Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

М. М. Меженная

ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ. КУРСОВОЕ ПРОЕКТИРОВАНИЕ

*Рекомендовано УМО по образованию
в области информатики и радиоэлектроники для специальностей
1-40 05 01 «Информационные системы и технологии (по направлениям)»,
1-58 01 01 «Инженерно-психологическое обеспечение
информационных технологий»
в качестве пособия*

Минск БГУИР 2019

УДК 004.42(076)
ББК 32.973.3я73
М43

Рецензенты:

кафедра интеллектуальных систем
Белорусского национального технического университета
(протокол №5 от 04.12.2018);

старший научный сотрудник лаборатории логического проектирования
государственного научного учреждения «Объединенный институт
проблем информатики Национальной академии наук Беларуси»,
кандидат технических наук С. Н. Кардаш

Меженная, М. М.
М43 Основы конструирования программ. Курсовое проектирование : пособие / М. М. Меженная. – Минск : БГУИР, 2019. – 80 с. : ил.
ISBN 978-985-543-507-6.

Пособие посвящено рассмотрению вопросов конструирования и реализации компьютерных программ на языке C++ в рамках процедурной парадигмы. Содержит методические указания к выполнению курсовой работы.

УДК 004.42(076)
ББК 32.973.3я73

ISBN 978-985-543-507-6

© Меженная М. М., 2019
© УО «Белорусский государственный
университет информатики
и радиоэлектроники», 2019

Содержание

Введение 4
1 Цель и задачи курсовой работы, требования к ее выполнению 5
1.1 Исходные данные 5
1.2 Функциональные требования 6
1.3 Требования к программной реализации 10
2 Структура и описание разделов пояснительной записки 12
3 Порядок защиты и критерии оценки курсовой работы 16
4 Рекомендации по проектированию программы 18
4.1 Принципы и уровни проектирования программы в курсовой работе 18
4.2 Способы организации работы по чтению/записи в файл 20
4.3 Принципы организации работы с массивами и векторами 21
4.4 Минимизация области видимости переменных 27
4.5 Разделение программы на независимые cpp-файлы и их подключение с помощью заголовочных h-файлов 28
5 Рекомендации по разработке алгоритмов работы программы 32
6 Рекомендации по программированию курсовой работы 45
6.1 Типичные ошибки начинающих при написании кода. Способы отслеживания и устранения ошибок. Создание exe-файла проекта 45
6.2 Структуры. Запись и чтение из файла 48
6.3 Перевыделение памяти с целью увеличения размера динамически созданного массива 60
6.4 Проверка корректности вводимых данных 62
6.5 Определение текущей даты и времени 65
ПРИЛОЖЕНИЕ А Задания для курсовой работы 69
ПРИЛОЖЕНИЕ Б Образец титульного листа курсовой работы 79
Список использованных источников 80

Введение

Настоящее пособие представляет собой методические указания к выполнению курсовой работы по дисциплине «Основы конструирования программ».

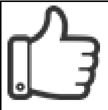
Курсовая работа ориентирована на студентов, изучающих C++, и направлена на формирование базового навыка – умения программировать в процедурной парадигме. Это в свою очередь создаст фундамент для перехода в будущем к объектно-ориентированному программированию.

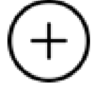
В пособии подробно изложены цель и задачи курсовой работы, требования к ее выполнению, содержание пояснительной записки, порядок защиты и критерии оценки курсовой работы. В приложении к данному пособию приведены варианты заданий для курсовой работы.

Особое внимание уделено процессам проектирования программ, принципам организации работы с файлами, массивами, векторами. Подробно раскрыта тема разработки алгоритмов; рассмотрены примеры кода для реализации отдельных функциональных задач курсовой работы. Многочисленные рекомендации и примеры, содержащиеся в настоящем пособии, сформированы по результатам анализа типичных ошибок и проблем, возникающих у студентов в процессе выполнения курсовой работы.

Для визуального выделения информационных блоков с различной смысловой нагрузкой в пособии используются следующие условные обозначения.

	Важно: информация, требующая особого внимания. В основу информации данного блока положены принципиально важные аспекты курсовой работы, а также предостережения ввиду часто возникающих ошибок.

	**Рекомендуется:** полезные советы и инструменты, поясняющие или существенно облегчающие процесс работы над курсовой, а также способствующие созданию качественных программ.

	**Дополнительно:** предложения по расширению функциональных возможностей курсовой работы, зачастую предусматривающие самостоятельное изучение материала для их реализации.

1 Цель и задачи курсовой работы, требования к ее выполнению

Целью курсовой работы является получение студентами теоретических знаний и практических навыков по конструированию и реализации программ на языке C++ в рамках процедурной парадигмы.

На уровне конструирования задачей курсовой работы является освоение технологии нисходящего проектирования и принципов методологии структурного программирования. Результат этапа конструирования представляется в форме графических блок-схем алгоритмов.

На уровне программирования задачей курсовой работы является реализация графических блок-схем алгоритмов на языке C++ с соблюдением правил соглашения о коде (C++ Code Convention). Результат этапа программирования представляется в форме проекта в среде разработки приложений на языке C++ (например, Microsoft Visual Studio).

1.1 Исходные данные

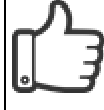
1. Тема курсовой работы выбирается из списка, приведенного в приложении А.
2. Язык программирования C++.
3. Среда разработки Microsoft Visual Studio.
4. Вид приложения – консольное.
5. Парадигма программирования – процедурная (*по согласованию с преподавателем допускается реализация программы в рамках объектно-ориентированной парадигмы программирования*).
6. Способ организации данных – структуры (struct) (*либо поля классов в случае объектно-ориентированного программирования*).
7. Способ хранения данных – файлы (*по согласованию с преподавателем допускается подключение баз данных*).
8. Каждая логически завершенная подзадача программы должна быть реализована в виде отдельной функции (*метода в случае объектно-ориентированного программирования*).
9. Построение программного кода должно соответствовать соглашению о коде «C++ Code Convention».
10. К защите курсовой работы представляются: консольное приложение и пояснительная записка.

11. Текст пояснительной записки оформляется в соответствии со стандартом предприятия СТП 01–2017.

	Важно: код курсовой работы должен быть уникальным. Об уникальности отдельно взятой курсовой работы свидетельствуют соответствующие тематике имена переменных, констант, функций; комментарии в коде; проработка исключительных ситуаций; реализация дополнительных функциональных возможностей.

Допускается использовать классы string и vector, библиотеку algorithm.

Для защиты курсовой работы необходимо представить распечатанную пояснительную записку с листингом кода и продемонстрировать работу программы.

	**Рекомендуется:** делать резервные копии своей программы на случай внезапной неисправности персонального компьютера. Используйте облачные технологии или внешние носители для резервного хранения информации.

1.2 Функциональные требования

Первым этапом работы программы является авторизация – предоставление прав доступа.

В рамках данного этапа необходимо считать данные из файла с учетными записями пользователей следующего вида:

– login;

– password;

– role (данное поле служит для разделения в правах администраторов и пользователей).

После ввода пользователем своих персональных данных (логина и пароля) и сверки со считанной из файла информацией необходимо предусмотреть возможность входа:

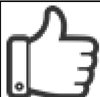
– в качестве администратора (в этом случае, например, role = 1);

– в качестве пользователя (в этом случае, например, role = 0).

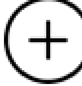
	Важно: если файл с учетными записями пользователей не существует, то необходимо программно создать его и записать учетные данные администратора.

Дополнительно: по соображениям безопасности в форме авторизации целесообразно маскировать пароль с помощью символов «звездочки» *.

Регистрация новых пользователей осуществляется администратором в режиме работы с учетными записями пользователей (т. е. администратор сам создает для пользователей аккаунты).



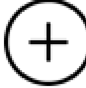
Рекомендуется: выполнять проверку новых учетных записей на уникальность логина.



Дополнительно: в курсовой работе можно реализовать еще один – альтернативный – способ регистрации, а именно регистрацию самим пользователем путем ввода желаемых логина и пароля и ожидания подтверждения администратором новой учетной записи. Для реализации этого способа в структуре учетных записей пользователей необходимо предусмотреть дополнительное поле access:

- login;
- password;
- role;
- access (данное поле служит для подтверждения или блокировки администратором учетных записей).

По умолчанию access = 0 при попытке зарегистрироваться; далее администратор меняет значение на access = 1 и тем самым подтверждает новую учетную запись: пользователь может осуществить вход в систему.



Дополнительно: в действительности по соображениям безопасности пароли учетных записей никогда не хранятся в открытом виде. Целесообразно выполнять хеширование пароля с «солью» («соль» – случайная строка, специально сгенерированная для данной учетной записи). И «соль», и результат применения хеш-функции к паролю с «солью» хранятся в базе данных. В этом случае структрура учетных записей пользователей будет иметь вид:

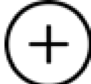
- login;
- salted_hash_password (результат хеширования пароля с «солью»);
- salt («соль»);
- role;
- access.

Вторым этапом работы программы является собственно работа с данными, которая становится доступной только после прохождения авторизации. Данные хранятся в отдельном файле и имеют вид, описанный подробно в каждом варианте к курсовой работе.

Для работы с данными должны быть предусмотрены два функциональных модуля: модуль администратора и модуль пользователя.

Модуль администратора включает следующие подмодули (с указанием функциональных возможностей):

1. Управление учетными записями пользователей:
   - просмотр всех учетных записей;
   - добавление новой учетной записи;
   - редактирование учетной записи;
   - удаление учетной записи.



Дополнительно: при реализации альтернативного способа регистрации (самим пользователем) необходимо предусмотреть еще две функциональных возможности администатора:
– подтверждение учетной записи;
– блокировка учетной записи.

Допускается создавать учетные записи для нескольких администраторов.

Важно: при реализации функционала удаления учетных записей запретить возможность удаления администратором собственной учетной записи. Это позволит исключить ситуацию удаления всех администраторов (что сделало бы дальнейшее редактирование учетных записей и данных невозможным).

2. Работа с данными:

   а) режим редактирования:
   - просмотр всех данных;
   - добавление новой записи;
   - удаление записи;
   - редактирование записи;

   б) режим обработки данных:
   - выполнение индивидуального задания;
   - поиск данных (как минимум по трем различным параметрам);
   - сортировка (как минимум по трем различным параметрам).

Модуль пользователя включает подмодуль работы с данными со следующими функциональными возможностями:

– просмотр всех данных;
– выполнение индивидуального задания;
– поиск данных (как минимум по трем различным параметрам);
– сортировка (как минимум по трем различным параметрам).

Для реализации перечисленных модулей/подмодулей необходимо создавать меню с соответствующими пунктами (примеры приведены на рисунках 1.1-1.3).

```
Введите логин - пользователь

Введите пароль- 12345
------------Меню пользователя-----------------------------------

1.Просмотр данных;
2.Поиск и фильтрация;
3.Сортировка;
0.Выход в главное меню.
```

Рисунок 1.1 – Пример авторизации и меню для пользователя

```
------------- Поиск студента -----------------------------------

ФИО искомого студента Петр

          Информация о найденых студентах

-----------------------------------------------------------------
¦  №  ¦     ФИО     ¦ № гр.¦Матем.¦  Физ ¦ Инф. ¦ Ср.б.¦Актив.¦Стипендия ¦

-----------------------------------------------------------------
¦  1  ¦  Петров     ¦12    ¦  4   ¦  5   ¦  5   ¦ 4.67 ¦  1   ¦  7500.00 ¦
-----------------------------------------------------------------
¦  2  ¦  Петрова    ¦21    ¦  5   ¦  3   ¦  5   ¦ 4.33 ¦  0   ¦  0.00    ¦
-----------------------------------------------------------------
```

Рисунок 1.2 – Пример поиска

```
------------Меню сортировки ------------------------------------
 1.По алфавиту;
 2.По среднему баллу;
 3.По стипендии
 0.Назад.
1

    Сортированная информация
```

№	ФИО	№ гр.	Матем.	Физ	Инф.	Ср.б.	Актив.	Стипендия
1	Антонов	21	3	5	5	4.33	1	5000.00
2	Иванов	12	5	5	5	5.00	0	6250.00
3	Петров	12	4	5	5	4.67	1	7500.00
4	Петрова	21	5	3	5	4.33	0	0.00
5	Сидоров	11	4	5	4	4.33	1	7500.00
6	Чернов	23	4	4	5	4.33	0	6250.00

Рисунок 1.3 – Пример сортировки

В курсовой работе необходимо предусмотреть:

1. Обработку исключительных ситуаций:
   - введенные пользователем данные не соответствуют формату поля *(например, символы в числовом поле)*;
   - введенные пользователем данные нелогичны *(например, отрицательная цена товара)*;
   - файл с данными для чтения не существует;
   - ничего не найдено по результатам поиска;
   - номер удаляемой записи выходит за пределы массива/вектора.
2. Возможность возврата назад (навигация).
3. Запрос на выполнение необратимых действий, а именно подтверждение удаления вида «Вы действительно хотите удалить файл (запись)?».
4. Обратную связь с пользователем, например вывод сообщения об успешности удаления/редактирования записи и т. д.

1.3 Требования к программной реализации

1. Все переменные и константы должны иметь осмысленные имена в рамках тематики варианта курсовой работы. Переменным рекомендуется присваивать имена, состоящие из букв нижнего регистра; для формирования составного имени

используется нижнее подчеркивание (например, number_of_students) или «верблюжья нотация» (например, flagExit). Константам рекомендуется присваивать имена, состоящие из букв верхнего регистра (например, SIZE_ARR_OF__ACCOUNTS, FILE_OF_ACCOUNTS).

2. Имена функций должны быть осмысленными, начинаться с буквы нижнего регистра, строиться по принципу глагол + существительное (например, addAccount, findStudentBySurname). Если функция выполняет проверку и возвращает результат типа bool, то ее название должно начинаться с глагола is (например, isNumberNumeric, isLoginUnique).

3. Не допускается использование оператора прерывания goto.

4. Код не должен содержать неименованных числовых констант («магических» чисел), неименованных строковых констант (например, имен файлов и др.). Подобного рода информацию следует представлять как глобальные константы. По правилам качественного стиля программирования тексты всех информационных сообщений, выводимых пользователю в ответ на его действия, также оформляются как константы.

5. Код необходимо комментировать (как минимум в части объявления структур, массивов/векторов, прототипов функций, нетривиальной логики).

6. Код не должен дублироваться – для этого существуют функции!

7. Одна функция решает только одну задачу (например, не допускается в одной функции считывать данные из файла и выводить их на консоль – это две разные функции). При этом внутри функции возможен вызов других функций.

8. Выполнение операций чтения/записи в файл должно быть сведено к минимуму (т. е. после однократной выгрузки данных из файла в массив/вектор дальнейшая работа ведется с этим массивом/вектором, а не происходит многократное считывание данных из файла в каждой функции).

9. Следует избегать глубокой вложенности условных и циклических конструкций: вложенность блоков должна быть не более трех.

10. Следует избегать длинных функций: текст функции должен умещаться на один экран (размер текста не должен превышать 25–50 строк).

11. Следует выносить код логически независимых модулей в отдельные .cpp файлы и подключать их с помощью заголовочных .h файлов.

2 Структура и описание разделов пояснительной записки

Структура пояснительной записки следующая.

Титульный лист *(образец приведен в приложении Б).*
Задание по курсовой работе *(заполненное и подписанное студентом и преподавателем).*
Содержание.
1. Требования к программе *(см. пояснения ниже).*
2. Конструирование программы.
   2.1 Разработка модульной структуры программы *(см. пояснения ниже).*
   2.2 Выбор способа организации данных *(см. пояснения ниже).*
   2.3 Разработка перечня пользовательских функций программы *(см. пояснения ниже).*
3. Разработка алгоритмов работы программы *(см. пояснения ниже).*
   3.1 Алгоритм функции main.
   3.2 Алгоритм функции … *(выбирается из перечня пользовательских функций).*
   3.3 Алгоритм функции … *(выбирается из перечня пользовательских функций).*
4. Описание работы программы *(см. пояснения ниже).*
   4.1 Авторизация.
   4.2 Модуль администратора.
   4.3 Модуль пользователя.
   4.4 Исключительные ситуации.

Приложение (обязательное): листинг кода с комментариями *(приводится ВЕСЬ код с авторским форматированием и комментариями; допускается применение в листинге кода шрифта 12 пунктов и выше).*

Далее приведены пояснения к разделам пояснительной записки.

Требования к программе включают:

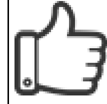
– полный текст варианта задания;

– исходные данные для курсовой работы (см. подраздел 1.1);

– функциональные требования к курсовой работе (рекомендуется взять за основу материал подраздела 1.2 и конкретизировать его для определенного варианта, например, прописать индивидуальное задание, разновидности поиска и сортировки, возможные исключительные ситуации);

– требования к программной реализации (см. подраздел 1.3).

Не следует в разделе требований писать то, что точно не будет реализовано в действительности.

	**Рекомендуется:** после завершения работы над программой вернуться к требованиям и проверить соответствие вашей программы тому, что вы заявляли.

Разработка модульной структуры программы подразумевает графическое представление структуры программы с указанием модулей, подмодулей и их функциональных возможностей (рисунок 2.1). *Для объектно-ориентированного программирования в данном подразделе приводится UML-диаграмма классов.*

АВТОРИЗАЦИЯ
МОДУЛЬ АДМИНИСТРАТОРА
МОДУЛЬ ПОЛЬЗОВАТЕЛЯ
РАБОТА С УЧЕТНЫМИ ЗАПИСЯМИ
ПРОСМОТР
ДОБАВЛЕНИЕ
РЕДАКТИРОВАНИЕ
УДАЛЕНИЕ
РАБОТА С ДАННЫМИ
ПРОСМОТР
ИНД. ЗАДАНИЕ
ПОИСК
СОРТИРОВКА
РАБОТА С ДАННЫМИ
РЕЖИМ РЕДАКТИРОВАНИЯ
ПРОСМОТР
ДОБАВЛЕНИЕ
РЕДАКТИРОВАНИЕ
УДАЛЕНИЕ
РЕЖИМ ОБРАБОТКИ
ИНД. ЗАДАНИЕ
ПОИСК
СОРТИРОВКА

Рисунок 2.1 – Пример модульной структуры программы

	Важно: рисунок 2.1 с примером модульной структуры программы носит обучающий характер и не предназначен для копирования в пояснительную записку! Данный пример нуждается в доработке, а именно: конкретизации индивидуального задания, детализации вариантов сортировки, поиска. Целесообразно также отразить в модульной структуре программы подмодули, отвечающие за чтение и запись информации, с указанием моментов их вызова в процессе использования программы.

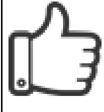
Выбор способов организации данных:

1. В качестве выбора способа описания входных данных приводится описание следующих типов struct (с указанием конкретных полей):
   – для учетных записей пользователей;
   – для данных.

В случае объектно-ориентированного программирования приводятся названия предполагаемых классов и содержащихся в них полей. При работе с базой данных дополнительно приводится структура таблиц.

2. В качестве способа объединения входных данных указывается:
   – использование массивов (статически или динамически создаваемых) или векторов;
   – их выбранная область видимости (локальные или глобальные).

