Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01. (ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Библиотека прикладных модулей для адаптивной системы моделирования

Студент ИУ6-53Б (Группа) — 15.12.2021 (Подпись, дата) — А.А. Бушев (И.О. Фамилия)

Руководитель курсовой работы (старший преподаватель «Компьютерные системы и сети») — 15.12.2021 (Подпись, дата) — С.С. Данилюк (И.О. Фамилия)

50

Москва, 2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ6__
(Индекс)
_____ А.В. Пролетарский
(И.О.Фамилия)
« 13 » сентября 2021 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине ____Технология разработки программных систем_____

Студент группы ____ИУ6-53Б_____

_____Бушев Антон Алексеевич_____
(Фамилия, имя, отчество)

Тема курсовой работы ____Библиотека прикладных модулей для адаптивной системы моделирования_____
_____

Направленность КР (учебная, исследовательская, практическая, производственная, др.)
_учебная_____

Источник тематики (кафедра, предприятие, НИР) ____кафедра_____

График выполнения КР: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Техническое задание ___см. техническое задание в приложении А_____

Оформление курсовой работы:

1. Расчетно-пояснительная записка (РПЗ) на 25-30 листах формата А4.
2. Техническое задание на 5-9 листах формата А4 – оформляется в качестве приложения А к РПЗ.
3. Руководство пользователя на 6-8 листах формата А4 – оформляется в качестве приложения Б к РПЗ (если предусмотрено в техническом задании).
4. Графический и иллюстративный материал оформляется в виде рисунков и помещается в РПЗ.
5. Все материалы и исходный текст программы закачать на страницу дисциплины на сайте кафедры.

Дата выдачи задания «_1_ » _сентября_ 2021 г.

Руководитель курсовой работы _____13.09.2021_____С.С. Данилюк___
(Подпись, дата) (И.О.Фамилия)

Студент _____13.09.2021_____А.А. Бушев_____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

РЕФЕРАТ

РПЗ 62 страницы, 3 части, 15 рисунков, 1 таблиц, 5 источников, 4 приложения

БИБЛИОТЕКА, ОДУ, SIMODO, МОДЕЛИРОВАНИЕ

Объектом разработки является библиотека прикладных модулей для адаптивной системы моделирования SIMODO.

Цель работы – проектирование и реализация библиотеки прикладных модулей для адаптированной системы моделирования, предназначенной для упрощения создания моделей.

В результате разработки была спроектирована библиотека прикладных модулей для адаптивной системы моделирования, позволяющая упростить процесс формирования моделей, а также было проведено тестирование данного программного продукта методами структурного контроля и модульного тестирования.

Библиотека прикладных модулей для адаптивной системы моделирования предназначена для использования пользователями адаптивной системы моделирования SIMODO.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ТЗ – техническое задание.

ПП – программный продукт.

Стек технологий – набор инструментов, применяющийся при работе в проектах и включающий языки программирования, фреймворки, СУБД

ОДУ – обыкновенное дифференциальное уравнение

СДУ – система дифференциальных уравнений

io – модуль ввода-вывода

tf – модуль табличных функций

math – математический модуль

scene – модуль сцены

ЭВМ – электронно-вычислительная машина

ВВЕДЕНИЕ

Работа посвящена проектированию и разработке библиотеки прикладных модулей для адаптивной системы моделирования (далее «библиотека»). Разработанная библиотека может быть использована пользователями адаптивной системы моделирования SIMODO, которая разрабатывается на кафедре ИУ6 МГТУ им. Н.Э. Баумана. Программный продукт позволяет упростить создание моделей и процесс моделирования, используя модули библиотеки при формировании моделей на специализированных языках.

Актуальность данной разработки обосновывается отсутствием аналогов библиотек для адаптивной системы моделирования SIMODO.

Предполагается, что потребителями данного программного продукта должны быть пользователи, работающие с адаптивной системой моделирования SIMODO.

1. Анализ требований и уточнение спецификаций

1.1 Анализ задания и выбор технологии, языка и среды разработки

Модуль ввода-вывода (io).

Для реализации модуля ввода-вывода следует выбрать соответствующую библиотеку. В соответствии с ТЗ (прил. А, пункт 4.5.2) языком программирования при реализации библиотеки должен быть C++1z. Существует три наиболее известных библиотек ввода-вывода C++ [4-6]:

1. stdio
2. STL
3. Boost.Asio

Традиционная библиотека stdio унаследована языком C++ от языка C. Данная библиотека проста в обращении, но является опасной ввиду того, что на программиста возлагаются проверки выхода за границы массивов и переполнения буферных массивов, при этом ошибки возникающие при невнимательности могут привести повреждению файловой системы[6].

Стандартная библиотека шаблонов C++ STL обладает более удобным интерфейсом и изначально проектировалась как безопасная замена замена стандартной библиотеке C[6]. Библиотека ввода-вывода C++ STL была стабильной с первой версии[4].

Библиотека ввода-вывода Boost.Asio обладает расширенным функционалом по сравнению с STL, например, асинхронный ввод-вывод. Библиотека Boost.Asio является экспериментальной[5], поэтому стабильность данной библиотеки недостаточна для разрабатываемой библиотеки.

Стандартная библиотека шаблонов C++ STL обладает преимуществом перед двумя другими вариантами: она сочетает в себе стабильность и безопасность, что является основанием для выбора данной библиотеки ввода-вывода при реализации модуля ввода-вывода.

Модуль табличных функций (tf).

Табличная функция – это функция, заданная в табличной форме. Она может быть сделана непрерывной с помощью интерполяции и экстраполяции[8].

Табличные функции обычно используются для задания сложных нелинейных зависимостей, которые не могут быть описаны с помощью стандартных функций, или для приведения собранных с какой-то периодичностью и заданных в виде таблицы экспериментальных данных к непрерывному виду.

Табличная функция работает следующим образом: пользователь задаёт функцию путём задания таблицы значений, в которой первые столбцы соответствуют аргументам функции, а последний столбец соответствует значению функции в этой точке.

Приближение функции, заданной таблично, другой непрерывной функцией называется аппроксимацией[9]. Существует три вида решения аппроксимации[9]:

1. Линейная интерполяция
2. Интерполяционный полином Лагранжа
3. Интерполяция сплайнами

При линейной интерполяции табличные значения функции в смежных узловых точках соединяются отрезками прямых, и функция приближается ломаной с вершинами в данных точках. Уравнения каждого отрезка ломаной в общем случае разные. Это наиболее простой и достаточно распространённый способ интерполяции[9], однако при линейной интерполяции интерполирующая функция имеет изломы в узлах интерполяции и разрывы значений производных, а погрешность интерполяции определяется расстояниями между узлами интерполяции[9].

Погрешность линейной интерполяции обусловлена тем, что график интерполирующей функции имеет изломы в узлах интерполяции. Эти изломы можно устранить, если в качестве интерполирующей использовать такую функцию, график которой представляет собой плавную кривую (например, полином), проходящий точно через заданные в таблицей точки. Интерполяция полиномом Лагранжа дает высокую точность, если значения функции в смежных узлах, заданные в таблице, изменяются достаточно медленно[9].

Полиномиальная интерполяция не всегда дает удовлетворительные результаты при аппроксимации функций. Интерполирующая функция может иметь значительные отклонения между узлами[7]. Увеличение степени интерполяционного многочлена не всегда приводит к уменьшению погрешности. При этом поведение полинома в окрестности какой-либо точки определяет его поведение в целом.

На практике для проведения гладких кривых через узловые значения функции используют гибкую упругую линейку, совмещая её с заданными точками. Математическая теория такой аппроксимации называется теорией сплайн-функций (от английского слова spline − рейка, линейка)[9]. Сплайн-интерполяцию выгодно применять при небольшом числе узловых точек (до 5 −7)[9].

Итог: точность линейной интерполяции зависит от таблицы табличной функции, интерполяционный полином Лагранжа обладает высокой точностью при небольших изменениях на большом количестве заданных точек, интерполяция сплайнами выгодна при

небольшом числе узловых точек. Ввиду неточности линейной интерполяции и эффективности интерполяции сплайнами на малом числе точек, при реализации модуля tf будет использоваться интерполяционный полином Лагранжа, потому что этот метод позволяет однозначно задать функцию, аппроксимирующую табличную.

Интерполяционный многочлен Лагранжа – многочлен минимальной степени, принимающий заданные значения в заданном наборе точек[7].

Пусть задана $n+1$ пара чисел $(x_0, y_0), (x_0, y_0), ..., (x_n, y_n)$ где все x_j различны[7-9]. Требуется построить многочлен $L(x)$ степени не более n, для которого $L(x_j) = y_j$.

Ж. Л. Лагранж предложил следующий способ вычисления таких многочленов[7]:

$$L(x) = \sum_{i=0}^{n} y_i \, l_i(x), \tag{1}$$

где базисные полиномы l_i определяются по формуле[7]

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j} = \frac{x-x_0}{x_i-x_0} \cdot \ldots \cdot \frac{x-x_{i-1}}{x_i-x_{i-1}} \cdot \frac{x-x_{i+1}}{x_i-x_{i+1}} \cdot \ldots \cdot \frac{x-x_n}{x_i-x_n}. \tag{2}$$

Итого значение функции от одного аргумента в точке будет равно[7]

$$L(x) = \sum_{i=0}^{n} y_i \left(\frac{x-x_0}{x_i-x_0} \cdot \ldots \cdot \frac{x-x_{i-1}}{x_i-x_{i-1}} \cdot \frac{x-x_{i+1}}{x_i-x_{i+1}} \cdot \ldots \cdot \frac{x-x_n}{x_i-x_n} \right) \tag{3}$$

Модуль сцены (scene).

При математическом моделировании ряда технических устройств используются системы дифференциальных нелинейных уравнений. Такие модели используются не только в технике, они находят применение в экономике, химии, биологии, медицине, управлении[10].

Исследование функционирования таких устройств требуют решения указанных систем уравнений. Поскольку основная часть таких уравнений являются нелинейными и нестационарными, часто невозможно получить их аналитическое решение.

Возникает необходимость использовать численные методы, наиболее известными из которых является метод Рунге-Кутты и метод Фельберга с переменным шагом.

Классический метод Рунге-Кутты обладает фиксированным шагом и точностью четвёртого порядка, но он прост в реализации. Метод Фельберга обладает точностью пятого порядка и переменным шагом, что усложняет формулы, однако приводит к более точным результатам[10-11].

Было решено, что в реализации модуля сцены будет использоваться классический метод Рунге-Кутты, потому что он проще в реализации, а разница в точности между четвёртым и пятым порядком не носит критический характер.

Для одного дифференциального уравнения n – го порядка, задача Коши состоит в нахождении функции, удовлетворяющей равенству[10]:

$$y^{(n)} = f\left(t, y', \dots, y^{(n-1)}\right) \tag{4}$$

и начальным условиям

$$y(t_0) = y_1^0, y'(t_0) = y_2^0, \dots, y^{(n-1)}(t_0) = y_n^0$$

Перед решением эта задача должна быть переписана в виде следующей СДУ[10]

$$\begin{aligned} &\frac{dy_1}{dt} = f_1(y_1, y_2, \dots, y_n, t) \\ &\frac{dy_2}{dt} = f_2(y_1, y_2, \dots, y_n, t) \\ &\dots \\ &\frac{dy_n}{dt} = f_n(y_1, y_2, \dots, y_n, t) \end{aligned} \tag{5}$$

С начальными условиями

$$y_1(t_0) = y_1^0, y_2(t_0) = y_2^0, \dots, y_n(t_0) = y_n^0 \tag{6}$$

По методу Рунге—Кутта четвёртого порядка общее решение имеет вид[10-11]

$$y^{n+1} = y^n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4, \tag{7}$$

где расчётные коэффициенты равны[11]

$$k_1 = \tau f(t_n, y^n), \tag{8}$$

$$k_2 = \tau f\left(t_n + \frac{\tau}{2}, y^n + \frac{1}{2}k_1\right), \tag{9}$$

$$k_3 = \tau f\left(t_n + \frac{\tau}{2}, y^n + \frac{1}{2}k_2\right), \tag{10}$$

$$k_4 = \tau f(t_n + \tau, y^n + k_3). \tag{11}$$

Для реализации модуля ввода-вывода была выбрана стандартная библиотека C++ STL. Для модуля табличных функций был выбран метод интерполяционного полинома Лагранжа. Для модуля сцены был выбран классический метод Рунге-Кутты 4 порядка.

Выбор технологии, языка и среды разработки.

Разрабатываемая библиотека состоит из четырёх программных модулей: io (модуль ввода-вывода), math (математический модуль), tf (модуль табличных функций), scene (модуль сцены). При разработке этих частей использовались различные технологии программирования.

В качестве технологии программирования модулей io, tf, math было выбрано функционально программирование, в котором процесс вычисления трактуется как вычисление значений функций в математическом понимании последних. Данный подход был выбран, потому что предметная область модулей io, tf, math либо носит математический характер (tf, math), либо предполагает, что информации с которой работает модуль уделяется основное внимание (io).

Для программирования модуля сцены было выбрано объектно-ориентированное программирование программирование, потому что предметная область модуля – объектная, и она содержит явные взаимодействия между сущностями.

Объектно-ориентированное программирование определяется как технология создания сложного программного обеспечения, основанная на представлении программы в виде совокупности объектов. Взаимодействие программных объектов в системе осуществляется путём передачи сообщений.

В качестве языка программирования был выбран C++ стандарта C++1z, так как это является требованием к информационной и программной совместимости в соответствии с ТЗ (прил. А, пункт 4.5.2). Для создания динамически подключаемых библиотек использовался Boost – собрание библиотек классов, использующих функциональность языка C++ и предоставляющих удобный кроссплатформенный высокоуровневый интерфейс для лаконичного кодирования различных повседневных подзадач программирования.

Разработка библиотеки производилась в среде CLion, которая включает в себя средства автоматического завершения кода, интерактивную подсветку и многие другие функции для упрощения процесса разработки[12]. Из альтернатив можно выделить QtCreator и CodeBlocks, обладающие бесплатной версией для открытого программного обеспечения, однако бесплатность CLion для студентов нивелирует преимущество альтернатив[12-14]. QtCreator также позволяет разрабатывать графические интерфейсы, что не является востребованным для библиотеки прикладных модулей для адаптивной среды моделирования.

