МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пермский государственный национальный исследовательский
университет»

УДК 004.42

Кафедра математического обеспечения
вычислительных систем

Генерация исходного кода программ по высокоуровневой спецификации задачи

Выпускная квалификационная работа

Работу выполнила студентка группы
ПМИ-1,2-2014 НБ 4 курса механико-
математического факультета
Хакимуллина Анжелика
Альвертовна
«___» _____2018 г.
Научный руководитель:
к.ф.-м.н., доцент кафедры МОВС
Дацун Наталья Николаевна
«___» _____2018 г.

Пермь 2018

Аннотация

Работа посвящена разработке генератора исходного кода на нескольких языках по высокоуровневой спецификации. Выделены общие требования к системе, проведен анализ существующих средств. Произведен анализ и выбор типа приложения, инструментов пользовательского интерфейса, методов генерации. Спроектирована архитектура распределенной системы, описаны особенности реализации внутреннего представления алгоритмов. Результат работы – разработанное веб-приложение, которое обеспечивает доступ к построению диаграммы, вид которой удовлетворяет методологии IDEF0, и генерация исходного кода по полученной диаграмме на трех различных языках программирования.

Работа состоит из 74 страниц основного текста, 7 таблиц, 12 иллюстраций и 5 приложений.

Оглавление

Введение

В последнее время интерес к программированию значительно вырос. Становится популярным изучение различных языков и парадигм программирования. В связи этим можно наблюдать множество ресурсов, предлагающих курсы по программированию.

Курсы крупных компаний нацелены на изучение конкретного языка или конкретной парадигмы. Зачастую курсы могут ранжироваться по уровню пользователя: людям, только начинающим интересоваться это темой, предлагается изучить основы, людям, уже знакомым с программированием, сразу предлагается ознакомление с конкретным языком или технологией. В основном подача материала осуществляется в виде лекций. Они могут быть в печатном виде, где видна структура темы, демонстрация примеров кода. Могут быть в устной, где либо предоставляется запись лекции, либо преподаватель присутствует онлайн и имеется возможность задавать ему вопросы. Однако уже было доказано, что лекции являются не самым эффективным способом обучения [1].

Некоторые из ресурсов предлагают интерактивные задания на протяжении изучения лекций, где необходимо вписать код, который перед этим демонстрировался. Но в процессе обучения на таких курсах не формируется навык написания хорошо структурированных программ. Также в них обычно для воспитания культуры кода не предусмотрены какие-либо обучающие материалы. В качестве книг и статей существуют материалы [2] которые не привязаны к языку программированию, но диктуют сильные требования, которым должны подчиняться все языки. Также существуют материалы, обычно опирающиеся на конкретный язык [3]. Но знание лишь конструкций и синтаксиса языка не гарантирует грамотное, структурированное написание кода. Программа может быть написана правильно, но при этом быть совершенно нечеловекочитаемой.

Разработка приложения, демонстрирующего код на разных языках, но с единой структурой алгоритмов и программы в целом, может решить данную проблему.

Система, помимо отображения кода, должна иметь инструмент модульного программирования, с помощью которого алгоритмы представляются в виде блоков, и их можно собирать в необходимом порядке. В качестве представления взята высокоуровневая спецификация IDEF0, по которой должен генерироваться исходный код на выбранной структуре данных и языке. В рамках разработки исследовательского прототипа, демонстрация кода осуществляется на трех языках: Си, Паскаль и Пролог. Отобраны алгоритмы для базовых структур данных, а именно нахождение максимума, минимума, суммы и количества элементов, среднего арифметического, сортировка. Были выбраны именно эти алгоритмы, так как они являются фундаментальными и могут использоваться в различных статистических задачах. Таким образом, можно сказать, что они нацелены на практическое применение.

Несмотря на то, что языка программирования только три, в них представлены две отличных друг от друга парадигмы программирования, а именно процедурная и логическая. Также однотипный подход к отображению алгоритмов позволяет обеспечивать расширение системы как в сторону увеличения числа языков, так и структур данных, а также алгоритмов.

Использование системы позволяет продемонстрировать, что оперируя разными парадигмами, используя разные структуры данных, можно выделить одинаковый подход. И хорошо владея знаниями в одной области, не возникает трудности с их применением в другой.

Объектом исследования работы является задача демонстрации кода по высокоуровневой спецификации. *Предметом исследования* является

разработка генератора исходного кода программ по высокоуровневой спецификации.

Целью данной работы является реализация генератора исходного кода программ по высокоуровневой спецификации.

Система должна выполнять следующие функции: осуществление выбора языка программирования, в зависимости от него выбор структуры данных, выбор алгоритма из имеющихся для соответствующей структуры, предоставлять работу с высокоуровневой спецификацией, выполнять генерацию исходного кода по ней. В пользовательском представлении должна быть возможность добавления и удаление блоков, связывания блоков по данным, именование данных.

Для достижения поставленной цели необходимо решить следующие *задачи*:

1) провести анализ и формализацию требований к разработке системы генерации исходного кода по высокоуровневому представлению:
    a) определить модель пользовательского представления;
    b) формализовать сценарий работы с системой;
    c) конкретизировать требования к системе генерации исходного кода;
    d) проанализировать несколько доступных систем, способных осуществлять генерацию исходного кода по высокоуровневому представлению;
    e) формализовать выявленные требования;
2) выполнить проектирование:
    a) алгоритмов, которые будут генерироваться;
    b) спроектировать структуры системы генерации исходного кода;
    c) определить технологии реализации пользовательского представления
    d) определить технологии реализации внутреннего представления;

e) выбрать методы решения задачи;

3) реализовать исследовательский прототип системы генерации:

f) реализовать архитектуру системы;

g) реализовать построение диаграммы и генерацию кода по ней;

h) реализовать интерфейс;

4) протестировать реализованную систему по принципу белого и черного ящиков.

В ходе решения поставленных задач предполагается использовать следующие методы исследования: сравнительный анализ существующих решений, синтез требований к разрабатываемой системе, сравнительный анализ инструментов и технологий разработки веб-программирования, модульное программирование.

Практическая значимость заключается в применении данного средства в процессе обучения программированию как совместно с преподавателем, так и самостоятельно.

Результатом является приложение, включающее в себя создание программы с помощью высокоуровневого представления – диаграммы с блоками. Возможность адаптации системы показана на примере трех языков программирования, пяти структур данных и семи алгоритмов для каждой структуры.

Глава 1. Анализ и формализация требований к разработке системы генерации исходного кода по высокоуровневому представлению

Данная глава посвящена анализу предметной области, который состоит из этапов:

1) определение спецификации пользовательского представления;
2) описание сценариев взаимодействия с системой;
3) определение требований к системе (результат отражен в техническом задании);
4) исследование нескольких доступных систем, способных генерировать исходный код по высокоуровневому представлению;
5) формальная постановка задачи.

1.1 Определение модели пользовательского представления

Алгоритм – последовательность четких указаний, предписывающих выполнение тех или иных действий в строго определенном порядке.

Для описания алгоритмов существуют различные способы:

1) словесное описание;
2) с помощью псевдокода;
3) графическое описание.

Словесное описание алгоритма представляет собой описание структуры алгоритма при помощи естественно языка. Примерами могут служить описание инструкций по эксплуатации каких-либо приборов. Такое описание не имеет единого стандарта, так как может быть представлено на любом естественном языке, с помощью разных конструкций, разных слов.

Приведем пример словесного описания нахождения максимума:

Шаг 1. Назначить максимальным элементом первый элемент структуры.

Шаг 2. Взять следующий элемент структуры.

Шаг 3. Сравнить текущий элемент с максимальным.

Шаг 4. Если максимальный элемент меньше текущего, назначить максимальным текущий элемент.

Шаг 5. Если есть еще элементы, вернуться на шаг 2, иначе вывести максимальный элемент.

Не смотря на то, что с помощью такого способа можно описать любые алгоритмы, он достаточно многословен и у него отсутствует наглядность. Данная инструкция имеет место в человеческой жизни, но её использование для обучения общения с машиной является нецелесообразным и трудозатратным.

Описание с помощью **псевдокода** более приближено к текущей задаче. Псевдокод – это описание структуры алгоритма на формализованном естественном языке. Строгих синтаксических правил для записи не предусмотрено, но о них можно договориться в рамках одного проекта.

Пример описания с помощью псевдокода:

```
начало
max = 1-ый элемент
цикл по элементам i = 2
        если max > i-ый элемент, то
                max = i-ый элемент
вывод max
конец алгоритма
```

Ключевым недостатком такого представления является то, что на изучение конструкций псевдокода потребуется дополнительное время.

Графическое описание алгоритма – это способ представление алгоритма с помощью общепринятых графических фигур. Такое представление является наиболее наглядным из всех и строгим. Оно может быть представлено разными способами, рассмотрим их подробней.

1.1.1 Блок-схема

Последовательность блоков и линий, где внутри блоков описываются команды или условия, – называется блок-схемой. Блок-схема отображает инструкции на конкретном языке и на конкретной структуре данных. Её использование в качестве представления пользователя является неподходящим, потому что тогда обучающийся уже должен знать, как выглядит алгоритм.

Описание с помощью блок-схемы продемонстрировано на рисунке 1.1.

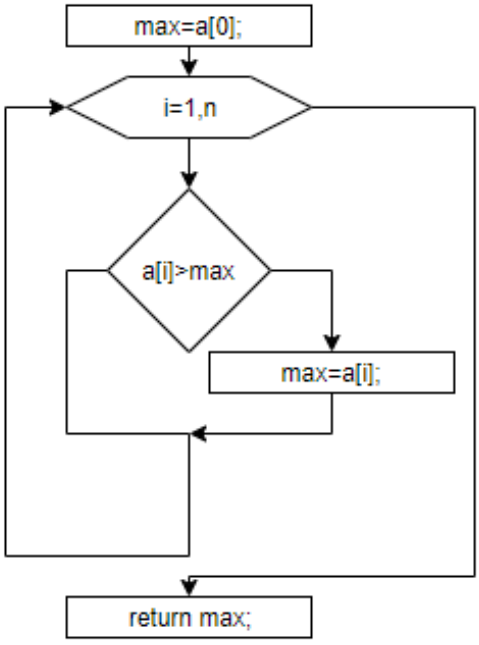


Рис. 1.1 Блок-схема нахождения максимума

1.1.2 Граф переходов

Данное описание удобно использовать для алгоритмов, где происходит смена состояний, логические операции, операции приближенные к машинным командам, что накладывает некоторые ограничения на описания алгоритмов. Граф переходов содержит в вершинах описание переменных, который в действительности у алгоритма может оказаться много и при совмещении нескольких алгоритмов картина окажется достаточно

громоздкой. Главным их недостатком для использования данной системы является то, что они трудно понимаемы человеком.

1.1.3 Методология IDEF0

«SADT - (аббревиатура выражения Structured Analysis and Design Technique - методология структурного анализа и проектирования) - это методология, разработанная специально для того, чтобы облегчить описание и понимание искусственных систем, попадающих в разряд средней сложности» [4].

С помощью методологии SADT можно получить полное и точное описание системы. Целью модели называется конкретное назначение, которое имеет система. Оно следует из формального определения модели в SADT:

«M есть модель системы S, если M может быть использована для получения ответов на вопросы относительно S с точностью A» [4].

В данной работе рассматривается IDEF0-модель, являющаяся подмножеством SADT-модели и используемая для создания функциональной модели. Под функциональной моделью понимается структурированное изображение функций производственной системы (среды). Функции могут быть связаны при помощи информации и объектов, которые также включены в функциональную модель [5].

IDEF0-методология описывает функции графически с помощью блочного моделирования. IDEF0-диаграммы отображают операцию в виде блока, а интерфейсы входа/выхода в/из операции представляются дугами, соответственно входящими в блок или выходящими из него (см. Рис. 1.2) [5].

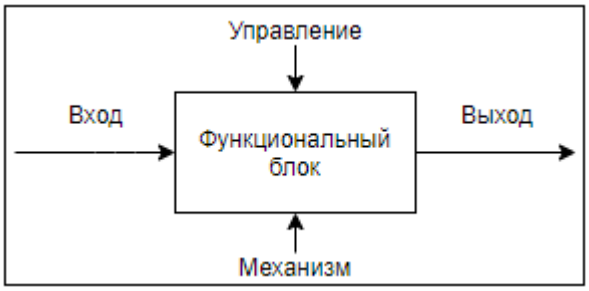


Рис. 1.2 Функциональный блок IDEF0

В качестве управления могут выступать условия, накладываемые на функцию, например условие выполнения алгоритма только для положительных элементов. Под механизмом в данном случае понимается структура, на которой выполняется функция, но в данном случае сама структура данных подается на вход функциям, поэтому эту связь можно опустить.

Блоки и дуги в IDEF0-модели используются для представления связей между несколькими функциями на диаграмме (см. рис. 1.3) [5].

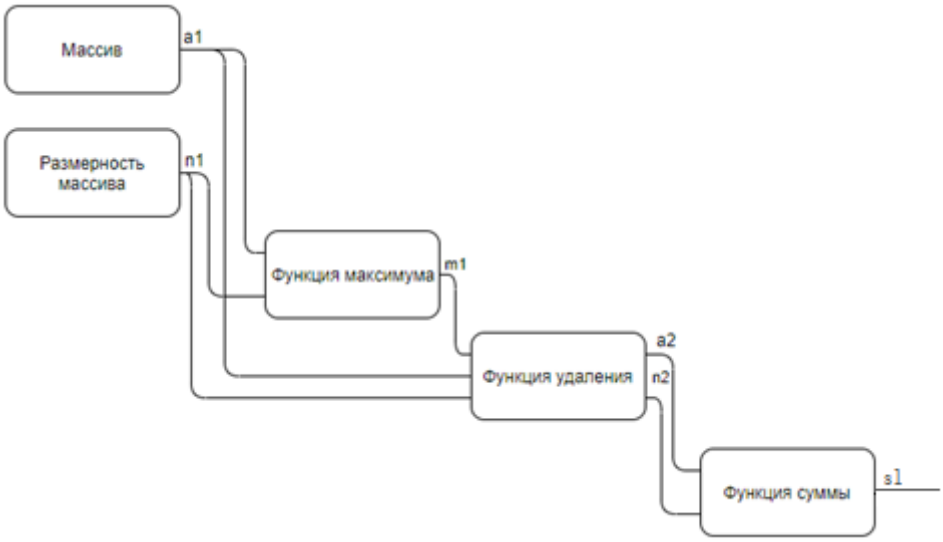


Рис. 1.3 Связи между блоками выполняемых функций

При помощи диаграмм SADT можно совершать развертку функций, т.е. производить их постепенную детализацию (см. рис. 1.4). В результате может быть получена иерархия функций. Так первый блок диаграммы A0 может соответствовать нескольким блокам на дочерней диаграмме A3. Номер дочерней диаграммы говорит о том, какой блок детализирован [6].