Разработка перечня пользовательских функций программы подразумевает перечисление прототипов функций, необходимых для реализации программы, и краткие комментарии к ним. Код функций не приводится, так как на текущем этапе проектирования он еще не существует.

	**Рекомендуется:** прототипы функций разбивать на тематические группы в соответствии с модульной структурой программы. При программной реализации целесообразно тематические группы функций реализовать в отдельных cpp-файлах и подключить их к основному файлу с помощью одноименых заголовочных h-файлов.

В случае использования объектно-ориентированного программирования приводятся методы для классов.

В сущности, второй раздел пояснительной записки представляет собой программный интерфейс курсовой работы: пользовательские типы данных (структуры), переменные массивов или векторов, прототипы всех предполагаемых функций. В дальнейшем необходимо будет к интерфейсу добавить реализацию.

Разработка алгоритмов работы программы включает блок-схемы алгоритмов (с кратким словесным описанием их работы в тексте пояснительной записки) для функции main и двух любых других пользовательских функций.

В случае использования объектно-ориентированного программирования разрабатываются алгоритмы для двух любых методов классов, а также алгоритм функции main.

	Важно: так как алгоритмы разрабатываются до непосредственного кодирования, то они не могут содержать просто копии строчек кода. Алгоритм функции должен отражать логику решения задачи; может содержать словесные инструкции с упоминанием имен структур, массивов/векторов, других функций.

Критерий качественного алгоритма: если разработанный алгоритм без дополнительных пояснений понятен другому человеку, владеющему основами программирования, и может быть однозначно реализован в коде, значит алгоритм достиг своей цели. Признаком хорошего алгоритма также является его относительная компактность (занимает не более одной страницы формата А4); иная ситуация свидетельствует либо о чрезмерно сложной логике решения проблемы, либо о недостаточной функциональной декомпозиции решаемой задачи (т. е. содержимое функции следовало разбить на несколько функций).

Алгоритм должен быть оформлен согласно ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. Пояснения и примеры разработки алгоритмов приведены в разделе 5 данного пособия.

	Рекомендуется: при разработке алгоритмов использовать профессиональные графические редакторы диаграмм и блок-схем, например Microsoft Visio.

Описание работы программы подразумевает краткое словесное описание работы программы со скриншотами консоли.

	Важно: объем пояснительной записки не регламентируется. Основным критерием выступает качество работы.

3 Порядок защиты и критерии оценки курсовой работы

Защита курсовой работы включает три этапа:

1. Демонстрация функциональных возможностей программы студентом (во избежание проблем совместимости версий и подключения внешних файлов рекомендуется проводить демонстрацию программы на собственных ноутбуках). Отдельный носитель (диск) с программой не требуется.
2. Анализ содержимого и качества оформления пояснительной записки (обязательно: распечатать пояснительную записку и вложить лист-задание).
3. Аудит кода (code review) и вопросы по коду.

Итоговая отметка за курсовую работу формируется в результате суммирования баллов за каждый из перечисленных выше этапов защиты.

Критерии получения минимальной удовлетворительной оценки за курсовую работу изложены в таблице 3.1.

Таблица 3.1 – Критерии получения минимальной удовлетворительной отметки за курсовую работу

Результирующая отметка	**Этап защиты, который подлежит оценке**	**Критерии оценки**
4 балла	1. Демонстрация функциональных возможностей программы	Реализация базового функционала, а именно работа только с данными (без реализации работы с учетными записями пользователей): просмотр, добавление, удаление, редактирование данных, выполнение индивидуального задания, поиск, сортировка
	2. Анализ содержимого и качества оформления пояснительной записки	Оформленная в соответствии с требованиями записка, но без алгоритмов
	3. Аудит кода (code review)	Владение кодом (исчерпывающие ответы на вопросы по собственному коду)

Если все изложенные в таблице 3.1 требования соблюдены, то за каждый этап защиты к минимальной удовлетворительной отметке (4 балла) прибавляются дополнительные баллы. Это происходит при выполнении описанных в таблице 3.2 критериев. Следует отметить, что за каждый из трех этапов защиты максимально можно набрать по два балла; итого максимальная результирующая отметка за курсовую работу составит 10 баллов.

Таблица 3.2 – Критерии начисления дополнительных баллов за каждый из этапов защиты (баллы прибавляются к исходной отметке, равной 4)

Этап защиты, который подлежит оценке	**Результирующая отметка**	**Критерии оценки**
1. Демонстрация функциональных возможностей программы	+ 1 балл (итог: 5 баллов)	Реализация полного функционала (авторизация, работа с учетными записями: просмотр, добавление, удаление, редактирование, работа с данными)
	+ 2 балла (итог: 6 баллов)	Реализация продвинутого функционала, а именно: обратная связь с пользователем (запрос на подтверждение удаления, сообщения об успешности выполнения действий), обработка исключительных ситуаций (проверка форматов вводимых данных, проверка существования номера записи для редактирования/удаления, проверка на уникальность нового логина и т. д.); любые другие подходы по усовершенствованию программы (например, хеширование паролей с «солью»)
2. Анализ содержимого и качества оформления пояснительной записки	+ 1 балл (итог: **7** баллов)	Частичная реализация алгоритмов (т. е. один или два вместо трех) или наличие в алгоритмах ошибок
	+ 2 балла (итог: 8 баллов)	Реализация трех алгоритмов без ошибок
3. Аудит кода (code review)	+ 1 балл (итог: 9 баллов)	Качество кода: хорошее (нет дублирования одного и того же кода: вместо этого – функции; каждая функция решает одну задачу; осмысленные имена переменных, констант, функций)
	+ 2 балла (итог: 10 баллов)	Качество кода: отличное (функция main не перегружена кодом; сведены к минимуму операции чтения и записи в файл; код снабжен комментариями; нет «хардкода»: вместо этого – константы; код разбит на модули в виде отдельных cpp-файлов, которые подключаются посредством заголовочных h-файлов; все действия логичны и понятны)

4 Рекомендации по проектированию программы

4.1 Принципы и уровни проектирования программы в курсовой работе

Для создания качественных (надежных, производительных, легко модифицируемых) программ необходимо следовать ключевым принципам проектирования:

1. *Минимизация сложности.* Управление сложностью – самый важный технический аспект разработки программного обеспечения. На уровне архитектуры сложность проблемы можно снизить, разделив систему на подсистемы. В разделении сложной проблемы на простые фрагменты и заключается цель всех методик проектирования программного обеспечения. Чем более независимы подсистемы, тем безопаснее сосредоточиться на одном аспекте сложности в конкретный момент времени.

Технологически процесс разделения системы на подсистемы целесообразно выполнять путем декомпозиции исходной задачи на подзадачи. Данный подход получил название нисходящего проектирования и представляет собой процесс дробления задачи на подзадачи, установления логических связей между ними. После этого переходят к уточнению выделенных подзадач. Этот процесс детализации продолжается до уровня, позволяющего достаточно легко реализовать подзадачу на выбранном языке программирования, т. е. до уровня очевидно реализуемых модулей.

2. *Простота сопровождения.* Проектируя приложение, не забывайте о программистах, которые будут его сопровождать. Постоянно представляйте себе вопросы, которые могут возникать у них при взгляде на создаваемый вами код. Проектируйте систему так, чтобы ее работа была очевидной.

3. *Слабое сопряжение.* Слабое сопряжение предполагает сведение к минимуму числа соединений между разными частями программы. Это позволит максимально облегчить интеграцию, тестирование и сопровождение программы.

4. *Расширяемость.* Расширяемостью системы называют свойство, позволяющее улучшать систему, не нарушая ее основной структуры. Изменение одного фрагмента системы не должно влиять на ее другие фрагменты. Внесение наиболее вероятных изменений должно требовать наименьших усилий.

5. *Возможность повторного использования.* Проектируйте систему так, чтобы ее фрагменты можно было повторно использовать в других системах.

6. *Минимальная, но полная функциональность.* Этот аспект подразумевает отсутствие в системе лишних частей: дополнительный код необходимо разработать, проанализировать, протестировать, а также пересматривать при изменении других фрагментов программы.

7. *Соответствие стандартам.* Попытайтесь придать всей системе привычный для разработчиков облик, применяя стандартные подходы и следуя соглашениям о коде (Code Convention).

Применительно к данной курсовой работе, предполагающей реализацию задач в процедурном стиле (без использования объектно-ориентированного программирования), процесс проектирования программы целесообразно представить в виде последовательности уровней (рисунок 4.1):

1. Уровень програмной системы: анализ исходной задачи, подлежащей программной реализации.

2. Выбор способов описания и хранения данных: выбор способов хранения информации (файлы), продумывание пользовательских типов данных (структур), выбор способов объединения объектов структур в наборы данных (массивы или контейнеры, например векторы), определение области видимости для выбранных наборов данных (глобальные, локальные).

3. Разделение системы на физически независимые модули (отдельные cpp-файлы) и их подключение с помощью одноименных заголовочных h-файлов.

4. Выделение в рамках каждого модуля прототипов функций (записываются в заголовочные h-файлы).

5. Реализация (описание) функций в соответствующих cpp-файлах.

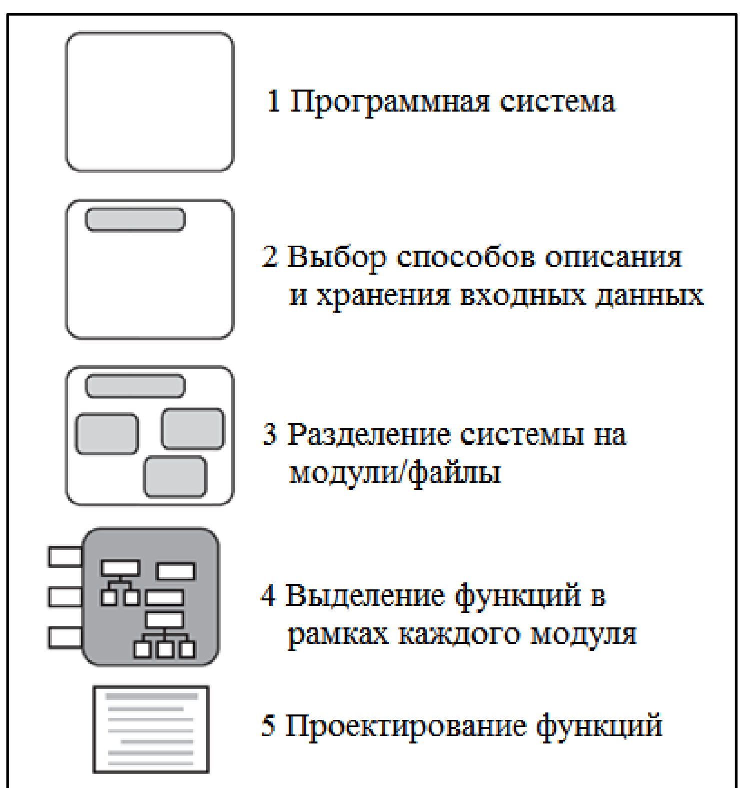


Рисунок 4.1 – Уровни проектирования программы в курсовой работе

4.2 Способы организации работы по чтению/записи в файл

Выполнение операций чтения/записи в файл должно быть сведено к минимуму (т. е. после однократной выгрузки данных из файла в массив/вектор дальнейшая работа ведется с этим массивом/вектором, а не происходит многократное считывание данных из файла в каждой функции).

Реализовать данный принцип можно двумя способами (рисунки 4.2, 4.3).

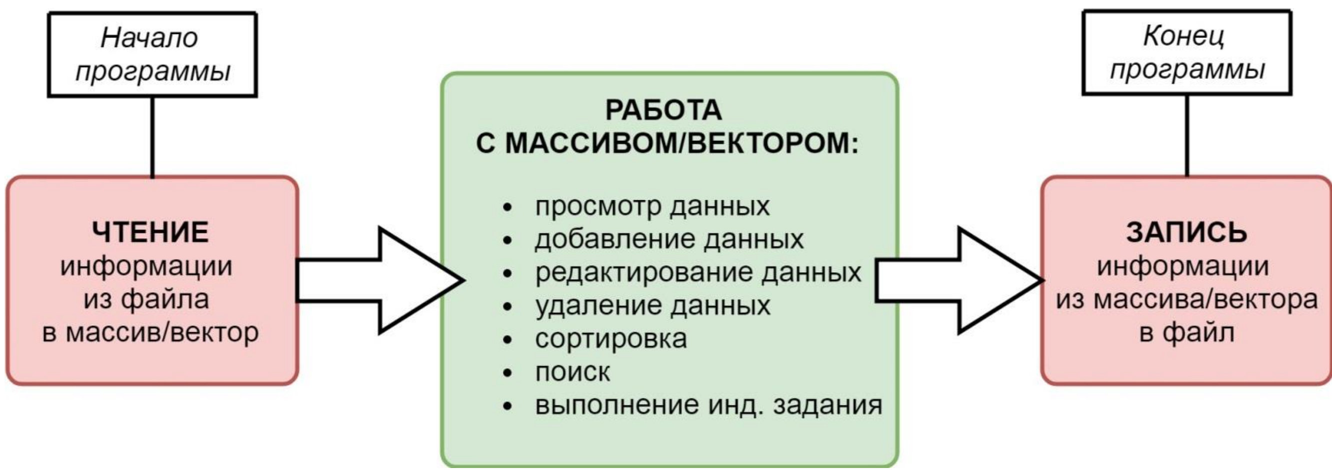


Рисунок 4.2 – Первый способ организации работы с файлами

Рисунок 4.3 – Второй способ организации работы с файлами

При первом способе (см. рисунок 4.2) в начале программы осуществляется чтение (импорт) информации из файла в массив или вектор, далее все операции с

данными производятся посредством массива/вектора, в конце программы (непосредственно перед ее закрытием) происходит запись (экспорт) информации из массива или вектора в файл.

При втором способе (см. рисунок 4.3) в начале программы осуществляется чтение (импорт) информации из файла в массив или вектор, далее все операции с данными производятся посредством массива/вектора, однако при каждом изменении информации дополнительно происходит запись (экспорт) информации из массива или вектора в файл.

	Важно: первый способ сводит операции перезаписи файла к минимуму (данная операция производитя только один раз – в конце программы). Однако в случае некорректного прерывания сессии все промежуточные изменения будут утеряны. В этом отношении **второй способ более надежный**, так как осуществляет динамическую выгрузку информации в файл, а потому **и более предпочтительный**. Следует, однако, отметить, что второй способ при этом уступает первому по временным затратам.

4.3 Принципы организации работы с массивами и векторами

В качестве способа объединения входных данных рекомендуется использовать массивы (динамически создаваемые) или векторы.

Массивы представляют собой набор однотипных пронумерованных данных.

Общие особенности работы с массивами:

1. Индексация с 0; индексы массива – только целые положительные числа.
2. Имя массива является указателем на первый байт нулевого элемента массива и хранит соответствующий адрес.
3. Не производится проверка выхода за границы массива (любая попытка проверки делает язык программирования более медленным), в связи с чем требуется анализ кода на предотвращение ситуаций, потенциально приводящих к выходу за пределы массива.
4. Массив в функцию всегда передается по указателю. Вместе с массивом в функцию необходимо передавать его размер, так как внутри функции узнать размер массива программным способом будет невозможно:

```
// Вывод содержимого массива на экран
void showStudentArray(Student *arr_of_students, int number_of_students);
// Удаление студента из массива
void delStudentFromArray(Student *arr_of_students, int &number_of_students);
```

Рекомендации по работе со статически создаваемыми массивами:

1. Размер памяти, выделяемой для статического массива, необходимо указывать как глобальную константу (например, RESERVE_SIZE):

```
 struct Student
 {
      string name;
      string surname;
      int age;
 };
const int RESERVE_SIZE = 100;
void main()
{
      Student arr_of_students[RESERVE_SIZE];
      int number_of_students=0;
      // работа программы
}
```

В примере кода, приведенном выше, RESERVE_SIZE имет смысл физического размера массива, а переменная number_of_students – логического размера массива.

2. Способ увеличения размера статически созданного массива (для добавления нового элемента в массив): изначально резервируем память физического размера RESERVE_SIZE, а используем логический размер n<= RESERVE_SIZE. При необходимости n увеличиваем, но при этом контролируем, чтобы n<= RESERVE_SIZE.

	Важно: при недостатке памяти для статически созданного массива потребуется внести правки в программный код (в части увеличения константы RESERVE_SIZE, отвечающей за размер массива) и перекомпилировать программу. В этом смысле программа в рамках курсовой работы, основанная на статических массивах, является ненадежной в практическом использовании; а применение динамически создаваемых массивов или векторов является более целесообразным.

3. При выполнении операции чтения информации из файла в статически созданный массив контролируем выход за пределы массива (далее в качестве логического размера массива фигурирует переменная number_of_students):

```
void readFileStudents(Student *arr_of_students, int &number_of_students)
{
	ifstream fin(FILE_OF_DATA, ios::in); // Открыли файл для чтения
	if (!fin.is_open()) cout << "Указанный файл не существует!" << endl;
	else
	{
		int i = 0;
		while (!fin.eof())
		{
			if (i < RESERVE_SIZE)
			{
				fin >> arr_of_students[i].name
					>> arr_of_students[i].surname
					>> arr_of_students[i].age;
				i++;
			} else
			{
				cout << "Недостаточно памяти для чтения всех данных!"
<< endl;
				break;
			}
		}
		number_of_students = i;
	}
	fin.close();  //Закрыли файл
}
```

4. При добавлении новых элементов в статически созданный массив контролируем выход за пределы массива (в качестве логического размера массива фигурирует переменная number_of_students):

```
void addStudentInArray(Student *arr_of_students, int &number_of_students)
{
	//добавление студента, если не происходит выход за пределы массива
	if (number_of_students + 1 <= RESERVE_SIZE)
	{
		number_of_students++;
		cout << "Введите имя студента: ";
		cin >> arr_of_students[number_of_students - 1].name;
		cout << "Введите фамилию студента: ";
```

```
            cin >> arr_of_students[number_of_students - 1].surname;
            cout << "Введите возраст студента: ";
            cin >> arr_of_students[number_of_students - 1].age;
            writeEndFileStudents(arr_of_students[number_of_students - 1]);
        }
        else cout << "Недостаточно памяти для добавления нового элемента!" << endl;
}
```

5. Для удаления элемента из статически созданного массива необходимо сдвинуть все элементы на одну позицию, начиная с номера удаляемого элемента, а затем уменьшить логический размер массива:

```
void delStudentFromArray(Student *arr_of_students, int &number_of_students)
{
        int number_of_deleted_item;
        cout << "Введите номер удаляемой записи: ";
        cin >> number_of_deleted_item;
        number_of_deleted_item--; // пользователь мыслит с 1, но индексы
        нумеруются с 0
        if (number_of_deleted_item >= 0 &&
            number_of_deleted_item < number_of_students)
        {
            for (int i = number_of_deleted_item; i < number_of_students - 1; i++)
            {
                arr_of_students[i] = arr_of_students[i + 1];
            }
            number_of_students--;
            writeFileStudents(arr_of_students, number_of_students);
        }
        else cout << "Введен некорректный номер удаляемой записи!" << endl;
}
```

Рекомендации по работе с динамически создаваемыми массивами:

1. Для динамического создания массива необходимо запрашивать память в процессе выполнения программы. В контексте курсовой работы объем запрашиваемой памяти соответствует количеству структур в файле. Рекомендуется запрограммировать отдельную функцию (например, getSizeOfFileWithStudents()) для вычисления количества структур в файле перед созданием динамического массива. Примеры кода, реализующие данную задачу, приведены в подразделе 6.2.

```
int number_of_students = getCountOfStucturesInFile(FILE_OF_DATA);
int *arr = new int[number_of_students]; // динамически создаваемый массив
```

2. Для динамически создаваемых массивов необходимо освобождать выделенную память в конце программы (в противном случае – утечки памяти):

```
delete []arr; // освобождение памяти, выделенной под динамический массив
```

3. Способ увеличения размера динамически созданного массива (для добавления нового элемента в массив): создаем новый массив, переносим туда содержимое старого массива, старый массив удаляем. Пример перевыделения памяти с целью увеличения размера динамически созданного массива приведен в подразделе 6.3.