Решающим параметром CLion является более проработанное автодополнение кода ввиду собственного анализатора кода[12].

Тестирование проводилось в среде SIMODO edit, которая разработана на кафедре ИУ6 МГТУ им. Н.Э. Баумана и которая включает в себя автоматического завершения кода, интерактивную подсветку и многие другие функции для упрощения процесса разработки на языке SBL, который разрабатывается на кафедре ИУ6 МГТУ им. Н.Э. Баумана. Альтернативы среде SIMODO edit не могут быть задействованы, потому что язык SBL поддерживается только средой SIMODO edit.

1.2 Выбор модели жизненного цикла программного обеспечения

При разработке программных продуктов широко применяется метод прототипирования, который базируется на создании прототипов. Прототипом называют действующий программный продукт, реализующий отдельные функции и внешние интерфейсы разрабатываемого программного обеспечения.

В связи с этим, при разработке использована спиральная модель жизненного цикла создаваемого программного продукта, которая основывается на методе прототипирования. Спиральная модель позволяет быстро спроектировать, реализовать и протестировать первый рабочий прототип, который можно показать для дальнейшего уточнения требуемых условий и характеристик проекта.

Все принимаемые решения апробируются путём создания прототипов. На завершающем шаге каждого витка спирали создаётся версия, на которой уточняются все требуемые условия и характеристики проекта и планируется следующая итерация. Также, спиральная модель облегчает дальнейшее создание новых версий программного продукта.

1.3 Разработка концептуальной диаграммы классов модуля scene

При анализе предметной области и технического задания были выявлены следующие основные объекты:

- Сцена
- Актор
- Степпер
- Шаг

На их основе была построена концептуальная диаграмма классов, которая продемонстрирована на рисунке 1.

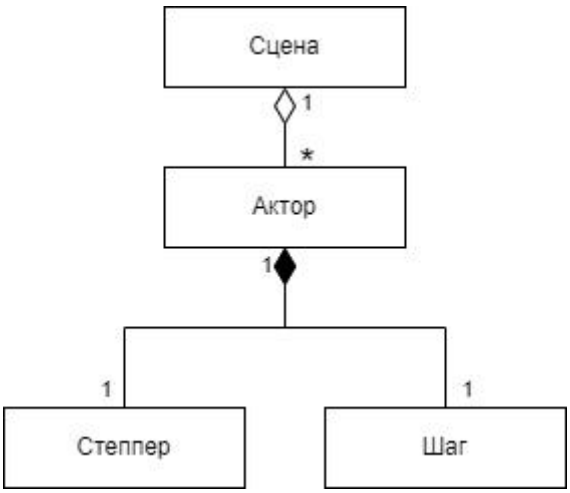

Рисунок 1 – Концептуальная диаграмма классов модуля scene

Сцена – контекст моделирования; управляет процессом моделирования. Актор – непосредственный объект моделирования, то есть акторы, находясь в сцене, моделируют определённое поведение, а сцена управляет ими. Степпер – непосредственный исполнитель шага моделирования. От реализации степпера зависит то, какое поведение моделирует актор. Актор обладает определённым состоянием, которое отражает шаг. Степпер позволяет получить следующий шаг моделирования, чтобы обновить состояние актора.

Итого, сцена может быть пуста или содержать в себе акторов; акторы обладают состоянием ввиде шага и поведение в виде степпера.

2. Проектирование структуры и компонентов программного продукта

2.1. Разработка структурной схемы программного продукта

Процесс проектирования программного обеспечения начинается с уточнения его структуры, т.е. определения структурных компонентов и связей между ними. Результат уточнения структуры может быть представлен в виде структурной схемы программного продукта и описания (спецификаций) компонентов. Разработку структурной схемы программного продукта выполняют методом пошаговой детализации.

Исходя из задач и функций, выполняемых библиотекой в целом и её модулями, была получена структурная схема модулей библиотеки. Структурная схема модулей показана на рисунках 2-5.

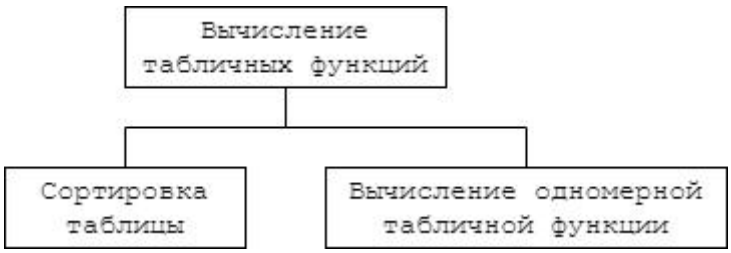

Рисунок 2 – Схема структурная программного обеспечения модуля tf

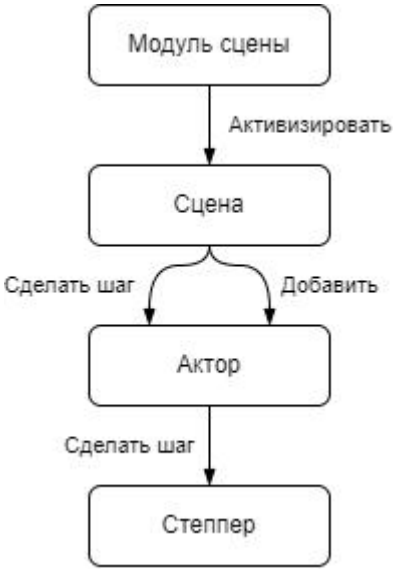

Рисунок 3 – Схема структурная программного обеспечения модуля scene

Модуль tf («Вычисление табличных функций») содержит в себе две подпрограммы: подпрограмму «Сортировка таблицы» и подпрограмму «Вычисление одномерной табличной функции». Подпрограмма «Сортировка таблицы» сортирует таблицу, которой задана функция, значение которой в заданной точке ищется. Подпрограмма «Вычисление одномерной табличной функции» вычисляет значение табличной функции от одного аргумента.

Модуль scene («Модуль сцены») предоставляет интерфейс для взаимодействия со сценой вне модуля. Также модуль scene активизирует сцену. Сцена может добавлять акторов и обновлять их состояние на каждом шаге моделирования. Актор обновляет своё состояние через выполнение шага степпером.

Модуль io содержит следующие функции:

- «print» выводит текстовую информацию стандартный поток вывода.
- «printE» является версией функции «print», возвращающей код ошибки.
- «printLine» выводит текстовую информацию и символ конца строки в стандартный потов вывода.
- «printLineE» является версией функции «printLine», возвращающей код ошибки.
- «writeFile» записывает текстовую информацию в файл.

– «writeFileE» является версией функции «writeFile», возвращающей код ошибки.
– «writeFileLine» записывает текстовую информацию и символ конца строки в файл.
– «writeFileLineE» является версией функции «writeFileLine», возвращающей код ошибки.
– «readLine» считывает строку из стандартного потока ввода.
– «readLineE» является версией функции «readLine», но помимо результата возвращает код ошибки.
– «readFile» считывает всё содержимое текстового файла.
– «readFileE» является версией функции «readLine», но помимо результата возвращает код ошибки.

Некоторые перечисленные функции используют внутренние подпрограммы, не входящие в интерфейс модуля io:

– «error» является подпрограммой генерирующей код ошибки на основе сообщения.
– «printFileValues» является подпрограммой, выводящей несколько текстовых значений в файл.
– «printFileValuesE» является версией подпрограммой «printFileValues», возвращающей код ошибки.
– «printValue» является подпрограммой, выводящей текстовое значение в файл
Модуль math. содержит в себе следующие функции:
– «тангенс» возвращает значение математической функции тангенс в точке, заданной аргументом функции.
– «косинус» возвращает значение математической функции косинус в точке, заданной аргументом функции.
– «синус» возвращает значение математической функции синус в точке, заданной аргументом функции.
– «знак выражения» возвращает «истина», если выражение, заданное аргументом функции, отрицательное, и «ложь» в противном случае.
– «арксинус» возвращает значение математической функции арксинус в точке, заданной аргументом функции.
– «арксинус от дроби» возвращает значение математической функции арксинус в точке, заданной аргументами функции в виде числителя и знаменателя дроби.
– «арккосинус» возвращает значение математической функции арккосинус в точке, заданной аргументом функции.

- «арккосинус от дроби» возвращает значение математической функции арккосинус в точке, заданной аргументами функции в виде числителя и знаменателя дроби.
- «арктангенс» возвращает значение математической функции арктангенс в точке, заданной аргументом функции.
- «арктангенс от дроби» возвращает значение математической функции арктангенс в точке, заданной аргументами функции в виде числителя и знаменателя дроби.
- «натуральный логарифм» возвращает значение математической функции натуральный логарифм в точке, заданной аргументом функции.
- «экспонента» возвращает значение математической экспоненциальной функции в точке, заданной аргументом функции.
- «четырёхквадрантный арктангенс» возвращает значение математической функции четырёхквадрантный арктангенс в точке, заданной аргументом функции.
- «квадратный корень» возвращает значение математической функции квадратный корень в точке, заданной аргументом функции.

2.2 Разработка структурной карты Константайна модулей io, tf, math

В соответствии с техническим заданием были разработаны карты Константайна для модулей ввода-вывода, табличных функций и математического модуля. Карты Константайна изображены на рисунках 6–8.

На картах Константайна показаны отношения модулей.

Модуль tf («Вычисление табличных функций») вызывает подпрограмму «Извлечение аргументов», передавая подпрограмме свои аргументы, и получает таблицу функции и точку на этой функции, значение в которой нужно найти. Подпрограмма «Извлечение аргументов» вызывает подпрограмму «Сортировка таблицы», передавая подпрограмме таблицу функции, и получает отсортированную таблицу функции. Модуль tf циклически вызывает подпрограмму «Вычисление одномерной табличной функции», передавая подпрограмме таблицу функции от одного аргумента и точку на этой функции, в которой нужно вычислить значение, и получает значение функции от одного аргумента в заданной точке.

Модуль io следующие функции: «print», «printE», «printLine», «printLineE», «writeFile», «writeFileE», «writeFileLine», «writeFileLineE», «readLine», «readLineE», «readFile», «readFileE». Этим функциям поступают на вход аргументы. Функции чтения «readLine», «readLineE», «readFile», «readFileE» возвращают результат операции чтения.

