Начало	Конец
25VVV1	2	gym	-	9:50	11:25
25VVV1	3	physics	lecture	11:40	13:15
25VVV1	4	logic foundations/curatorial hour	lecture/lesson	13:45	15:20
25VVV2	2	gym	-	9:50	11:25
25VVV2	3	physics	lecture	11:40	13:15
25VVV2	4	logic foundations/curatorial hour	lecture/lesson	13:45	15:20
25VVV3	2	gym	-	9:50	11:25
25VVV3	3	physics	lecture	11:40	13:15
25VVV3	4	logic foundations/curatorial hour	lecture/lesson	13:45	15:20
25VVV4	2	gym	-	9:50	11:25
25VVV4	3	physics	lecture	11:40	13:15
25VVV4	4	logic foundations/curatorial hour	lecture/lesson	13:45	15:20
25V0A1	2	gym	-	9:50	11:25
25V0A1	3	physics	lecture	11:40	13:15
25V0A1	4	nothing/curatorial hour	nothing/lesson	13:45	15:20
25V0A2	2	gym	-	9:50	11:25
25V0A2	3	physics/nothing	lecture/nothing	11:40	13:15
25V0A2	4	nothing/curatorial hour	nothing/lesson	13:45	15:20

```

=== Система управления расписанием ===
1. Показать все расписание
2. Найти по группе
3. Найти по дню недели
0. Выход
Выберите действие: |

```

Последнее, что осталось — это пункт «Выход». Нажав на него, пользователь выйдет из консоли.

Отладка и возникшие трудности

При разработке ассемблерных функций я столкнулся с несколькими трудностями. Первая проблема была связана с синтаксисом встроенного ассемблера. В отличие от отдельных ассемблерных файлов, где используется синтаксис MASM или NASM, встроенный ассемблер в Visual Studio имеет свои особенности. Например, нужно явно указывать размер операнда с помощью `byte ptr`, `word ptr` или `dword ptr`, иначе компилятор не сможет определить, сколько байт нужно читать или записывать.

Вторая сложность заключалась в правильном сохранении и восстановлении регистров. Сначала я забыл сохранить регистры `esi` и `edi`, и программа работала нестабильно: иногда всё было нормально, а иногда возникали странные ошибки в других частях программы. Это произошло потому, что я случайно перезаписывал значения регистров, которые использовались в вызывающей функции. После того как я добавил команды `push` и `pop`, проблема исчезла.

Третья трудность была связана с передачей параметров и возвратом результата. Я потратил некоторое время, чтобы разобраться, как правильно обращаться к параметрам функции внутри блока `__asm`. Оказалось, что компилятор позволяет использовать имена переменных напрямую, и он сам подставляет правильные адреса. Это очень удобно и избавляет от необходимости вручную вычислять смещения в стеке.

Для отладки ассемблерного кода я использовал встроенный отладчик Visual Studio, который позволяет пошагово выполнять программу и наблюдать за значениями регистров и памяти. Это очень помогло понять, что происходит на каждом шаге выполнения функции. Я ставил точки останова на входе в функцию и внутри цикла, смотрел, какие значения находятся в регистрах `al` и `bl`, проверял, правильно ли устанавливаются флаги процессора после команды `cmp`.

Ещё одна проблема, с которой я столкнулся, — это работа с русскими символами. Строки в программе используют кодировку Windows-1251, где каждый русский символ занимает один байт. Функции `asm_strcmp` и `asm_strcmpu` работают побайтно, поэтому они корректно обрабатывают такие строки. Однако если бы программа использовала кодировку UTF-8, где русские символы занимают два или более байт, эти функции работали бы неправильно, так как они не учитывают многобайтовые символы. Это важное ограничение, о котором нужно помнить при работе со строками на низком уровне.

Заключение

В ходе выполнения курсовой была разработана программа для автоматизации работы с расписанием занятий в учебном заведении с применением метода нисходящего программирования [3, 5, 16]. Поставленная цель достигнута: создана функционирующая система, которая позволяет загружать данные о занятиях из файла, осуществлять поиск по различным критериям и выводить результаты в удобном табличном формате.