4. Для удаления элемента из динамически созданного массива необходимо сдвинуть все элементы на одну позицию, начиная с номера удаляемого элемента, а затем уменьшить логический размер массива (см. пример кода, реализующего данную задачу, в пояснениях к статически создаваемым массивам).

В качестве альтернативы массивам выступают векторы из стандартной библиотеки шаблонов. Векторы безопаснее и удобнее, чем массив, но в общем случае медленнее с точки зрения производительности.

Рекомендации по работе с векторами:

1. Для полноценной работы с векторами необходимо подключить следующие библиотеки:

```
#include <vector>
#include <iterator>
#include <algorithm>
```

2. Для минимизации операций перевыделения памяти для вектора при считывании информации из файла рекомендуется после объявления вектора зарезервировать для него память. При этом целесообразно отталкиваться от количества структур в читаемом файле. Наиболее удобно запрограммировать отдельную функцию (например, getCountOfStucturesInFile(string file_path)) для вычисления количества структур в файле, примеры кода, реализующие данную задачу, приведены в подразделе 6.2. Тогда:

```
vector <Student> vector_of_students;
/* резервируем память на getSizeOfFileWithStudents() элементов, но ничем не заполняем: */
vector_of_students.reserve(getCountOfStucturesInFile(FILE_OF_DATA));
```

3. Передача вектора в функцию осуществляется либо по значению, либо по ссылке (например, если в функции происходят изменения вектора). Размер вектора при этом в функцию передавать не нужно (он доступен внутри функции посредством библиотечного метода size()). Примеры:

```
void showStudents(vector <Student> vec_of_students); // вывод массива на экран
void delStudent(vector <Student> &vec_of_students); // удаление элемента массива
```

4. Добавление элемента в конец вектора осуществляется посредством библиотечного метода push_back:

```
void readFileStudents(vector <Student> &vec_of_students)
{
	ifstream fin(FILE_OF_STUDENTS, ios::in);
	if (!fin.is_open()) cout << "Файл не существует!";
	else
	{
		Student student_temp;
		while (!fin.eof())
		{
			fin >> student_temp.surname
			    >> student_temp.name
			    >> student_temp.age;
			vec_of_students.push_back(student_temp);
		}
	}
	fin.close();
}
```

5. Удаление элемента из вектора, расположенного по индексу index_for_delete, осуществляется посредством библиотечного метода erase (при этом необходимо с помощью итератора begin() выполнить позиционирование в начало вектора, а далее осуществить сдвиг на требуемое число позиций):

```
vec_of_students.erase(vec_of_students.begin() + index_for_delete);
```

6. Для доступа к элементу вектора используется метод at:

```
void writeFileStudents(vector <Student> vec_of_students)
```

```
{
	ofstream fout(FILE_OF_STUDENTS, ios::out);
	for (int i = 0; i < vec_of_students.size(); i++)
	{
			fout << vec_of_students.at(i).surname << " "
			    << vec_of_students.at(i).name << " "
			    << vec_of_students.at(i).age;
			if (i < vec_of_students.size() - 1)
				fout << endl;
	}
	fout.close();
}
```

7. Сортировку вектора можно выполнить с помощью метода sort из библиотеки algorithm, при этом требуется создать дополнительную функцию-компаратор, определяющую, по какому полю и в каком порядке (по возрастанию/по убыванию) будет выполнена сортировка:

```
void sortStudentsBySurname(vector <Student> &vec_of_students) // сортировка
{
	sort(vec_of_students.begin(), vec_of_students.end(), mySortBySurname);
}

bool mySortBySurname(Student student_a, Student student_b) // функция-компаратор
{
	return student_a.surname < student_b.surname; // по алфавиту: от а до я
}
```

4.4 Минимизация области видимости переменных

Областью видимости называют фрагмент программы, в котором переменная известна и может быть использована. В С++ переменная может иметь область видимости, соответствующую фрагменту кода, заключенному в фигурные скобки, в котором переменная объявлена и используется (локальная переменная) или всей программе (глобальная переменная; объявляется вне блоков описания функций).

Важно: минимизировать область видимости переменной, сделав ее как можно более локальной.

Несмотря на то, что глобальные переменные облегчают доступ к ним, так как не нужно беспокоиться о списках параметров и правилах области видимости, удобство доступа к глобальным переменным не может компенсировать связанную с этим опасность. Так программу, в которой каждый метод может вызвать любую переменную в любой момент времени, сложнее понять, чем код, основанный на грамотно организованных методах. Сделав данные глобальными, вы не сможете ограничиться пониманием работы одного метода: вы должны будете понимать работу всех других методов, которые вместе с ним используют те же глобальные данные. Подобные программы сложно читать, сложно отлаживать и сложно изменять. В то время как локальная область видимости способствует интеллектуальной управляемости: чем больше информации скрыто, тем меньше нужно удерживать в уме в каждый конкретный момент времени и тем ниже вероятность того, что разработчик допустит ошибку, забыв одну из многих деталей, о которых нужно было помнить.

	Важно: выбирать локальную область видимости для массивов/векторов. Глобальные переменные имеют два главных недостатка: функции, обращающиеся к глобальным данным, не знают о том, что другие функции также обращаются к этим данным; или же функции знают об этом, но не могут контролировать, что именно другие функции делают с глобальными данными.

4.5 Разделение программы на независимые cpp-файлы и их подключение с помощью заголовочных h-файлов

Следует выносить код логически независимых модулей в отдельные cpp- файлы и подключать их с помощью заголовочных h-файлов (рисунок 4.4).

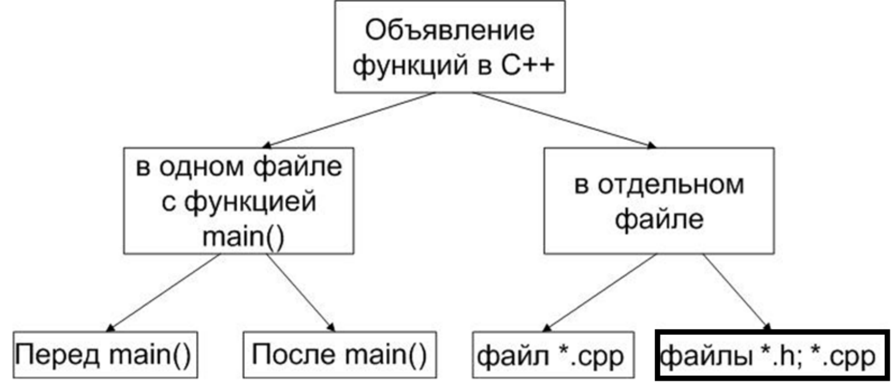


Рисунок 4.4 – Способы объявления функций в С++

Разделение исходного кода программы на несколько файлов становится необходимым по ряду причин:

1. Обеспечение удобства работы с небольшими по объему фрагментами кода, локализованными в отдельных файлах.

2. Разделение программы на логически завершенные модули, которые решают конкретные подзадачи.

3. Разделение программы на физически независимые модули с целью повторного использования этих модулей в других программах.

4. Разделение интерфейса и реализации.

Последний принцип (разделение интерфейса и реализации) требует особого пояснения. Отдельный модуль состоит из двух файлов: заголовочного h-файла (интерфейс) и cpp-файла реализации. Для удобства h-файл и cpp-файл называют одинаково; имя должно соответствовать смысловой нагрузке данного модуля.

Заголовочный файл подключается в одноименный с ним cpp-файл с реализацией программной логики модуля, а также в файл, который будет осуществлять вызов функций данного модуля (например, в файл с запускающей функцией main()). В этом смысле заголовочный файл представляет собой связующее звено между файлом, который задействует логику модуля, и файлом, собственно реализующим эту логику. Подклчение заголовочного файла производится посредством директивы препроцессора include, например:

#include "validation.h"

Сам по себе заголовочный файл не является единицей компиляции: на этапе обработки исходного кода препроцессором директива #include "validation.h" заменяется текстом из файла validation.h.

Заголовочный файл может содержать:

- директивы препроцессора;
- глобальные константы;
- описание типов пользователя;
- прототипы функций.

Заголовочный файл не должен содержать описание функций!

Заголовочный файл должен иметь механизм защиты от повторного включения. Защита от повторного включения реализуется директивой препроцессора:

#pragma once

Файл реализации содержит описание функций. Следует отметить, что при несовпадении сигнатуры функции в прототипе (h-файл) и в определении (cpp-файл) компилятор выдаст ошибку. Также файл реализации может содержать объявления. Объявления, сделанные в файле реализации, будут лексически локальны для этого

файла, т. е. будут действовать только для этой единицы компиляции. При этом в файле реализации не должно быть объявлений, дублирующих объявления в соответствующем заголовочном файле.

В файле реализации должна быть директива включения соответствующего заголовочного файла.

Далее на рисунках 4.5–4.10 приведены примеры выделения в программе независимого модуля, выполняющего проверку вводимых данных на корректность, и его подключения к файлу main.cpp. Данный модуль состоит из заголовочного файла validation.h и файла реализации validation.cpp.

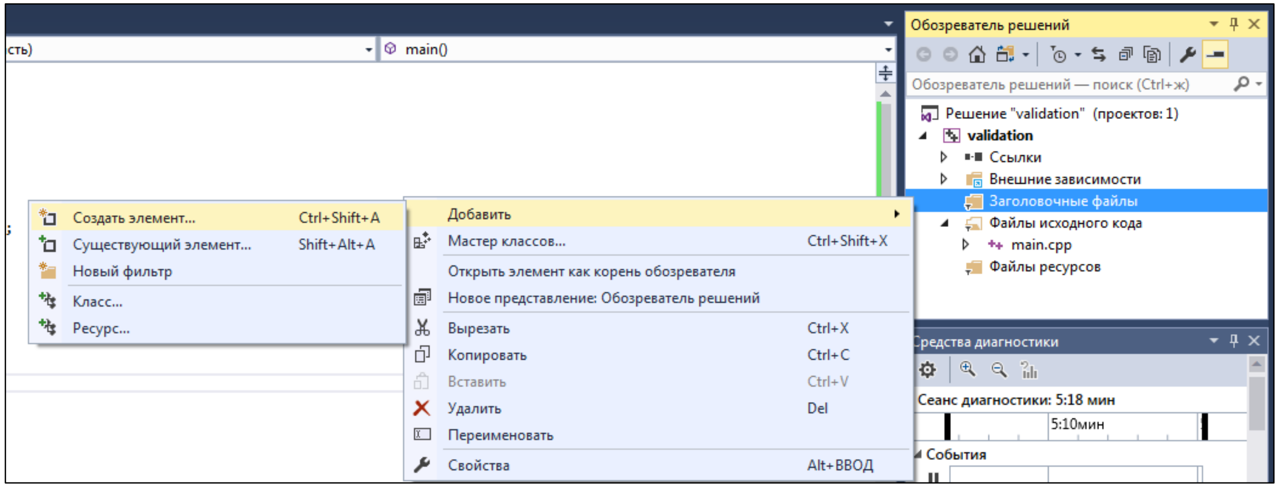


Рисунок 4.5 – Добавление в проект заголовочного файла

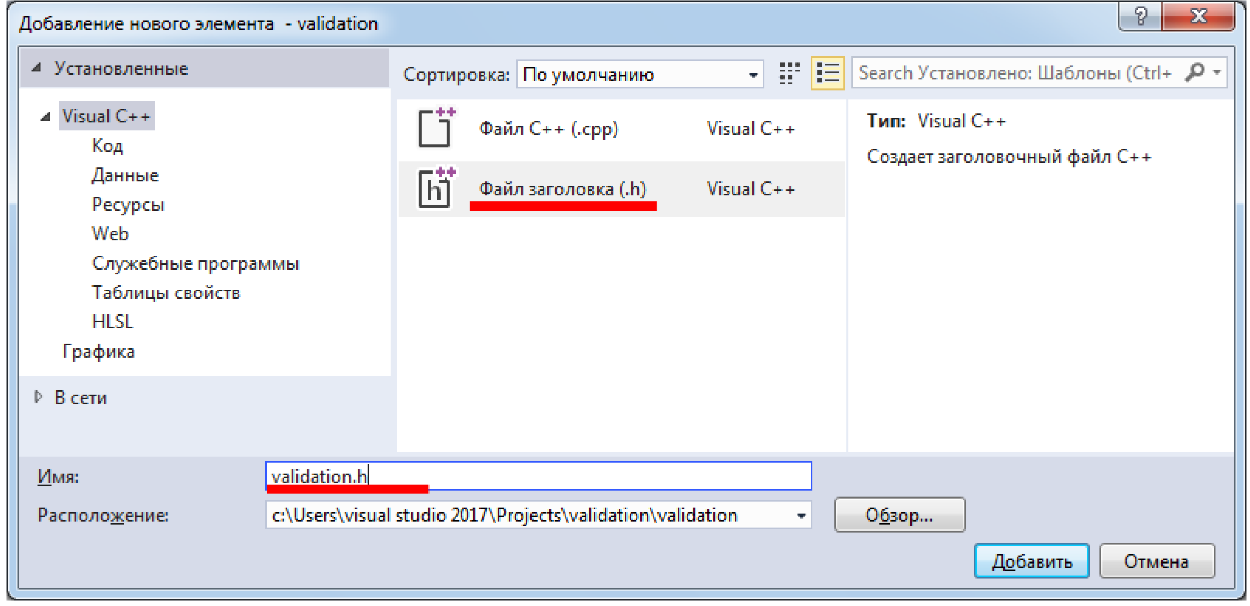


Рисунок 4.6 – Создание заголовочного файла validation.h

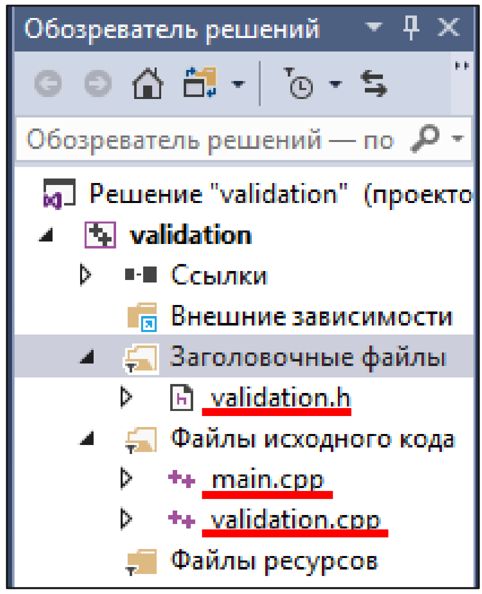


Рисунок 4.7 – Структура проекта с запускающим файлом main.cpp, а также модулем validation, состоящим из заголовочного файла validation.h и файла реализации validation.cpp

main.cpp validation.h validation.cpp

validation

```
1  #include "validation.h"
2
3  #include <iostream>
4
5  using namespace std;
6
7  void main()
8  {
9      setlocale(0, "rus");
10
11     cout << "Введите Ваш возраст: ";
12     int age = inputNumber(LEFT_RANGE_OF_AGE, RIGHT_RANGE_OF_AGE);
13     cout << age;
14
15     system("pause");
16
17 }
```

Рисунок 4.8 – Пример программной реализации файла main.cpp

```cpp
#pragma once

#include <iostream>
using namespace std;

const int LEFT_RANGE_OF_AGE = 1;
const int RIGHT_RANGE_OF_AGE = 99;

int inputNumber(int left_range, int right_range);
bool isNumberNumeric();
bool isNumberRangeCorrect(int number, int left_range, int right_range);
```

Рисунок 4.9 – Пример программной реализации файла validation.h

```cpp
#include "validation.h"

int inputNumber(int left_range, int right_range)
{
    int number;
    while (true)
    {
        cin >> number;
        if (isNumberNumeric() && isNumberRangeCorrect(number, left_range, right_range))
        {
            return number;
        }
        else
        {
            cout << "Некорректный ввод!" << endl;
            cout << "Введите корректные данные: " << endl;
        }
    }
}

bool isNumberNumeric()
{
    if (cin.get() == '\n') return true;
    else
    {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        return false;
    }
}

bool isNumberRangeCorrect(int number, int left_range, int right_range)
{
    if ((number >= left_range) && (number <= right_range)) return true;
    else return false;
}
```

Рисунок 4.10 – Пример программной реализации файла validation.cpp

5 Рекомендации по разработке алгоритмов работы программы

Алгоритм – набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата. В самом упрощенном виде разработку простейшей программы можно представить в виде схемы: анализ задачи → разработка (обдумывание) алгоритма ее решения → программирование (реализация алгоритма с использованием конкретного алгоритмического языка программирования). Разработать алгоритм решения задачи означает разбить задачу на последовательно выполняемые шаги. При этом должны быть четко указаны как содержание каждого шага, так и порядок выполнения шагов.

В блок-схеме алгоритма отдельные шаги изображаются в виде блоков различной формы, соединенных между собой линиями, указывающими направление последовательности. Правила оформления алгоритмов регламентируются ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.

В таблице 5.1 приведены основные символы блок-схем алгоритмов и комментарии по их применению.

	Важно: при начертании элементов необходимо придерживаться строгих размеров, определяемых двумя значениями a и b. Значение a выбирается из ряда 15, 20, 25, ... мм, b рассчитывается из соотношения 2a = 3b.

Минимальное количество текста, необходимого для понимания функции символа, следует помещать внутри данного символа. Если объем текста внутри символа превышает его размеры, следует использовать символ комментария.