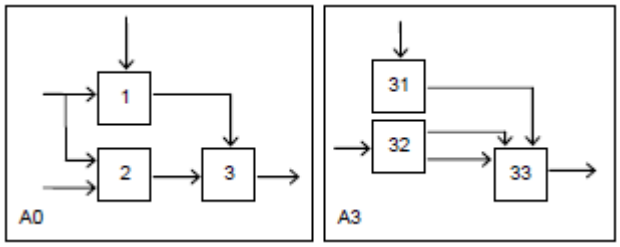


Рис. 1.4 Детализация функций на дочерних диаграммах

Главной особенностью моделей, описанных с помощью IDEF0, является то, что они могут быть применены в абсолютно любых областях деятельности человека. В основном IDEF0 используется для описания бизнес-процессов, но это не является ограничением. Такие модели обладают наглядностью и понятны людям, которые даже не являются ИТ-специалистами.

1.2 Описание процесса взаимодействия с системой

Для описания и анализа функциональных требований к системе были использованы прецеденты. Основная идея прецедентов заключается в выявлении и описании ситуаций, когда пользователи взаимодействуют с системой [6].

Определим границы системы, в нее входят: набор алгоритмов и приложение, которое занимается выдачей результата. Пользователи системы будут называться акторами.

При анализе были выделены прецеденты. Среди них были выделены главные, второстепенные и дополнительные прецеденты. Главные прецеденты были описаны в развёрнутой форме.

Название: Генерация исходного кода.

Акторы: Пользователь.

Описание: Пользователь выбирает структуру данных и алгоритм, после чего в диаграмму добавляется новый блок. По построенной диаграмме приложение генерирует исходный код.

Основной поток:

Действия акторов	Отклик системы
Пользователь выбирает нужные параметры. (E1)	Система заносит блоки в диаграмму
Пользователь работает с диаграммой, выполняется подпоток S1	Система изменяет список блоков
Пользователь запускает генерацию (E2)	Система генерирует код по составленной диаграмме

Подпотоки:

S1: пользователь выбирает алгоритм, блок размещается на диаграмме, устанавливает между блоками связи, именует параметры.

Альтернативные потоки:

E1: Очистка диаграммы при выборе другой структуры данных или языка.

E2: Очистка диаграммы. Пользователь решает очистить диаграмму. Выбранные параметры возвращаются в значение по умолчанию, прецедент завершается.

Диаграмма прецедентов представлена на рис. 1.5.

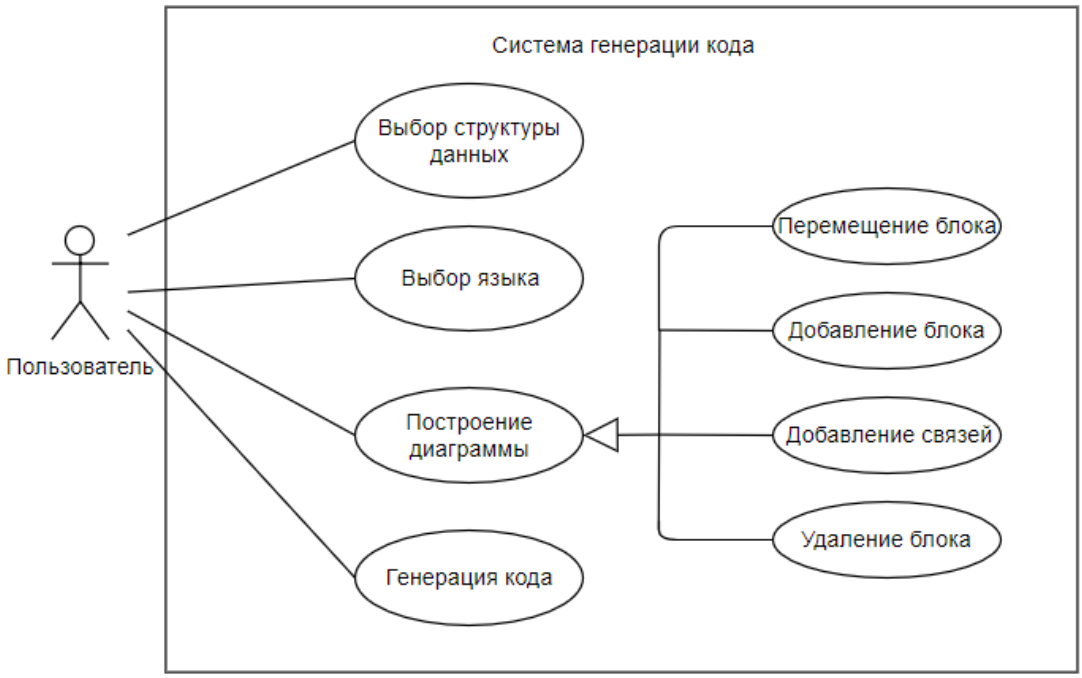


Рис. 1.5 Диаграмма прецедентов

Для завершения анализа смоделируем поведение системы, т.е. построим модель, где опишем бизнес-процессы. Для этого построим диаграмму активностей (см. рис. 1.6), которая также описывает алгоритм выполняемых действий.

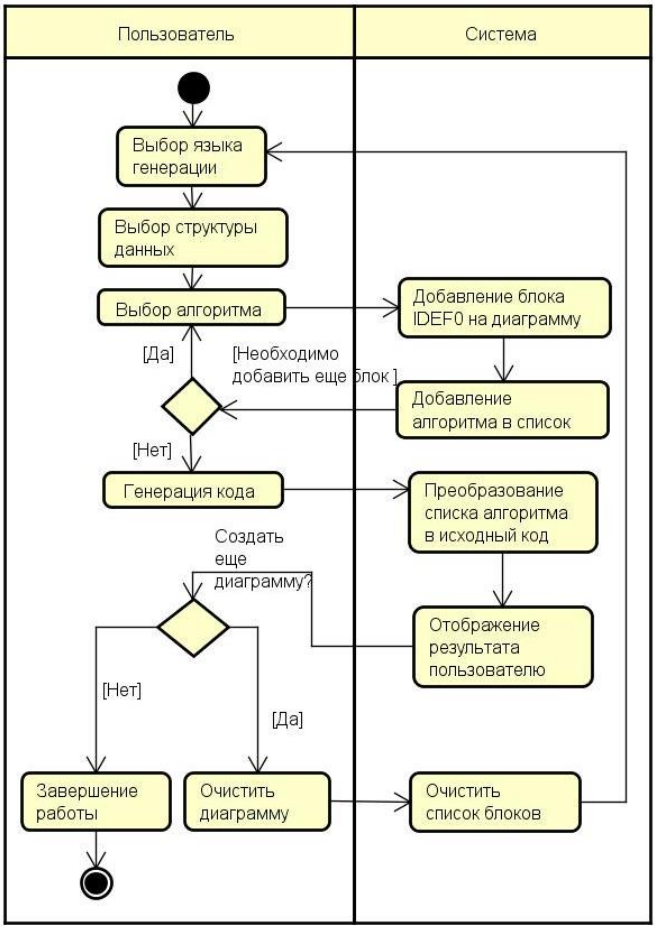


Рис. 1.6 Диаграмма активностей

На рис. 1.7 приведено описание операции «сгенерировать код».

Имя: сгенерироватьКод(списокБлоков).
Обязанности: генерация кода по списку заданных на диаграмме блоков.
Ссылки: прецедент «Генерация кода»
Примечание: проверять соответствие связей между блоками.
Исключение: Если список блоков пуст, не генерировать.
Предусловия: Выбраны блоки, составлена диаграмма.
Постусловия:

1. Создан объект :Генератор связанный с объектом :Диаграмма
2. Создан объект :Результат
3. Атрибуты объекта :Результат заданы

Рис. 1.7 Описание операции «Сгенерировать код»

1.3 Описание требований к работе системы

Прежде всего, стоит описать целевую группу, для которой данная система предназначена. Ей являются школьники, студенты, изучающие программирование. То есть люди уже знакомы с основами и должны углубить свои знания в области написания структурного кода. Тем не менее, важно, чтобы средство обучения было понятно людям вне зависимости от их уровня компетенции пользователя в сфере IT. Его графическое представление должно быть легко читаемым и четко понимаемым. Этим требованиям удовлетворяет методология IDEF0.

Основные возможности, которые должны предоставляться системой:

1) указание целевого языка программирования для генерируемого кода;
2) выбор структуры данных;
3) работа с диаграммой:
4) добавление блока на диаграмму (выбор из алгоритмов соответствующей структуры данных);
5) связывание блоков по входным и выходным параметрам;
6) редактирование блока: изменение имен параметров и удаление блока;
7) генерация исходного кода на различные языки программирования в соответствии с диаграммой на выбранном языке.

1.4 Анализ действующих решений

Идея генерация кода по высокоуровневой модели давно привлекает человеческий ум. Попыток создания таких генераторов достаточно, многие из них создаются на коммерческой основе. Рассмотрим несколько их них.

1.4.1 MetaAuto

UML-диаграммы используются в программировании повсеместно: как при проектировании будущих систем, так и для описания существующих. Они прекрасно представляют функциональные требования и статическое

структурное представление. Но моделирование динамических аспектов программы на языке UML затруднено потому как в диаграмме классов визуализируется лишь статичная картина работы программы. В связи с чем разработчики инструментального средства MetaAuto, используют в качестве графического представления – граф переходов. Его отображение осуществляется с помощью редактора MS Visio. Данное средство преобразует граф в исходные коды программ на любых языках программирования, для которых предварительно созданы соответствующие шаблоны. В качестве внутреннего представления MetaAuto использует xml-формат и XSLT-преобразование [7].

Граф переходов хорош тем, что позволяет отражать динамические аспекты программы. Но само построение графа, его понимание может занимать отдельный курс по его изучению. Таким образом, существенным недостатком является то, что понимание графа переходов предъявляет высокий уровень к компетенции пользователя.

1.4.2 Flexberry Designer

«Flexberry Designer - инструментарий проектирования, который представляет собой CASE-инструмент и состоит из UML-редактора Flexberry Designer и модуля расширения «Flexberry ORM», предназначенным для генерации C#-кода и SQL. Данный инструментарий позволяет по диаграмме классов сгенерировать C#-классы объектов данных и БД. Проектирование с использованием данного инструментария подразумевает принцип Model-First, когда все изменения в модели производятся в CASE, а изменения в коде выполняются во время генерации. Также реализован механизм, позволяющий программистам писать код, который при перегенерации не будет потерян, так называемые, «скобки программиста». Использование инструментария проектирования позволяет с лёгкостью вносить изменения в модель любому участнику команды разработки, не опасаясь что-нибудь сломать» [8].

Продукт Flexberry Designer направлен на то, чтобы уменьшить количество затрачиваемого времени на разработку рутинных вещей, таких как создание форм, привязка к базам данных, создание простейших обработчиков событий и выполнено это для языка C#.

Данное средство обладает многими преимуществами. В первую очередь у него имеется высокоуровневый редактор. Также возможна интеграция с системами проектного управления. Есть множество готовых компонент для уменьшения времени, которое тратится на рутинную разработку. Доступна дальнейшая поддержка кода после генерации. Дополнительно есть совместимость с различными СУБД и многое другое.

В текущей же работе разрабатываемая программа направлена на процесс обучения, в котором хочется показать разнообразие языков и подходов к программированию, поэтому генерация только в один язык является, в данном случае, недостатком.

1.4.3 QReal

«Средство QReal — это кроссплатформенный свободно распространяемый под лицензией GNU GPL инструмент с открытым исходным кодом, предназначенный для создания специализированных сред визуального программирования» [9].

Разработчики Qreal используют технологию метамоделирования. Так изначально графически описываются метаязыки представления элементов на диаграмме классов (средствами встроенного в QReal графического редактора векторных изображений), но на физическом уровне метамодель хранится в виде XMI-описания. Генерация кода осуществляется в язык программирования C++. Метаязык компилируется в динамическую библиотеку – подключаемый модуль. Для быстрого создания трансляторов визуальных диаграмм в исходный код на целевом языке в QReal используется специальный язык описания генераторов, который позволяет

для визуального языка задать правила обхода созданных с его помощью моделей и генерации кода по ним. В дальнейшем эти правила интерпретируются для конкретных диаграмм, порождая соответствующий им исходный код.

В 2010 году в Qreal была добавлена возможность генерации исходного кода на C# по UML-диаграмме активностей [10]. Была выбрана именно эта диаграмма в виду того, что она демонстрирует динамические аспекты программы. При попытках генерации кода из данной диаграммы может возникнуть нетривиальная проблема ее неструктурированности. Под неструктурированной понимается диаграмма, не отображающаяся напрямую на структурные конструкции текстовых языков программирования. Также если использовать диаграмму активностей, то возникают проблема неоднозначности кода. Всё это является существенными недостатками для генерации.

1.4.4 MetaLanguage

Разработкой системы MetaLanguage занимается Высшая школа экономики города Перми [11]. Эта система специализируется на разработке визуальных предметно-ориентированных языков моделирования. Первым этапом разработки является создание конструкций метаязыка с помощью средства визуализации MindFusion, задание отношений между ними и задание каких-либо ограничений на объекты метамодели. Когда метаязык создан, можно приступить к созданию моделей. Модели могут создаваться только с помощью тех конструкций, которые имеются в основе метаязыка.

Разработанные модели могут быть преобразованы в любой желаемый язык программирования, при условии описания правил трансформации модель-текст. В процессе написания данных правил необходимо знать и учитывать семантику результирующего языка. Там, где это нужно

описывается вставка имен объектов модели и/или их свойств, которые подставляются в результате трансформации.

Данный подход удобен, имеет ряд преимуществ, в том числе возможность генерации в любой язык. Однако, графическое представление, которое присутствует в данной системе, не очень наглядно, поскольку не указывает связи параметров между узлами на диаграмме.

1.4.5 SciVi

Данная система разрабатывается в Пермском государственном национально-исследовательском университете. Она является универсальной мультиплатформенной клиент-серверной системой научной визуализации [12]. Но в данном случае необходимо заострить внимание на том, как реализована фильтрация данной системы. Для этого используется диаграмма потока данных, вершины которой соответствуют фильтрам, а дуги – связям по данным. Количество входных и выходных параметров может отличаться у разных вершин, в зависимости от того, какой фильтр они представляют собой.

Добавление самих вершин, точнее, их разнообразие определяется количеством онтологий описания фильтров. В каждой из них хранится описание роли фильтра, множества входных и выходных данных фильтра, множество допустимых полей и настроек фильтра, реализация фильтра в виде программного кода.