Метод нисходящего программирования продемонстрировал свою эффективность при решении задачи подобного класса [16, 22, 23]. Разбиение сложной задачи на иерархию модулей позволило сосредоточиться на логике работы программы на каждом уровне абстракции, не отвлекаясь на детали реализации. Главная функция координирует работу модулей верхнего уровня, которые, в свою очередь, используют функции более низкого уровня для выполнения конкретных операций. Такая структура обеспечила прозрачность кода и упростила процесс разработки [10, 15].

Первая глава курсовой была посвящена теоретическим основам нисходящего программирования и анализу предметной области. Были рассмотрены ключевые принципы метода: разделение задачи на подзадачи, абстракция и сокрытие информации, пошаговая детализация, модульность и слабая связанность [2, 3, 5, 6]. Изучение этих принципов позволило понять, как правильно проектировать структуру программы и организовывать взаимодействие между модулями [4, 21]. Анализ предметной области выявил основные сущности системы расписания: группы студентов, преподаватели, аудитории, дисциплины и временные слоты. Были определены функциональные требования к программе и выявлены типичные конфликты, которые необходимо учитывать при составлении расписания [18, 24].

Вторая глава была посвящена практической реализации программы. На этапе проектирования была разработана модульная архитектура, включающая три основных блока: модуль управления данными, модуль формирования расписания и модуль проверки корректности [4, 6, 21]. Каждый модуль был

детализирован до уровня конкретных функций, что позволило чётко распределить ответственность между компонентами системы. Реализация программы велась на языке C с использованием структур данных для хранения информации о занятиях [8, 25]. Для демонстрации возможности интеграции низкоуровневого кода были реализованы функции сравнения и копирования строк на языке ассемблера [12, 13, 20].

Разработанная программа успешно справляется с поставленными задачами. Модуль загрузки данных корректно читает информацию из CSV-файла, обрабатывает различные форматы записей и заполняет массив структур. Модуль поиска позволяет быстро находить занятия по группе или дню недели, что особенно важно для пользователей, которым необходим оперативный доступ к расписанию. Модуль вывода обеспечивает наглядное представление результатов в табличном виде с выравниванием столбцов.

Тестирование программы показало её надёжность и корректность работы [10, 11]. Были проверены различные сценарии использования: загрузка файлов разного размера, поиск существующих и несуществующих записей, обработка некорректного ввода пользователя. Программа стабильно работает с файлами, содержащими до тысячи записей, что вполне достаточно для типичного учебного заведения. Время отклика на запросы пользователя остаётся незаметным, что обеспечивает комфортную работу с системой.

Использование ассемблерных вставок продемонстрировало возможность сочетания высокоуровневого и низкоуровневого программирования в рамках одного проекта [12, 13, 17, 20]. Хотя в данном случае выигрыш в производительности минимален, опыт работы с ассемблером оказался ценным для понимания принципов работы процессора, организации памяти и механизмов вызова функций [14]. Реализация функции сравнения строк на ассемблере потребовала глубокого понимания архитектуры x86, работы с регистрами и управления потоком выполнения программы.

Модульная архитектура упрощает доработки: можно добавить поиск по преподавателям, экспорт в Excel, поиск по преподавателю, фильтрацию по типу занятий, экспорт расписания в различные форматы, графический интерфейс пользователя. Каждое новое расширение может быть реализовано в виде отдельного модуля, который интегрируется в существующую систему без необходимости переписывать уже работающий код. Это одно из главных преимуществ нисходящего подхода: программа остаётся открытой для модификаций, но закрытой для изменений в уже реализованных модулях.

Практическая значимость разработанной программы заключается в возможности её использования в реальных условиях учебных заведений. Автоматизация работы с расписанием позволяет сократить временные затраты методистов и диспетчеров учебных отделов, минимизировать количество ошибок при поиске информации и обеспечить быстрый доступ к данным для студентов и преподавателей. Программа может быть адаптирована под специфику конкретного учебного заведения путём модификации формата входных данных и добавления необходимых функций.

В итоге курсовая продемонстрировала, что метод нисходящего программирования является эффективным инструментом для разработки программ сложной структуры [3, 5, 16, 23]. Систематический подход к декомпозиции задачи, чёткое определение интерфейсов модулей и последовательная детализация функций позволяют создавать качественное ПО, которое легко понять, протестировать и сопровождать. Полученный опыт применения нисходящего подхода может быть использован при разработке других программных систем, где требуется управление сложностью и обеспечение надёжности кода [18, 24].