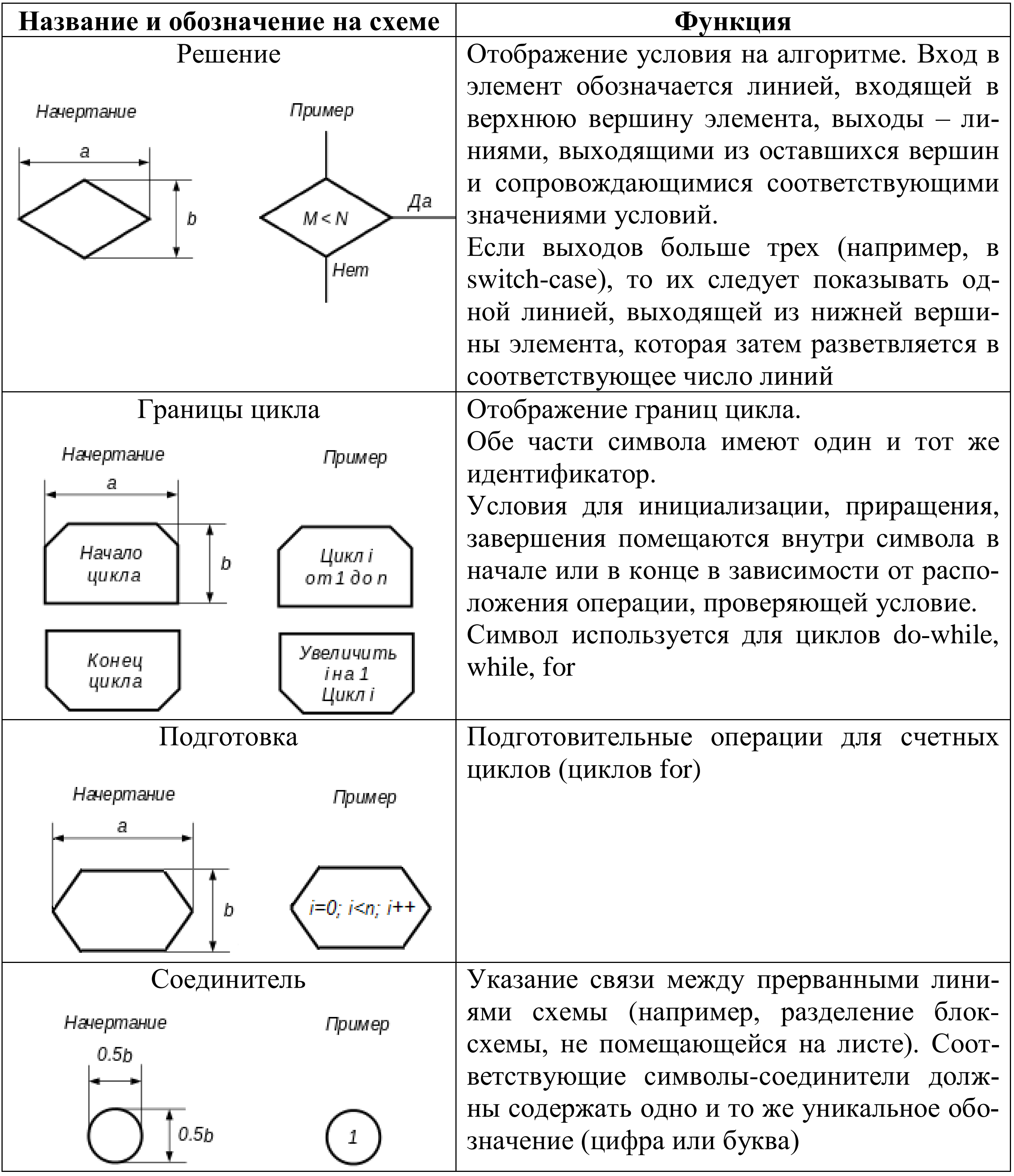
В таблице 5.2 приведены обозначения соединений между символами блок-схем алгоритмов. Соединения между символами показываются линиями и носят название потоков. Направление потока слева направо и сверху вниз считается стандартным, стрелки в таких потоках не указываются. В случаях если поток имеет направление, отличное от стандартного (справа налево или снизу вверх), применение стрелок обязательно. Стрелки выполняются с развалов в 60°.

	Важно: чтобы линии в схемах **подходили** к центру символа **сверху**, а **исходили снизу**. Исключениями из этого правила являются символы соединитель, терминатор (с альтернативным сценарием программы), решение и подготовка, допускающие иные направления входных и/или выходных потоков.
	Важно: пересечения линий следует избегать, так как это существенно затрудняет восприятие алгоритма.

Таблица 5.1 – Символы блок-схем алгоритмов

Название и обозначение на схеме	**Функция**
Терминатор *Начертание* *Пример* a, 0.5b Начало	Выход во внешнюю среду и вход из внешней среды (начало или конец программы). На практике имеют смысл следующие описания терминаторов: начало/конец, запуск/остановка, ошибка (подразумевает завершение алгоритма с ошибкой), исключение (подразумевает генерацию программного исключения), наимнование действия (например, имя функции/метода), возврат (подразумевает возврат из функции/метода в основную программу каких-либо данных)
Данные *Начертание* *Пример* a, b Ввести n	Ввод или вывод данных
Процесс *Начертание* *Пример* a, b sum=0	Обработка данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации). Например, инициализация переменной, арифметические действия над данными и др.
Предопределенный процесс *Начертание* *Пример* a, b Сортировать массив	Отображение процесса, состоящего из шагов программы, которые определены в другом месте. Используется для вызова функции/метода. Внутри блока записывается имя вызываемой функции или метода

Продолжение таблицы 5.1



Название и обозначение на схеме	Функция
Решение Начертание: a, b Пример: M < N, Да, Нет	Отображение условия на алгоритме. Вход в элемент обозначается линией, входящей в верхнюю вершину элемента, выходы – линиями, выходящими из оставшихся вершин и сопровождающимися соответствующими значениями условий. Если выходов больше трех (например, в switch-case), то их следует показывать одной линией, выходящей из нижней вершины элемента, которая затем разветвляется в соответствующее число линий
Границы цикла Начертание: a, b, Начало цикла, Конец цикла Пример: Цикл i от 1 до n, Увеличить i на 1 Цикл i	Отображение границ цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие. Символ используется для циклов do-while, while, for
Подготовка Начертание: a, b Пример: i=0; i<n; i++	Подготовительные операции для счетных циклов (циклов for)
Соединитель Начертание: 0.5b, 0.5b Пример: 1	Указание связи между прерванными линиями схемы (например, разделение блок-схемы, не помещающейся на листе). Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение (цифра или буква)

Продолжение таблицы 5.1

Название и обозначение на схеме	Функция
Комментарий Начертание / Пример Входные параметры: массив array[20]; строка str.	Добавление пояснительных записей. Пунктирные линии в комментарии связаны с соответствующим символом или группой символов (при этом группа выделяется замкнутой пунктирной линией). Текст комментария помещается около ограничивающей его по всей высоте скобки. Комментарий также используется, когда объем текста, помещаемого внутри некоего символа, превышает его размер. Комментарии используют с терминаторами для описания входных аргументов функции/метода

Таблица 5.2 – Соединения между символами блок-схемы алгоритма

Наименование	Обозначение на схеме	Функция
Линии потока		Указание направления линии потока: – без стрелки, если линия направлена слева направо или сверху вниз; – со стрелкой в остальных случаях
Излом линии под углом 90°		Изменение направления потока
Пересечение линий потока		Пересечение двух несвязанных потоков **следует избегать!**
Слияние двух линий потока		Слияние двух линий потока, каждая из которых направлена к одному и тому же символу на схеме

На рисунках 5.1–5.3 приведены примеры, поясняющие использование различных графических символов на блок-схемах алгоритмов.

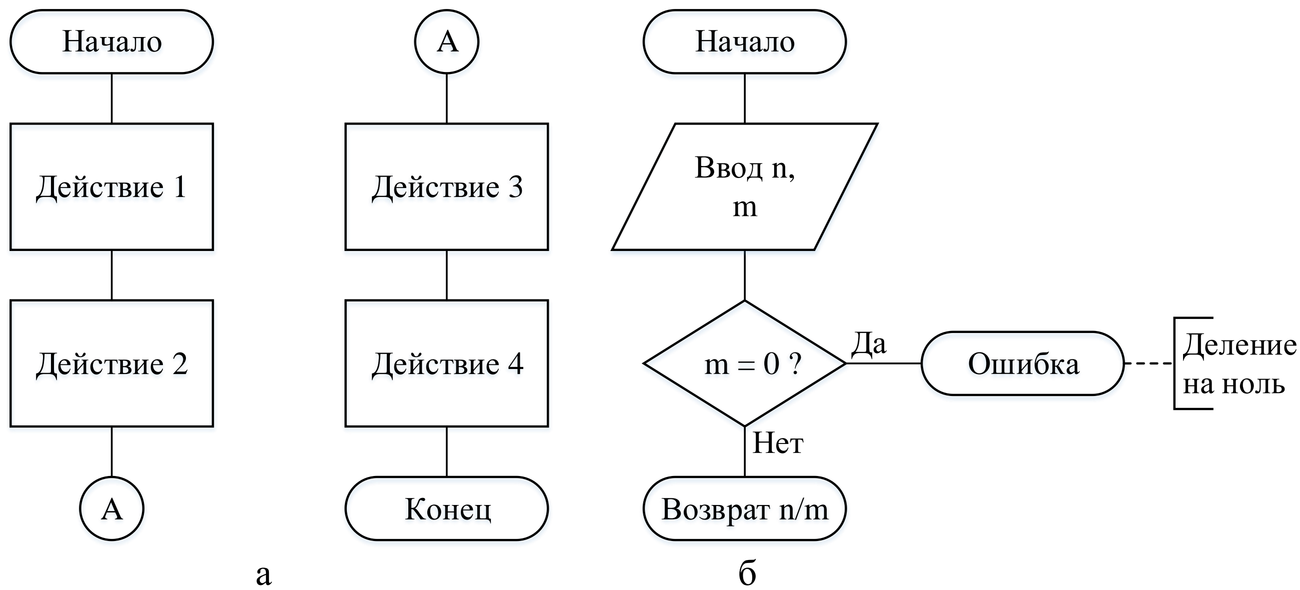


Рисунок 5.1 – Пример использования терминатора и соединителя (а), ввода данных, условия и комментария (б) на блок-схеме алгоритма

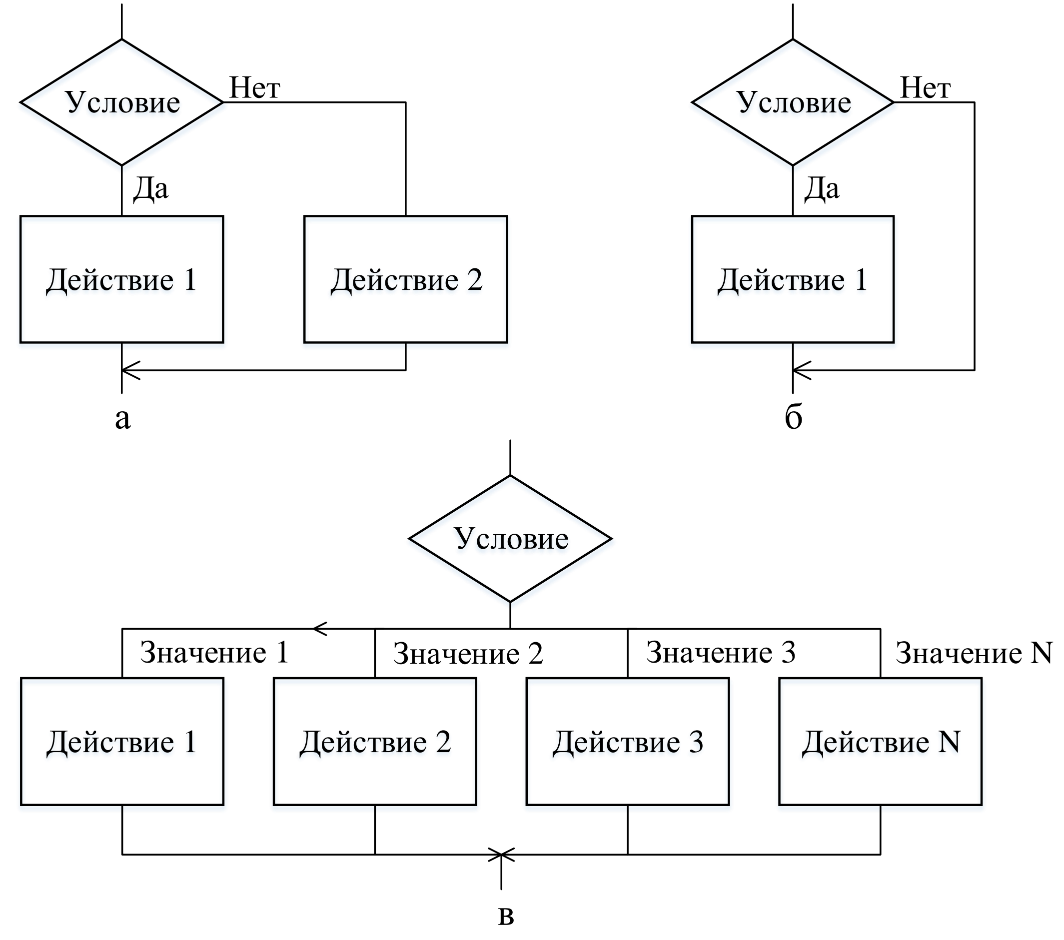


Рисунок 5.2 – Отображение условной конструкции if-else (а), if (б), оператора выбора switch-case (в) на блок-схеме алгоритма

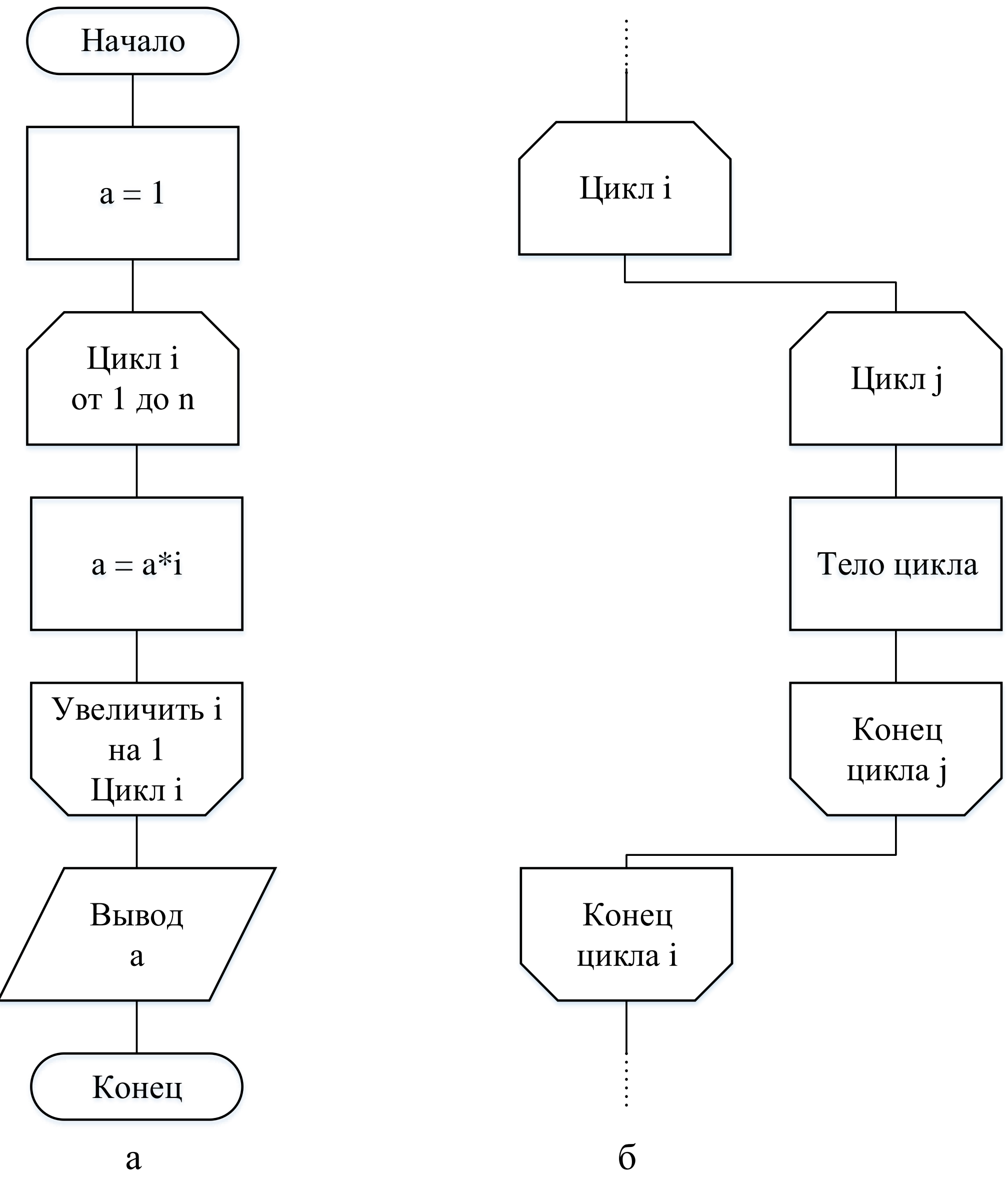


Рисунок 5.3 – Примеры отображения одного цикла (а) и вложенных циклов (б) на блок-схеме алгоритма

Далее на рисунках 5.4–5.9 приведены примеры алгоритмов ряда функций для курсовой работы.

	Важно: данные примеры носят обучающий характер, не предназначены для копирования в курсовую работу, не являются единственно верным вариантом реализации логики курсовой работы. В частности, данные алгоритмы могут выглядеть совершенно иначе вследствие авторского подхода к функциональной декомпозиции задач курсовой работы, а также могут быть расширены за счет дополнительных функциональных возможностей, обработки исключительных ситуаций.