Онтологический подход к описанию и хранению фильтров позволяет расширять их набор без модификации ранее отлаженного программного кода клиента и сервера, легко адаптируя систему визуализации к новым задачам [12]. Данный подход достаточно близок, к реализуемому в данной работе, что непосредственно будет учтено в дальнейшей разработке. Недостатком является лишь то, что по данной диаграмме потока данных нет генерации.

1.4.6 Результат обзора

В результате анализа близких к разрабатываемому средству аналогов, была составлена сравнительная таблица 1.1. В качестве критериев были взяты: используемое пользовательское представление, внутреннее программное представление, язык программирования, в который идет генерация. А также перечислены преимущества и недостатки.

Таблица 1.1 Инструментальные средства для генерации исходного кода

Критерии	**MetaAuto**	**Flexberry**	**QReal**	**MetaLanguage**	**Подзадача системы SciVi**
Пользовательское представление	Граф переходов	Диаграмма классов	Диаграмма активностей	Граф «сущность-связь»	Диаграмма потока данных
Внутреннее представление	XML-формат	Объектная метамодель (C#), JSON (JavaScript)	Графовое представление	Объектная метамодель	Онтология
Генерируемый язык	Любой	C# и JavaScript	C#	Любой	Нет
Преимущества	Возможность генерации в любой язык.	Возможность увеличения числа языков генерации	Генерация кода программы, отражающего динамические аспекты системы	Возможность генерации в любой язык	Расширяемость и гибкость, благодаря описанию с помощью онтологий
Недостатки	Высокий уровень требований к уровню компетенции пользователя	Коммерческий продукт	Неструктурированность диаграммы, неоднозначность генерации	Требуется эксперт для создания метаязыка	Нет генерации

Требуемая система должна обладать понятным интерфейсом для людей, начинающих изучать программирование. Все перечисленные пользовательские представления уступают ранее описанной методологии IDEF0. Потому что она зарекомендовала себя, как достаточно наглядную диаграмму во многих областях.

1.5 Постановка задачи

1) Исходные данные:

nA: целое {количество алгоритмов}

A: множество, A $= \{a_i\}, i \in [1, nA]$ {набор алгоритмов}

a_i: максимум, минимум, среднее, сумма, количество, удаление, сортировка

nSd: целое {количество структур данных}

Sd: множество, $Sd = \{sd_i\}$, где $i \in [1, nSd]$ {структуры данных }

Sd ={ массив, список, строка, текстовый файл, бинарный файл}

nGl: целое { количество целевых языков}

Gl: множество, $Gl = \{gl_i\}$, где $i \in [1, nGl]$ {целевые языков}

Gl ={ c, pascal, prolog}

nIN: целое {количество входных параметров}

$In_i = \{in_j^k\}$, где $j \in [1, nA], k \in [1, nSd]$, $i \in [1, nIN]$ {входные параметры блока}

nOUT: целое {количество выходных параметров}

$Out_i = \{out_j^k\}$, где $j \in [1, nA], k \in [1, nSd]$, $i \in [0, nOUT]$ {выходные параметры блока}

$V_i =< a_j, sd_k, gl_l, In_i, Out_j^k >$ {блок спецификации IDEF0}

$L = \{l_j^k\}$, где $j \in [1, nA], k \in [1, nSd]$; {библиотека кодов функций}

2) Ограничения:

nA > 0, nSd > 0, nGl > 0, nIN > 0, nOUT ≥ 0,

$A \neq \emptyset, Sd \neq \emptyset, Gl \neq \emptyset, V \neq \emptyset, L \neq \emptyset$

3) Результаты:

Sc: файл, $Sc = l_j^k \times a_n \times sd_m \times < a_n.\text{имя}, In_n, Out_n >, j = n, k = m$ {исходный код}

4) Связь:

$\exists$ i $\in [1, \infty), j \in [0, nOUT], k \in [1, \infty), m \in [1, nIN], i \neq k : V_i.out_j = V_k.in_m$

Глава 2. Проектирование системы

Данная глава содержит следующие необходимые этапы для проектирования системы:

1) написание алгоритмов, выявление единого стиля, структуры;
2) определена общая структура системы;
3) выполнен анализ и выбор технологий разработки пользовательского интерфейса;
4) обоснован выбор типа приложения;
5) выбраны методы решения задачи генерации исходного кода программы по высокоуровневой спецификации.

2.1 Построение алгоритмов

Зачастую знакомство с программированием начинается с языков Си и Паскаль, которые являются языками процедурной парадигмы. Поэтому эти языки были выбраны в качестве целевых для генерации программного кода. Кроме них, был выбран язык программирования Пролог, представляющий логическую парадигму. В качестве структур данных были выбраны массив, строка, список, текстовый и бинарный файл, так как они являются базовыми при изучении программирования. Для демонстрации работы со структурами выбраны следующие алгоритмы для работы со структурами: нахождение максимального, минимального элемента, суммы, количества элементов, среднего арифметического, выполнение удаления элемента и сортировки. Алгоритмы выбраны не случайно, так как могут использоваться в статистических задачах, таким образом, направлены на практическое применение.

В качестве отображения алгоритмов на диаграмму ранее была выбрана методология IDEF0, поэтому структура алгоритмов построена определенным образом. Возврат результата работы алгоритма выполняется через параметр

функции, что демонстрирует передачу параметров по ссылке, а также четкое разделение на входные и выходные параметры.

Для одного алгоритма, выполняемого на всех структурах, выделялись инварианты. Поэтому нетрудно пронаблюдать структуру алгоритма, которая не меняется при переходе из одной структуры в другую. Помимо этого, алгоритмы для разных языков одной структуры данных также имеют схожий каркас. Таким образом, можно пронаблюдать, что понимая построение алгоритма для одного языка, одной структуры, осуществимо применение этих знаний для работы в ином языке, иной структуре. Сравнение одного алгоритма на разных языках представлено в таблице 2.1

Таблица 2.1 Сравнение алгоритма на разных языках и структурах

Нахождение максимума на языке Си в текстовом файле	**Нахождение максимума на языке Пролог в списке**	**Нахождение максимума на языке Паскаль в массиве**

`void max(FILE *file, int &max)`
`{`
 	
`fseek(file,0L, SEEK_SET);`
`fscanf(file, "%d", &max);`
 	
`int num;`
`while (!feof(file))`
`{`
`fscanf(file, "%d", &num);`
`if (num > max)`
`max = num;`
 	
`}`
`}`
	
	`max([X], X).`
	`max([X, Y | T], Max) :-`
	`X <= Y,`
	`max([Y | T], Max).`
	`max([X, Y | T], Max) :-`
	`X > Y,`
	`max([X | T], Max).`
		`Procedure Max(a:array of integer; n:integer; var max : integer);`
		`var i:integer;`
		`begin`
		`max:=a[0];`
		
		`i:=1;`
		`while (i < n) do`
		`begin`
		
		`if (a[i] > max) then max:=a[i];`
		`inc(i);`
		`end;`
		`end;`

Все алгоритмы можно вызывать последовательно друг за другом в любом порядке – это демонстрирует четкое распределение обязанностей, что также направлено на проявление структурированности кода. Описания алгоритмов имеют такой вид не случайно, оно выдержаны в едином стиле:

1) каждый оператор выполняется в новой строке;
2) равные отступы от начала строки;

3) имена алгоритмов одинаковы для всех структур и языков;

4) сначала перечисляются входные параметры, а затем выходные.

Советы по оформлению кода были подчерпнуты из материала [13]. В целом в системе используется 77 алгоритмов, которые описаны в приложении В.

2.2 Проектирование структуры системы

В данной системе можно выделить два технологических аспекта: построение диаграммы в рамках нотации IDEF0 и генерация кода по ней. Прежде всего, пользователь должен четко обозначить с каким языком и какой структурой данных он работает, потому что именно от этих двух параметров главным образом зависит исходный код алгоритмов.

После того, как эти два параметра зафиксированы, может быть сформировано меню выбора алгоритмов, которые могут быть добавлены на диаграмму. Один и тот же алгоритм для разных структур и разных языков может отличаться числом входных и/или выходных данных, поэтому при схеме языка или структуры происходит очистка диаграммы и вновь формирование меню редактора.

По окончании построения диаграммы пользователь вызывает генерацию кода. Помимо списка алгоритмов, которые были добавлены на диаграмму, для генерации понадобятся зафиксированные параметры в начале, для того чтобы выбрать из библиотеки алгоритмов соответствующие описание. Происходит компоновка описания функций и генерация вызовов этих функций. Полученный результат отображается пользователю на экран. Модель описанной системы можно наблюдать на рис. 2.1.

Для построения системы было решено применить модульную архитектуру, где каждый модуль отвечает за решение какой-то подзадачи. В данном случае можно выделить модули способные выполнить свою

функцию самостоятельно без помощи остальных модулей, лишь на основе входящих данных. В число отобранных частей, входят следующие модули:

1) редактор диаграммы;
2) генератор кода;
3) библиотека алгоритмов.

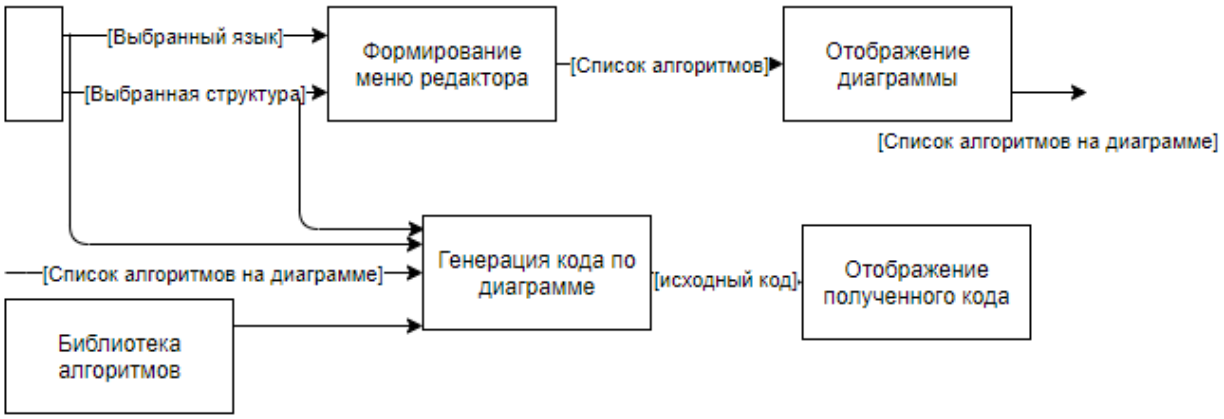


Рис. 2.1 Модель системы генерации кода по диаграмме

При таком подходе возможно пополнение содержимого библиотеки, при добавлении нового языка программирования и соответственно новых алгоритмов. Параллельно с этим допустимо изменение изображение блока на диаграмме, например, удаление или добавление числа параметров, то есть линий связанных с блоком. В то же время имеется возможность дополнения генератора кода, в связи с добавлением алгоритмов.

2.3 Технологии пользовательского представления

В человеко-машинном взаимодействии с разработкой системы форма диалога управляемая пользователем, так как именно он определяет, какие алгоритмы будут в итоговой программе, какова будет последовательность выполнения алгоритмов, как они будут связаны друг с другом, а также какие будут имена параметров [14].

Конкретно наиболее уместной будет объектно-ориентированная форма диалога, в которой используется меню, окна и пиктограммы. В данном случае пиктограммы – это блоки диаграммы, меню – выбор языка и

структуры данных, окна содержат диаграмму вместе с меню и вывод исходного кода.

Требуется обратить внимание на то, что при создании диаграммы необходимо решать задачу построения графа. В эту задачу включены:

1) возможность изменения положения блока, относительно других;
2) построение дуги между двумя точками с заданным радиусом кривизны;

В работе [15] с похожей задачей, в качестве средства создания пользовательского интерфейса используется библиотека Qt. С помощью неё действительно возможно создание интерфейса при помощи графики, однако, в данной работе задача построения не является предметом исследования. Поэтому было принято решение воспользоваться готовым продуктом.

На данный момент существует несколько редакторов узлов графа (node editor), где под узлом понимается прямоугольник с обозначениями входных и выходных параметров. В таких редакторах возможно связывание узлов с помощью параметров, а некоторые даже предоставляют несложную логику вычислений.

2.3.1 D3 Node Editor

D3 Node Editor [16] - это JavaScript библиотека для создания редакторов узлов для решения задач визуального программирования. Она имеет поддержку актуальных версий и является наиболее скачиваемой на данный момент. Библиотека полностью берет на себя работу по отображению и обработке узлов. Использует D3.js и Angular Light. Название d3 расшифровывается как data driven document[17]. Это тоже JavaScript библиотека, ориентированная на работу с данными и их визуальное представление для веба, включая загрузку данных, визуализацию в режиме реального времени и другие возможности.

Компоненты, входящие в библиотеку:

1) socket позволяет описать тип параметра, что позволяет запретить соединение параметров разных типов;
2) control является полем ввода для приема данных;
3) node – объект, представляющий собой прямоугольник на диаграмме с входами и выходами.
4) engine – это компонент, с помощью которого может быть реализована логика вычислений для данных из control.
5) editor – объект, управляющий node для отображения и взаимодействия с пользователем.

Данная библиотека позволяет менять стилистику узлов, заниматься обработкой данных, содержащихся в узлах. Она является открытой, что позволяет без каких-либо трудностей внести в неё необходимые свойства в узлы, которые могут отсутствовать в стандартной поставке. Также библиотека имеет небольшую, но качественную документацию и используется для веб-приложений.

2.3.2 Node Editor Winforms

Node Editor Winforms – библиотека на языке C#, предоставляющая редактор узлов, а именно элемент управления, который может быть использован в WindowsForms приложении [18].

Редактор содержит интерфейс, через который можно описать блоки:

1) количество входных и выходных параметров;
2) типы параметров;
3) как связываются входные параметры с выходными, т.е. можно заложить некоторую логику вычислений.

В данном реакторе имеется поле, через которое можно изменять некоторые параметры узла, например значения входных параметров. Также он способен автоматически сформировать контекстное меню при обнаружении экземпляров класса, наследующего вышеприведенный

интерфейс. Элементы в редакторе можно перемещать, даже по несколько штук сразу, и соединять друг с другом. В качестве недостатка данной библиотеки выступает невозможность установление других типов параметров, кроме численного. Поэтому указать имена параметров, не представляется возможным. Данная библиотека не имеет документации, но является открытой для изменений.

Для выбора технологии построим сравнительную таблицу, в которой будут отображены критерии, необходимые для создания редактора.