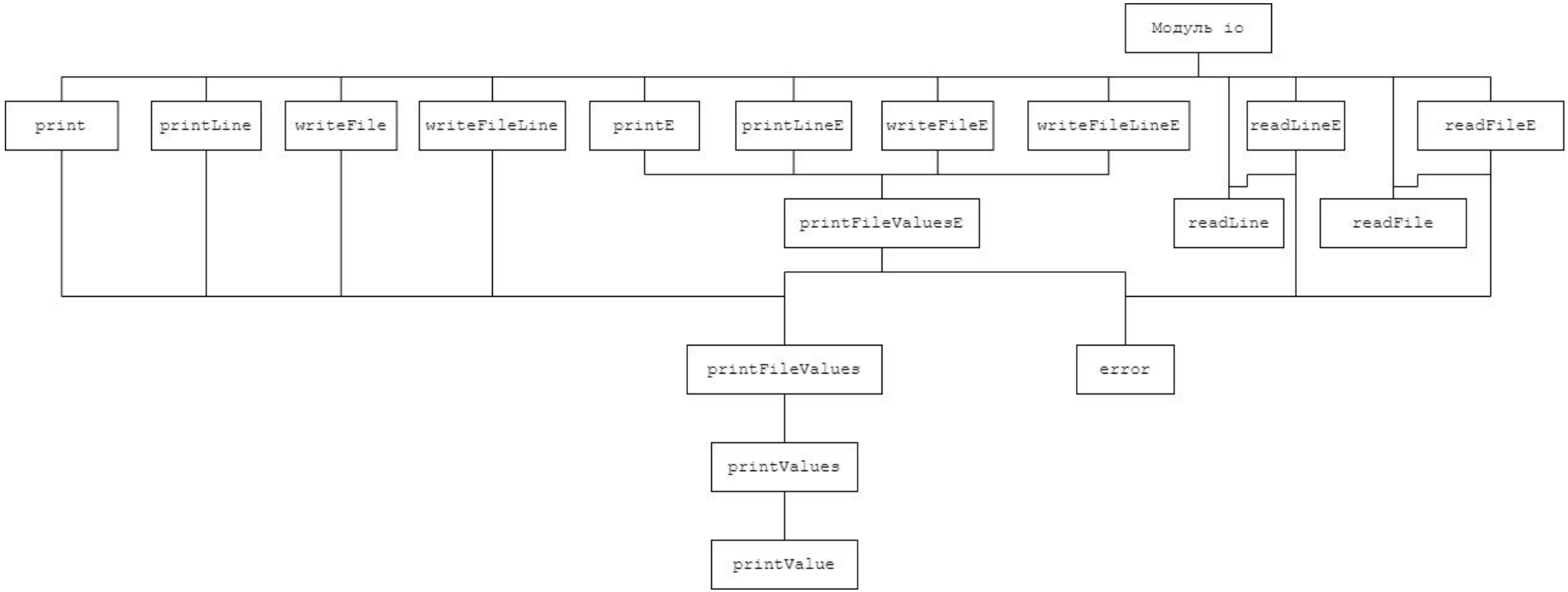

Рисунок 4 – Схема структурная программного обеспечения модуля io

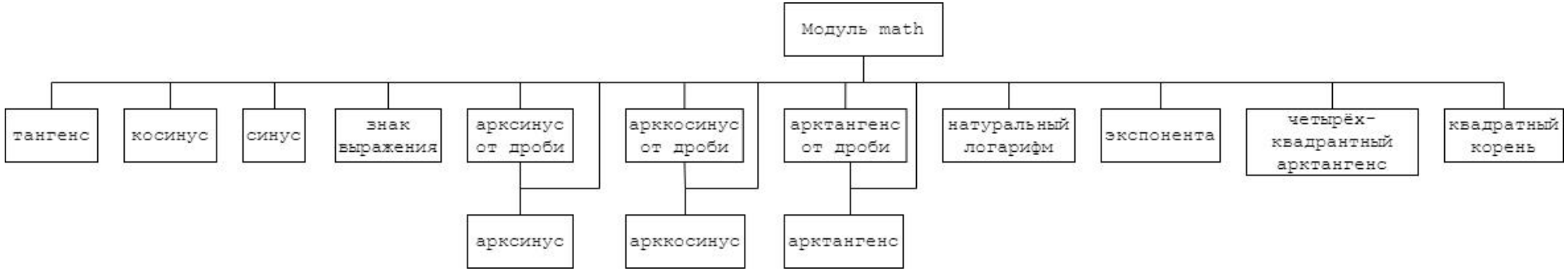

Рисунок 5 – Схема структурная программного обеспечения модуля math

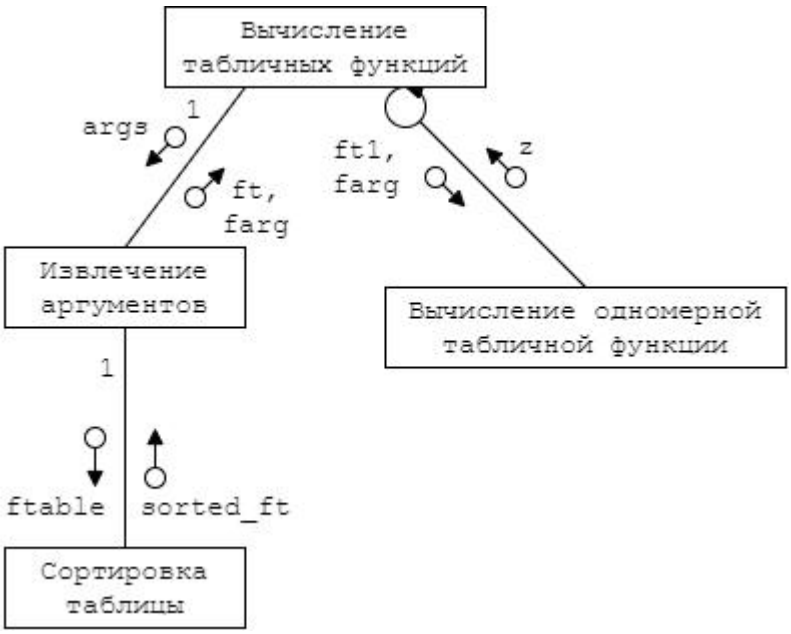

Рисунок 6 – Схема взаимодействия модулей Константайна модуля tf

Функции «readLineE» и «readFileE» вызывают свои версии функции, которые не проверяют на ошибки выполнение операций чтения. Функции с символом «E» возвращают код ошибки. Функции «readLineE», «readFileE» и подпрограмма «printFileValuesE» получают код ошибки из вызываемой подпрограммы «error», передавая подпрограмме сообщение об ошибке. Функции записи без проверки ошибок операций записи вызывают подпрограмму «printFileValues», передавая подпрограмме значения для записи в файл или стандартный поток вывода, опционально имя файла и режим записи (либо удалить файл, либо дополнить его). Функции записи с проверкой ошибок вызывают подпрограмму «printFileValuesE», передавая подпрограмме те же аргументы, что и функции записи без проверки ошибок. Подпрограмма «printFileValuesE» вызывает подпрограмму «printFileValues», передавая ей значения для записи, поступившие на вход. Подпрограмма «printFileValues» вызывает подпрограмму «printValues» передавая значения для записи, поступившие на вход. Подпрограмма «printValues» циклически вызывает подпрограмму «printValue», передавая подпрограмме значение для записи. Функции «readLine», «readFile» и подпрограмма «printValue» обращаются к библиотеке ввода-вывода для вызова операций чтения/записи.

Модуль math содержит в себе функции, которые обращаются к библиотеке математических функций, получают аргументы соответствующих математических функций и возвращают значение этих математических функций в точке, заданной аргументом, полученным на входе. Функции арк- математических функций от дроби получают на вход два аргумента, которые задают точку, в которой нужно найти значение соответствующей математической функции, в виде числителя и знаменателя дроби. Такие функции модуля вызывают соответствующие функции модуля от одного аргумента.

2.3 Разработка схем алгоритмов модулей io, tf

Структурная декомпозиция – частный случай процедурной, при структурной декомпозиции запрещены «циклические» или обратные вызовы подпрограмм. Схема структурной декомпозиции иерархическая и напоминает дерево, перевёрнутое корнем вверх. На верхнем уровне, в корне дерева находится основная программа, которая вызывает подпрограммы следующего уровня, а эти подпрограммы, в свою очередь, вызывают только подпрограммы более низких уровней. Вызов подпрограмм, расположенных в иерархии вызовов выше или на том же уровне, что и вызывающая подпрограмма, является циклическим и, следовательно, запрещён.

Разработанные схемы алгоритмов представлены на рисунках 9-10.

Основная функция модуля tf формирует наборы точек с координатами, которые совпадают по всем измерениям кроме одного, в котором будет вычислено значение одномерного «среза» изначальной многомерной табличной функции. Вычисление табличной функции от одного аргумента производится в соответствии с формулой (3).

Модуль io состоит из функций, которые выполняют узкую задачу, например, функции записи только извлекают необходимые аргументы или добавляют символ конца строки и передают их подпрограмме выполняющей открытие и закрытие файла или передачу стандартного потока вывода, вызывая подпрограмму, в цикле выводящую текстовые значения с помощью другой подпрограммы, которая уже работает напрямую с библиотекой ввода-вывода.

2.4 Разработка диаграммы классов предметной области модуля scene

В соответствии с концептуальной диаграммой классов предметной области и с инфологической моделью предметной области были спроектированы следующие классы, представленные на диаграмме классов.

Разработанная диаграмма классов представлена на рисунке 11.

Сцена – абстрактный класс, который содержит методы «addActor», «clearActors» и чистый виртуальный метод «start» и содержит поле времени текущего шага «time», конца моделирования «endTime», шага времени «deltaTime», функции обратного вызова «callback», акторов «actors». Актор содержит поле текущего состояния «state» и Степпер актора «stepper» и метод перехода на следующий шаг «calcStep». Шаг представляет из себя структуру с независимым аргументом «x» и значениями функций «y». Степпер является интерфейсом предоставляющий один чисто виртуальный метод «doStep».