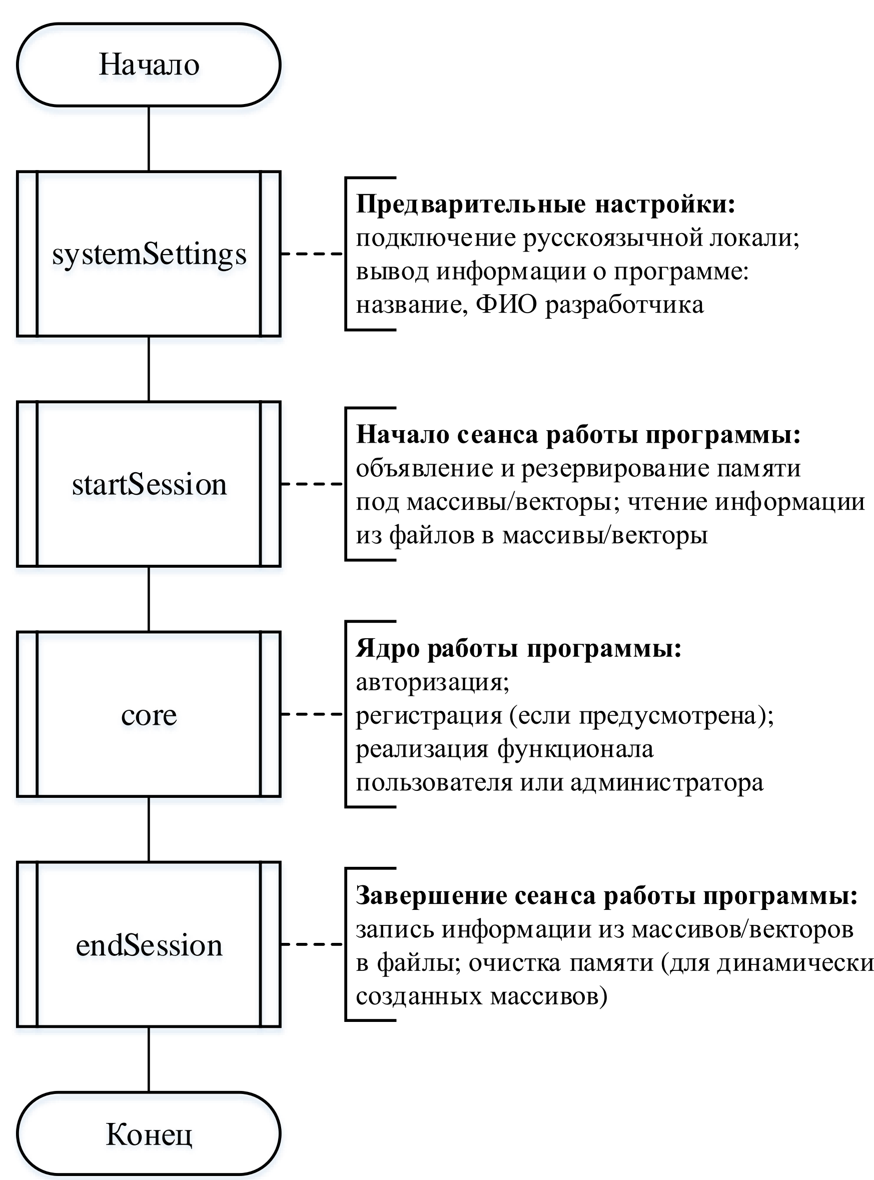


Рисунок 5.4 – Пример алгоритма главной функции main

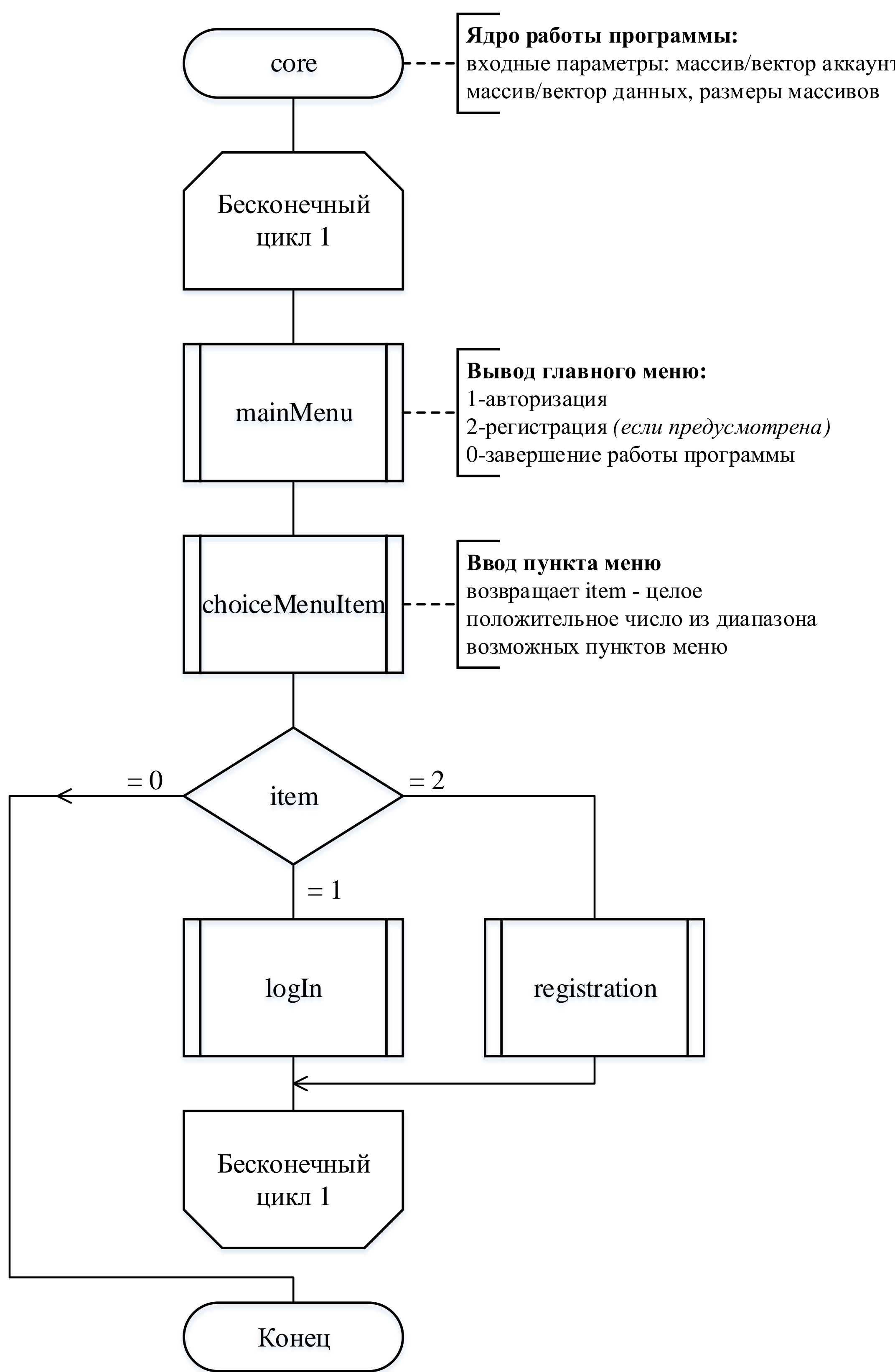


Рисунок 5.5 – Пример алгоритма функции core: основная логика программы

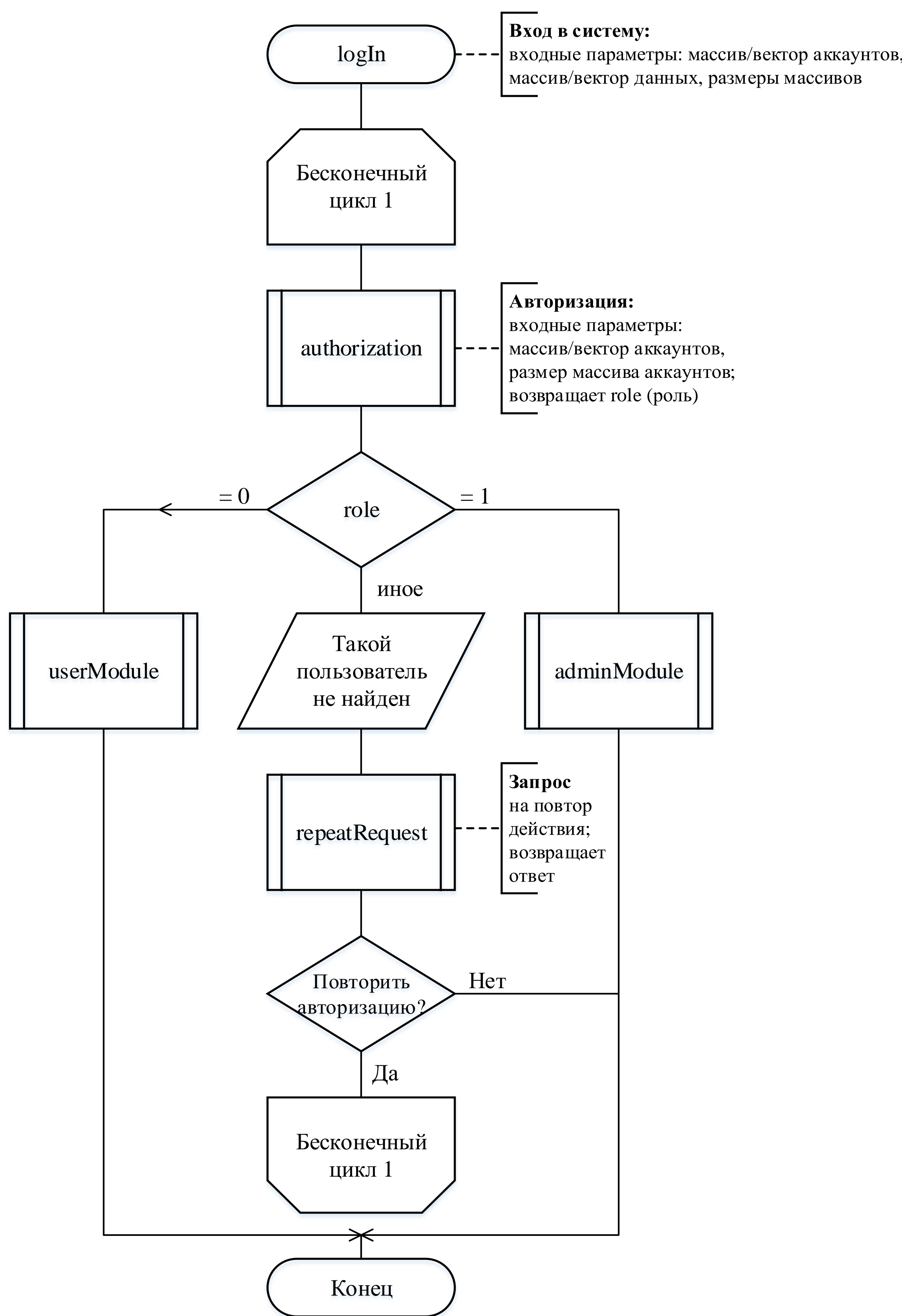


Рисунок 5.6 – Пример алгоритма функции logIn: вход в систему

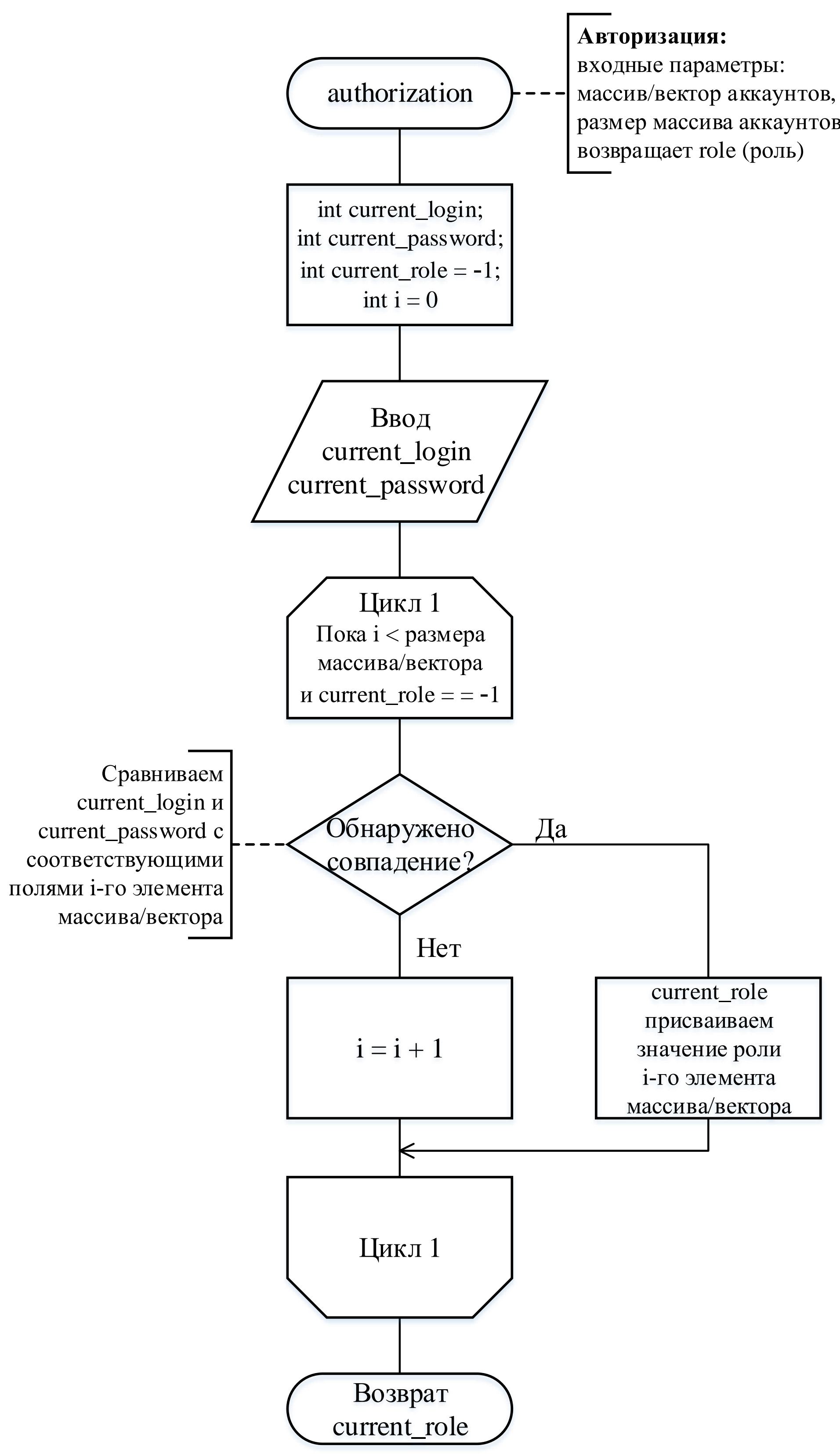


Рисунок 5.7 – Пример алгоритма функции authorization: авторизация

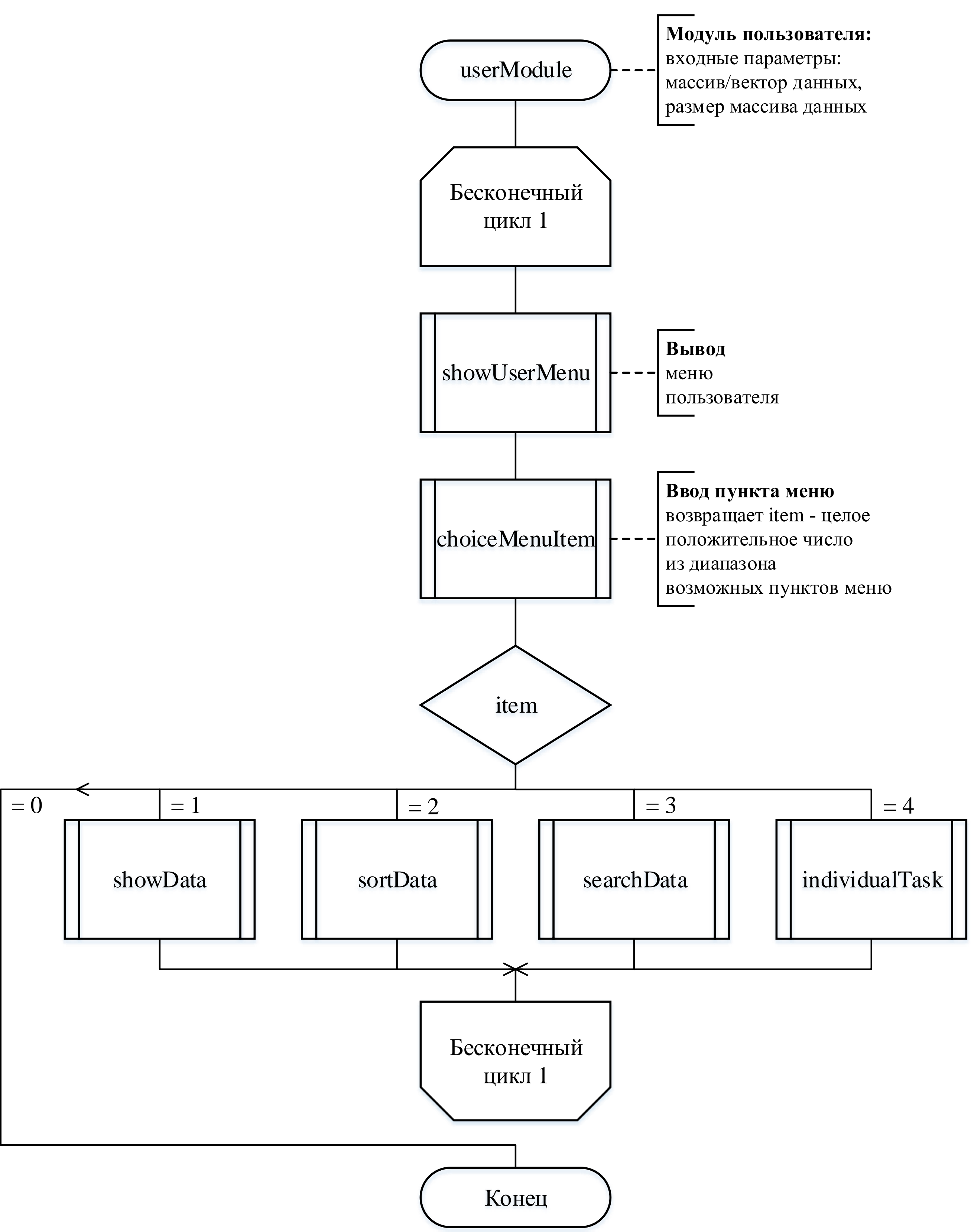


Рисунок 5.8 – Пример алгоритма функции userModule: функционал пользователя

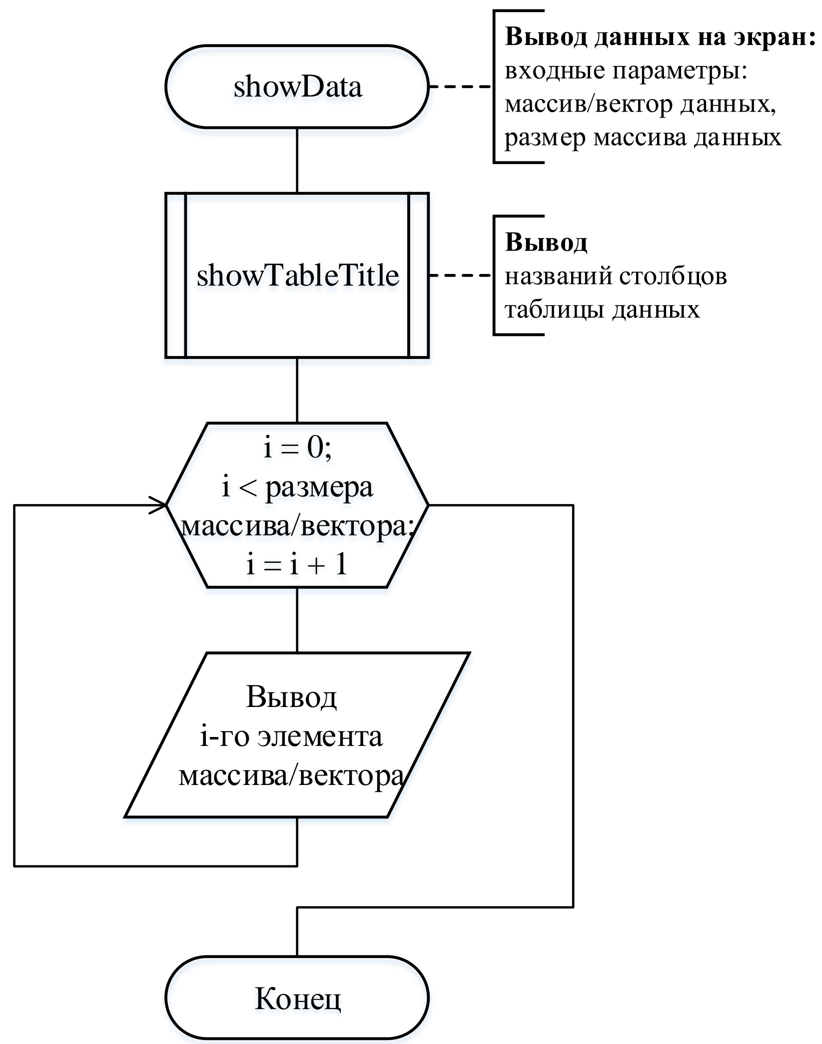


Рисунок 5.9 – Пример алгоритма функции showData: вывод данных на экран

6 Рекомендации по программированию курсовой работы

6.1 Типичные ошибки начинающих при написании кода. Способы отслеживания и устранения ошибок. Создание exe-файла проекта

Типичные ошибки начинающих при написани кода:

1. Пропуск «;» в конце инструкции или размещение «;» там, где это не нужно.
2. Путаница в количестве открывающихся «{» и закрывающихся «}» скобок (пишите обе скобки одновременно!).
3. Использование в условных конструкциях знака присвоения «=» вместо «==».
4. Ни о чем не говорящие имена переменных и функций.
5. Объявление служебных переменных и попытка их использования без инициализации (без присвоения начального значения).
6. «Магические» числа и строки в коде.
7. Отсутствие освобождения памяти (с помощью оператора delete) после ее ручного выделения (с помощью оператора new).
8. Использование слишком длинных функций, которые в реальности следовало бы разбить на несколько функций.
9. Сложная (неочевидная, неоптимальная) логика решения задачи.
10. Аргументация «Но программа же работает!» в ответ на замечания по логике работы программы, «недружественному» интерфейсу и оформлению кода.