Сравнение технологий представлено в таблице 2.2:

Таблица 2.2 Сравнение пользовательских представлений

Критерии	**D3 Node Editor**	**Node Editor Winforms**
Документация	+	-
Установка совместимых типов параметров	+	+
Установка имен для параметров	+	-
Открытый доступ	+	+
Установка параметра любого типа	+	-
Возможность перемещения объектов	+	+

В результате сравнения, с учетом имеющихся характеристик и выявленных требования, можно сделать вывод, что в данном случае более предпочтительно использовать библиотеку D3 Node Editor.

2.4 Обоснование выбора типа приложения

В данном случае стоит выбор между созданием настольным и веб-приложением. Создавая настольные приложения, пользователю предоставляется наиболее быстрый отклик от системы, так как данные не

передаются по сети. Также разработку настольных приложений не ограничивают возможности браузера и их ошибки. В тоже время нужно понимать, с какими проблемами можно столкнуться при развертывании приложения на машине у пользователя: различные версии операционной системы, установленные библиотеки, реестры и другое.

Если выбрать веб-разработку, то обновление становиться проще, так как не нужно принудительно проверять последнюю версию приложения на машине. Всё, что может потребоваться от пользователя это обновить страницу. Также можно гарантировать работу сайта на любом устройстве. Узким местом при использовании веб-приложений является наличие доступа в интернет для этого. Если в компьютерном классе сеть не будет обеспечена, то работать с обучающимися будет невозможно. Сравнение по выделенным критериям представлено в таблице 2.3

Таблица 2.3 Сравнение типов приложений

Критерии	**Веб-приложение**	**Настольное приложение**
Возможность использования на компьютерах разных ОС	+	-
Внесение изменений, без проблем для пользователей	+	-
Удаленный доступ	+	+
Использование вне доступа в интернет	-	+

Таким образом, можно сделать вывод о том, что стоит отдать предпочтение созданию веб-приложения.

2.5 Методы решения

Главной функцией обработки в проектируемой системе является генерация кода по диаграмме. От того, как будет производиться обход полученного графа, будет зависеть скорость ответа системы пользователю. В

результате построения диаграммы, от редактора можно получить список блоков добавленных на нее, с которым и предстоит работать. Каждый блок в этом списке имеет следующие свойства:

1) номер;
2) массив входных параметров;
3) массив выходных параметров;
4) имя функции.

Если параметр связан с параметром другого блока, то в массиве содержится объект, представляющий собой пару: номер блока, с которым связан и номер параметра в этом блоке. Возможно несколько способов генерации по полученному списку. В данном случае генерация заключается в методе обхода блоков на диаграмме.

Рекурсивный подход.

Создаем три списка: список всех блоков listAll, список выполненных блоков listComplete. Будем выполнять алгоритм для всех элементов списка listAll:

Здесь начинает выполняться рекурсивный алгоритм. Если ли текущий элемент не принадлежит списку listComplete, то проверяем, есть ли у блока входные параметры. Если их нет, то необходимо запустить генерацию вызова этого блока и положить этот блок в список выполненных.

Если входные параметры есть, то необходимо проверить соединены ли они с каким-либо выходом другого блока. Если нет, то это означает, что в диаграмме ошибка. В ином случае, если параметры соединены с блоками, принадлежащие списку listComplete, то сгенерировать код для текущего блока и добавить список listComplete, иначе запустить обработку для соединенных блоков.

Итерационный подход.

Данный метод также как и предыдущий, прежде всего, инициализирует список всех блоков listAll, список выполненных блоков listComplete. К этому

добавляется стэк текущих обрабатываемых блоков stackTemp. Метод продемонстрирован на рис. 2.2.

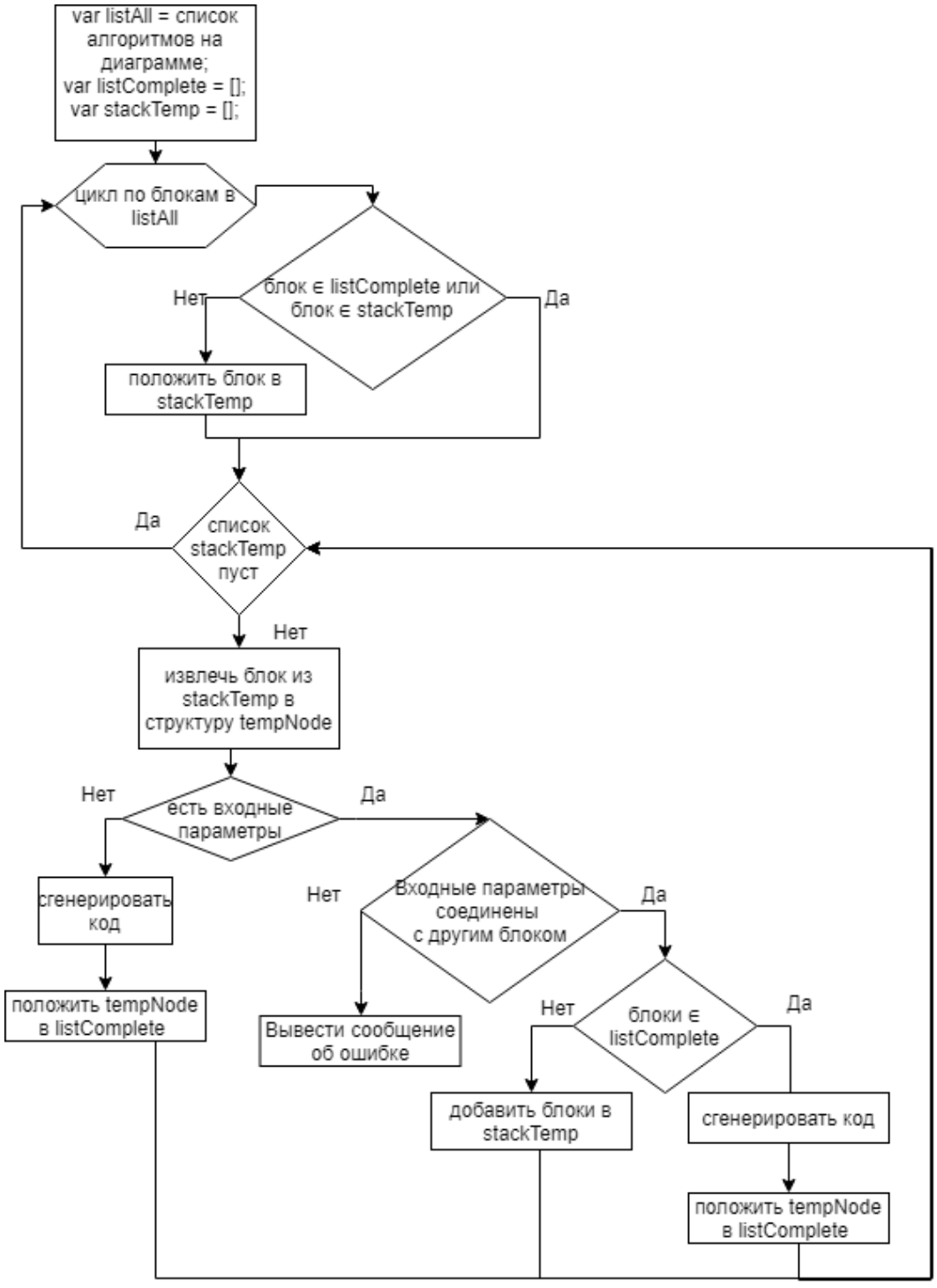


Рис. 2.2 Итерационный алгоритм обхода

В рекурсивном алгоритме у каждой переменной появляется вполне определённое назначение, которое не может быть изменено. Подход, когда данные управляют программой, даёт множество преимуществ, основными из

которых являются: лёгкость понимания кода и простота написания обработчиков ложных данных, независимость одних частей кода от других.

Но главным недостатком рекурсии является её расточительность. Она может быстро привести к переполнению стека, так как помимо запоминания блока, к которому необходимо вернуться, в памяти фиксируется состояния всех переменных во время вызова. Поэтому лучше создать стек, который будет хранить только те переменные, которые необходимо обработать, и применить итерационный алгоритм.

Глава 3. Реализация системы

Данная глава содержит описание реализованной архитектуры системы. Одной из задач, решаемой на данном этапе также является апробация системы на частном примере (получение исходного кода на трех языках программирования: Си, Паскаль, Пролог). Содержится реализация построения диаграммы и генерации кода по ней. Также обоснован выбор и разработан интерфейс пользователя.

3.1 Архитектура системы

Система должна состоять из нескольких компонент. Во-первых, это компонента описание алгоритмов, а именно файл, доступ к которому обеспечивается максимально быстро. Доступ к файловой системе может выполняться только на стороне сервера, однако такое обращение достаточно медленное и нагружает систему. Большинство алгоритмов имеют описание, помещающееся на экране без прокрутки, поэтому их можно сразу загружать вместе со всей системой.

Во-вторых, компонент библиотеки пользовательского представления. В-третьих, компонент, с помощью которого обеспечивается обращение к этой библиотеке и в целом работа с ней, т.е. описание блоков, которые будут отражены на диаграмме. Следующим, компонентом является описание пользовательского интерфейса, который дает возможность выбора языка, структуры, работы с диаграммой, просмотр кода. Последний компонент – работа по выполнению генерации по построенной диаграмме.

Архитектура разработанной системы приведена на рис. 3.1

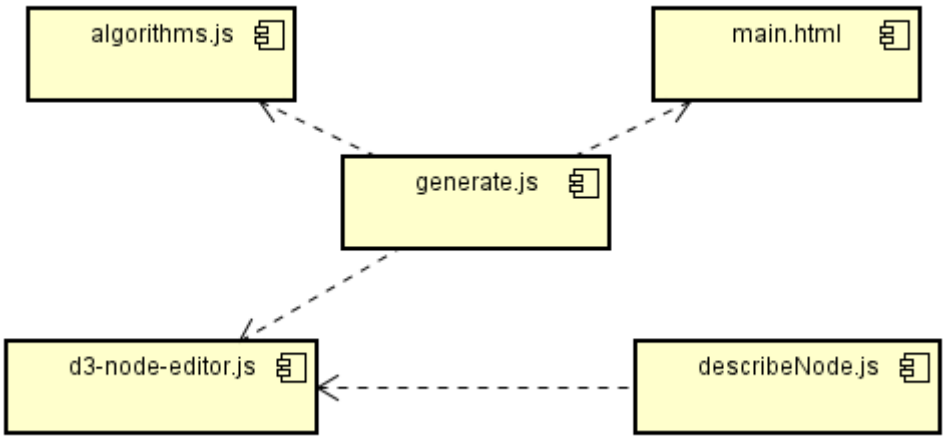


Рис. 3.1 Диаграмма компонентов

Файл *algorithms.js* содержит функцию, на вход которой подается имя алгоритма, описание которого требуется. Она содержит в себе оператор выбора по именам алгоритмов и возвращает соответствующий текст. Часть функции приведена на рис. 3.2

```
function code(name)
{
   switch (name){
   case "mas_c_max":
   return "void max(int a[], int n, int &max)\n\
{\n\
   max = a[0];\n\
   int i = 1;\n\
   while (i < n)\n\
   {\n\
     if (a[i] > max)\n\
        max = a[i];\n\
     i++;\n\
   }\n\
}\n"
   break;
...
}
```

Рис. 3.2 Текст функции в файле algorithms

3.2 Реализация построения диаграммы и генерации кода по ней

Файл *describeNode.js* содержит внутренне описание алгоритмов и то, как что у них будет отображаться на диаграмме. С помощью библиотеки d3-

node-editor алгоритмы отображаются на диаграмме единообразно. Каждый блок стандартно имеет следующие свойства:

1) номер (id);
2) имена (значения) входных и выходных параметры (data);
3) позиция на диаграмме (position);
4) заголовок (title).

Для генерации их не достаточно, так как необходимы название функции для её описания вызова (nameFunction), типы параметров (dataType) и связь с описанием функции в общем файле (describe), поэтому эти свойства были добавлены. Например, для алгоритма суммы элементов массива на языке Си, это выглядит следующим образом:

```
builder(node) {
    node.nameFunction = "sum";
    node.dataType = {0:"int"};
    node.describe = "mas_c_sum";
```

Также в описание блока необходимо добавить такие элементы как socket и control. Для описания типов параметров, в начале файла describeNode необходимо создать несколько элементов socket. Первым параметром которого является id – имя, которое уникально, а вторым является тип сокета. Тип необходимо указать, чтобы соединять socket только одного типа. Если понадобиться описать socket без ограничения к соединению, то можно указать тип ‘Any type’. Пример создания элемента socket:

```
var numSocket = new D3NE.Socket('number', 'Number value', 'hint');
var arraySocket = new D3NE.Socket('array', 'Array value', 'hint');
var listSocket = new D3NE.Socket('list', 'List value', 'hint');
```

В объект, который представляется блоком, добавляются элементы Input и Output, обозначающие входные аргументы и выходные соответственно. Первым параметром для создания таким элементов является имя, которое будет отображаться на диаграмме, в данном случае это обозначает тип аргумента; вторым параметром является socket. Пример создания аргументов:

```
var sizeIn = new D3NE.Input("Размерность", numSocket);
```

```
var arrayIn = new D3NE.Input("Массив", arraySocket);
var resultOut = new D3NE.Output("Результат", numSocket);
```

Для того чтобы обозначить имена аргументов, используется элемент control. Для него обязательно нужно указать, какого типа данные будут в него заноситься, по какому имени они будут доступны, какое начальное значение будет иметь поле ввода control. Также необходимо обозначить какие действия требуется отслеживать, чтобы поддерживать значения данных в актуальном состоянии. В данной программе отслеживается нажатие мыши и ввод данных. Пример создания control представлен ниже:

```
var resultControl = new D3NE.Control('<input type="string">',
    (el, c) => {
      el.value = c.getData('0') || "имя выходного значения";

      function upd() {
        c.putData("0", el.value);
      }

      el.addEventListener("input", ()=>{
        upd();
        editor.eventListener.trigger("change");
      });
      el.addEventListener("mousedown",
function(e){e.stopPropagation()});//   prevent   node   movement   when
selecting text in the input field
      upd();
    }
  );
```

В заключение описания блока следует указать, что он возвращает, а именно node с добавленными в него ранее описанных элементов input, output, и control:

```
 return node
  .addInput(arrayIn)
  .addInput(sizeIn)
  .addControl(resultControl)
  .addOutput(resultOut);
```

При выборе языка или структуры происходит заполнение контекстного меню выбора алгоритмов для добавления их на диаграмму. Например, для языка Си это выглядит таким образом:

```
menu = new D3NE.ContextMenu({
  'Массив' : arrayComp,
```

```
  'Сумма': sumComp,
  'Максимум': maxComp,
  'Минимум':minComp,
  'Удаление': delComp,
  'Количество': kolComp,
  'Среднее' : srComp,
  'Сортировка':sortComp
});
```

Для создания редактора на странице кроме меню понадобятся компоненты, которые отображаются на диаграммы, т.е. ранее описанные блоки и контейнер.

```
components=[arrayComp,sumComp,maxComp,minComp, delComp,kolComp,srComp,
sortComp];
container = document.querySelector('#d3ne');
editor = new D3NE.NodeEditor('demo@0.1.0', container, components,
menu);
```

При смене языка или структуры необходимо начать новый сеанс работы с редактором. Поэтому имеются соответствующие функции, которые сохраняют выбранные параметры и загружают новую страницу с новым сеансом и переданными параметрами. Передача параметров указывается ниже:

```
var url = new URL(window.location.href);
var language = url.searchParams.get("language");
var structure = url.searchParams.get("structure");
if (language == null || structure == null)
{
  language = "0";
  structure = "0";
}
Пример адреса страницы: idef0.ru/main.html?structure=0&language=0
```

Генерация кода выполняется следующим образом: происходит инициализация частей кода, которые будут дополняться по мере обхода алгоритмов. Например, в языке Си таких частей две: там, где описываются функции и там, где описывается её вызов. В языке Прологе их три: описание предикатов, описание правил и вызов.