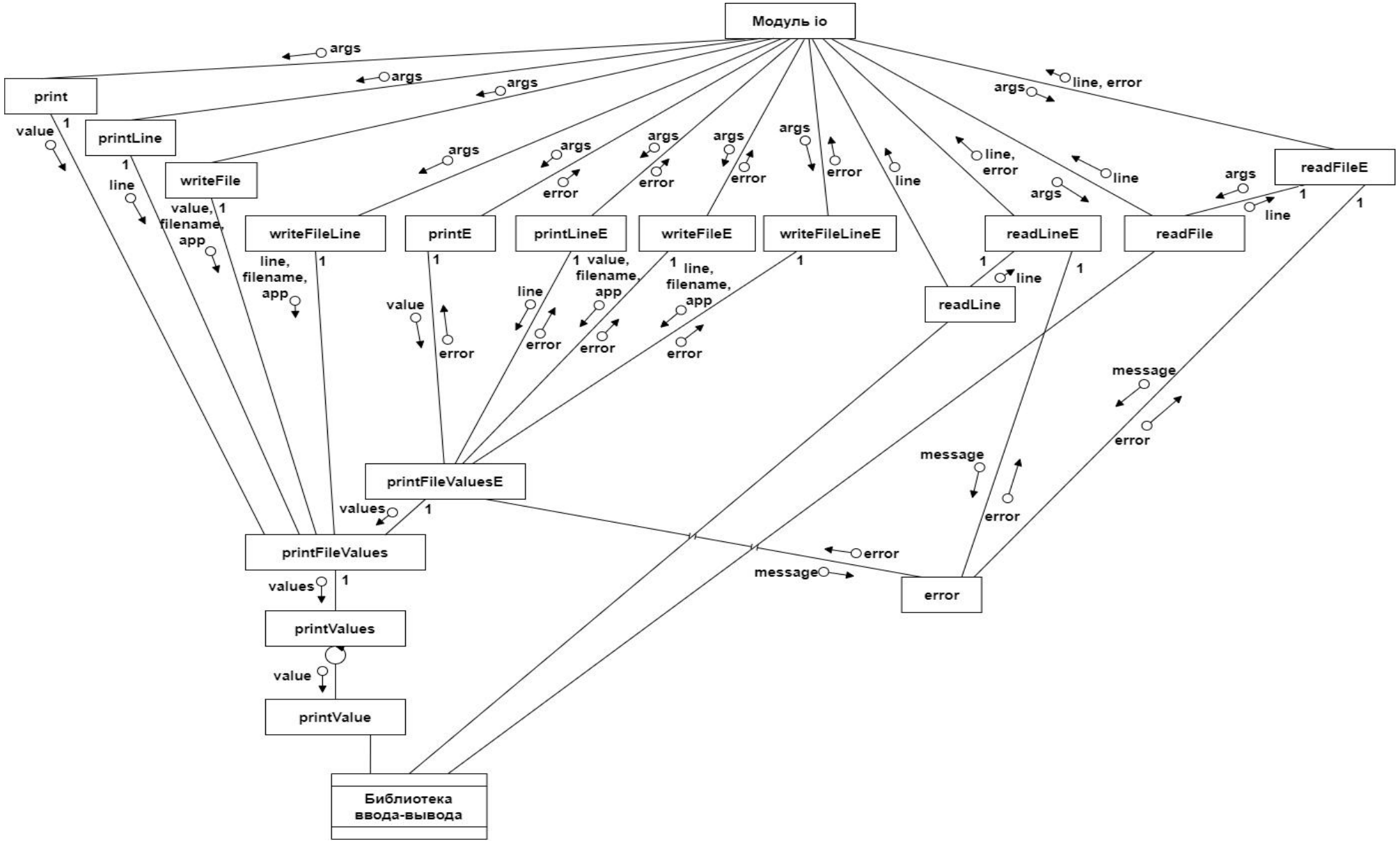

Рисунок 7 – Схема взаимодействия модулей Константайна модуля io

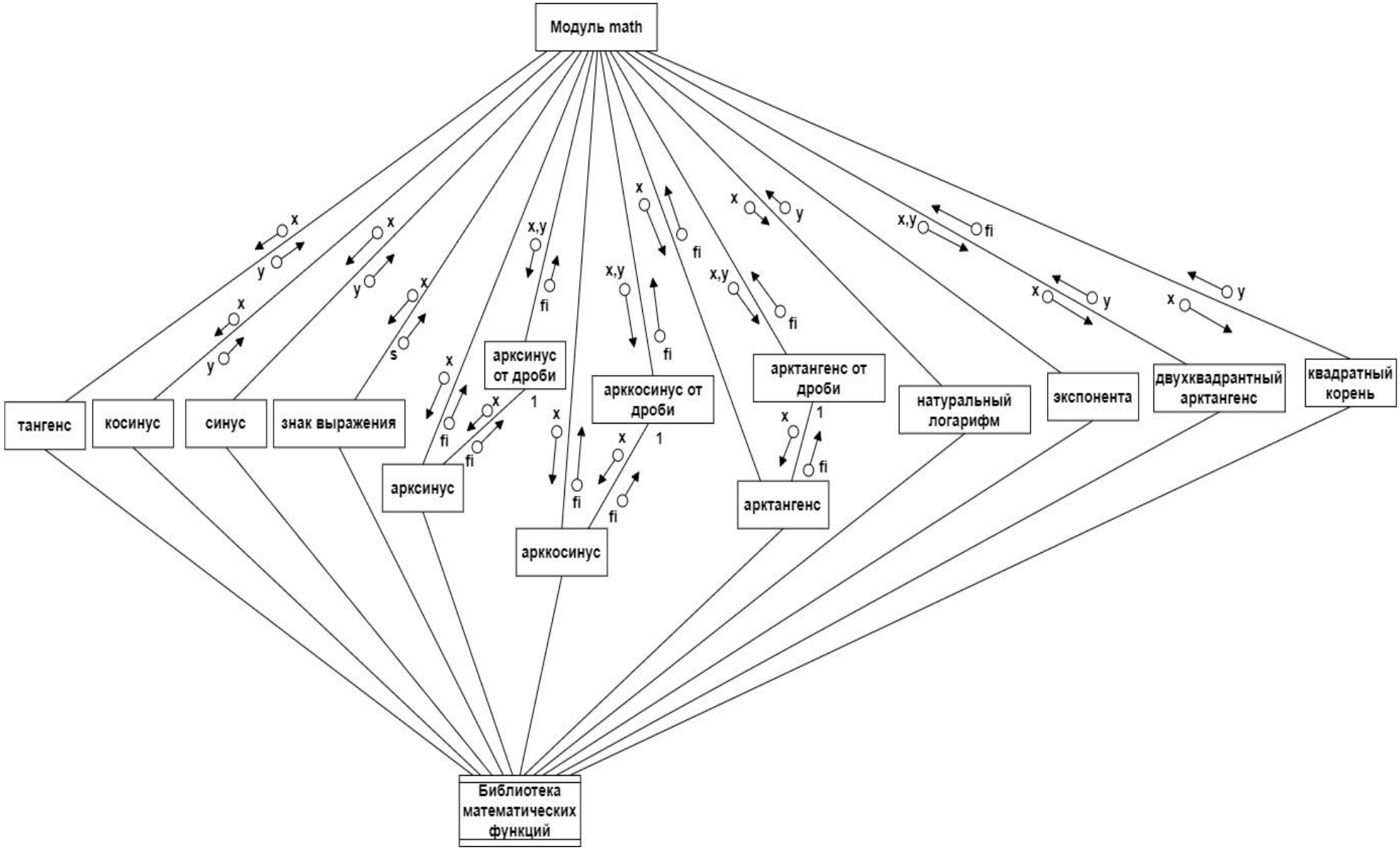

Рисунок 8 – Схема взаимодействия модулей Константайна модуля math

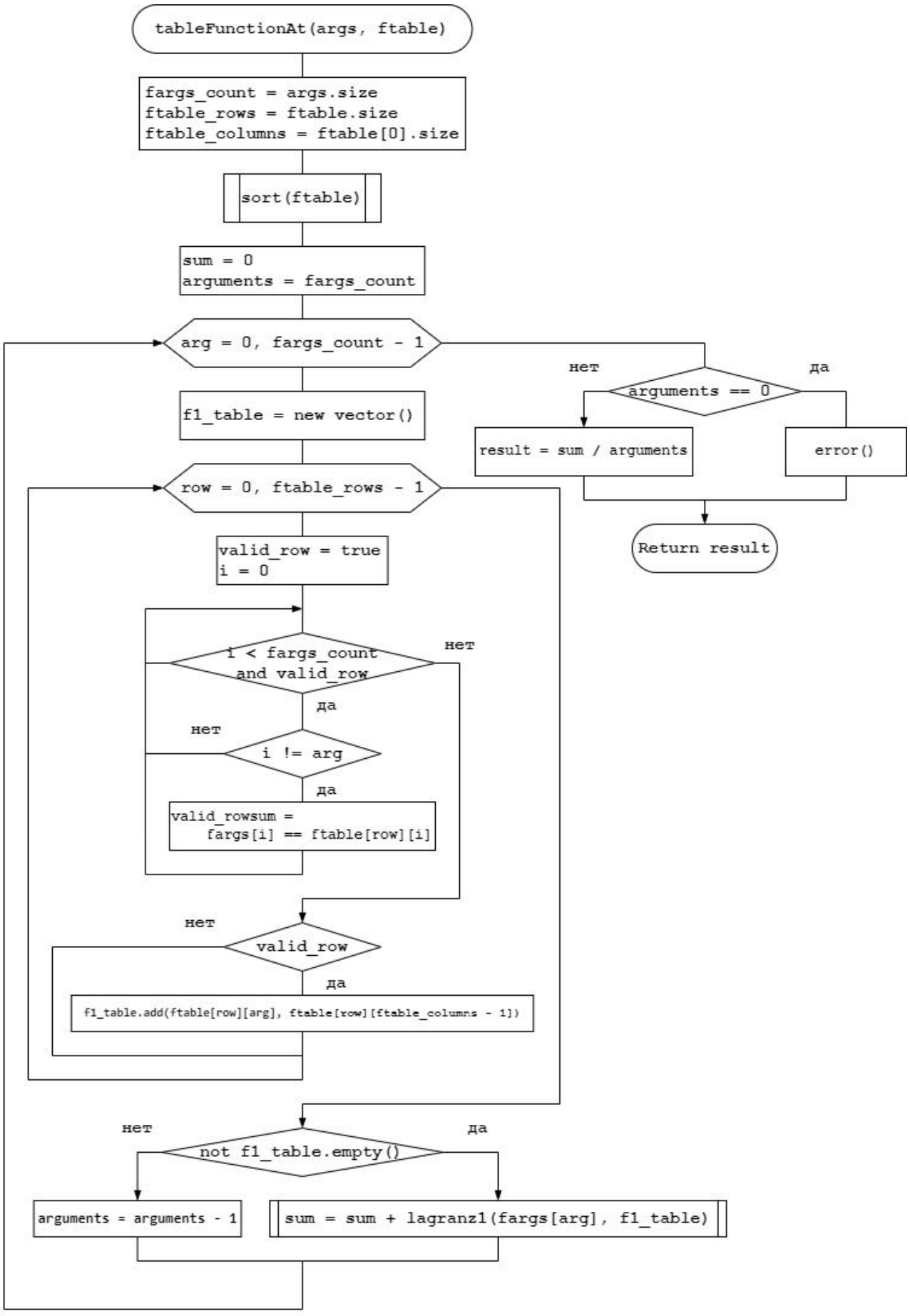

Рисунок 9 – Схемы алгоритмов модулей (подпрограмм) модуля tf

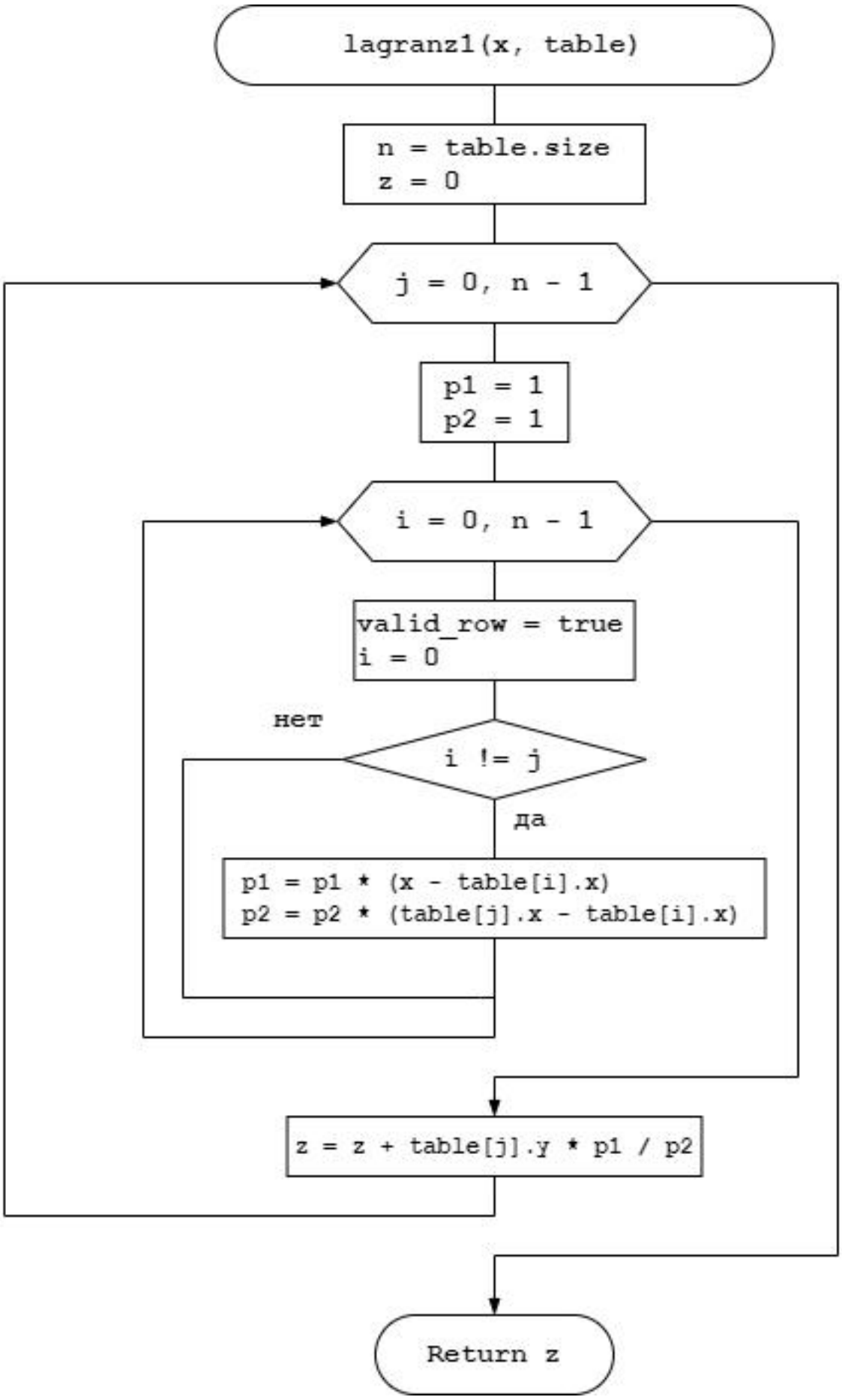

Рисунок 9 – Продолжение

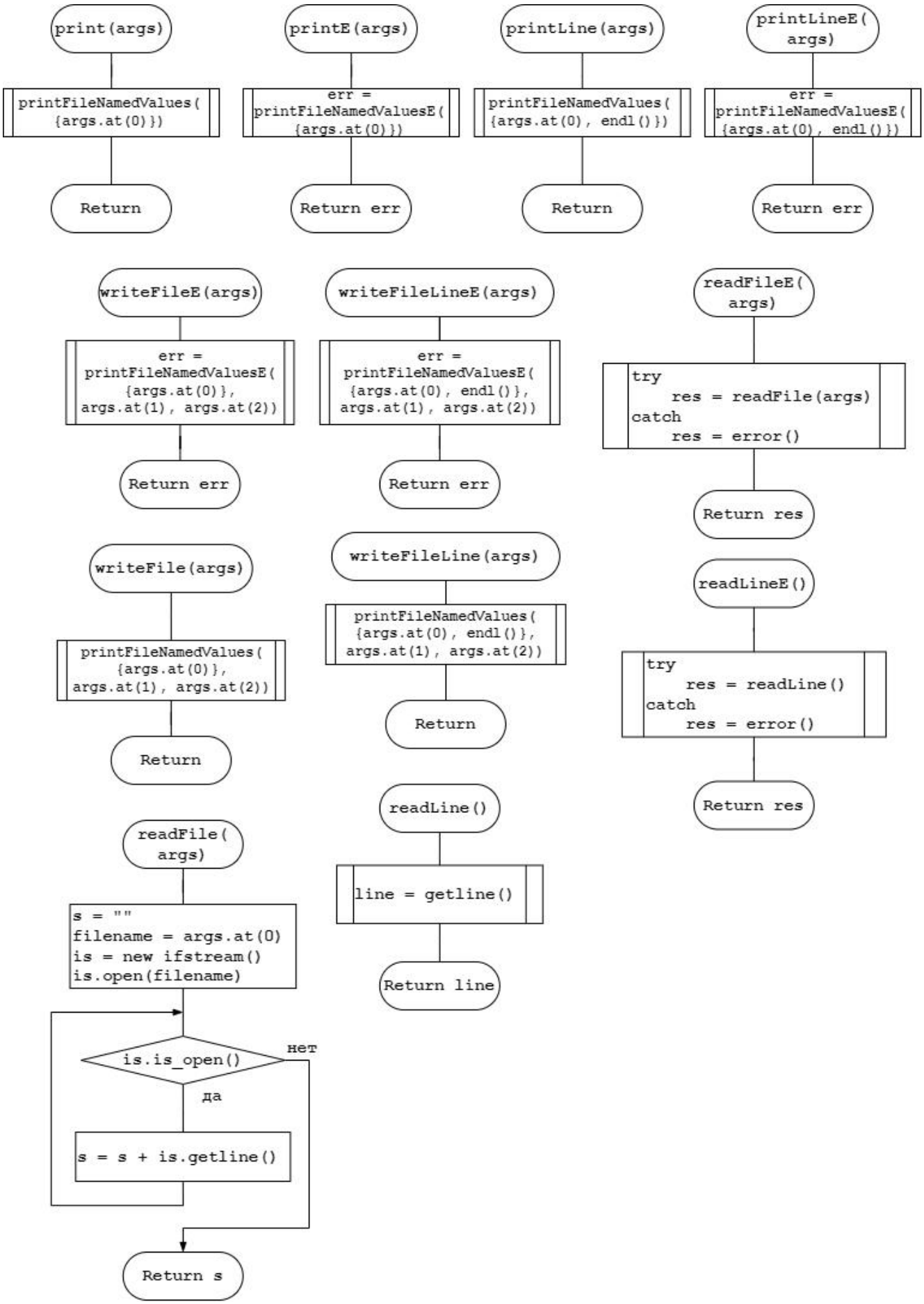

Рисунок 10 – Схемы алгоритмов модулей (подпрограмм) модуля io

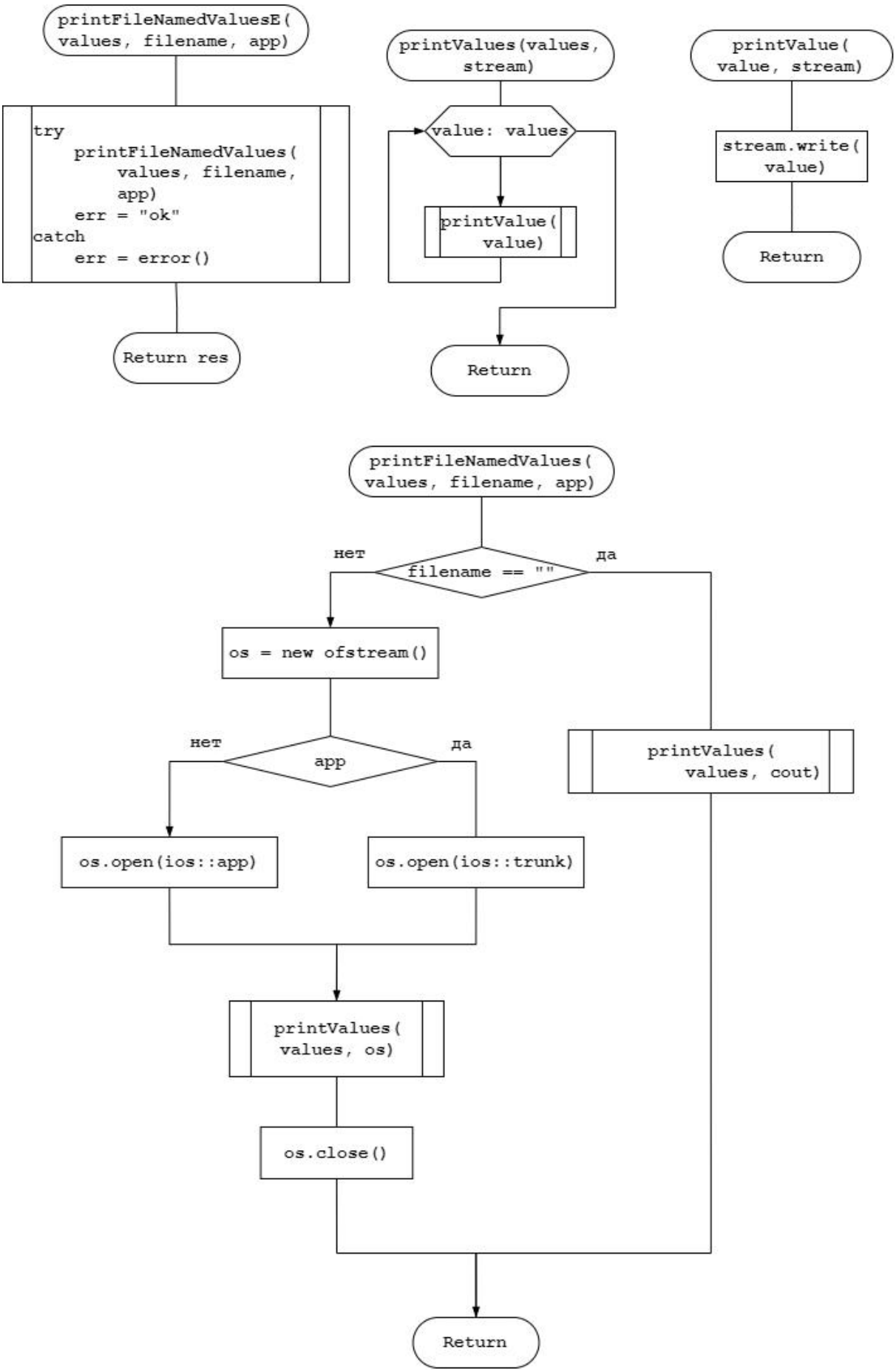

Рисунок 10 – Продолжение

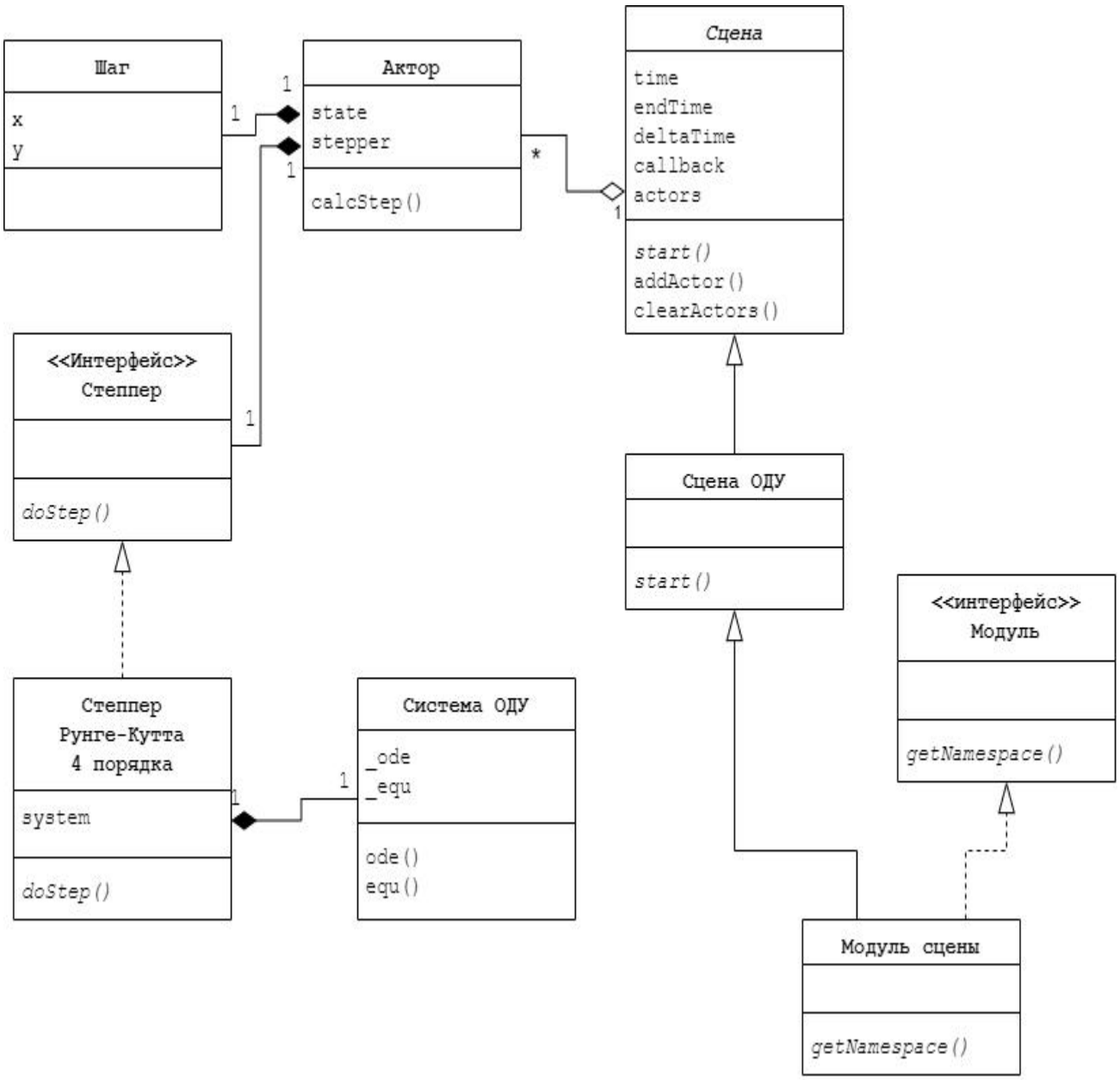

Рисунок 11 – Диаграммы классов предметной области модуля scene

Степпер Рунге-Кутта 4 порядка является реализацией Степпера, используя классический алгоритм Рунге-Кутта 4 порядка в соответствии с формулами (7), (8), (9), (10), (11) и Система ОДУ. Система ОДУ представляет собой структуру из двух функций и двух методов, их вызывающих. Сцена ОДУ реализует интерфейс абстрактного класса Сцена. Модуль сцены реализует интерфейс Модуль и наследуется от класса Сцена ОДУ.

2.4 Разработка диаграммы компоновки

Для более детального понимания связи компонентов и работы библиотеки в целом, была разработана диаграмма компоновки о связи файлов, показанная на рисунке 12.

Все модули зависят от ядра языка SBL. В модуле сцены активно применялоь обобщённое программирование с использованием шаблонов C++, поэтому заголовочных файлов больше, чем файлов модулей. Модули представлены файлами «math.cpp», «tf.cpp», «io.cpp», «scenemodule.cpp». Файлы «scenemodule.h», «scene.h», «abstractscene.h», «abstractscene.cpp», «odescene.h», «odescene.cpp», «rungekutta4stepper.h», «odesystem.h» принадлежат модулю сцены.

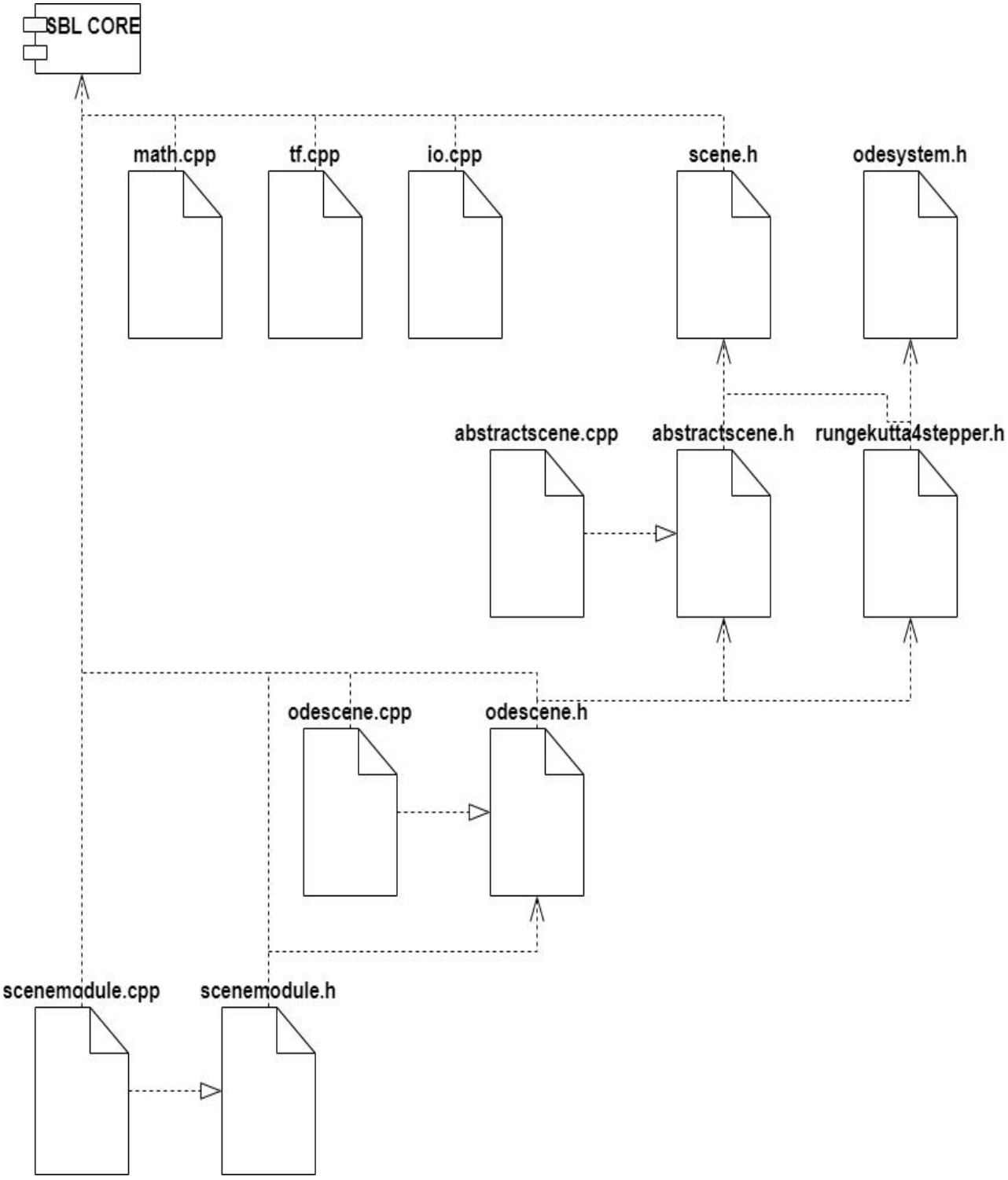

Рисунок 12 – Диаграмма компоновки программных компонентов

3. Выбор стратегии тестирования и разработка тестов

3.1 Выбор стратегии тестирования и разработка тестов

Был выбран метод тестирования структурным контролем и модульное тестирование. Данный выбор обосновывается тем, что структурный контроль, позволяет на раннем этапе реализации программного продукта выявлять ошибки кодирования, а модульные тесты позволяют независимо тестировать небольшие части программ.

Ни один метод тестирования и никакое число тестов не может гарантировать безошибочной работы программного продукта.

3.2 Тестирование структурным контролем

Первым этапом тестирования программного продукта был выбран метод тестирования структурным контролем. Данный вид тестирования помогает определить типовые ошибки, часто совершаемые в коде. Результаты тестирования представлены в таблице 1.

Таблица 1 – Тестирование структурным контролем