Для отслеживания и устранения ошибок рекомендуется использовать следующие подходы в том порядке, в котором они перечислены далее:

1. Построение (сборка) проекта в среде разработки.
2. Отладка в пошаговом режиме.
3. Рефакторинг.

Частая сборка проекта, выполняемая в процессе работы над кодом, позволяет быстро выявлять ошибки компиляции и компоновки, например неверный синтаксис, несоответствия типов и другое (рисунок 6.1) и своевременно их устранять.

Успешная сборка – это подтверждение правильности синтаксиса кода приложения и корректного разрешения всех статических ссылок на библиотеки. Однако успешная сборка не исключает наличие логических ошибок, о которых будут свидетельствовать неверные результаты работы программы. Логические ошибки могуть быть эффективно локализованы посредством отладки программы, позволяющей узнать текущие значения переменных; выяснить, по какому пути выполнялась программа. Существуют две взаимодополняющие технологии отладки:

1. Пошаговое выполнение программы с остановками на строчках исходного кода (рисунок 6.2).
2. Логирование – вывод текущего состояния программы с помощью расположенных в критических точках программы операторов вывода.

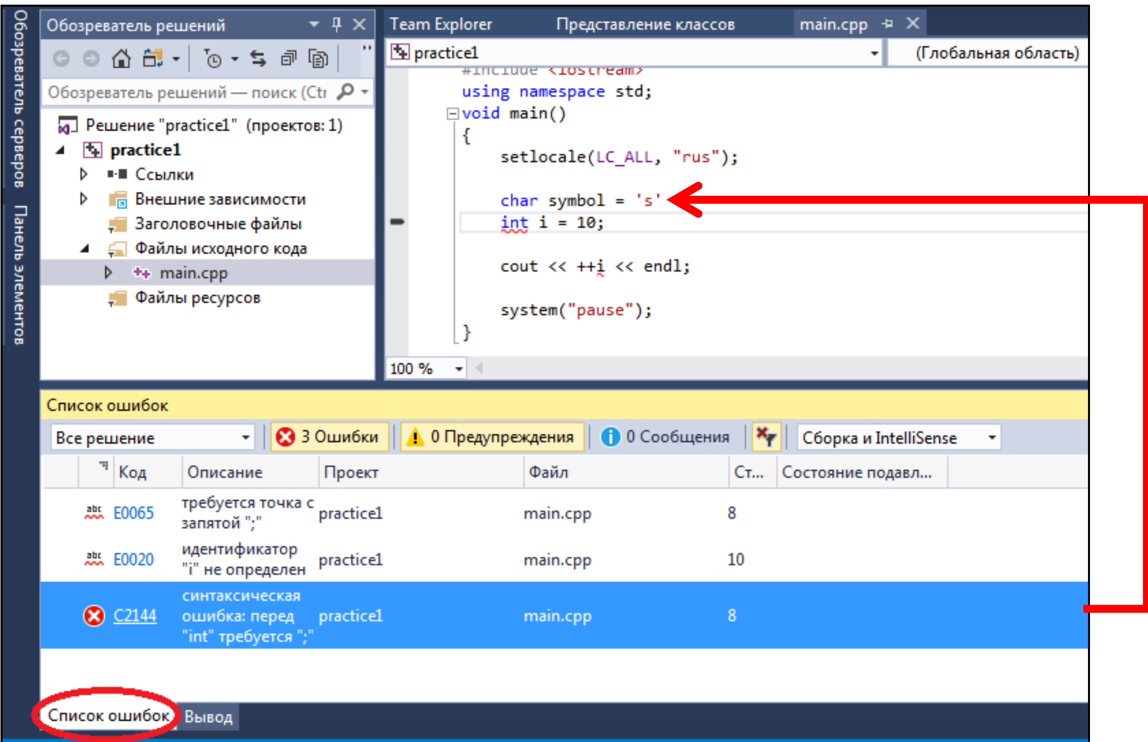
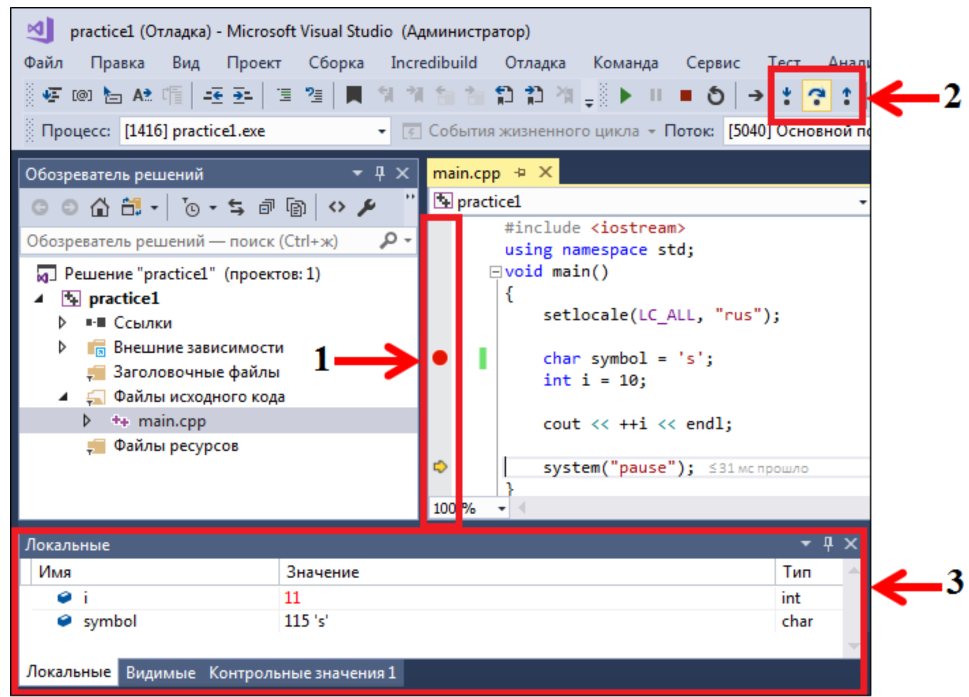


Рисунок 6.1 – Пример обнаружения ошибки сборки проекта



1 – устанавливаем точку остановки, 2 – после запуска программы переходим от точки остановки к следующей инструкции посредством панели инструментов для пошаговой отладки; 3 – отслеживаем значения переменных в процессе отладки

Рисунок 6.2 – Пример отладки программы в пошаговом режиме

Помимо синтаксических и логических ошибок собственно качество кода программы зачастую нуждается в улучшении – рефакторинге. Рефакторинг – процесс изменения внутренней структуры программы, не затрагивающий ее внешнего поведения и имеющий целью облегчить понимание ее работы. Ни о чем не говорящие имена переменных и функций, «магические» числа и строки в коде, использование слишком длинных функций, сложная (неочевидная, неоптимальная) логика – вот примеры проблем, которые необходимо решить на стадии рефакторинга.

После полного тестирования приложения и рефакторинга целесообразно скомпилировать итоговую release-версию проекта (помимо отладочной debug-версии) – рисунки 6.3, 6.4.

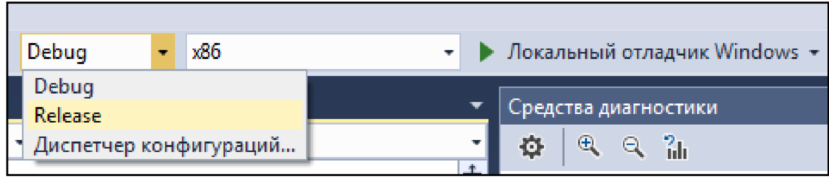


Рисунок 6.3 – Выбор версии проекта для сборки: debug или release

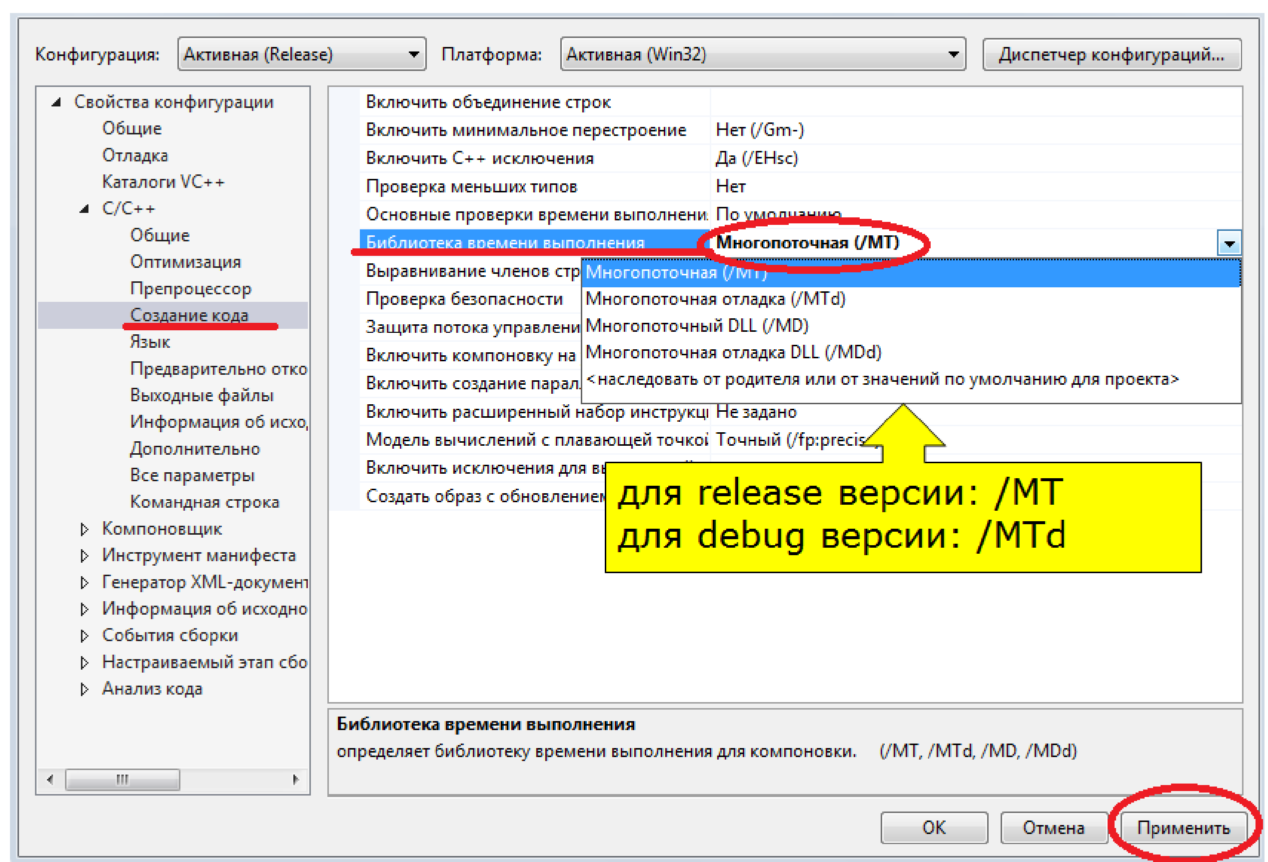


Рисунок 6.4 – Изменение динамического связывания библиотек времени выполнения на статическое связывание

Следует отметить, что приложения, созданные с помощью Microsoft Visual Studio, зависят от Visual C++ Redistibutable и динамически связаны с библиотеками MSVCR**.dll (Microsoft C Runtime Library). Для последующих запусков exe-файла на других компьютерах, которые потенциально могут не содержать требуемых библиотек (не установлена Microsoft Visual Studio или версия установленной Microsoft Visual Studio ниже требуемой) перед сборкой итоговой версии необходимо изменить динамическое связывание библиотек времени выполнения на статическое связывание. Для этого требуется перейти к свойствам проекта (щелкнув правой кнопкой мыши на имени проекта в *Обозревателе решений*), а в разделе С/С++ → *Создание кода* найти параметр *Библиотека времени выполнения*. Его необходимо изменить с *Многопоточная DLL (/MD)* на *Многопоточная (/MT)* – см. рисунок 6.4.

6.2 Структуры. Запись и чтение из файла

В языке С++ потоки, которые позволяют читать и записывать информацию в файл, являются объектами классов ifstream, ofstream. Они находятся в библиотеке с заголовочным файлом <fstream>:

- ifstream: класс, функции которого используются для чтения файлов;
- ofstream: класс, функции которого используются для записи файлов.

Название класса эквивалентно типу переменной, поэтому после названия класса объявляется объект, тип которого будет соответствовать классу. Для инициализации объектов в конструктор класса необходимо передать параметры: первый параметр – путь к файлу, второй параметр – режим работы с файлом. Константы режима файлов:

- ios::in – открыть файл для чтения;
- ios::out – открыть файл для записи;
- ios::ate – перейти к концу файла после открытия;
- ios::app – добавлять к концу файла;
- ios::binary – бинарный файл.

Далее приведены два примера записи и чтения в файл.

Первый пример: работа с файлом выполняется в текстовом режиме. В качестве строковых полей структуры используем класс string, структуры объединяем в статически создаваемые локальные массивы, запись и чтение выполняем посредством указания всех полей структуры.

Приведенный ниже пример ориентирован на статически создаваемые массивы, а именно включает целый ряд проверок выхода за пределы зарезервированной под статический массив памяти.

С точки зрения скорости данный вариант работы с файлом уступает второму варианту, однако сам файл содержит информацию в текстовом виде (также при ручной записи в файл информации требуемого формата она впоследствии может быть корректно считана программно).

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

struct Student
{
	string name;
	string surname;
	int age;
};

// Запись в файл (если что-то было в файле, то исходные данные будут удалены):
void writeFileStudents(Student *arr_of_students, int number_of_students);
// Добавление в конец файла одной строки:
void writeEndFileStudents(Student new_student);
// Чтение из файла в массив:
void readFileStudents(Student *arr_of_students, int &number_of_students);

// Заполнение массива студентов
void generateStudentArray(Student *arr_of_students, int &number_of_students);
// Добавление студента в массив
void addStudentInArray(Student *arr_of_students, int &number_of_students);
// Удаление студента из массива
void delStudentFromArray(Student *arr_of_students, int &number_of_students);
// Вывод содержимого массива на экран
void showStudentArray(Student *arr_of_students, int number_of_students);

const string FILE_OF_DATA = "MyFile.txt"; //Путь к файлу
const int RESERVE_SIZE = 100; //Максимальное количество элементов массива
```

```
void main()
{
	setlocale(LC_ALL, "rus");

	Student arr_of_students[RESERVE_SIZE];
	int number_of_students = 0;
	generateStudentArray(arr_of_students, number_of_students);
	writeFileStudents(arr_of_students, number_of_students);

	addStudentInArray(arr_of_students, number_of_students);
	showStudentArray(arr_of_students, number_of_students);

	delStudentFromArray(arr_of_students, number_of_students);
	showStudentArray(arr_of_students, number_of_students);

	Student		arr_new_of_students[RESERVE_SIZE];  /* Создаем новый массив
	исключительно для того, чтобы продемонстрировать корректность чтения
	данных из файла */
	int new_number_of_students = 0;
	readFileStudents(arr_new_of_students, new_number_of_students);
	showStudentArray(arr_new_of_students, new_number_of_students);

	system("pause");
}

void generateStudentArray(Student *arr_of_students, int &number_of_students)
{
	number_of_students = 2;

	arr_of_students[0].name = "Alex";
	arr_of_students[0].surname = "Black";
	arr_of_students[0].age = 20;

	arr_of_students[1].name = "Ivan";
	arr_of_students[1].surname = "Ivanov";
	arr_of_students[1].age = 25;
}

void addStudentInArray(Student *arr_of_students, int &number_of_students)
{
```

```
	//добавление студента, если не происходит выход за пределы массива
	if (number_of_students + 1 <= RESERVE_SIZE)
	{
		number_of_students++;
		cout << "Введите имя студента: ";
		cin >> arr_of_students[number_of_students - 1].name;
		cout << "Введите фамилию студента: ";
		cin >> arr_of_students[number_of_students - 1].surname;
		cout << "Введите возраст студента: ";
		cin >> arr_of_students[number_of_students - 1].age;

		writeEndFileStudents(arr_of_students[number_of_students - 1]);
	}
	else cout << "Недостаточно памяти для добавления нового элемента!" <<
endl;
}

void delStudentFromArray(Student *arr_of_students, int &number_of_students)
{
	int number_of_deleted_item;
	cout << "Введите номер удаляемой записи: ";
	cin >> number_of_deleted_item;
	// пользователь мыслит с 1, но индексы нумеруются с 0:
	number_of_deleted_item--;
	if (number_of_deleted_item >= 0 &&
	    number_of_deleted_item < number_of_students)
	{
		for (int i = number_of_deleted_item; i < number_of_students - 1; i++)
			arr_of_students[i] = arr_of_students[i + 1];

		number_of_students--;
		writeFileStudents(arr_of_students, number_of_students);
	}
	else cout << "Введен некорректный номер удалемой записи!" << endl;
}

void showStudentArray(Student *arr_of_students, int number_of_students)
{
	for (int i = 0; i<number_of_students; i++)
		cout << arr_of_students[i].name << " "
```

```
                << arr_of_students[i].surname << " "
                << arr_of_students[i].age << endl;
}

void writeFileStudents(Student *arr_of_students, int number_of_students)
{
        ofstream fout(FILE_OF_DATA, ios::out);  // Открыли файл для записи
        for (int i = 0; i<number_of_students; i++)
        {
                fout << arr_of_students[i].name << " "
                        << arr_of_students[i].surname << " "
                        << arr_of_students[i].age;
                if (i<number_of_students - 1)
                {
                        fout << endl;
                }
        }
        fout.close();
}

void writeEndFileStudents(Student new_student)
{
        ofstream fadd(FILE_OF_DATA, ios::app);  // Открыли файл для дозаписи
        fadd << endl;
        fadd << new_student.name << " "
                << new_student.surname << " "
                << new_student.age;
        fadd.close();
}

void readFileStudents(Student *arr_of_students, int &number_of_students)
{
        ifstream fin(FILE_OF_DATA, ios::in); // Открыли файл для чтения
        if (!fin.is_open()) cout << "Указанный файл не существует!" << endl;
        else
        {
                int i = 0;
                while (!fin.eof())
                {
                        if (i < RESERVE_SIZE)
```

```
			{
				fin >> arr_of_students[i].name
					>> arr_of_students[i].surname
					>> arr_of_students[i].age;
				i++;
			} else
			{
			cout << "Недостаточно памяти для чтения всех данных!" << endl;
			break;
			}
		}
		number_of_students = i;
	}
	fin.close();  //Закрыли файл
}
```

При работе с динамически создаваемыми массивами целесообразно запрашивать память в процессе выполнения программы, соответствующую количеству структур, записанных в файле. Для этого ниже приведена функция определения количества структур, записанных в текстовый файл (что соответствует количеству строчек в файле) – getCountOfStucturesInFile(string file_path).