Список использованных источников

Печатные издания

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – 2-е изд. – М. : Вильямс, 2008. – 1184 с.
2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – М. : ДМК Пресс, 2010. – 272 с.
3. Дейкстра, Э. Дисциплина программирования / Э. Дейкстра. – М. : Мир, 1978. – 275 с.
4. Йодан, Э. Структурное проектирование и конструирование программ / Э. Йодан. – М. : Мир, 1979. – 415 с.
5. Касьянов, В. Н. Методы построения трансляторов / В. Н. Касьянов, И. В. Поттосин. – Новосибирск : Наука, 1986. – 344 с.
6. Керниган, Б. Язык программирования С / Б. Керниган, Д. Ритчи. – 2-е изд. – М. : Вильямс, 2009. – 304 с.
7. Кнут, Д. Искусство программирования. Том 1. Основные алгоритмы / Д. Кнут. – 3-е изд. – М. : Вильямс, 2006. – 720 с.
8. Макконнелл, С. Совершенный код. Мастер-класс / С. Макконнелл. – М. : Русская редакция, 2010. – 896 с.
9. Майерс, Г. Надежность программного обеспечения / Г. Майерс. – М. : Мир, 1980. – 360 с.
10. Марков, Е. П. Программирование на ассемблере для процессоров Intel Pentium / Е. П. Марков. – СПб. : БХВ-Петербург, 2008. – 512 с.

11. Пильщиков, В. Н. Программирование на языке ассемблера IBM PC / В. Н. Пильщиков. – М. : Диалог-МИФИ, 2005. – 288 с.

12. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум. – 5-е изд. – СПб. : Питер, 2007. – 844 с.

13. Фаулер, М. Рефакторинг: улучшение существующего кода / М. Фаулер. – СПб. : Символ-Плюс, 2009. – 432 с.

Электронные ресурсы

14. Введение в структурное программирование [Электронный ресурс]. – Режим доступа: <https://intuit.ru/studies/courses/2247/160/lecture/4448> (дата обращения: 15.04.2026).

15. Документация Microsoft Visual C++ [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/cpp/> (дата обращения: 10.04.2026).

16. Методы проектирования программного обеспечения [Электронный ресурс]. – Режим доступа: <https://www.intuit.ru/studies/courses/1/1/info> (дата обращения: 12.04.2026).

17. Нисходящее и восходящее проектирование [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/276593/> (дата обращения: 14.04.2026).

18. Основы программирования на ассемблере [Электронный ресурс]. – Режим доступа: <https://www.codenet.ru/progr/asm/> (дата обращения: 16.04.2026).

19. Принципы модульного программирования [Электронный ресурс].
– Режим доступа: <https://www.intuit.ru/studies/courses/3481/698/lecture/15107>
(дата обращения: 11.04.2026).

20. Разработка алгоритмов методом пошаговой детализации
[Электронный ресурс]. – Режим доступа:
<https://informatics.mccme.ru/course/view.php?id=156> (дата обращения:
13.04.2026).

21. Структурное программирование: основные концепции
[Электронный ресурс]. – Режим доступа:
<https://www.intuit.ru/studies/courses/3470/708/info> (дата обращения: 17.04.2026).

22. Технологии разработки программного обеспечения [Электронный
ресурс]. – Режим доступа: <https://www.intuit.ru/studies/courses/497/353/info>
(дата обращения: 18.04.2026).

23. Язык программирования С: справочное руководство
[Электронный ресурс]. – Режим доступа: <https://en.cppreference.com/w/c> (дата
обращения: 19.04.2026).