Вопрос	**Результат проверки**	**Вывод**
Обращения к данным		
Все ли переменные инициализированы?	Да, выбранная в п. 1.1. IDE отобразила бы ошибку, если бы были неинициализированные переменные.	Все переменные инициализированы.
Не превышены ли максимальные (или реальные) размеры массивов и строк?	У классов стандартной библиотеки STL нет ограничений на размеры массивов и строк.	Размеры массивов и строк не превышены.
Присутствуют ли переменные со сходными именами?	Нет, выбранная в п. 1.1. IDE отобразила бы ошибку именования	Нет переменных со сходными именами

Продолжение таблицы 1

При вводе из файла проверяется ли завершение файла?	При чтении из файла в цикле проверяется завершение файла	Нет ошибок при работе с файлами
Соответствуют ли типы записываемых и читаемых значений?	В C++ при несоответствии типов записываемых и читаемых значений компилятор выдаёт ошибку, чего не наблюдается	Типы соответствуют
Не выходят ли индексы за границы массивов?	Нет, иначе мы бы увидели сообщение об ошибке в консоли терминала	За границу массивов программа не выходит.
Передача управления		
Будут ли корректно завершены циклы?	Некорректное завершение циклов привело бы к некорректному поведению программу или к зависанию. Зависаний не происходит, а функциональность будет проверена в следующем пункте	Циклы завершены корректно
Будет ли завершена программа?	Программа будет работать без остановки, пока запущено моделирование	Программа будет работать до подачи сигнала SIGTERM или SIGKILL
Интерфейс		
Соответствуют ли списки параметров и аргументов по порядку, типу, единицам измерения?	Соответствуют, иначе бы компилятор отобразил ошибку	Соответствуют

Продолжение таблицы 1

Не изменяет ли подпрограмма аргументов, которые не должны изменяться?	Все изменения с данными происходят в локальных переменных	Подпрограммы не изменяют аргументы
Не происходит ли нарушения области действия глобальных и локальных переменных с одинаковыми именами?	Глобальные переменные не используются	Ошибок областей видимости нет.

Тестирование структурным контролем показало, что ошибки, распространённые среди программистов, отсутствуют. После этого можно приступить к тестированию функциональности продукта с помощью других методов тестирования.

3.3 Модульное тестирование

Данный вид тестирования предназначен для проверки корректности функционирования отдельных компонентов библиотеки и удобен тем, что является автоматизированным. Были разработаны двадцать четыре теста, проверяющих работоспособность основных компонентов. В качестве примера написанных модульных тестов приведены некоторые тесты в приложении Д.

Результат модульного тестирования представлен в таблице 2.