```
// Определение количества структур в файле (при необходимости)
int getCountOfStucturesInFile(string file_path)
{
	ifstream file(file_path, ios::in); // Открыли текстовый файл для чтения
	int number_of_strings = 0;
	if (file.is_open())
	{
		string buffer;
		while (getline(file, buffer))
			number_of_strings++;
	}
	file.close();
	return number_of_strings;
}
```

Второй пример: работа с файлом выполняется в бинарном режиме. Сначала приведем ряд пояснений по особенностям чтения и записи в файл в бинарном режиме.

При записи единичной структуры в бинарный файл записываем всю структуру целиком и сразу:

```
ofstream fadd(FILE_OF_DATA, ios::binary | ios::app);
fadd.write((char*)&new_student, sizeof(new_student)); /* Записываем структуру new_student в открытый нами файл; для этого узнаем адрес структуры new_student и приводим указатель на new_student к однобайтовому типу; указываем, что в структуре new_student находится sizeof(new_student) байт */
fadd.close();
```

Из файла можно прочитать всю структуру целиком и сразу:

```
ifstream fin(FILE_OF_DATA, ios::binary | ios::in);
if (!fin.is_open()) cout << "Указанный файл не существует!" << endl;
else
{
   fin.read((char*) &temp_student, sizeof(temp_student));
}
fin.close();
```

При записи массива структур в бинарный файл записываем весь массив целиком и сразу:

```
ofstream fout(FILE_OF_DATA, ios::binary | ios::out);
fout.write((char*)&arr_of_students[0], sizeof(Student)*number_of_students);
fout.close();
```

Из файла целесообразно читать весь массив структур целиком и сразу, однако для этого необходимо знать количество структур в бинарном файле:

```
int getCountOfStucturesInFile(string file_path)
{
        //Открываем файл и перемещаем указатель в конец файла
        ifstream file(FILE_OF_DATA, ios::ate | ios::binary);

        /* file.tellg() возвращает значение типа int, которое показывает, сколько указателем пройдено в байтах от начала файла до текущей позиции */
        int number_of_strings = file.tellg() / sizeof(Student);
        file.close();
```

```
    return number_of_strings;
}
```

Тогда собственно чтение в массив структур из файла программно реализуется следующим образом:

```
ifstream fin(FILE_OF_DATA, ios::binary | ios::in);
if (!fin.is_open()) cout << "Указанный файл не существует!" << endl;
else
{
  fin.read((char*)&arr_of_students[0],
            sizeof(Student)* getCountOfStucturesInFile(FILE_OF_DATA));
}
fin.close();
```

Когда приходится записывать структуру или объект в файл, то внутри структуры может быть очень много информационных полей и при записи структуры в файл в текстовом режиме придется записывать каждый элемент структуры, а это отнимет много времени. В этом отношении данный вариант работы с файлом имеет преимущество по скорости. Но заполненный таким способом файл при открытии в текстовом режиме нечитабельный, а попытка вручную записать информацию, чтобы считать ее программно, не приведет к желаемому результату.

Далее приведен пример работы с файлом в бинарном режиме. В качестве строковых полей структуры используем строковые массивы, структуры объединяем в статически создаваемые локальные массивы.

Приведенный ниже пример ориентирован на статически создаваемые массивы, а именно, включает целый ряд проверок выхода за пределы зарезервированной под статический массив памяти.

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

const int SIZE_CHAR = 20;
const int RESERVE_SIZE = 100;
const string FILE_OF_DATA = "MyFile.txt"; //Путь к файлу
```

```
struct Student
{
	char name[SIZE_CHAR];
	char surname[SIZE_CHAR];
	int age;
};

// Запись в файл (если что-то было в файле, то исходные данные будут удалены):
void writeFileStudents(Student *arr_of_students, int number_of_students);
// Добавление в конец файла одной строки:
void writeEndFileStudents(Student new_student);
// Чтение из файла в массив:
void readFileStudents(Student *arr_of_students, int &number_of_students);
// Определение количества структур в файле (при необходимости)
int getCountOfStucturesInFile(string file_path);

//0 Заполнение массива студентов
void generateStudentArray(Student *arr_of_students, int &number_of_students);
// Добавление студента в массив
void addStudentInArray(Student *arr_of_students, int &number_of_students);
// Удаление студента из массива
void delStudentFromArray(Student *arr_of_students, int &number_of_students);
// Вывод содержимого массива на экран
void showStudentArray(Student *arr_of_students, int number_of_students);

void main()
{
	setlocale(LC_ALL, "rus");

	Student arr_of_students[RESERVE_SIZE];
	int number_of_students = 0;
	generateStudentArray(arr_of_students, number_of_students);
	writeFileStudents(arr_of_students, number_of_students);

	addStudentInArray(arr_of_students, number_of_students);
	showStudentArray(arr_of_students, number_of_students);

	delStudentFromArray(arr_of_students, number_of_students);
	showStudentArray(arr_of_students, number_of_students);
```

```
	Student arr_new_of_students[RESERVE_SIZE]; /* Создаем новый массив исключительно для того, чтобы продемонстрировать корректность чтения данных из файла */
	int new_number_of_students = 0;
	readFileStudents(arr_new_of_students, new_number_of_students);
	showStudentArray(arr_new_of_students, new_number_of_students);

	system("pause");
}

void generateStudentArray(Student *arr_of_students, int &number_of_students)
{
	number_of_students = 2;

	strcpy_s(arr_of_students[0].name, "Alex");
	strcpy_s(arr_of_students[0].surname, "Black");
	arr_of_students[0].age = 20;

	strcpy_s(arr_of_students[1].name, "Alex1");
	strcpy_s(arr_of_students[1].surname, "Black1");
	arr_of_students[1].age = 27;
}

void addStudentInArray(Student *arr_of_students, int &number_of_students)
{
	//добавление студента, если не происходит выход за пределы массива
	if (number_of_students + 1 <= RESERVE_SIZE)
	{
		number_of_students++;
		cout << "Введите имя студента: ";
		cin >> arr_of_students[number_of_students - 1].name;
		cout << "Введите фамилию студента: ";
		cin >> arr_of_students[number_of_students - 1].surname;
		cout << "Введите возраст студента: ";
		cin >> arr_of_students[number_of_students - 1].age;

		writeEndFileStudents(arr_of_students[number_of_students - 1]);
	}
```

```
        else cout << "Недостаточно памяти для добавления нового элемента!" <<
endl;
}

void delStudentFromArray(Student *arr_of_students, int &number_of_students)
{
        int number_of_deleted_item;
        cout << "Введите номер удаляемой записи: ";
        cin >> number_of_deleted_item;
        // пользователь мыслит с 1, но индексы нумеруются с 0:
        number_of_deleted_item--;
                if (number_of_deleted_item >= 0 &&
                        number_of_deleted_item < number_of_students)
                {
                        for (int i = number_of_deleted_item; i < number_of_students - 1; i++)
                                arr_of_students[i] = arr_of_students[i + 1];

                         number_of_students--;
                         writeFileStudents(arr_of_students, number_of_students);
                }
                else cout << "Введен некорректный номер удалемой записи!" << endl;
}

void showStudentArray(Student *arr_of_students, int number_of_students)
{
        for (int i = 0; i < number_of_students; i++)
                cout << arr_of_students[i].name << " "
                << arr_of_students[i].surname << " "
                << arr_of_students[i].age << endl;
}

void writeFileStudents(Student *arr_of_students, int number_of_students)
{
        //Открываем файл для записи:
        ofstream fout(FILE_OF_DATA, ios::binary | ios::out);
        fout.write((char*)&arr_of_students[0], sizeof(Student)*number_of_students);
        fout.close();
}

void writeEndFileStudents(Student new_student)
```

```
{
	//Открываем файл для дозаписи:
	ofstream fadd(FILE_OF_DATA, ios::binary | ios::app);
	fadd.write((char*)&new_student, sizeof(new_student)); //Записали структуру
	fadd.close();
}

void readFileStudents(Student *arr_of_students, int &number_of_students)
{
	ifstream fin(FILE_OF_DATA, ios::binary | ios::in);

	if (!fin.is_open()) cout << "Указанный файл не существует!" << endl;
	else
	{
		// определяем количество строк в файле
		int sizeOfFileWithStudents = getCountOfStucturesInFile(FILE_OF_DATA);
		/* если выделенная память под статический массив вмещает все строч-
		ки в файле */
		 if (sizeOfFileWithStudents <= RESERVE_SIZE)
		 {
			// будем считывать все строчки
			number_of_students = sizeOfFileWithStudents;
		 }
		 else
		 {
		 // иначе считаем ровно столько, насколько хватает места в массиве
			number_of_students = RESERVE_SIZE;
			cout << "There is not enough memory for read all data!" << endl;
		 }
		 /* читаем сразу number_of_students-строчек из файла и сохраняем их в
		 массиве */
		 fin.read((char*)&arr_of_students[0], sizeof(Student)*number_of_students);
	}
	fin.close();
}

int getCountOfStucturesInFile(string file_path)
{
	//Открываем файл и перемещаем указатель в конец файла
	ifstream file(FILE_OF_DATA, ios::ate | ios::binary);
```

```
        /*file.tellg() возвращает значение типа int, которое показывает, сколько ука-
        зателем пройдено в байтах от начала файла до текущей позиции */
        int number_of_strings = file.tellg() / sizeof(Student);
        file.close();

        return number_of_strings;
}
```

6.3 Перевыделение памяти с целью увеличения размера динамически созданного массива

При работе с динамически создаваемыми массивами может возникнуть необходимость увеличения размера памяти, требуемой для хранения информации (например, для добавления в последующем новых данных). Для решения такой задачи целесообразно выполнить перевыделение памяти под массив.

Далее приведен пример программы, выполняющей динамическое создание массива, его заполнение случайными числами из отрезка [a; b], вывод результата заполнения на консоль, перевыделение памяти с целью увеличения размера массива на один элемент, добавление нового элементы и повторный вывод содержимого массива на консоль.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

void generateElementsOfArray(int *arr, int n);
void showArray(int *arr, int n);
int* getRememberForArray(int *arr, int &n, int m);
int inputNumber();

void main()
{
        setlocale(LC_ALL, "RUS");
        int n;
        cout << "Введите размер массива n: ";
        cin >> n;
```

```
	int *arr = new int[n];
	generateElementsOfArray(arr, n);
	cout << "Массив arr (исходный):"<<endl;
	showArray(arr, n);

	arr = getRememberForArray(arr, n, n+1);
	arr[n-1] = inputNumber();
	cout << "Новый массив arr:" << endl;
	showArray(arr, n);
	delete[]arr;
	system("pause");
}

void generateElementsOfArray(int *arr, int n)
{
	srand(time(NULL));
	int a, b;
	cout << "Введите a: ";
	cin >> a;
	cout << "Введите b: ";
	cin >> b;
	for (int i = 0; i < n; i++)
	{
		arr[i] = rand() % (b - a + 1) + a;
	}
}

void showArray(int *arr, int n)
{
	for (int i = 0; i<n; i++) {
		cout << arr[i] << " ";
	}
	cout << endl;
}

int inputNumber()
{
	int new_element;
	cout << "Введите новый элемент массива: ";
```

```
        cin >> new_element;
        return new_element;
}

int* getRememberForArray(int *arr, int &n, int m)
{
        int *arr_new;
        if (n < m)
        {
                arr_new = new int[m];
                for (int i = 0; i < n; i++)
                {
                        arr_new[i] = arr[i];
                }
                for (int i = n; i < m; i++)   // дополняем массив нулевыми значениями
                {
                        arr_new[i] = 0;
                }
                delete[]arr;
                arr = arr_new;
                n = m;
        }
        return arr;
}
```

6.4 Проверка корректности вводимых данных

При программировании логики курсовой работы необходимо предусмотреть обработку целого ряда исключительных ситуаций. Наиболее часто встречающиеся исключительные ситуации связаны с ошибками пользователя на этапе ввода данных, а именно: введенные пользователем данные не соответствуют формату поля *(например, символы в числовом поле)*; введенные пользователем данные нелогичны *(например, отрицательная цена товара)*.

Так как пользователи могут ошибаться и изменить этот факт невозможно, то необходимо корректно обработать возможные ошибки ввода. Далее приведен пример программы, реализующей ввод пользователем возраста с проверкой на принадлежность целым числам из интервала (0; 100). Результат работы программы отражен на рисунке 6.5.

Пояснения к используемым в коде библиотечным методам по работе с потоковым вводом/выводом на консоль:

1. cin.get() – возвращает последний символ из потока. Если все символы были цифрами (допустимо использование в качестве первого символа знака минус при необходимости ввода отрицательных чисел), то последним символом потока будет переход на новую строку, т. е. \n. В противном случае можно сделать заключение о нецифровом вводе.

2. cin.clear() – сбрасывает флаги ошибок, в противном случае повторный вызов cin не сработает.

3. cin.ignore(numeric_limits<streamsize>::max(), '\n') – очищает поток (отбрасываем максимальное число символов, которое может содержать поток, до первого перевода строки включительно).

Если в проекте подключена библиотека windows.h, то возникнет конфликтная ситуация с методом max(). Тогда вместо

cin.ignore(numeric_limits<streamsize>::max(), '\n');

используйте

cin.ignore(256, '\n');

В качестве 256 можно указать и другое число. Важно понимать: это то максимально возможное количество символов, которое будет удалено из потока.

```
Введите Ваш возраст: qwerty
Некорректный ввод! Повторите попытку: qwerty18
Некорректный ввод! Повторите попытку: 18qwerty
Некорректный ввод! Повторите попытку: 18.5
Некорректный ввод! Повторите попытку: 18,5
Некорректный ввод! Повторите попытку: -45
Некорректный ввод! Повторите попытку: 1000
Некорректный ввод! Повторите попытку: 0
Некорректный ввод! Повторите попытку: 18
18
Для продолжения нажмите любую клавишу . . .
```

Рисунок 6.5 – Результат работы программы, запрашивающей возраст пользователя

```
#include <iostream>
using namespace std;

const int LEFT_RANGE_OF_AGE = 0;
const int RIGHT_RANGE_OF_AGE = 100;

int inputNumber(int left_range, int right_range);
bool isNumberNumeric();
bool isNumberRangeCorrect(int number, int left_range, int right_range);
```

```
void main()
{
	setlocale(LC_ALL, "rus");
	cout << "Введите ваш возраст: ";
	int age = inputNumber(LEFT_RANGE_OF_AGE, RIGHT_RANGE_OF_AGE);
	cout << age << endl;
	system("pause");
}

int inputNumber(int left_range, int right_range)
{
	int number;
	while (true)
	{
		cin >> number;
		if (isNumberNumeric() && isNumberRangeCorrect(number, left_range,
right_range))
			return number;
		else
			cout << "Некорректный ввод! Повторите попытку: ";
	}
}

bool isNumberNumeric()
{
	if (cin.get() == '\n')
		return true;
	else
	{
		cin.clear();
		cin.ignore(std::numeric_limits<streamsize>::max(), '\n');
		return false;
	}
}

bool isNumberRangeCorrect(int number, int left_range, int right_range)
{
	if ((number > left_range) && (number < right_range))
		return true;
	else
		return false;
}
```

6.5 Определение текущей даты и времени

В курсовой работе может возникнуть необходимость работы с текущей датой и/или текущим временем. Например, для вывода на консоль всех товаров, хранящихся более x месяцев (x вводится с клавиатуры), необходимо знать текущий месяц. Для вывода на консоль списка сотрудников пенсионного возраста требуется знать текущую календарную дату. Часто для проверки вводимых данных на корректность (например, даты рождения) необходимо программно запрашивать текущий год и проверять, чтобы вводимый пользователем год рождения по крайней мере не превышал текущий календарный год.

Рассмотрим две программы для определения календарной даты и текущего времени, в результате которых осуществляется вывод на консоль системной даты в формате ДД.ММ.ГГГГ и системного времени в формате Часы:Минуты:Секунды.

Первая программа, приведенная ниже, работает со структурой struct tm (таблица 6.1), в которой посредством функции localtime_s библиотеки time.h сохраняется системная дата и время (с учетом часового пояса). Для удобства последующей работы с параметрами даты и времени в программе введены новые пользовательские типы данных – Date и Time с полями, характеризующими дату и время соответственно посредством полей типа int.

Таблица 6.1 – Параметры структуры struct tm из библиотеки time.h

Поле структуры tm	Тип	Значение	Диапазон
tm_sec	int	Количество секунд	0-59
tm_min	int	Количество минут	0-59
tm_hour	int	Количество часов	0-23
tm_mday	int	День месяца	1-31
tm_mon	int	Количество месяцев с января	0-11
tm_year	int	Количество лет, прошедших с 1900 года	

```
#include <iostream>
#include <time.h>
using namespace std;

struct Date
{
	int day;
	int month;
	int year;
```

```
};
struct Time
{
	int hour;
	int minute;
	int second;
};

Date getCurrentDate();
Time getCurrentTime();
void showDate(Date date);
void showTime(Time time);

void main()
{
	showDate(getCurrentDate());
	showTime(getCurrentTime());
	system("pause");
}

Date getCurrentDate()
{
	struct tm localtime;
	time_t now = time(NULL);
	localtime_s(&localtime, &now);
	Date currentDate;
	currentDate.day = localtime.tm_mday;
	currentDate.month = 1 + localtime.tm_mon;  // tm_mon: months in range [0-11]
	currentDate.year = 1900 + localtime.tm_year; // tm_year: years since 1900
	return currentDate;
}

Time getCurrentTime()
{
	struct tm localtime;
	time_t now = time(NULL);
	localtime_s(&localtime, &now);
	Time currentTime;
	currentTime.hour = localtime.tm_hour;
	currentTime.minute = localtime.tm_min;
	currentTime.second = localtime.tm_sec;
```

```
	return currentTime;
}

void showDate(Date date)
{
	cout << date.day << "." << date.month << "." << date.year << endl;
}

void showTime(Time time)
{
	cout << time.hour << ":" << time.minute << ":" << time.second << endl;
}
```

Вторая программа, приведенная ниже, работает со структурой SYSTEMTIME tm, в которой посредством функции GetLocalTime библиотеки windows.h сохраняется системная дата и время (с учетом часового пояса). Для удобства последующей работы с параметрами даты и времени в программе введены новые пользовательские типы данных – Date и Time с полями, характеризующими дату и время соответственно посредством полей типа string.