Приложение а. Листинг программы

Файл «Source.cpp»

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Windows.h>
#include <locale.h>
#include "main.h"
Lesson schedule[MAX_LESSONS];
int lesson_count = 0;
int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    char* locale = setlocale(LC_ALL, "");
    char filename[256] = "uploads/расписание.csv";
    char input[100];
    int choice;
    printf("Загрузка расписания из файла: %s\n", filename);
    int loaded = load_schedule_csv(filename);
    if (loaded == 0) {
        printf("Не удалось загрузить расписание или файл
пуст.\n");
        printf("Нажмите Enter для выхода...");
        getchar();
        return 1;
    }
    printf("Загружено занятий: %d\n", lesson_count);
    while (1) {
        show_menu();
        if (scanf("%d", &choice) != 1) {
            printf("Ошибка ввода. Введите число.\n");
            while (getchar() != '\n');
            continue;
        }
        getchar();
    }
```

```

switch (choice) {
case 1:
    print_all_schedule();
    break;
case 2:
    printf("Введите название группы: ");
    if (fgets(input, sizeof(input), stdin) != NULL)
    {
        input[strcspn(input, "\n")] = 0;
        find_by_group(input);
    }
    break;
case 3:
    printf("Введите день недели: ");
    if (fgets(input, sizeof(input), stdin) != NULL)
    {
        input[strcspn(input, "\n")] = 0;
        find_by_day(input);
    }
    break;
case 0:
    printf("Выход из программы.\n");
    return 0;
default:
    printf("Неверный выбор. Попробуйте снова.\n");
}
return 0;
}

```

Файл «main.h»

```

#include <stdio.h>
#define MAX_LESSONS 1000
typedef struct {
    char group[20];
    char day[20];

```

```

    int pair;
    char subject[150];
    char type[30];
    char start[10];
    char end[10];
} Lesson;

int asm_strcmp(const char* str1, const char* str2);
void asm_strcpy(char* dest, const char* src);
void trim_string(char* str);
void safe_copy(char* dest, const char* src, size_t dest_size);
int load_schedule_csv(const char* filename);
void print_all_schedule();
void find_by_group(const char* group);
void find_by_day(const char* day);
void show_menu();

```

Файл «asm_strcmp.cpp»

```

#include "main.h"

int asm_strcmp(const char* str1, const char* str2) {
    int result;
    __asm {
        push esi
        push edi
        mov esi, str1
        mov edi, str2
        compare_loop :
        mov al, byte ptr[esi]
        mov bl, byte ptr[edi]
        cmp al, bl
        jne not_equal
        test al, al
        je equal
        inc esi
        inc edi
        jmp compare_loop
        not_equal :
    }
}

```

```

        sub al, bl
        movsx eax, al
        jmp done
    equal :
    xor eax, eax
    done :
    mov result, eax
    pop edi
    pop esi
}
return result;
}

```

Файл «asm_strcpy.cpp»

```

#include "main.h"

void asm_strcpy(char* dest, const char* src) {
    __asm {
        push esi
        push edi
        mov edi, dest
        mov esi, src
    copy_loop :
        mov al, byte ptr[esi]
        mov byte ptr[edi], al
        test al, al
        je copy_done
        inc esi
        inc edi
        jmp copy_loop
    copy_done :
        pop edi
        pop esi
    }
}

```

Файл «show_menu.cpp»

```

#include "main.h"

```

```

#include <stdio.h>

void show_menu() {
    printf("\n=== Система управления расписанием ===\n");
    printf("1. Показать все расписание\n");
    printf("2. Найти по группе\n");
    printf("3. Найти по дню недели\n");
    printf("0. Выход\n");
    printf("Выберите действие: ");
}

```

Файл «print_all_schedule.cpp»

```

#include "main.h"
#include <stdio.h>
#include <string.h>
extern Lesson schedule[MAX_LESSONS];
extern int lesson_count;
void print_all_schedule() {
    if (lesson_count == 0) {
        printf("\nРасписание пусто.\n");
        return;
    }
    printf("\n=== Полное расписание ===\n");
    printf("%-6s %-6s %-5s %-40s %-20s %-8s %-8s\n",
        "Группа", "День", "Пара", "Предмет", "Тип", "Начало",
"Конец");
    printf("-----\n");
    for (int i = 0; i < lesson_count; i++) {
        printf("%-6s %-6s %-5d %-40s %-20s %-8s %-8s\n",
            schedule[i].group, schedule[i].day,
schedule[i].pair,
            schedule[i].subject, schedule[i].type,
            schedule[i].start, schedule[i].end);
    }
}

```

Файл «load_schedule_csv.cpp»