Таблица 2 – Отчёт о прохождении всех модульных тестов

№	Файл теста	Успешность прохождения
1	printLineE-01.sbl	+
2	printLineE-02.sbl	+
3	printLineE-03.sbl	+
4	printLineE-04.sbl	+
5	printUndefinedE-01.sbl	+
6	printUndefinedE-02.sbl	+
7	printUndefinedE-03.sbl	+
8	printUndefinedE-04.sbl	+
9	readFileE-01.sbl	+
10	writeFileE-01.sbl	+

Продолжение таблицы 2

11	writeFileLineE-01.sbl	+
12	scene-01.sbl	+
13	scene-02.sbl	+
14	scene-03.sbl	+
15	scene-04.sbl	+
16	scene-05.sbl	+
17	scene-06.sbl	+
18	math-01.sbl	+
19	tableFunctionAt-01.sbl	+
20	tableFunctionAt-02.sbl	+
21	tableFunctionAt-03.sbl	+
22	tableFunctionAt-04.sbl	+
23	tableFunctionAt-05.sbl	+
24	tableFunctionAt-06.sbl	+

На основании полученного тестового отчёта был сделан вывод, что все модульные тесты были пройдены успешно и компоненты программного продукта работают корректно.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была спроектирована и реализована библиотека прикладных модулей для адаптивной системы моделирования, полностью удовлетворяющее всем требованиям технического задания. Разработанная библиотека упрощает создание моделей для численного моделирования.

Для реализации модуля ввода-вывода была выбрана стандартная библиотека ввода-вывода C++ STL и функциональное программирование. Для реализации модуля табличных функций был выбран метод интерполяционного полинома Лагранжа и функциональное программирование. Для реализации модуля сцены был выбран классический метод Рунге-Кутты 4 порядка и объектно-ориентированное программирование. Для создания динамически подключаемых библиотек использовался Boost. Разработка библиотеки производилась в среде CLion и SIMODO edit. При разработке библиотеки использована спиральная модель жизненного цикла. Разработана концептуальная диаграммы классов модуля scene

Разработана структурная схема программного продукта; структурная карта Константайна модулей io, tf, math; схемы алгоритмов модулей io, tf; диаграммы классов предметной области модуля scene; диаграммы компоновки программных компонентов.

Произведено тестирование библиотеки методом структурного контроля и модульного тестирования. На основании полученного тестового отчёта был сделан вывод, что ошибки, распространённые среди программистов, отсутствуют и все компоненты программного продукта работают корректно.

У библиотеки большой потенциал к расширению. В будущих версиях будет выпущено:

- параллельные вычисления в процессе моделирования;
- распределение вычислений акторов сцены между разными ЭВМ;
- улучшенный метод численного интегрирования с пятым порядком точности;
- улучшенный алгоритм численного интегрирования с переменным шагом;
- асинхронный ввод-вывод.

СПИСОК ИСТОЧНИКОВ

1. Иванова Г.С. – Технология программирования: учебник / Г.С. Иванова. – 3-е изд., стер. – М. : КНОРУС, 2016. – 334 с. – (Бакалавриат)
2. Иванова Г.С., Ничушкина Т.Н., Пугачёв Е.К., Самарёв Р.С., Фетисов М.В. – Методические указания по выполнению курсовой работы по дисциплине «Технология разработки программных систем»: Электронное учебное издание. – МГТУ им. Н. Э. Баумана, 2019. – 41 с.
3. Буферизированный (потоковый) ввод-вывод [Электронный ресурс]. – https://intuit.ru/studies/courses/17500/496/lecture/11252 (дата обращения 25.10.2021)
4. Галовиц Я. – C++17 STL. Стандартная библиотека шаблонов. – СПб.: Питер, 2018. – 432 с.: ил. – (Серия «Библиотека программиста»). – ISBN 978-5-4461-0680-6
5. Библиотека Boost [Электронный ресурс]. – https://www.boost.org/ (дата обращения 25.10.2021)
6. Стандарт C++1z [Электронный ресурс]. – https://en.cppreference.com/w/cpp/17 (дата обращения 25.10.2021)
7. Методы обработки экспериментальных данных [Электронный ресурс]. – https://portal.tpu.ru/SHARED/m/MOE/Ucheba/Tab3/Tab/LK_%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%9C_%D0%9B%D0%B0%D0%B3%D1%80%D0%B0%D0%BD%D0%B6%D0%B0.pdf (дата обращения 22.10.2021).
8. Методы вычисления элементарных функций [Электронный ресурс]. – https://scask.ru/p_book_pta.php?id=45 (дата обращения 22.10.2021).
9. Аппроксимация функций [Электронный ресурс]. – https://vgasu.ru/attachments/aproximacia.pdf (дата обращения 22.10.2021).
10. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Численные методы. - М.: Наука, 1987. - 630 с.
11. Арушанян О. Б., Залеткин С. Ф. Решение задачи Коши для обыкновенных дифференциальных уравнений одношаговыми разностными методами: практикум на ЭВМ по вычислительным методам. - М.: МГУ, 2002. - 51 с.
12. CLion IDE [Электронный ресурс]. – https://www.jetbrains.com/clion/ (дата обращения 01.10.2021).
13. Codeblocks IDE [Электронный ресурс]. – https://www.codeblocks.org/ (дата обращения 01.10.2021).
14. Qt [Электронный ресурс]. – https://www.qt.io/ (дата обращения 01.10.2021).

ПРИЛОЖЕНИЕ А

Техническое задание

Листов 8

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

БИБЛИОТЕКА ПРИКЛАДНЫХ МОДУЛЕЙ ДЛЯ АДАПТИВНОЙ СИСТЕМЫ МОДЕЛИРОВАНИЯ

Техническое задание на курсовую работу
по дисциплине Технология разработки программных систем

Листов 8

Студент гр. ИУ6-53 (Группа) — 15.12.2021 (Подпись, дата) А.А. Бушев (И.О. Фамилия)

Руководитель курсовой работы,
(старший преподаватель «Компьютерные системы и сети») — 15.12.2021 (Подпись, дата) С.С. Данилюк (И.О. Фамилия)

Москва, 2021

1 ВВЕДЕНИЕ

Настоящее техническое задание распространяется на разработку библиотеки прикладных модулей для адаптивной системы моделирования. Библиотека планируется использоваться в адаптивной системе моделирования SIMODO, которая разрабатывается на кафедре ИУ6 МГТУ им. Н.Э. Баумана. Библиотека будет состоять из следующих модулей: io (модуль ввода-вывода), math (математический модуль), tf (модуль табличных функций), scene (модуль сцены). Библиотека будет использоваться пользователями адаптивной системы моделирования для упрощения создания моделей.

Актуальность данной разработки обосновывается отсутствием аналогов библиотек для адаптивной системы моделирования SIMODO.

2 ОСНОВАНИЯ ДЛЯ РАЗРАБОТКИ

Библиотека прикладных модулей для адаптивной системы моделирования разрабатывается в соответствии с тематикой кафедры на основе учебного плана кафедры ИУ6 «Компьютерные системы и сети» факультета ИУ «Информатика и системы управления».

3 НАЗНАЧЕНИЕ РАЗРАБОТКИ

Библиотека, разрабатываемая для адаптивной системы моделирования SIMODO, должна выполнять следующие задачи:

- запись и чтение текстовой информации стандартных потоков ввода и вывода;
- запись и чтение текстовой информации файлов;
- вычисление математических функций;
- вычисление многомерных табличных функций;
- моделирование на основе дифференциальных уравнений.

4 ТРЕБОВАНИЯ К ПРОГРАММНОМУ ИЗДЕЛИЮ

4.1 Требования к функциональным характеристикам

4.1.1 Выполняемые функции

4.1.1.1 Модуль ввода-вывода:

- чтение текстовых данных из стандартного потока ввода;
- запись текстовых данных в стандартный поток вывода;
- чтение текстовых данных из файла;
- запись текстовых данных в файл.

4.1.1.2 Математический модуль:

- вычисление функции синуса в точке;
- вычисление функции косинуса в точке;
- вычисление функции тангенса в точке;
- вычисление функции арксинуса в точке;
- вычисление функции арккосинуса в точке;
- вычисление функции арктангенса в точке;
- вычисление функции квадратного корня в точке;
- вычисление экспоненциальной функции в точке;
- вычисление функции натурального логарифма в точке;
- вычисление функции четырёхквадрантного арктангенса в точке;
- определение знака выражения в точке;
- вычисление функции арксинуса от двух аргументов в точке;
- вычисление функции арккосинуса от двух аргументов в точке;
- вычисление функции арктангенса от двух аргументов в точке;
- значение числа пи.

4.1.1.3 Модуль табличных функций:

- вычисление одномерной табличной функции в точке;
- вычисление двумерной табличной функции в точке;
- вычисление многомерной двоичной функции в точке.

4.1.1.4 Модуль сцены моделирования:

- здание и получение времени начала моделирования;

−задание и получение времени конца моделирования;
−задание и получение временного шага моделирования;
−задание и получение моделей дифференциальных уравнений;
−очищение сцены;
−задание и получение функции обратного вызова;
−запуск моделирования.

4.1.2 Исходные данные:

4.1.2.1 Модуль ввода-вывода:

−строка для записи в стандартный поток вывода или файл;
−управляющий флаг добавления в конец файла;
−имя файла для записи или чтения в или из файла соответственно.

4.1.2.2 Математический модуль:

−параметр функции;
−параметр функции, представленный числителем и знаменателем дроби.

4.1.2.3 Модуль табличных функций:

−многомерная таблица значений табличной функции;
−координаты точки, в которой ищется значение функции.

4.1.2.4 Модуль сцены моделирования:

−время начала моделирования;
−время конца моделирования;
−временной шаг моделирования;
−функция обратного вызова;
−период вызова функции обратного вызова;
−модели дифференциальных уравнений.

4.2 Требования к надежности

4.2.1 Предусмотреть контроль вводимой информации.

4.2.2 Предусмотреть блокировку некорректных действий пользователя.

4.3 Условия эксплуатации

4.3.1 Условия эксплуатации в соответствии с СанПиН 2.2.2/2.4.1340-03.

4.4 Требования к составу и параметрам технических средств

4.4.1 Программное обеспечение должно функционировать на IBM-совместимых персональных компьютерах.

4.4.2 Минимальная конфигурация технических средств:

4.4.2.1 Тип процессора ..... Pentium.

4.4.2.2 Объем ОЗУ..... 4 Гб.

4.5 Требования к информационной и программной совместимости

4.5.1 Программное обеспечение должно работать под управлением операционных систем семейств GNU/Linux.

4.5.2 Программное обеспечение должно быть реализовано с использованием языка C++ стандарта C++1z.

5 ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

5.1 Разрабатываемые программные модули должны быть самодокументированы, т.е. тексты программ должны содержать все необходимые комментарии и соответствовать соглашениям о кодировании SIMODO.

5.2 В состав сопровождающей документации должны входить:

5.2.1 Расчетно-пояснительная записка на 25-30 листах формата А4 (без приложений 5.3.2, 5.3.3 и 5.3.4).

5.2.2 Техническое задание (Приложение А).

5.2.3 Фрагмент исходного текста модулей (Приложение Б).

5.2.4 Примеры работы с модулями (Приложение В).

5.3 Графическая часть должна быть включена в расчетно-пояснительную записку в качестве иллюстраций:

5.3.1 Концептуальная диаграмма классов.

5.3.2 Схема структурная программного обеспечения.

5.3.3 Схема взаимодействия модулей Константайна.

5.3.4 Схемы алгоритмов модулей (подпрограмм).

5.3.5 Диаграммы классов предметной области.

5.3.6 Диаграммы компоновки программных компонентов

5.3.7 Таблицы тестов.

6 СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

Этап	**Содержание этапа**	**Сроки и объем**	**Представляемые результаты**	
			Спецификации и программный продукт	**Документы**
1.	Выбор темы, составление задания, решение организационных вопросов	1..2 недели (10 %)	-	**Заполненный бланк задания на курсовую работу – вывешивается на сайт кафедры для получения утверждающей подписи заведующего кафедрой**
2.	Анализ предметной области, разработка ТЗ. Исследование методов решения, выбор основных проектных решений	3..4 недели	Результаты декомпозиции предметной области. Эскизный проект: интерфейс, схемы, возможно, часть программы (выбранные готовые решения).	Фрагмент расчетно-пояснительной записки с обоснованием выбора средств и подходов к разработке
3.	**Сдача ТЗ**	**4 неделя (25 %)**	**-**	**Техническое задание – утверждается руководителем**
4.	Проектирование и реализация основных компонентов – ядра программы	5..7 недели	Технический проект основной части: структура программы, алгоритмы программ, описания структур данных, диаграмма классов – в зависимости от выбранной технологии разработки. Программный продукт, реализующий основные функции (демонстрируется руководителю)	Фрагмент расчетно-пояснительной записки с обоснованием разработанных спецификаций Тексты части программного продукта, реализующего основные функции.

Этап	Содержание этапа	Сроки и объем	Представляемые результаты	
			Спецификации и программный продукт	**Документы**
5.	**Сдача прототипа программного продукта**	**7 неделя (50 %)**	**Прототип программного продукта – демонстрируется руководителю**	
6.	Разработка компонентов, обеспечивающих функциональную полноту	8..10	Рабочий проект программы. Готовая программа	Черновик расчетно-пояснительной записки. Тексты программного продукта.
7.	**Сдача программного продукта**	**11 неделя (75 %)**	**Готовая программа – оценивается руководителем в баллах**	**-**
8.	Тестирование программы и подготовка документации	12..14	Тесты и результаты тестирования.	РПЗ и Руководство пользователя.
9.	**Оформление и сдача документации**	**14 неделя (90 %)**	**–**	**Расчетно-пояснительная записка и Руководство пользователя – проверяются и подписываются руководителем**
10.	Защита курсовой работы	15..16 недели (100%)	–	Доклад (3-5 минут). Защита курсовой работы. Подписанная документация – вывешивается на сайт кафедры

7 ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

7.1 Порядок контроля

Контроль выполнения осуществляется руководителем еженедельно.

7.2 Порядок защиты

Защита осуществляется комиссии преподавателей кафедры.

7.3 Срок защиты

Срок защиты: 15-16 недели.

8 ПРИМЕЧАНИЕ

В процессе выполнения работы возможно уточнение отдельных требований технического задания по взаимному согласованию руководителя и исполнителя.