Затем определяется три списка:

1) список listCompleted выполненных блоков, который изначально пуст;

2) список listAll всех блоков, рассматриваемый по порядку возрастания значения id;
3) стек listTemp текущих рассматриваемых блоков.

Идея прохода по блокам заключается в следующем: рассматриваются все блоки по порядку, и проверяется, может ли текущий блок выполниться. Блок может выполниться, если его входные параметры определены или входных параметров нет. Входные параметры определены, когда они связаны с блоком, который находится в списке выполненных блоков. Таким образом, первым всегда выполнится блок, у которого нет входных параметров, а такие блоки отвечают за структуру данных, которая будет объявлена в начале.

Последующие блоки знают имена входных параметров из полей ввода для имен выходных параметров у блока, связанного своим выходом с входом таких блоков. Но не все алгоритмы имеют поля ввода. Так, например, сортировка принимает структуру по ссылке, поэтому возвращает значение в неё же. Поэтому перед запуском генерации кода выполняется предварительная подготовка: у всех блоков определяется свойство data. Заполнение этого свойства выполняется также с помощью итерационного алгоритма. Сначала проверяется: есть ли свойство data. Если его нет, то блок необходимо добавить в список текущих обрабатываемых блоков listTemp, и пока этот список не пуст, запускать алгоритм добавления свойства data.

```
function compliteData(){
  for (var i in listAll){
       if (Object.keys(listAll[i].data).length == 0){
            if(find(listTemp,listAll[i]) == -1)
            listTemp.push(listAll[i]);
            while (listTemp.length != 0 && noerror)
                 noerror = addData();  }
       if (!noerror) return noerror;}
  listTemp = [];
  return true; }
```

В вызываемом алгоритме проверяется, соединен ли блок с каким-либо предшествующим блоком, который бы мог определить свойство. Если нет, то это означает, что в диаграмме допущена ошибка. Иначе по номеру блока и номеру выхода, с которыми соединен текущий блок, можно определить имя параметра:

```
   function addData(){
    tempNode = listTemp[listTemp.length - 1];
    for(var j=0;j<tempNode.inputs.length && noerror;j++){
    if (tempNode.inputs[j].connections.length == 0){
      noerror = false;
      return noerror;
     }
     if
(Object.keys(listAll[tempNode.inputs[j].connections[0].node].data)
.length == 0)
     listTemp.push(listAll[tempNode.inputs[j].connections[0].node]);
          else
tempNode.data[j]=listAll[tempNode.inputs[j].connections[0].node]
.data[tempNode.inputs[j].connections[0].output];
   }}
```

Сама генерация зависит от языка, но не зависит от структуры и алгоритма. Достаточно описать одну функцию для обработки блока текущего языка, в этом главное преимущество единого внутреннего представления. Генерация для алгоритмов на языке Си выглядит следующим образом:

```
if (tempNode.controls != 0)
 for (var i in tempNode.data)
 {
 main += "\t" + tempNode.dataType[i] + " "
 + tempNode.data[i] + ";\n";
 }
 main += "\t" + tempNode.nameFunction + "(";
 for (var i=0; i<tempNode.inputs.length;i++)
 {
 if (i != 0) main += ", ";
main+=listAll[tempNode.inputs[i].connections[0].node].data[tempNode.in
puts[i].connections[0].output];
```

```
}
if (tempNode.controls != 0)
for (var i in tempNode.data)
main += ", " + tempNode.data[i];
main += ");\n";
codestr += code(tempNode.describe);
```

По окончании обхода, все части, в которых создавался код, объединяются в одну и выводятся пользователю.

В результате объем написанного кода составляет около одной тысячи строк кода. Текст программы описан в приложении D.

3.3 Реализация интерфейса

Для реализации интерфейса был выбран шаблон «центральная сцена» [19]. Так как его основной задачей является акцентировать внимание на содержимом, которое позволяет выполнить определенную задачу, в данном случае – построить диаграмму.

Взгляд человека движется сверху-вниз слева-направо, поэтому стоит расположить элементы панели настроек диаграммы в соответствующем порядке. В верхнем левом углу расположено описание того, что сейчас нужно сделать. Далее предлагается выбрать язык и структуру. Во избежание загромождения и возможного дальнейшего пополнения данных, выбор реализован через выпадающий список. Ниже располагается «центральная сцена» - динамическая рабочая область для построения диаграммы. После неё справа сверху расположена красная кнопка генерации кода, чтобы достаточно контрастировать с остальным фоном, а также побуждающая к действию. При нажатии на неё вызывается метод, запускающий обход диаграммы и последующую генерацию кода по ней. В результате в правой колонке имеется возможность видеть код и сравнивать с тем, что находится в рабочей области.

Весь описанный интерфейс (см. рис. 3.3) представлен в файле *main.html*.

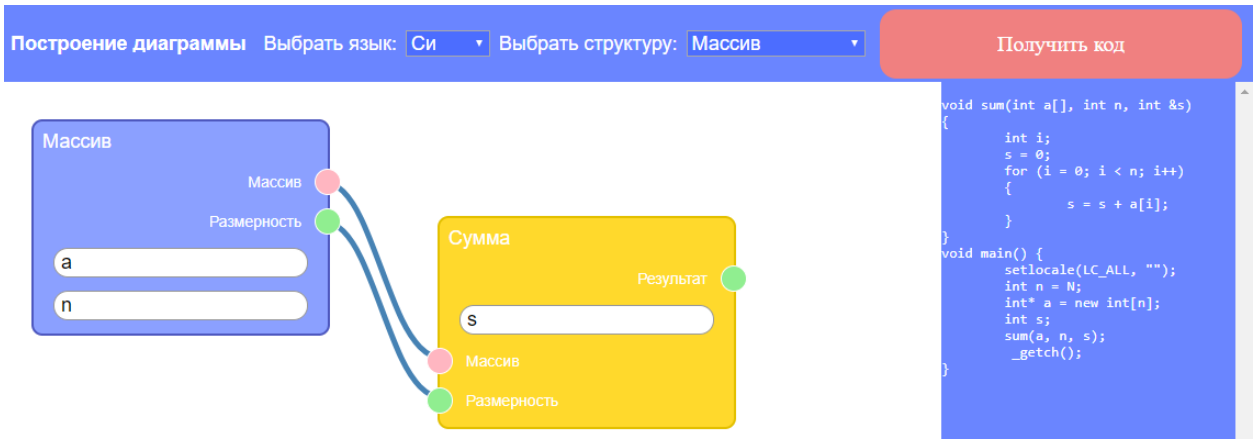


Рис. 3.3 Интерфейс разработанной системы

Глава 4. Тестирование реализованной системы

В данной главе содержится описание тестирования части программы с помощью методов черного и белого ящика.

4.1 Метод черного ящика

Целью тестирования является выявление ситуации, при которой результат, полученный в итоге программы, не соответствует данным, поданным на вход [20].

Тестирование с помощью данного метода представляет собой проверку результатов программы без учета её внутренней структуры. В качестве критериев, с помощью которых проходит проверка, были взяты:

1) выполнение требуемых функций системы с учетом входных и выходных параметров;
2) проверка допустимых значений отображения редактора: нормальные условия (средние), граничные (экстремальные) и исключительные условия (выход за границы);
3) проверка длины последовательности, которую можно занести в поле ввода: пустой набор, единичный набор, слишком короткий набор, набор минимально возможной длины, нормальный набор, набор из нескольких частей, набор максимально возможной длины, слишком длинный набор.

Таблица 4.1 Тестирование методом черного ящика

№	**Описание теста**	**Вход**	**Ожидаемый результат**	**Результат выполнения на компьютере**
1	Выбор языка программирования	Выбран Паскаль из выпадающего списка	Обновление страницы и Паскаль выбран	+

			как язык	
2	Выбор структуры данных	Выбрана структура данных «список»	Обновление страницы и в пользователь	+
3	Добавление блока на диаграмму	Выбор алгоритма из контекстного меню	На диаграмму добавлен блок, алгоритм соответствующей структуры и языка	+
4	Перемещение блока по диаграмме	Начальное положение блока	Блок перемещен в конечное положение блока	+
5	Связывание блоков	Входной параметр блока, выходной параметр другого блока	Прорисовка связи между блоками	+
6	Проверка на допустимое число блоков на диаграмме в нормальных условиях	Пять блоков	Отображение блоков, имеется возможность их перемещать	+
7	Проверка на допустимое число блоков на диаграмме в экстремальных условиях	Семь блоков (большее число блоков трудно для восприятия человеком)	Отображение блоков, имеется возможность их перемещать	+
8	Проверка на допустимое число блоков на диаграмме в исключительных условиях	Двадцать блоков	Отображение блоков, имеется возможность их перемещать	-
9	Ввод пустого имени параметра	Пустое имя	Сообщение об ошибке ввода	+
10	Ввод набора из	Имя с пробелами	Сообщение об	+

	нескольких частей		ошибке ввода	
11	Ввод слишком короткого набора	Имя из пробела	Сообщение об ошибке ввода	+
12	Ввод набора минимально возможной длины	Имя из одного символа	Продолжение работы программы	+
13	Ввод некорректного набора	Имя из цифр	Сообщение об ошибке ввода	+
14	Ввод набора максимально возможной длины	Имя из 5000 знаков	Приложение продолжает свою работу	+

4.2 Метод белого ящика

Тестирование с помощью этого метода включает в себя учёт структуры программы. В качестве критерия взято минимальное грубое тестирование, его достаточно для работы с небольшими системами. Он заключается в том, чтобы написать такое количество тестов, которое по возможности покрывает решения и условия, при этом имеющий дополнительные требования при проверке цикла. Если встречается цикл с предусловием или со счетчиком, то тело цикла должно быть проверено при нулькратном, однократном и многократном повторении тела цикла [20]. Каждое условие должно сработать как истина и как ложь.

Главным образом необходимо протестировать функцию генерации кода. Поэтому составим для неё таблицу, где отметим, какой тест какие ветки программы затрагивает.

Таблица 4.2 Минимально глубокое тестирование функции aisle()

№			**Тест 1**	**Тест 2**	**Тест 3**	**Тест4**
1	for (var i in listAll)	=0	+			
		=1		+		
		>1			+	+
2	if (find(listTemp,listAll[i]) == -1	+		+	+	+

	&& find(listComplited,listAll[i]) == -1)	-			+	
3	find(listTemp,listAll[i]) == -1	+		+		+
		-			+	
4	find(listComplited,listAll[i]) == -1	+		+		+
		-			+	
5	while (listTemp.length != 0)	=0				+
		=1		+		
		>1			+	

Таблица 4.3 Минимально глубокое тестирование функциии treeNodeWork()

№			T1	T2	T3	T4	T5	T6
1	if (tempNode.inputs.length == 0)	+	+					
		-		+	+	+	+	+
2	for(i=0;i<tempNode.inputs.length;i++)	=0		+				
		=1			+			
		>1				+	+	+
3	if (tempNode.inputs[i] .connections.length == 0)	+			+			
		-				+	+	+
4	for (var j=0;j<tempNode.inputs[i] .connections.length;j++)	=0				+		
		=1					+	
		>1						+
5	if (find(listComplited, listAll[tempNode.inputs[i] .connections[j].node]) == -1)	+				+		+
		-					+	
6	if (find(listTemp,listAll[tempNode .inputs[i].connections[j].node]) == -1)	+				+		
		-						+

В результате проведенного тестирования были найдены ошибки, но в последствие исправлены, поэтому проводилось несколько этапов тестирования. Критерии покрытия ветвей, решений или условий не гарантируют полное тестирование на ошибки, как и критерий метод глубокого тестирования. Но он является компромиссом между надежностью и сложностью тестирования [20].

Заключение

В рамках данной работы выполнено проектирование и разработка системы генерации исходного кода по высокоуровневой спецификации задачи.

Работоспособность системы проверена на примере работ трех языков программирования с пятью структурами данных и семью алгоритмами на каждой. В качестве инструмента построения диаграммы была использована открытая библиотека d3-node-editor, позволившая выполнить все требования, которые были поставлены перед пользовательским интерфейсом. Также благодаря ей, было использовано единое внутреннее представление блоков, которое было расширено в виду исследовательской задачи. Такое представление обеспечило создание единообразного подхода к обработке алгоритмов и дальнейшей генерации кода.

Созданная система является веб-приложением, поэтому при расширении функционала не возникнет проблем с его развертыванием на машине пользователя.

Выбранная модульная архитектура позволяет дополнять каждый компонент независимо и четко разделять логику и доступ к данным между ними. При добавлении нового языка или структуры, не нужно изменять внутреннее представление алгоритмов, а вследствие этого метод обхода, т.е. генерации кода. В результате объем написанного кода составляет 1079 строк на языке программирования JavaScript.

Апробация работы была проведена: на “Всероссийской научно-практической конференции молодых ученых с международным участием «Математика и междисциплинарные исследования – 2018»“ , за доклад был получен диплом I степени в секции «Современные методики обучения»; на студенческих научно-практических конференциях механико-

математического факультета «Математическое и программное обеспечение информационных и интеллектуальных систем» 2017 г. и 2018 г.

Список публикаций:

1) Хакимуллина А.А., Дацун Н.Н. Высокоуровневое моделирование при обучении основам программирования. Математика и междисциплинарные исследования – 2018 [Электронный ресурс]. Пермь, 2018. С. 124-126.