```

#include "main.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
extern Lesson schedule[MAX_LESSONS];
extern int lesson_count;
int load_schedule_csv(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (!file) {
        printf("Ошибка: не удалось открыть файл %s\n",
filename);
        return 0;
    }
    char line[512];
    int skip_header = 1;
    lesson_count = 0;
    while (fgets(line, sizeof(line), file) && lesson_count <
MAX_LESSONS) {
        if (skip_header) {
            skip_header = 0;
            continue;
        }
        if (line[0] == '\n' || line[0] == '\r' || line[0] ==
'\0') {
            continue;
        }
        Lesson temp = { 0 };
        char* token;
        int col = 0;
        token = strtok(line, ";");
        while (token != NULL && col < 7) {
            trim_string(token);
            switch (col) {
                case 0: safe_copy(temp.group, token,
sizeof(temp.group)); break;

```

```

        case 1:      safe_copy(temp.day,      token,
sizeof(temp.day)); break;
        case 2: temp.pair = atoi(token); break;
        case 3:      safe_copy(temp.subject,  token,
sizeof(temp.subject)); break;
        case 4:      safe_copy(temp.type,     token,
sizeof(temp.type)); break;
        case 5:      safe_copy(temp.start,    token,
sizeof(temp.start)); break;
        case 6:      safe_copy(temp.end,      token,
sizeof(temp.end)); break;
    }
    token = strtok(NULL, ";");
    col++;
}
if (strlen(temp.group) > 0) {
    schedule[lesson_count++] = temp;
}
}
fclose(file);
return lesson_count;
}

```

Файл «find_by_group.cpp»

```

#include "main.h"
#include <stdio.h>
extern Lesson schedule[MAX_LESSONS];
extern int lesson_count;
void find_by_group(const char* group) {
    printf("\n=== Расписание для группы %s ===\n", group);
    printf("%-6s %-5s %-40s %-20s %-8s %-8s\n",
        "День", "Пара", "Предмет", "Тип", "Начало", "Конец");
    printf("-----\n");
    printf("-----\n");

    int found = 0;
    for (int i = 0; i < lesson_count; i++) {

```

```

        if (asm_strcmp(schedule[i].group, group) == 0) {
            printf("%-6s %-5d %-40s %-20s %-8s %-8s\n",
                schedule[i].day,                schedule[i].pair,
schedule[i].subject,
                schedule[i].type,                schedule[i].start,
schedule[i].end);
            found = 1;
        }
    }
    if (!found) {
        printf("Занятия для группы %s не найдены.\n", group);
    }
}

void find_by_day(const char* day) {
    printf("\n=== Расписание на %s ===\n", day);
    printf("%-6s %-5s %-40s %-20s %-8s %-8s\n",
        "Группа",    "Пара",    "Предмет",    "Тип",    "Начало",
"Конец");
    printf("-----\n");

    int found = 0;
    for (int i = 0; i < lesson_count; i++) {
        if (asm_strcmp(schedule[i].day, day) == 0) {
            printf("%-6s %-5d %-40s %-20s %-8s %-8s\n",
                schedule[i].group,                schedule[i].pair,
schedule[i].subject,
                schedule[i].type,                schedule[i].start,
schedule[i].end);
            found = 1;
        }
    }
    if (!found) {
        printf("Занятия на %s не найдены.\n", day);
    }
}

```

Файл «trim_string.cpp»

```
#include "main.h"
#include <string.h>

void trim_string(char* str) {
    if (!str) return;
    char* start = str;
    while (*start == ' ' || *start == '"' || *start == '\\t')
start++;
    char* end = start + strlen(start) - 1;
    while (end >= start && (*end == ' ' || *end == '"' ||
*end == '\\n' || *end == '\\r' || *end == '\\t')) {
        *end = '\\0';
        end--;
    }
    if (start != str) {
        memmove(str, start, strlen(start) + 1);
    }
}

void safe_copy(char* dest, const char* src, size_t dest_size)
{
    if (dest_size == 0) return;
    size_t src_len = strlen(src);
    if (src_len >= dest_size) {
        src_len = dest_size - 1;
    }
    char temp[512];
    strncpy(temp, src, src_len);
    temp[src_len] = '\\0';
    asm_strcpy(dest, temp);
}
```