ПРИЛОЖЕНИЕ Б

Фрагмент исходного текста модулей

Листов 5

В качестве примера фрагмента программы приведён код файла odescene.cpp, в котором реализуется класс сцены ОДУ OdeScene, отвечающий за математическое моделирование.

odescene.cpp

```
#include "odescene.h"
#include "rungekutta4stepper.h"

#include "simodo/core/convertion_functions.h"
#include "simodo/core/Exception.h"
#include "simodo/sbl/OpModelSupport_statics.h"

using namespace std;

using namespace simodo;
using namespace simodo::core;
using namespace simodo::core::sbl;

namespace simodo::module::scene
{
  // Конструктор сцены
  OdeScene::OdeScene(const core::SemanticRulesHost_interface
&interpreter)
      : AbstractScene(interpreter)
  {}

  // Добавление актора
  void OdeScene::addActor(Actor actor)
  {
    AbstractScene::addActor(actor);
  }

  // Добавления актора на основе модели
  void OdeScene::addActor(const core::sbl::NamedValue
&object_reference)
  {
    // Проверка валидности модели
    if (object_reference.declared != DeclaredType::Reference
      && origin(object_reference).declared != DeclaredType::Object
      && origin(object_reference).declared !=
        DeclaredType::Type) /// \todo Не правильно работает
типизация должен быть объект, а не тип
      throw Exception("Scene::addSceneActor",
              "В качестве аргумента метода 'add' должен быть
объект, а задан " +

convertToU8(getDeclaredTypeName(origin(object_reference).declared))
);
```

```
    // Проверка на повторное добавление актора
    for (const Actor &a: _actors)
    {
      if (&(origin(object_reference)) ==
&(origin(a.get_object_reference())))
        throw Exception("Scene::addSceneActor",
                "Повторное добавление объекта '" +

simodo::core::convertToU8(origin(object_reference).name) +
                "'");
    }

    // Временные переменные для формирования актора
    vector<shared_ptr<NamedValue>> derivatives_variables_pointers;
    vector<shared_ptr<NamedValue>> output_variables_pointers;
    NamedValue ode_function;
    NamedValue equ_function;
    const auto &value_set =
get<ObjectValue>(origin(object_reference).variant);

    // Извлечение данных из модели
    for (const NamedValue &n: value_set)
    {
      // Извлечение интегрируемого параметра
      if (n.name.substr(0, 5) == u"__dt_")
      {
        u16string x_name = n.name.substr(5);
        size_t i = 0;

        for (; i < value_set.size(); ++i)
          if (value_set[i].name == x_name)
            break;

        if (i == value_set.size())
          throw Exception("Scene::addActor",
                  "Некорректная струтура объекта, переменная '" +
simodo::core::convertToU8(x_name) +
                  "'");

        NamedValue x =
simodo::core::sbl::OpModelSupport_statics::makeReference(
            const_cast<NamedValue &>(value_set[i]), {}, {});
        NamedValue dxdt =
simodo::core::sbl::OpModelSupport_statics::makeReference(const_cas
t<NamedValue &>(n),
                                                 {}, {});


output_variables_pointers.push_back(get<shared_ptr<NamedValue>>(x.
variant));

derivatives_variables_pointers.push_back(get<shared_ptr<NamedValue
>>(dxdt.variant));
```

```
      // Извлечение функции второстепенных параметров
      } else if (n.name == u"__EQU")
      {
        if (n.declared != DeclaredType::Reference &&
origin(n).declared != DeclaredType::Function)
          throw Exception("Scene::addSceneActor",
                  "Некорректная струтура объекта, переменная '" +
simodo::core::convertToU8(n.name) +
                  "'");

        equ_function = n;
      // Извлечение функции СДУ
      } else if (n.name == u"__ODE")
      {
        if (n.declared != DeclaredType::Reference &&
origin(n).declared != DeclaredType::Function)
          throw Exception("Scene::addSceneActor",
                  "Некорректная струтура объекта, переменная '" +
simodo::core::convertToU8(n.name) +
                  "', тип " +
core::convertToU8(getDeclaredTypeName(origin(n).declared)));

        ode_function = n;
      }
    }

    // Проверка валидности загруженной модели
    if (derivatives_variables_pointers.empty() ||
ode_function.declared == DeclaredType::Error)
      throw Exception("Scene::addSceneActor", "Неполная струтура
объекта");

    using StepperType = RungeKutta4Stepper<double>;

    auto *interpreter = &_interpreter;

    // Создание объекта СДУ
    StepperType::OdeSystemType ode_system =
StepperType::OdeSystemType(
        [output_variables_pointers,
            derivatives_variables_pointers,
            ode_function,
            interpreter](
            const double /*x*/,
            const std::vector<double/*, ODE_DIMENSION*/> &y
        )
        {
          {
            auto y_it = y.begin();
            auto o_it = output_variables_pointers.begin();
            while(y_it != y.end())
            {
              (*o_it++)->variant = *y_it++;
```

```
            }
          }

          if (origin(ode_function).declared == DeclaredType::Error)
          {
            throw Exception("Scene::stepper", "ODE функция
невалидна");
          }

          SemanticTreeWalkerState state = interpreter-
>callFunction(ode_function, {},

InterpreterMode::FullExecution);
          if (state == SemanticTreeWalkerState::Error)
            throw Exception("Scene::stepper", "Выполнение
прервано");

          return foreach<shared_ptr<NamedValue>, double>(
              derivatives_variables_pointers,
              [](const shared_ptr<NamedValue>& derivative_variable)
              {
                return get<double>(derivative_variable->variant);
              }
          );
        },
        [equ_function, interpreter]()
        {
          if (origin(equ_function).declared != DeclaredType::Error)
          {
            SemanticTreeWalkerState state = interpreter-
>callFunction(equ_function, {},

InterpreterMode::FullExecution);
            if (state == SemanticTreeWalkerState::Error)
              throw Exception("Scene::stepper", "Выполнение
прервано");
          }
        }
    );

    // Создание актора
    Actor &&actor = Actor(object_reference,
output_variables_pointers, static_pointer_cast<Stepper < double>>
    (make_shared<StepperType>(ode_system)),
        StepperType::Step{AbstractScene::get_time(),
foreach<shared_ptr<NamedValue>, double>(
            output_variables_pointers,
            [](const shared_ptr<NamedValue>
&output_variable_pointer)
            {
              return get<double>(output_variable_pointer->variant);
            }
        )}
```

```
    );

    // Добавление сформированного из модели актора
    addActor(actor);
  }

  void OdeScene::start()
  {
    if (_callback.declared != DeclaredType::Error)
    {
      SemanticTreeWalkerState state =
_interpreter.callFunction(_callback, {},
InterpreterMode::FullExecution);
      if (state == SemanticTreeWalkerState::Error)
        throw Exception("Scene::start", "Выполнение прервано");
    }

    const double time = _time;

    for (int n = 1; /*!_p_interpreter->stop_signal()*/; ++n)
    {
      const double old_time = _time;
      _time = time + n * _delta_time;

      if (_end_time > 0.0 && _time > _end_time)
        break;

      for (Actor &actor: _actors)
      {
        actor.calculateStep(old_time, _delta_time);
      }

      if (n % _callback_period == 0 && _callback.declared !=
DeclaredType::Error)
      {
        SemanticTreeWalkerState state =
_interpreter.callFunction(_callback, {},

InterpreterMode::FullExecution);
        if (state == SemanticTreeWalkerState::Error)
          throw Exception("Scene::start", "Выполнение прервано");
      }
    }
  }
}
```

Полный текст модулей расположен в репозитории по адресу:

https://bmstu.codes/lsx/simodo/loom/-/tree/dev

ПРИЛОЖЕНИЕ В

Примеры работы с модулями

Листов 3

Библиотека прикладных модулей для адаптивной системы моделирования позволяет упростить процесс формирования моделей. Продемонстрируем работу с библиотекой прикладных модулей для адаптивной системы моделирования.

Сначала следует определить модель, которая будет симулироваться. В примере на рисунке В.1 приведена модель летательного аппарата, который перемещается в направлении точки, достигает её и начинает новое движение к этой же точке.

В модели можно увидеть три блока кода: блок переменных, блок интегрируемых значений и блок системы уравнений и блок системы обыкновенных дифференциальных уравнений, представленные двумя функциями.

После задания модели можно приступать к написанию программы на языке SBL. Код программы представлен на рисунке В.2.