В рамках данной работы работоспособность системы проверена на трех языках. Для превращения системы в программы продукт необходимо выполнить следующие действия:

1) разработать инструмент, который бы позволял дополнять систему алгоритмами при помощи единого интерфейса без непосредственного внесения изменений в код;
2) выделить инварианты алгоритмов для их единообразного хранения.

Библиографический список

1. *Grant M.* Dynamics of undergraduate student generic problem-solving skills captured by a campus-wide study [Электронный ресурс] [Режим доступа: https://news.ok.ubc.ca/2016/12/14/lecturing-likely-not-effective-for-developing-problem-solving-skills-in-students/] [Проверено: 04.06.2018].
2. *Хьюз Дж.* Структурный подход к программированию / *Дж. Хьюз, Дж. Мичтом*. М.:Мир. 1980. 276с.
3. *Мартин Р.* Чистый код: создание, анализ и рефакторинг. Библиотека программиста / *Р. Мартин*. СПб.: Питер. 2016. 464 с.
4. *Марка Д.А*. Методология структурного анализа и проектирования (SADT) / *Д.А. Марка, К. МакГоуэн*. Пер. с англ. М.: МетаТехнология. 1993. 243 с.
5. Методология IDEF0. Стандарт. Русская версия. М.: Метатехнология, 1993. 93 с.
6. *Шаврин С.М.* Моделирование и проектирование информационных систем. / *С.М. Шаврин, Л.Н. Лядова, С.И. Чуприна*. Пермь: Пермский университет. 2007. 151 с.
7. *Канжелев С.Ю., Шалыто А.А.* Автоматическая генерация программ с явным выделением состояний. // Журнал "Информационно-управляющие системы". 2006. №6. С 60-63.
8. Flexberry. [Электронный ресурс] [Режим доступа: http://wiki.flexberry.ru/FlexberryDesigner.ashx] [Проверено: 20.12.2017].
9. QReal [Электронный ресурс.] [Режим доступа: http://qreal.ru/] [Проверено: 25.04.2018].
10. *Иванова О.О.* Генерация кода по диаграмме активностей. [Электронный ресурс] [Режим доступа: https://goo.gl/dcybvT] [Проверено: 25.04.2018].
11. *Сухов А.О.* Разработка инструментальных средств создания визуальных предметно-ориентированных языков [Текст]: дис. канд. ф.-м. наук:

05.13.11: защищена 05.12.2013 / Сухов Александр Олегович. Институт системного программирования Российской академии наук. 2013. 157 с.

12. *Рябинин К.В., Чуприна С.И., Бортников А.Ю.* Автоматизация настройки систем научной визуализации на специфику разнообразных источников данных. // Научная визуализация. М.: Национальный исследовательский ядерный университет МИФИ. 2016. С. 1–14
13. *Ван Тассел Д.* Стиль, разработка, эффективность, отладка и испытание программ / *Д. Ван Тассел*. Пер. с англ. М.: Мир. 1981. 320 с.
14. *Иванова Г.С.* Технология программирования: Учебник для вузов / Г.С. *Иванова.* М.: Изд-во МГТУ им. Н.Э. Баумана. 2002. 320 с.
15. *Kustov M., Guban B., Datsun N.* Application of SADT for source code generation in learning the programming fundamentals. // Труды 6-ого весеннего/летнего коллоквиума молодых исследователей в области программной инженерии (SYRCoSE 2012), 30-31 мая 2012 г. Пермь. С.28-33.
16. D3 Node Editor [Электронный ресурс] [Режим доступа: https://d3-node-editor.readthedocs.io/en/latest] [Проверено: 25.04.2018].
17. Введение в d3.js [Электронный ресурс] [Режим доступа: http://frontender.info/vvedenie-v-djs/] [Проверено 04.06.2018].
18. Node Editor Winforms Editor [Электронный ресурс] [Режим доступа: https://github.com/komorra/NodeEditorWinforms/blob/master/README.md] [Проверено: 25.04.2018].
19. *Тидвелл Дж.* Разработка пользовательских интерфейсов / Дж. Тидвелл. СПб.:Питер. 2008. 416 с.
20. *Плаксин М.А.* Тестирование и отладка для профессионалов будущих и настоящих [Электронный ресурс] / *М.А. Плаксин*. М.: БИНОМ. Лаборатория знаний. 2013.167 с.

21. *Хакимуллина А.А., Дацун Н.Н.* Высокоуровневое моделирование при обучении основам программирования. Математика и междисциплинарные исследования – 2018 [Электронный ресурс]. Пермь, 2018. С. 124-126.

Приложение А. Техническое задание

Наименование: генератор исходного кода программ по высокоуровневой спецификации задачи.

Область применения программы: в учебных учреждениях на ИТ-специальностях.

Краткая характеристика: разработанное приложение позволяет добавлять/удалять алгоритмы, которые должны быть сгенерированы в исходном коде, а также устанавливать и изменять связи между блоками спецификации. В результате пользователь может получить исходный код программы из составленной диаграммы.

Основания для разработки:

Основанием для разработки является задание на научно-исследовательскую работу кафедры математического обеспечения вычислительных систем 11 триместр.

Назначение разработки:

Функцией является генерация исходного кода программ по высокоуровневой спецификации задачи.

Требования к программе:

1. Требования к функциональным характеристикам

1.1. Приложение должно позволять осуществлять выбор соответствующих алгоритмов для каждой структуры данных.

1.2. Приложение должно строить диаграмму по выбранным алгоритмам.

1.3. Приложение должно позволять размещать элементы диаграммы, соединяя их по входам-выходам.

1.4. Приложение должно генерировать исходных код по построенной диаграмме на следующих целевых языках: Си, Pascal, Visual Prolog.

1.5. Приложение выполняет выбор внутреннего представления полученного графа для обеспечения относительной независимости генератора от целевого языка программирования.

Входные данные:

Целевой язык, структура данных, соответствующий ей алгоритм, диаграмма высокоуровневой спецификации.

Выходные данные:

Исходный код программы на языке высокого уровня.

2. Требования к надежности

Программа должна функционировать на протяжении всего времени работы с ней (24 х 7)

3. Условия эксплуатации

Квалификация пользователя для использования программы должна быть не ниже «Пользователь ПК».

4. Требования к составу и параметрам технических средств

4.1. В состав технических средств должен входить персональный компьютер.

4.2. ПК должен поддерживать корректную работу .exe файлов.

5. Требования к интерфейсу

5.1. Интерфейс должен содержать отдельную область для задания структуры данных.

5.2. Должна быть возможность выбора алгоритмов для каждой структуры данных.

5.3. Должна быть возможность соединения алгоритмов (блоков) между собой по входам и выходам.

5.4. Должна быть область представления диаграммы с возможностью добавления, изменения, удаления алгоритмов (блоков).

5.5. Должна присутствовать область, в которой просматривается сгенерированный исходный код.

6. Стадии и этапы разработки

Этапы разработки:

7.1. Этап анализа

7.2. Этап проектирования

7.3. Этап реализации

7.4. Этап тестирования

7.5. Этап документирования

7.6. Этап сопровождения

8. Порядок контроля и приёмки

Защита научно-исследовательской работы выполняется в комиссии в составе не менее 3 человек. Текст отчёта должен быть представлен руководителю на проверку не менее чем за 3 рабочих дня до защиты.

Приложение В. Библиотека алгоритмов

1	Массив	Список	Строка

Поиск максимума	`void FindMax(int a[], int n, int &max){`
	`  max = a[0];`
	`  int i = 1;`
	`  while (i < n){`
	`  if (a[i] > max)`
	`  max = a[i];`
	`  i++;`
	`  }`
	`}`
		`void FindMax(Node* head, int &max){`
		`  max = head->value;`
		`  Node* cur = head;`
		`  while (cur != NULL){`
		`  if (max < cur->value)`
		`  max = cur->value;`
		`  cur = cur->next;`
		`  }`
		`}`
			`void Max(char *str, char &max){`
			`  max = str[0];`
			`  int i = 1;`
			`  while (i < strlen(str)-1){`
			`  if (str[i] > max)`
			`  max = str[i];`
			`  i++;`
			`  }`
			`}`

Удаление элемента по значению	`void Delet(int *a, int &n, int key, int *a1, int &n1){`
	`  int i = 0, j = 0;`
	`  while (i < n){`
	`  while (a[i] == key){`
	`  for (int j = i; j < n; j++)`
	`  a[j] = a[j + 1];`
	`  n--;`
	`  }`
	`  i++;`
	`  }`
	`  for (int i = 0; i < n; i++)`
	`  a1[i] = a[i];`
	`  n1 = n;`
	`}`
		`void delete_value(PNode &head, int value, PNode &newHead){`
		`  Node *current, *previous;`
		`  previous = current = head;`
		`  while (current) {`
		`  if (current->value == value) {`
		`  (current == head) {`
		`  if (head->next == NULL) {`
		`  delete current;`
		`  return;`
		`  }`
		`  head = current->next;`
		`  delete current;`
		`  current = head;`
		`  }`
		`  else {`
		`if (current->next != NULL) {`
		`  previous->next = current->next;`
		`  delete current;`
		`current = previous->next;`
		`  }`
		`  else {`
		`  previous->next = NULL;`
		`  delete current;`
		`  return;`
		`  }`
		`  }`
		`  }`
		`  else{`
		`  previous = current;`
		`current = previous->next;`
		`  }`
		`  }`
		`  newHead = head;`
		`}`
			`void Delet(char *s, char key, char *s1)`
			`{`
			`  int i = 0, j = 0, n = strlen(s);`
			`  while (i < strlen(s)){`
			`  while (s[i] == key){`
			`  for (int j = i; j < strlen(s); j++)`
			`  s[j] = s[j + 1];`
			`  n--;`
			`  }`
			`  i++;`
			`  }`
			`  for (int i = 0; i < n; i++)`
			`  s1[i] = s[i];`
			`s1[i]='\0';`
			`}`

Подсчет элемента	`void ChRavElement(int a[], int n, int key, int &k){` `  k = 0;` `  int i = 0;` `  while(i < n – 1) {` `  if (a[i] == key)` `  k = k + 1;` `  i++;` `  }` `}`	`void ChRavElement(Node* head, int key, int &k){` `  int k = 0;` `  Node* cur;` `  cur = head;` `  while (cur != NULL) {` `  if (cur->value == key)` `  k = k + 1;` `  cur = cur->next;` `  }` `  return k;}`	`void Csimv (char *str, char key, int &k){` `  k = 0;` `  int i=0;` `  **while**(i < strlen(str) – 1)` `{` `  **if (**str[i] == **key)** ` **k = k + 1;** `  i++;` `  **}** `}`

	Массив	Список	Строка

Пузырьковая сортировка	`void BubbleSort(int* a, int n)`
	`{`
	`  int tmp, i, j;`
	`  for (i = 0; i < n - 1; i++)`
	`  {`
	`  for (j = i+1; j < n; j++)`
	`  {`
	`  if (a[i] > a[j])`
	`  {`
	`  tmp = a[i];`
	`  a[i] = a[j];`
	`  a[j] = tmp;`
	`  }`
	`  }`
	`  }`
	`}`
		`void llist_bubble_sort(Node *head)`
		`{`
		`  struct Node *a = NULL;`
		`  struct Node *b = NULL;`
		`  struct Node *c = NULL;`
		`  struct Node *e = NULL;`
		`  struct Node *tmp = NULL;`
		
		`  while (e != head->next){`
		`  c = a = head;`
		`  b = a->next;`
		`  while (a != e){`
		`  if (a->value > b->value{`
		`  if (a == head){`
		`  tmp = b->next;`
		`  b->next = a;`
		`  a->next = tmp;`
		`  head = b;`
		`  c = b;`
		`  }`
		`  else{`
		`  tmp = b->next;`
		`  b->next = a;`
		`  a->next = tmp;`
		`  c->next = b;`
		`  c = b;`
		`  }`
		`  }`
		`  else{`
		`  c = a;`
		`  a = a->next;`
		`  }`
		`  b = a->next;`
		`  if (b == e)`
		`  e = a;`
		`  }`
		`  }`
		`}`
			`void BSortStr(char *a)`
			`{`
			`  int tmp, i, j;`
			`  for (i = 0; i < strlen(a) - 1; i++)`
			`  {`
			`  for (j = 0; j < strlen(a) - 1; j++)`
			`  {`
			`  if (a[j + 1] > a[j])`
			`  {`
			`  tmp = a[j + 1];`
			`  a[j + 1] = a[j];`
			`  a[j] = tmp;`
			`  }`
			`  }`
			`  }`
			`}`

	Текстовый файл	Бинарный файл

Поиск максимума	void MaxInFile(FILE *file, int &max)
	{
	  fseek(file, 0L, SEEK_SET);
	  fscanf(file, "%d", &max);
	  int num;
	  while (!feof(file))
	  {
	  fscanf(file, "%d", &num);
	  if (num > max)
	  max = num;
	  }
	}
		void MaxInBfile(FILE *file, int n, int &max)
		{
		  fseek(file, 0L, SEEK_SET);
		  struct Data temp;
		  fread(&temp.day, sizeof(temp.day), 1, file);
		  fread(&max, sizeof(temp.mes), 1, file);
		  fread(&temp.god, sizeof(temp.god), 1, file);
		  int i = 1;
		  while (i<n){
		  fread(&temp.day, sizeof(temp.day), 1, file);
		  fread(&temp.mes, sizeof(temp.mes), 1, file);
		  fread(&temp.god, sizeof(temp.god), 1, file);
		  if (temp.mes > max)
		  max = temp.mes;
		  i++;
		  }
		}

	Текстовый файл	Бинарный файл

Удаление по значению	void DelSimv(FILE *file, char simv)
	{
	  FILE *out;
	  char buffer[50];
	  fseek(file, 0L, SEEK_SET);
	  fopen_s(&out, "binput222.txt", "w+");
	  while (!feof(file))
	  {
	  fgets(buffer, 50, file);
	  while (strstr(buffer, &simv) - buffer >= 0)
	  {
	  for (int i = strstr(buffer, &simv) - buffer; i < strlen(buffer); i++)
	  buffer[i] = buffer[i + 1];
	  }
	  fputs(buffer, out);
	  }
	  fclose(file);
	  fclose(out);
	  remove("input.txt");
	  rename("binput222.txt", "input.txt");
	  fopen_s(&file, "input.txt", "r+b");
	}
		void DelInBfile(FILE *file, int &n, int elem)
		{
		  FILE *binput;
		  struct Data temp;
		  int k = 0;
		  fseek(file, 0L, SEEK_SET);
		  fopen_s(&binput, "binput222.dat", "w+b");
		  for (int i = 0; i < n; i++)
		  {
		  fread(&temp.day, sizeof(temp.day), 1, file);
		  fread(&temp.mes, sizeof(temp.mes), 1, file);
		  fread(&temp.god, sizeof(temp.god), 1, file);
		  printf_s("%d %s\n", temp.day, temp.mes);
		  if (temp.day != elem)
		  {
		  fwrite(&temp.day, sizeof(temp.day), 1, file);
		  fwrite(&temp.mes, sizeof(temp.mes), 1, file);
		  fwrite(&temp.god, sizeof(temp.god), 1, file);
		  k++;
		  }
		  }
		  fclose(file);
		  fclose(binput);
		  remove("binput.dat");
		  rename("binput222.dat", "binput.dat");
		  fopen_s(&file, "binput.dat", "r+b");
		  n = k;
		}