```
#include <iostream>
#include <windows.h>
#include <time.h>
#include <string>
using namespace std;

struct Date
{
	string day;
	string month;
	string year;
};
struct Time
{
	string hour;
	string minute;
	string second;
};

Date getCurrentDate();
Time getCurrentTime();
```

```
void showDate(Date date);
void showTime(Time time);

void main()
{
	showDate(getCurrentDate());
	showTime(getCurrentTime());
	system("pause");
}

Date getCurrentDate()
{
	SYSTEMTIME tm;
	GetLocalTime(&tm);
	Date currentDate;
	currentDate.day = to_string(tm.wDay);
	currentDate.month = to_string(tm.wMonth);
	currentDate.year = to_string(tm.wYear);
	return currentDate;
}

Time getCurrentTime()
{
	SYSTEMTIME tm;
	GetLocalTime(&tm);
	Time currentTime;
	currentTime.hour = to_string(tm.wHour);
	currentTime.minute = to_string(tm.wMinute);
	currentTime.second = to_string(tm.wSecond);
	return currentTime;
}

void showDate(Date date)
{
	cout << date.day << "." << date.month << "." << date.year << endl;
}

void showTime(Time time)
{
	cout << time.hour << ":" << time.minute << ":" << time.second << endl;
}
```

Приложение А
(обязательное)
Задания для курсовой работы

1. Разработка программы учета товаров на складе

Программа предоставляет сведения о товарах, имеющихся на складе: наименование товара, количество единиц товара, цена единицы товара, дата поступления товара на склад, ФИО зарегистрировавшего товар.

Индивидуальное задание: вывести в алфавитном порядке список товаров, хранящихся более x месяцев, стоимость которых превышает y рублей (x, y вводятся с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

2. Разработка программы распределения мест в общежитии

Для получения места в общежитии формируется список студентов, который включает: ФИО студента, номер группы (шесть цифр), средний балл, участие в общественной деятельности, доход на одного члена семьи. Общежитие в первую очередь предоставляется тем студентам, чьи доходы на члена семьи меньше двух минимальных зарплат, затем остальным в порядке уменьшения среднего балла (при равных баллах приоритет отдается тем, кто участвовал в общественной деятельности).

Индивидуальное задание: вывести список очередности предоставления места в общежитии при условии, что размер минимальной зарплаты вводится с клавиатуры.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

3. Разработка программы расписания движения автобусов

В справочной автовокзала хранится расписание движения автобусов. Для каждого рейса указаны: номер рейса, тип автобуса, пункт назначения, время отправления, время прибытия на конечный пункт.

Индивидуальное задание: вывести информацию о всех рейсах, которыми можно воспользоваться для прибытия в пункт назначения не позднее чем за 12 часов до заданного времени (интересующее время прибытия вводится с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

4. Разработка программы продажи автобусных билетов

Автовокзал осуществляет продажу билетов на пригородные рейсы. Для каждого рейса указаны: номер рейса, тип автобуса, пункт назначения, дату отправления, время отправления, время прибытия на конечный пункт, стоимость одного билета, количество оставшихся для продажи билетов, количество проданных билетов.

Индивидуальное задание: обеспечить функциональную возможность покупки билетов на конкретный рейс в требуемом количестве (при этом количество оставшихся для продажи билетов и количество проданных билетов обновляется).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

5. Разработка программы учета переговоров абонентов сотовой связи

Оператор сотовой связи хранит информацию о разговорах своих абонентов: номер абонента, ФИО абонента, указание принадлежности вызова к исходящему или входящему, номер исходящего или входящего вызова, дата звонка, время звонка, продолжительность разговора, тариф одной минуты.

Индивидуальное задание: вывести по каждому абоненту за требуемый период времени: перечень входящих и исходящих вызовов, общее время входящих вызовов, общее время исходящих вызовов, общую сумму на исходящие вызовы (требуемый период времени вводится с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

6. Разработка программы расчета заработной платы сотрудников предприятия

Сведения о сотрудниках предприятия содержат: Ф.И.О. сотрудника, табельный номер, год, месяц, количество проработанных часов за месяц, почасовой тариф. Рабочее время свыше 144 часов считается сверхурочным и оплачивается в двойном размере.

Индивидуальное задание: рассчитать размер заработной платы каждого сотрудника за вычетом подоходного налога, который составляет 12 % от суммы заработной платы. Определить объем выплат конкретному сотруднику за требуемый период времени (требуемый период времени вводится с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

7. Разработка программы учета книг в библиотеке

Для книг, хранящихся в библиотеке, задаются: регистрационный номер книги, автор, название, год издания, издательство, количество страниц, номер читательского билета (шесть цифр) последнего читателя, отметка о нахождении книги у читателя или в библиотеке в текущий момент.

Индивидуальное задание: вывести список книг с фамилиями авторов в алфавитном порядке, изданных после заданного года (год вводится с клавиатуры). Вывести список книг, находящихся в текущий момент у читателей.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

8. Разработка программы учета выпускаемой предприятием продукции

Сведения о выпущенной продукции включают: дату, номер цеха, наименование продукции, количество выпущенных единиц, ФИО ответственного по цеху в данный день.

Индивидуальное задание: для заданного цеха необходимо вывести количество выпущенных изделий по каждому наименованию за требуемый период времени (требуемый период времени вводится с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

9. Разработка программы учета стажа сотрудников предприятия

Информация о сотрудниках предприятия содержит: ФИО сотрудника, дату рождения, название отдела, должность, дату начала работы.

Индивидуальное задание: вывести список сотрудников пенсионного возраста. Вывести список сотрудников в порядке убывания стажа.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

10. Разработка программы учета выплат заработной платы сотрудникам предприятия

Информация о сотрудниках предприятия содержит: ФИО сотрудника, название отдела, должность, размер заработной платы за месяц.

Индивидуальное задание: вычислить общую сумму выплат за месяц по каждому отделу, а также среднемесячный заработок сотрудников по каждому отделу. Вывести список сотрудников, у которых зарплата ниже введенной с клавиатуры.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

11. Разработка программы учета сведений об абонентах сотовой связи

Оператор сотовой связи хранит информацию о своих абонентах: ФИО абонента, номер телефона, год подключения, наименование текущего тарифного плана.

Индивидуальное задание: вывести список и подсчитать общее количество абонентов, подключенных с хххх года (год вводится с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

12. Разработка программы ассортимента игрушек в магазине

Сведения об ассортименте игрушек в магазине включают: название игрушки, цена, изготовитель, количество, минимальная рекомендуемая граница по возрасту.

Индивидуальное задание: вывести список игрушек, которые подходят детям в возрасте х лет (х вводится с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

13. Разработка программы ассортимента обуви в магазине

Сведения об ассортименте обуви в магазине включают: артикул, наименование, цена, изготовитель, размер, количество пар. Артикул начинается с буквы Ж – для женской обуви, М – для мужской, Д – для детской.

Индивидуальное задание: вывести список обуви артикула х и размера у (х, у вводятся с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

14. Разработка программы учета заказов сервисного центра

В сервисном центре хранится информация обо всех заказах: наименование ремонтируемого изделия (телевизор и т. д.), марка изделия, ФИО владельца, телефон владельца, стоимость ремонта, дата приемки, дата выдачи, статус (выполнен или нет).

Индивидуальное задание: вывести список заказов, невыполненных на текущий момент (сначала – просроченные, затем – ожидающие выполнения в плановом порядке). Вывести общий доход от даты х до даты у (х, у вводятся с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

15. Разработка программы учета успеваемости студентов

Сведения об успеваемости студентов содержат следующую информацию: номер группы (шесть цифр), ФИО студента, сведения о пяти зачетах (зачет/незачет), отметки по пяти экзаменам.

Индивидуальное задание: вывести всех студентов в порядке убывания количества задолженностей (количество задолженностей по каждому студенту необходимо указать). Вывести средний балл, полученный каждым студентом группы x (вводится с клавиатуры), и средний балл всей группы в целом.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

16. Разработка программы учета сведений о музыкальном конкурсе

Информация о конкурсе включает: ФИО участника, год рождения, название страны, наименование музыкального инструмента (гитара, фортепиано, скрипка, виолончель и др.), занятое место по результатам конкурса.

Индивидуальное задание: по каждому классу музыкальных инструментов вывести список первых трех мест с указанием возраста победителей. Вывести список самых молодых (до 12 лет) победителей конкурса в порядке увеличения возраста.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

17. Разработка программы учета сведений о пациентах медицинского центра

Сведения о пациентах медицинского центра содержат: ФИО пациента, пол, дату рождения, место проживания (город), контактный телефон, диагноз.

Индивидуальное задание: вывести иногородних пациентов. Вывести список пациентов старше x лет, у которых диагноз y (x, y вводятся с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

18. Разработка программы продажи железнодорожных билетов

Железнодорожный вокзал осуществляет продажу билетов на поезда дальнего следования. Для каждого поезда указаны: номер поезда, пункт назначения, дату отправления, время отправления, время прибытия на конечный пункт, стоимость

одного билета, количество оставшихся для продажи билетов, количество проданных билетов.

Индивидуальное задание: обеспечить функциональную возможность покупки билетов на конкретный рейс в требуемом количестве (при этом количество оставшихся для продажи билетов и количество проданных билетов обновляется). Вывести номер, время отправления и наличие билетов для поездов, прибывающих в город x в интервале от a до б (x, a, b вводятся с клавиатуры).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

19. Разработка программы планирования факультативных учебных дисциплин для студентов

Для формирования факультативных занятий необходимо обработать информацию следующего вида: ФИО студента, номер группы, средний балл успеваемости, пять возможных для факультативного посещения дисциплин. Выбираемая дисциплина помечается 1, невыбираемая – 0. Перечень предлагаемых дисциплин: математика, физика, программирование, английский язык, базы данных.

Индивидуальное задание: вывести список и общее количество студентов, желающих прослушать дисциплину x. Если число желающих больше 15, то отобрать 15 студентов с более высоким баллом успеваемости. Вывести предлагаемые дисциплины в порядке убывания популярности с указанием общего числа записавшихся на каждую из них.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

20. Разработка программы учета сведений об игроках хоккейной команды

Сведения об игроках хоккейной команды включают: ФИО игрока, дату рождения, количество сыгранных матчей, число заброшенных шайб, количество голевых передач, количество штрафных минут.

Индивидуальное задание: вывести шесть лучших игроков (голы + передачи) с указанием их результативности.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

21. Разработка программы продажи авиабилетов

Авиакомпания осуществляет продажу билетов на самолеты с указанием: номера рейса, типа самолета, пункта назначения, дату вылета, время вылета, время прилета, вместимости самолета, количества оставшихся билетов бизнес-класса, стоимости билета бизнес-класса, количества оставшихся билетов эконом-класса, стоимости билета эконом-класса.

Индивидуальное задание: обеспечить функциональную возможность покупки билетов на конкретный рейс в требуемом количестве (при этом количество оставшихся для продажи билетов обновляется). Если на интересующий рейс нет билетов требуемого класса, то при наличии билетов другого класса на этот рейс – вывести соответствующее информационное сообщение (например, «Билетов эконом-класса на данный рейс нет в наличии, но имеются билеты бизнес-класса в количестве 10»).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

22. Разработка программы учета автомобилей таксопарка

Таксопарк содержит информацию об имеющихся автомобилях: вид автомобиля (такси, микроавтобус, лимузин), вместимость, расход топлива, стоимость автомобиля, количество автомобилей данного вида.

Индивидуальное задание: подсчитать общую стоимость таксопарка. Подсчитать общую стоимость автомобилей каждого вида. Подобрать автомобили, по вместимости соответствующие заданному с клавиатуры диапазону.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

23. Разработка программы расчета стипендии

Для расчета стипендии необходимо обработать информацию следующего вида: номер группы, ФИО студента, форма обучения (платная/бюджетная), зачеты по пяти предметам (зачет/незачет), отметки по четырем предметам, признак участия в общественной работе: 1 – активное участие, 0 – неучастие.

Индивидуальное задание: рассчитать стипендию для студентов-бюджетников. При этом отличники (отметки 9, 10) и общественники получают 50 % надбавку, а просто отличники – 25 %. Студенты со средним баллом, равным или ниже 5, стипендию не получают. Базовый размер стипендии вводится с клавиатуры.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

24. Разработка программы начисления пособий по уходу за ребенком

Сведения о детях сотрудниц компании содержат следующую информацию: ФИО сотрудницы, дата рождения ребенка, ФИО ребенка. Для каждого ребенка сотрудницы создается отдельная запись указанного вида.

Индивидуальное задание: для каждой сотрудницы вывести количество детей, а также рассчитать общую сумму полагающихся пособий. Пособие начисляется для детей в возрасте до трех лет. Для первого ребенка в семье базовое пособие составляет x (вводится с клавиатуры), для второго 1,25x, для третьего и последующих 1,5x.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

25. Разработка программы планирования бюджета проекта

В компании по разработке программного обеспечения для планирования работ по проекту необходимо обработать информацию следующего вида: наименование проекта, вид работ (работа над требованиями, разработка архитектуры, реализация, тестирование), ФИО сотрудника, предполагаемое количество часов, стоимость одного часа.

Индивидуальное задание: для каждого проекта вывести его итоговую стоимость, перечень видов работ с указанием общего количества задействованных специалистов и стоимости данного этапа.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

26. Разработка программы расчета выплат по больничным листам

Сведения о больничных сотрудников компании имеют следующий вид: ФИО сотрудника, год, месяц, количество дней, пропущенных по болезни, оплата за один день.

Индивидуальное задание: для месяца x года y вывести список сотрудников с указанием выплат по больничным листам для каждого из них. Вывести общую сумму выплат по больничным листам за интересующий месяц. X, y вводятся с клавиатуры.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

27. Разработка программы подбора туристической путевки

Туристическая компания содержит сведения о предлагаемых путевках: тип путевки (отдых, экскурсионный тур, лечение, шопинг, круиз), страна пребывания, вид транспорта, количество дней, питание (завтраки/все включено), стоимость.

Индивидуальное задание: вывести все путевки требуемого типа (вводится с клавиатуры) стоимостью менее x в порядке убывания стоимости.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

28. Разработка программы учета продаж театральных билетов

Сведения о продажах билетной кассы содержат следующую информацию: дата, наименование театра, наименование спектакля, количество проданных билетов.

Индивидуальное задание: вывести список самых популярных театров в порядке убывания в указанный месяц. Вывести список самых популярных спектаклей в порядке убывания в указанный месяц. Месяц вводится с клавиатуры.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

29. Разработка программы учета продаж проездных билетов

Сведения о продажах проездных билетов содержат следующую информацию: год, месяц, наименование транспорта, количество проданных проездных билетов, стоимость одного проездного билета на данный вид транспорта.

Индивидуальное задание: определить общий доход от продажи проездных билетов за определенный месяц. Вывести список самых востребованных видов транспорта в порядке убывания в указанный месяц. Месяц вводится с клавиатуры.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

30. Разработка программы учета командировок сотрудников предприятия

Сведения о командировках сотрудников компании содержат следующую информацию: ФИО сотрудника, год, месяц, длительность командировки в днях, город, в который осуществлялся выезд, сумма командировочных расходов на один день.

Индивидуальное задание: определить общие выплаты командировочных за указанный месяц. Вывести список наиболее часто посещаемых городов в порядке убывания за указанный период (с месяца x по месяц y).

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

31. Разработка программы учета обучающихся в детском центре

Сведения об обучающихся в детском центре содержат следующую информацию: наименование секции, ФИО ребенка, дата рождения, ФИО родителя, контактный телефон, стоимость обучения в месяц, сумма задолженности по оплате (0 – в случае отсутствия задолженности).

Индивидуальное задание: вывести список должников в порядке убывания размера долга. Вывести имеющуюся в центре информацию в порядке увеличения возраста детей.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

32. Разработка программы учета сведений об игроках футбольной команды

Сведения об игроках футбольной команды включают: ФИО игрока, дату рождения, количество сыгранных матчей, число забитых мячей, количество голевых передач, количество желтых карточек, количество красных карточек.

Индивидуальное задание: вывести шесть лучших игроков (голы + передачи) с указанием их результативности. Вывести всех игроков, имеющих в активе красные карточки.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе, подраздел 1.2).

Приложение Б
(обязательное)
Образец титульного листа курсовой работы

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Основы конструирования программ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

РАЗРАБОТКА ПРОГРАММЫ УЧЕТА ТОВАРОВ НА СКЛАДЕ

Выполнил: *студент группы*______ *ФИО*

Проверил: *ФИО преподавателя*

Минск 20__

Список использованных источников

1. Навроцкий, А. А. Основы алгоритмизации и программирования в среде Visual C++ : учеб.-метод. пособие / А. А. Навроцкий. – Минск : БГУИР, 2014. – 160 с.

2. Шилдт, Г. С++ Базовый курс / Г. Шилдт ; пер. с англ. – 3-е изд. – М. : Изд. дом «Вильямс», 2015. – 624 с.

3. Макконнелл, С. Совершенный код. Мастер-класс / С. Макконнелл ; пер. с англ. – М. : Русская редакция, 2010. – 896 с.

4. Документация по Visual Studio [Электронный ресурс]. – Режим доступа : https://docs.microsoft.com/ru-ru/visualstudio/ide/?view=vs-2017.

5. Todd Hoff C++ Coding Standard [Электронный ресурс]. – Режим доступа : http://www.possibility.com/Cpp/c++_coding_standards.pdf.

6. Google C++ Style Guide [Электронный ресурс]. – Режим доступа : https://google.github.io/styleguide/cppguide.html.

7. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем.

8. Доманов, А.Т. Стандарт предприятия СТП 01-2017 / А. Т. Доманов, Н. И. Сорока. – Минск : БГУИР, 2017. – 169 с.

Св. план 2019, поз. 8

Учебное издание

Меженная Марина Михайловна

ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ. КУРСОВОЕ ПРОЕКТИРОВАНИЕ

ПОСОБИЕ

Редактор Е.И. Костина
Корректор Е.Н. Батурчик
Компьютерная правка, оригинал-макет

Подписано в печать. Формат. Бумага офсетная. Гарнитура «Таймс»
Отпечатано на ризографе.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о регистрации издателя, изготовителя,
распространителя печатных изданий №
№
ЛП №
220013, Минск, П.Бровки, 6