Подсчет элемента	void Csimv(FILE *file, int key, int &k)
	{
	k = 0;
	int num;
	fseek(file, 0L, SEEK_SET);
	
	while (!feof(file))
	{
	fscanf(file, "%d", &num);
	
	if (num == key)
	k = k + 1;
	
	}
	}
		void CinBfile(FILE *file, int n, int key, int &k)
		{
		k = 0;
		struct Man temp;
		fseek(file, 0L, SEEK_SET);
		
		int i = 0;
		while (i < n – 1)
		{
		fread(&temp.age, sizeof(temp.age), 1, file);
		fseek(file, ftell(file) + 1, SEEK_SET);
		fread(&temp.name, sizeof(temp.name), 1, file);
		if (temp.age == key)
		k = k + 1;
		i++;
		}
		}

	Текстовый файл	Бинарный файл

Сортировка	`void FBsort(FILE *filein, int lenstr, int cstr){`
	`  char buffer[11], buffer2[11];`
	`  int i = 0;`
	`  int N = cstr;`
	`  fseek(filein, 0, SEEK_SET);`
	`while (i<(sizeof(buffer))*(N-1)) {`
	`  //Читаем первую из сравниваемых записей`
	`fseek(filein, i, SEEK_SET);`
	`fread(buffer, (sizeof(buffer)), 1, filein);`
	`  buffer[10] = '\n';`
	`  printf("buffer %s\n", buffer);`
	`  int j = i + (sizeof(buffer))+1;`
	`  while (j < ((sizeof(buffer)))*N){`
	`  //Читаем вторую из сравниваемых записей`
	`  fseek(filein, j, SEEK_SET);`
	`  fread(buffer2, (sizeof(buffer)), 1, filein);`
	`  buffer2[10] = '\n';`
	`  printf("buffer2 %s\n", buffer2);`
	`  //Если первая запись больше второй, то ...`
	`  if (strcmp(buffer, buffer2) > 0){`
	`  //ОБМЕН ЗАПИСЕЙ`
	`  int pos = i;`
	`  fseek(filein, pos, SEEK_SET);`
	`  fwrite(buffer2, (sizeof(buffer)), 1, filein);`
	`  pos = j;`
	`  fseek(filein, pos, SEEK_SET);`
	`  fwrite(buffer, (sizeof(buffer)), 1, filein);`
	`  strncpy_s(buffer, buffer2, lenstr);`
	`  buffer[10] = '\n';`
	`  printf("buffer! %s\n", buffer);`
	`  printf("buffer!2 %s\n", buffer2);`
	`  }`
	`  j += (sizeof(buffer)) + 1;`
	`  }`
	`  i += (sizeof(buffer)) + 1;`
	`  }`
	`}`
		`void BFileBsort(FILE *filein, int N){`
		`  int i = 0, pos,j;`
		`  struct Man b;`
		`  struct Man a;`
		`  fseek(filein, 0, SEEK_SET);`
		`while (i<((sizeof(a)+1)*(N - 1))) {`
		`  //Читаем первую из сравниваемых записей`
		`  fseek(filein, i, SEEK_SET);`
		`  fread(&a.age, (sizeof(a.age)), 1, filein);`
		`  fseek(filein, ftell(filein) + 1, SEEK_SET);`
		`  fread(&a.name, (sizeof(a.age)), 1, filein);`
		`  printf_s("%d %s\n", a.age, a.name);`
		`  int j = i + sizeof(a.age) + sizeof(a.name) +1;`
		`  while (j < ((sizeof(a.age) + sizeof(a.name) + 1)*N)) {`
		`  //Читаем вторую из сравниваемых записей`
		`  fseek(filein, j, SEEK_SET);`
		`  fread(&b.age, (sizeof(b.age)), 1, filein);`
		`  fseek(filein, ftell(filein) + 1, SEEK_SET);`
		`  fread(&b.name, (sizeof(b.age)), 1, filein);`
		`  printf_s("%d %s\n", b.age, b.name);`
		`  //Если первая запись больше второй, то ...`
		`  if (a.age > b.age) {`
		`  //ОБМЕН ЗАПИСЕЙ`
		`  pos = i;`
		`  fseek(filein, pos, SEEK_SET);`
		`  fwrite(&b.age, sizeof(b.age)+1, 1, filein);`
		`  fwrite(&b.name, sizeof(b.name), 1, filein);`
		`  pos = j;`
		`  fseek(filein, pos, SEEK_SET);`
		`  fwrite(&a.age, sizeof(a.age) + 1, 1, filein);`
		`  fwrite(&a.name, sizeof(a.name), 1, filein);`
		`  a.age = b.age;`
		`  strncpy_s(a.name, b.name, strlen(a.name));`
		`  printf_s("a!!! %d %s\n", a.age, a.name);`
		`  printf_s("b!!! %d %s\n", b.age, b.name);`
		`  }`
		`  j += sizeof(a.age) + sizeof(a.name) + 1;`
		`  }`
		`  i += sizeof(a.age) + sizeof(a.name) + 1;`
		`  }`
		`}`

Приложение С. Пример сгенерированного кода

Си	Паскаль

#include <stdio.h>
using namespace std;
#include <String.h>
#include <locale.h>
#include <stdlib.h>
#include <limits.h>
#include <conio.h>
 	
#define N 5
void max(int a[], int n, int &max)
{
  max = a[0];
  int i = 1;
  while (i < n)
  {
  if (a[i] > max)
  max = a[i];
  i++;
  }
}
 	
void del(int *a, int &n, int key, int *a1, int &n1)
{
  int i = 0, j = 0;
 	
  while (i < n)
  {
  while (a[i] == key)
  {
  for (int j = i; j < n; j++)
  {
  a[j] = a[j + 1];
  }
  n--;
  }
  i++;
  }
  for (int i = 0; i < n; i++)
  {
  a1[i] = a[i];
  }
  n1 = n;
}
 	
void min(int a[], int n, int &min)
{
  int i = 1;
  min = a[0];
  while (i < n)
  {
  if (a[i] < min)
  min = a[i];
  i++;
  }
}
	Const N=5;
	Var a1:array of integer;
	  a2:array of integer;
	  a3:array of integer;
	  n1:integer;
	  n2:integer;
	  n3:integer;
	  m1:integer;
	  m2:integer;
	  k1:integer;
	  s1:integer;
	  sr1:real;
	  i:integer;
	Procedure FindMax(a:array of integer; n:integer; var max : integer);
	var i:integer;
	begin
	  max:=a[0];
	  i:=1;
	  while (i < n) do
	  begin
	  if (a[i] > max) then max:=a[i];
	  inc(i);
	  end;
	end;
	
	Procedure FindMin(a:array of integer; n:integer; var min : integer);
	var i:integer;
	begin
	  min:=a[0];
	  i:=1;
	  while (i < n) do
	  begin
	  if (a[i] < min) then min:=a[i];
	  inc(i);
	  end;
	end;
	
	Procedure Sum(a:array of integer; n:integer; var s : integer);
	var i:integer;
	begin
	  s:=0;
	i:=0;
	  while (i<n) do
	  begin
	  s:=s+a[i];
	  inc(i);
	  end;
	end;
	
	Procedure Delet(a: array of integer; var n:integer; key:integer; var a1 : array of integer; var n1 : integer);
	var i,m: Integer;
	begin

```
void sum(int a[], int n, int &s)
{
	int i;
	s = 0;
	for (i = 0; i < n; i++)
	{
		s = s + a[i];
	}
}

void sra(int s, int k, double &sr)
{
	sr = (double)s / k;
}

void kol(int a[], int n, int &k)
{
	k = n;
}

void main()
{
	int n1 = N;
	int *a1 = new int[n1];
	int m1;
	max(a1, n1, m1);
	int *a2 = new int[n1];
	int n2;
	del(a1, n1, m1, a2, n2);
	int m2;
	min(a2, n2, m2);
	int *a3 = new int[n2];
	int n3;
	del(a2, n2, m2, a3, n3);
	int s1;
	sum(a3, n3, s1);
	int k1;
	kol(a3, n3, k1);
	double sr1;
	sra(s1, k1, sr1);
	_getch();
}
```

```
  m:=0;
  for i:=0 to n-1 do
  if (a[i]=key) then m:=m+1 else a[i-
m]:=a[i];
  n1:=n-m;
	for i := 0 to n1-1 do
		a1[i] := a[i];
end;

Procedure Sra(s : integer; k : integer; var
sr : real);
begin
	sr:=s/k;
end;

Procedure kol(a: array of integer; var
n:integer; var k : integer);
begin
	k:=n;
end;

Begin
n1:=N;
setlength(a1,n1);
FindMax(a1,n1,m1);
n2:=N;
setlength(a2,n2);
Delet(a1,n1,m1,a2,n2);
FindMin(a2,n2,m2);
n3:=N;
setlength(a3,n3);
Delet(a2,n2,m2,a3,n3);
Sum(a3,n3,s1);
Kol(a3,n3,k1);
Sra(s1,k1,sr1);
end.
```

Приложение D. Текст программы

main.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="css/style.css">
    <link rel="stylesheet" href="css/d3-node-editor.css">
    <script src="js/jquery.js"> </script>
    <script src="js/generate.js"> </script>
    <script src="js/algorithm.js"> </script>

    <script
src="https://cdn.jsdelivr.net/npm/alight@0.14.0/alight.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/d3@4.10.2/build/d3.min.js"></script>
    <script src="js/d3-node-editor.js"></script>
    <title>Document</title>
</head>
<body>

<div class="ControlsContainer">
    <div class="header">Построение диаграммы</div>
    <div class="textik" >Выбрать язык:</div>
        <select id="Change-Language" onchange="Change()" style="font-size: 18px;
color:white; background-color:rgba(69, 103, 255, 0.8)!important ">
                <option value="0">Си</option>
                <option value="1">Prolog</option>
            </select>

            <div class="textik" >Выбрать структуру:</div>
            <select id="Combobox"  onchange="Change()" style="font-size: 18px;
color:white; background-color:rgba(69, 103, 255, 0.8)!important ">
                <option value="0">Массив</option>
                <option value="1">Список</option>
                <option value="2">Строка</option>
                <option value="3">Текстовый файл</option>
                <option value="4">Бинарный файл</option>
            </select>
            <button onclick="Generate()">Получить код</button>
        </div>
    <div class="flexBoxHolder">
        <div id="removable" >
            <div id="d3ne" class="node-editor"></div>
        </div>
```

```
        <div id="CodeWrapper" class="codeview">
            <xmp class="card-title " id="ProgramCode" style="color:white">Здесь
будет сгенерированный код...
            </xmp>
        </div>
    </div>

    <script src="js/script.js"> </script>
    <script src="js/change-structure.js"> </script>
</body>
</html>
```

describeNode.js

```
var numSocket = new D3NE.Socket('number', 'Number value', 'hint');
var arraySocket = new D3NE.Socket('array', 'Array value', 'hint');
var listSocket = new  D3NE.Socket('list', 'List value', 'hint');

var arrayComp = new D3NE.Component('Массив',{
    builder(node) {
        node.nameFunction = "array_c";
        node.dataType = {0:"int*" ,1:"int"};
        node.describe = "mas_c_sum";
        var sizeOut = new D3NE.Output('Размерность', numSocket);
        var arrayOut = new D3NE.Output('Массив', arraySocket);
        var sizeControl = new D3NE.Control('<input type="string">',
          (el, c) => {
             el.value = c.getData('1') || "имя размерности";

             function upd() {
                c.putData("1", el.value);
             }

             el.addEventListener("input", ()=>{
                upd();
                editor.eventListener.trigger("change");
             });
             el.addEventListener("mousedown",
function(e){e.stopPropagation()});// prevent node movement when selecting text in
the input field
            upd();
          }
       );
        var arrayControl = new D3NE.Control('<input type="string">',
        (el, c) => {
            el.value = c.getData('0') || "имя массива";

            function upd() {
                c.putData("0", el.value);
            }
```

```
            el.addEventListener("input", ()=>{
                upd();
                editor.eventListener.trigger("change");
            });
            el.addEventListener("mousedown", function(e){e.stopPropagation()});//
prevent node movement when selecting text in the input field
            upd();
        }
        );
        return node
                .addOutput(arrayOut)
                .addOutput(sizeOut)
                .addControl(arrayControl)
                .addControl(sizeControl);

    }});

var listComp = new D3NE.Component('Список',{
    builder(node) {
        var listOut = new D3NE.Output('Список', listSocket);
        var arrayControl = new D3NE.Control('<input type="string">',
            (el, c) => {
                el.value = c.getData('0') || "имя списка";

                function upd() {
                    c.putData("0", el.value);
                }

                el.addEventListener("input", ()=>{
                    upd();
                    editor.eventListener.trigger("change");
                });
                el.addEventListener("mousedown",
function(e){e.stopPropagation()});// prevent node movement when selecting text in
the input field
                upd();
            }
        );
        return node
            .addOutput(listOut)
            .addControl(arrayControl);
 }});

 var listCompProlog = new D3NE.Component('Список',{
    builder(node) {
        node.nameFunction = "list_prolog";
        var listOut = new D3NE.Output('Список', listSocket);
        var arrayControl = new D3NE.Control('<input type="string">',
            (el, c) => {
```

```
                el.value = c.getData('0') || "имя списка";

                function upd() {
                    c.putData("0", el.value);
                }

                el.addEventListener("input", ()=>{
                    upd();
                    editor.eventListener.trigger("change");
                });
                el.addEventListener("mousedown",
function(e){e.stopPropagation()});// prevent node movement when selecting text in
the input field
                upd();
            }
        );
        return node
            .addOutput(listOut)
            .addControl(arrayControl);
 }});

var sumComp = new D3NE.Component("Сумма", {
    builder(node) {
        node.nameFunction = "sum";
        node.dataType = {0:"int"};
        node.describe = "mas_c_sum";
  var sizeIn = new D3NE.Input("Размерность", numSocket);
  var arrayIn = new D3NE.Input("Массив", arraySocket);
  var resultOut = new D3NE.Output("Результат", numSocket);
  var resultControl = new D3NE.Control('<input type="string">',
        (el, c) => {
            el.value = c.getData('0') || "имя выходного значения";

            function upd() {
                c.putData("0", el.value);
            }

            el.addEventListener("input", ()=>{
                upd();
                editor.eventListener.trigger("change");
            });
            el.addEventListener("mousedown", function(e){e.stopPropagation()});//
prevent node movement when selecting text in the input field
            upd();
        }
    );
  return node
    .addInput(arrayIn)
    .addInput(sizeIn)
     .addControl(resultControl)
```

```
     .addOutput(resultOut);
    }});

var sumCompProlog = new D3NE.Component("Сумма", {
    builder(node) {
        node.nameFunction = "sum";
        node.dataType = {0:"integer*", 1:"integer [out]"}
        node.describe = "list_prolog_sum";
        node.functionType = "determ";
  var listIn = new D3NE.Input("Список", listSocket);
  var resultOut = new D3NE.Output("Результат", numSocket);
  var resultControl = new D3NE.Control('<input type="string">',
        (el, c) => {
            el.value = c.getData('0') || "имя_выходного_значения";
            function upd() {
                c.putData("0", el.value);
            }
            el.addEventListener("input", ()=>{
                upd();
                editor.eventListener.trigger("change");
            });
            el.addEventListener("mousedown", function(e){e.stopPropagation()});//
prevent node movement when selecting text in the input field
            upd();
        }
    );
  return node
    .addInput(listIn)
     .addControl(resultControl)
     .addOutput(resultOut);
    }});
var sumCompList = new D3NE.Component("Сумма", {
    builder(node) {
        node.nameFunction = "sum";
        node.dataType = {0:"int"};
        node.describe = "list_c_sum";
      var listIn = new D3NE.Input("Список", listSocket);
      var resultOut = new D3NE.Output("Результат", numSocket);
      var resultControl = new D3NE.Control('<input type="string">',
            (el, c) => {
                el.value = c.getData('0') || "имя выходного значения";

                function upd() {
                    c.putData("0", el.value);
                }

                el.addEventListener("input", ()=>{
                    upd();
                    editor.eventListener.trigger("change");
                });
```

```
                el.addEventListener("mousedown",
function(e){e.stopPropagation()});// prevent node movement when selecting text in
the input field
                upd();
            }
        );
      return node
         .addInput(listIn)
         .addControl(resultControl)
         .addOutput(resultOut);
    }
    });

var maxComp = new D3NE.Component("Максимум", {
    builder(node) {
        node.nameFunction = "max";
        node.dataType = {0:"int"};
        node.describe = "mas_c_max";
      var sizeIn = new D3NE.Input("Размерность", numSocket);
      var arrayIn = new D3NE.Input("Массив", arraySocket);
      var resultOut = new D3NE.Output("Результат", numSocket);
      var resultControl = new D3NE.Control('<input type="string">',
      (el, c) => {
         el.value = c.getData('0') || "имя выходного значения";

         function upd() {
            c.putData("0", el.value);
         }
         el.addEventListener("input", ()=>{
            upd();
            editor.eventListener.trigger("change");
         });
         el.addEventListener("mousedown", function(e){e.stopPropagation()});//
prevent node movement when selecting text in the input field
        upd();
      }
   );
      return node
      .addInput(arrayIn)
      .addInput(sizeIn)
         .addOutput(resultOut)
         .addControl(resultControl);
    }});


var url = new URL(window.location.href);
var language = url.searchParams.get("language");
var structure = url.searchParams.get("structure");
if (language == null || structure == null)
{
```

```
    language = "0";
    structure = "0";
}
objSel = document.getElementById("Combobox");

switch(language)
{
        case "0":
        //alert("вот этот свитч");
        objSel.options.length = 0;
        objSel.options[0] = new Option("Массив","0");
        objSel.options[1] = new Option("Список","1");
        objSel.options[2] = new Option("Строка","2");
        objSel.options[3] = new Option("Текстовый файл","3");
        objSel.options[4] = new Option("Бинарный файл","4");

        ChangeStructure();
        break;
        case "1":
        objSel.options.length = 0;
        objSel.options[0] = new Option("Список","0");
        structure = "0";
        ChangeStructure();
        break;
        case "2":
        objSel.options.length = 0;
        objSel.options[0] = new Option("Массив","0");
        objSel.options[1] = new Option("Список","1");
        objSel.options[2] = new Option("Строка","2");
        objSel.options[3] = new Option("Текстовый файл","3");
        objSel.options[4] = new Option("Бинарный файл","4");
        ChangeStructure();
        break;
    }
$("#Change-Language").val(language);
$("#Combobox").val(structure);
function Change(){

window.location.replace("file:///C:/Users/%D0%90%D0%BD%D0%B6%D0%B5%D0%BB%D0%B8%D0
%BA%D0%B0/Desktop/diploma/diploma/idef0.ru/1.html" +
     "?structure=" + $("#Combobox")[0].selectedIndex + "&language="+ $("#Change-
Language")[0].selectedIndex);
}

function ChooseLanguage()
{
    if (!document.getElementById("choose-language").aria)
    {
        document.getElementById("nav-language").className = "nav-item dropdown
show";
```

```
        document.getElementById("menu-language").className = "dropdown-menu
show";
        document.getElementById("choose-language").aria = true;
    }
    else
    {
        document.getElementById("nav-language").className = "nav-item dropdown";
        document.getElementById("menu-language").className = "dropdown-menu";
        document.getElementById("choose-language").aria = false;
    }
  //alert("tuk");
}
```

generate.js

```
function find(array, value) {
    if ([].indexOf) {

        var find = function(array, value) {
          return array.indexOf(value);
        }
      } else {
        var find = function(array, value) {
          for (var i = 0; i < array.length; i++) {
            if (array[i] === value) return i;
          }
          return -1;
        }
      }
    for (var i = 0; i < array.length; i++) {
      if (array[i] === value) return i;
    }
    return -1;
  }

function Generate(){
    listComplited = [];
    listTemp = [];
    listAll = [];
    notification();
    listAll = editor.toJSON().nodes;
    noerror = true;
    compliteData();
    if (!noerror)
    {
        codestr = "ошибка в диаграмме";
        //alert("тут ошибка");
    }
    else
    {
```

```
        if (!aisle())
            codestr = "ошибка в диаграмме";
        else
            toEnd();
        //alert(main);
        var str = $("#ProgramCode");
        str.html(codestr);
    }
}

function notification(){
    switch($("#Change-Language")[0].selectedIndex){
        case 0:
        codestr = "";
        main = 'void main() {\n \tsetlocale(LC_ALL, "");\n';
        break;
        case 1:
        predicate = "implement main\n\
    open core, console\n\
class predicates\n";
    codestr ="clauses\n";
    main = "\n\
    run() :-\n\
    init(),\n"
        break;
        ...}
}

function toEnd(){
    switch($("#Change-Language")[0].selectedIndex){
        case 0:
        codestr += main + "\t _getch();\n}";
        break;
        case 1:
        main += '\
    !,\n\
    nl,\n\
    _ = readline().\n\
    run() :-\n\
        console::write("Fail").\n\
end implement main\n';
        codestr =predicate + codestr + main;
        break;
        ...
    }
}

function compliteData(){
    for (var i in listAll){
        if (Object.keys(listAll[i].data).length == 0)
```

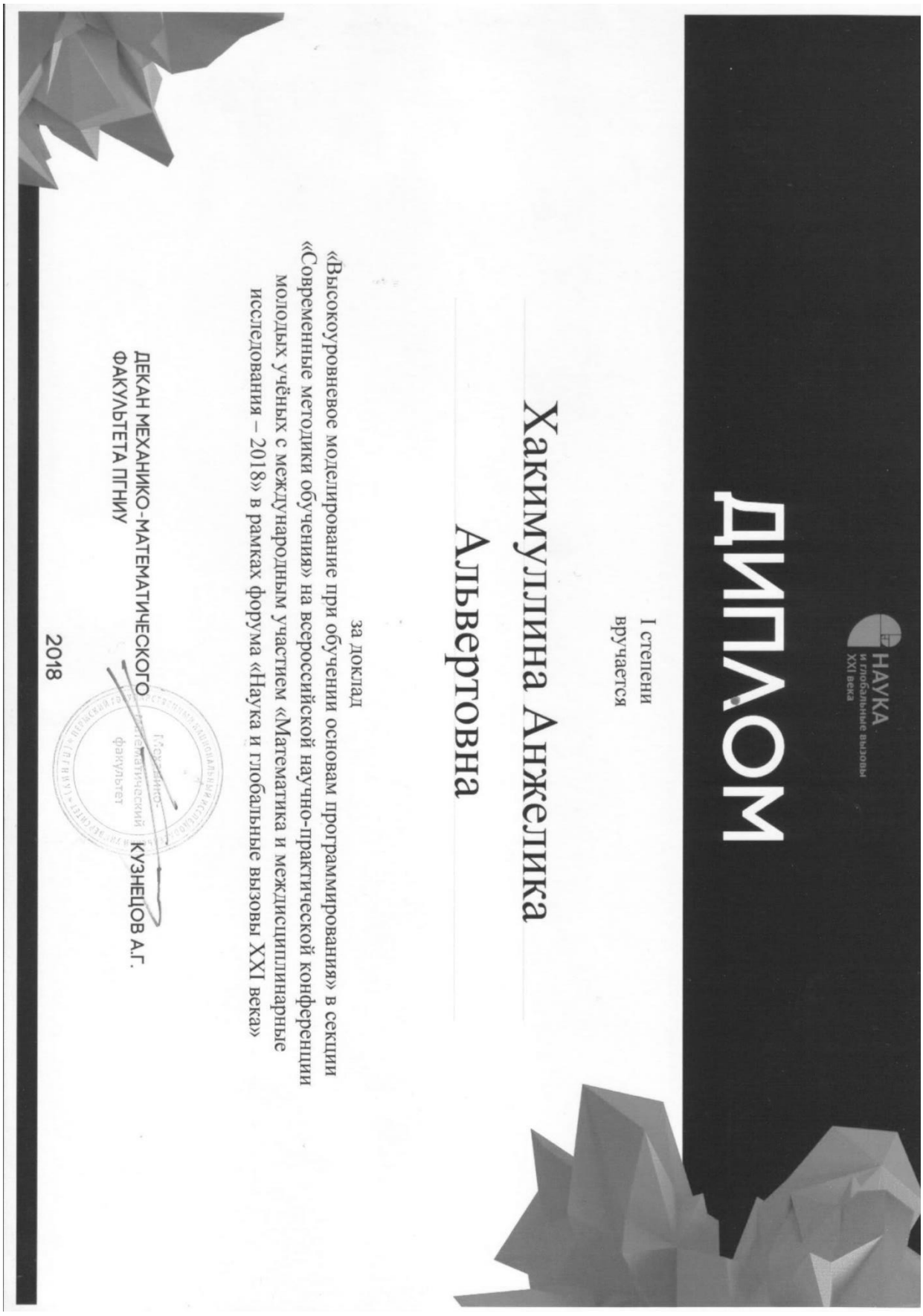
```
        {
            if(find(listTemp,listAll[i]) == -1)
            listTemp.push(listAll[i]);
            while (listTemp.length != 0 && noerror)
            {
                noerror = addData();
            }
        }
        if (!noerror) return noerror;
    }
    listTemp = [];
    return true;
 }

function addData(){
    tempNode = listTemp[listTemp.length - 1];
    for(var j=0;j<tempNode.inputs.length && noerror;j++){
        if (tempNode.inputs[j].connections.length == 0)
            {
                noerror = false;
                return noerror;
            }
        if
(Object.keys(listAll[tempNode.inputs[j].connections[0].node].data).length == 0){
            listTemp.push(listAll[tempNode.inputs[j].connections[0].node]);
        }
        else
            tempNode.data[j]=listAll[tempNode.inputs[j].connections[0].node].data
[tempNode.inputs[j].connections[0].output];
    }
}
 function aisle(){
    notError = true;
    for (var i in listAll)
    {
        if(find(listTemp,listAll[i]) == -1 && find(listComplited,listAll[i]) == -
1)
        {
            listTemp.push(listAll[i]);
            while (listTemp.length != 0 && notError){
                notError = tempNodeWork();
            }
        }
        if (!notError) return notError;
    }
    return notError;
}

function tempNodeWork(){
    var tempNode = listTemp[listTemp.length - 1];
```

```
    if (tempNode.inputs.length == 0)
    {
        SwitchWithoutInput(tempNode);
        listTemp.splice(listTemp.indexOf(tempNode),1); //удаляю из listTemp
        listComplited.push(tempNode);
    }
    else {
        var inputsComplited = true; //все входы ведут к выполненным блокам
        for (var i=0;i<tempNode.inputs.length;i++)
        {
            if (tempNode.inputs[i].connections.length == 0)
            return false;
            for (var j=0;j<tempNode.inputs[i].connections.length;j++)
            {
                if (find(listComplited,
                     listAll[tempNode.inputs[i].connections[j].node]) == -1)
                     {
                        inputsComplited = false;
                        if
(find(listTemp,listAll[tempNode.inputs[i].connections[j].node]) == -1)
                        listTemp.push(listAll[tempNode.inputs[i].connections[j].n
ode]);
                     }
            }
        }
        if (inputsComplited)
        {
            SwitchFunction(tempNode);
            listTemp.splice(listTemp.indexOf(tempNode),1);
            listComplited.push(tempNode);
        }
    }
    return true;
}
function SwitchWithoutInput(tempNode){
    switch(tempNode.nameFunction){
        case 'array_c':
        main += "\t" + tempNode.dataType[1] +" "+ tempNode.data[1] + " = N;\n";
        main += "\t" +  tempNode.dataType[0]+" "+tempNode.data[0] + " = new
int["+ tempNode.data[1] +"];\n";
        break;
        case 'list_prolog':
        main +="    "+ tempNode.data[0].toUpperCase() + "=[1, 2, 3, 4, 5, 20, 9,
10, 18, 1],\n";
        break;
        ...
    }
}
```

Приложение Е. Диплом конференции



НАУКА и глобальные вызовы XXI века

ДИПЛОМ

I степени
вручается

Хакимуллина Анжелика
Альвертовна

за доклад
«Высокоуровневое моделирование при обучении основам программирования» в секции «Современные методики обучения» на всероссийской научно-практической конференции молодых учёных с международным участием «Математика и междисциплинарные исследования – 2018» в рамках форума «Наука и глобальные вызовы XXI века»

ДЕКАН МЕХАНИКО-МАТЕМАТИЧЕСКОГО ФАКУЛЬТЕТА ПГНИУ КУЗНЕЦОВ А.Г.

2018