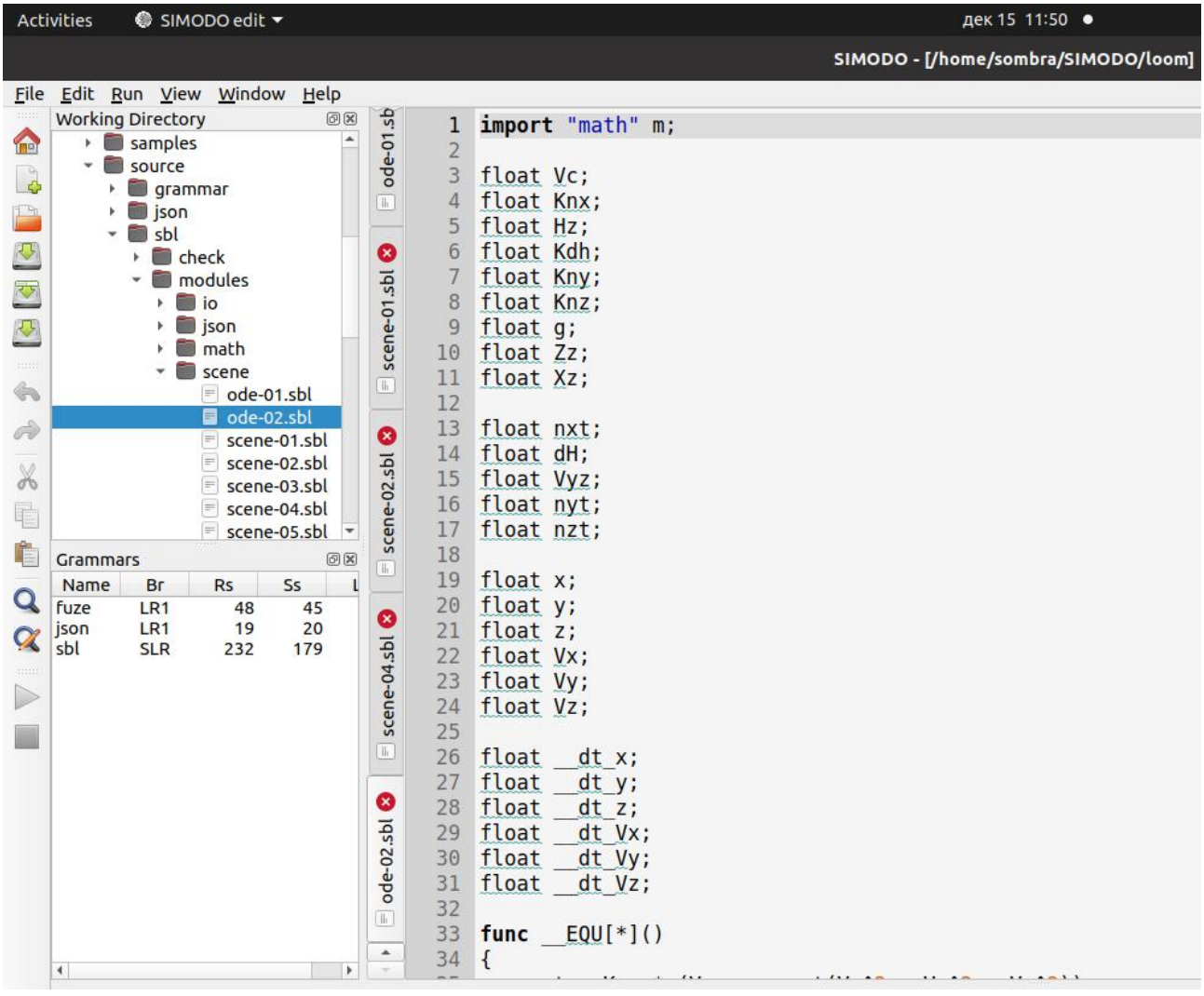

Рисунок В.1 – Модель на языке SBL

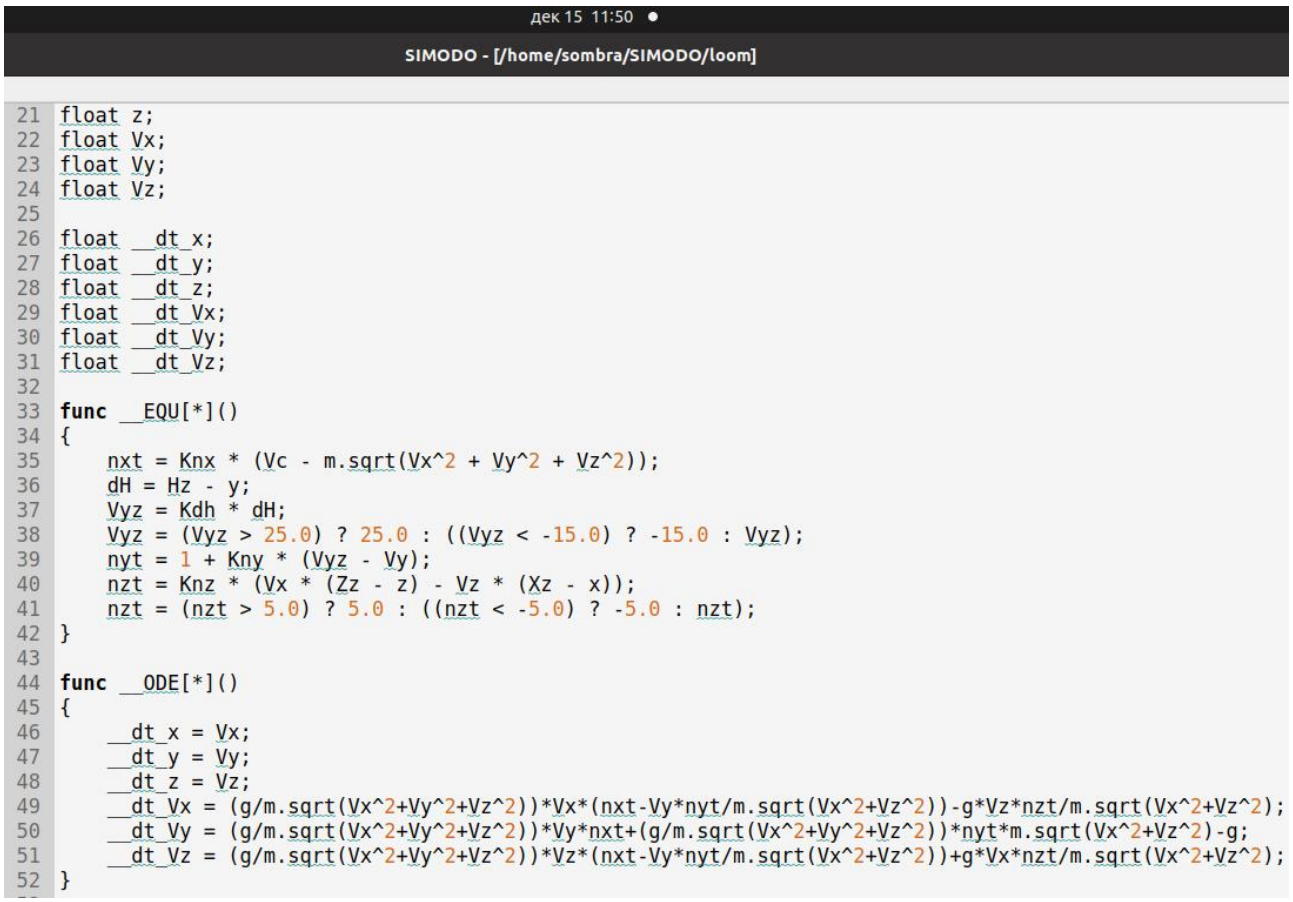

Рисунок В.1 – Продолжение

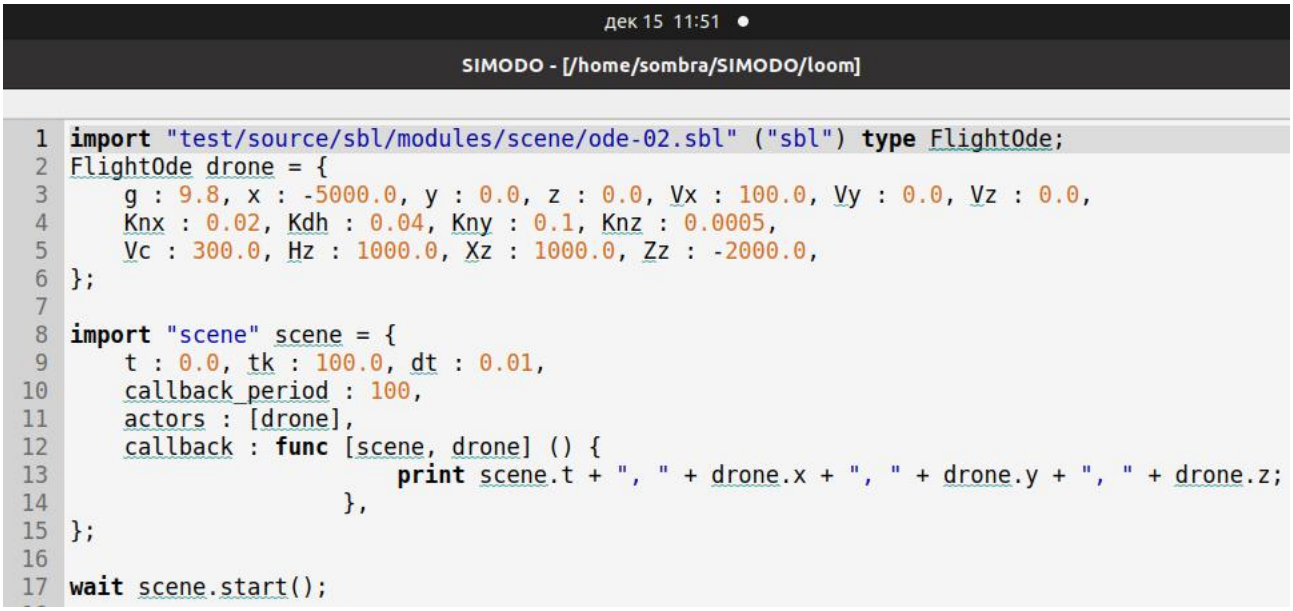

Рисунок В.2 – Программа на языке SBL

Первой строчкой импортируется модель, описанная ранее. Затем на основе модели инстанцируется объект этой модели и инициализируется начальными значениями. На восьмой строчке импортируется модуль сцены и инициализируется начальными значениями. Примечательно, что в функции обратного вызова модуля происходит вывод параметров объекта модели, созданного ранее. На основе этих данных будут построены графики

перемещения аппарата, который представлен заданной моделью. Семнадцатой строчкой запускается процесс моделирования. Процесс моделирования завершится сигнал SIGTERM.

На основе полученных данных были построены графики перемещения аппарата, представленного заданной моделью (рисунки В.3, В.4).

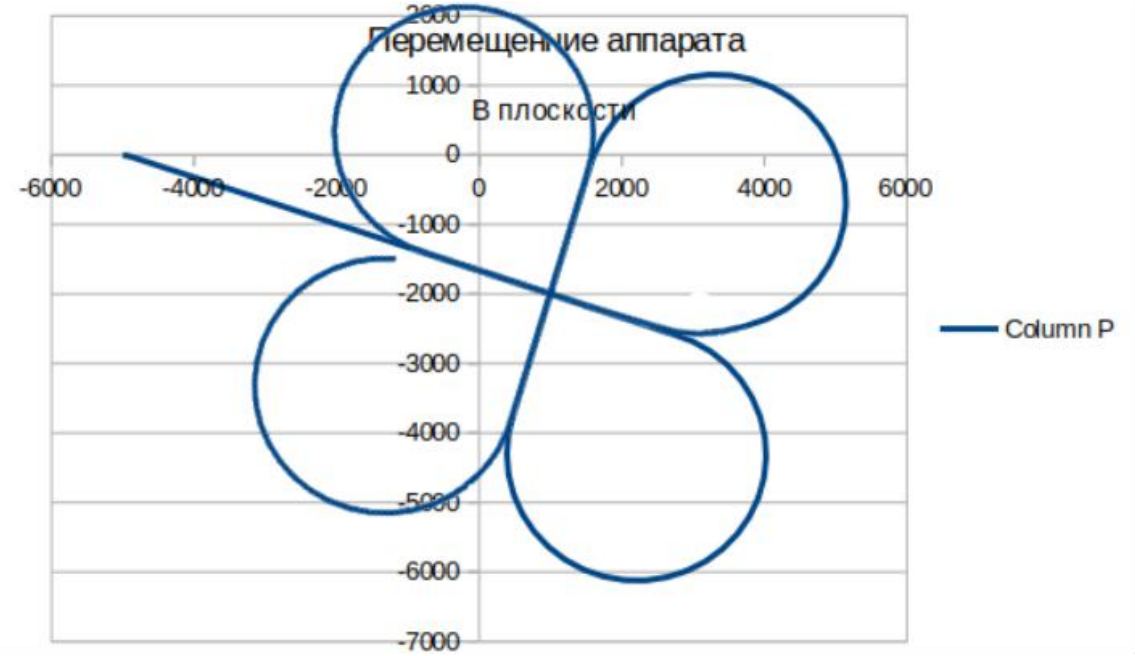

Рисунок В.3 – Перемещение аппарата в плоскости

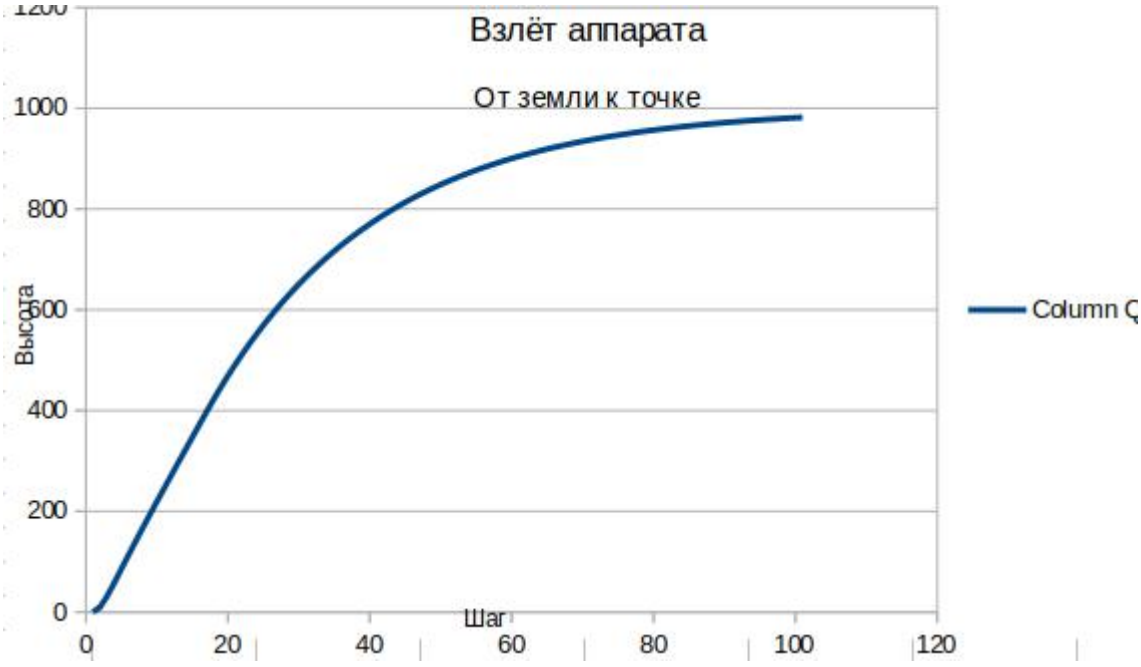

Рисунок В.4 – Перемещение аппарата по вертикали

Как видно из графиков В.3 и В.4, аппарат начал движение у поверхности и стал постепенно выходить на плоскость точки назначения. В это же время его движение очерчивает два лепестка, и видно, куда аппарат движется – на пересечение линий траектории.

На примере моделирования летательного аппарата были продемонстрированы возможности библиотеки прикладных модулей для адаптивной системы моделирования.

ПРИЛОЖЕНИЕ Г

Соглашение о кодировании SIMODO

Листов 7

Таблица Г.1 – Перечень основных правил соглашения о стиле кодирования

№	Правило	Пояснения
1. Запрещающие соглашения		
1.1.	Следует избегать использования глобальных переменных	
1.2.	Следует избегать использования макросов	
1.3.	Следует избегать использования оператора **`goto`**	
1.4.	Следует избегать использования указателей C++	Используйте интеллектуальные указатели: unique_ptr, shared_ptr, weak_ptr
1.5.	Следует избегать использования массивов C++	Используйте параметризованные контейнеры стандартной библиотеки C++: arrey, vector, deque и другие
2. Соглашения об именовании		
2.1.	Имена, представляющие типы, должны быть написаны в смешанном регистре, начиная с верхнего	Пример: `Line, SavingsAccount`

2.2.	Имена переменных должны быть записаны в нижнем регистре, слова должны разделяться знаком подчеркивания	Пример:
		`line, savings_account`
		Распространённая в настоящее время практика в сообществе разработчиков C++. Позволяет легко отличать переменные от типов, предотвращает потенциальные коллизии имён, например:
		`Line line;`
		Также, записанные таким образом переменные легко отличить от названия метода (см. далее).

2.3.	Названия методов и функций должны быть глаголами, быть записанными в смешанном регистре и начинаться с нижнего	Пример: `getName(), computeTotalWidth()`

Таблица Г.1 – Продолжение

№	Правило	Пояснения

2.4.	Аббревиатуры и сокращения в именах должны записываться в нижнем регистре	Пример:
		`exportHtmlSource();`
		`// НЕЛЬЗЯ: exportHTMLSource();`
		`openDvdPlayer();`
		`// НЕЛЬЗЯ: openDVDPlayer();`
		Использование верхнего регистра может привести к конфликту имён, описанному выше. Иначе переменные бы имели имена dVD, hTML и т. д., что не является удобочитаемым. Другая проблема уже описана выше; когда имя связано с другим, читаемость снижается; слово, следующее за аббревиатурой, не выделяется так, как следовало бы.

2.5.	Членам класса с модификатором private следует присваивать префикс-подчёркивание	Пример:
		`class SomeClass`
		`{`
		`private:`
		`  int _length;`
		`}`
		Не считая имени и типа, область видимости — наиболее важное свойство переменной. Явное указание модификатора доступа в виде подчёркивания избавляет от путаницы между членами класса и локальными переменными. Это важно, поскольку переменные класса имеют большее значение, нежели переменные в методах, и к ним следует относиться более осторожно.
		Дополнительным эффектом от префикса-подчёркивания является разрешение проблемы именования в методах, устанавливающих значения, а также в конструкторах:
		`void setDepth (int depth)`
		`{`
		`  _depth = depth;`
		`}`

Таблица Г.1 – Продолжение

№	Правило	Пояснения

2.6.	Все имена следует записывать по-английски	Пример:
		`file_name;`
		`// НЕ РЕКОМЕНДУЕТСЯ: imyaFayla`

2.7.	Переменные, имеющие большую область видимости, следует называть длинными именами, имеющие небольшую область видимости — короткими	Имена временных переменных, использующихся для хранения временных значений или индексов, лучше всего делать короткими. Программист, читающий такие переменные, должен иметь возможность предположить, что их значения не используются за пределами нескольких строк кода. Обычно это переменные i, j, k, l, m, n (для целых), а также c и d (для символов).

2.8.	Имена объектов не указываются явно, следует избегать указания названий объектов в именах методов	Пример:
		`line.getLength();`
		`// НЕ РЕКОМЕНДУЕТСЯ:`
		`line.getLineLength();`
		Второй вариант смотрится вполне естественно в объявлении класса, но совершенно избыточен при использовании, как это и показано в примере.

3. Особые правила наименования		

3.1.	Слова get/set должны быть использованы везде, где осуществляется прямой доступ к атрибуту	Пример:
		`employee.getName();`
		`employee.setName(name);`
		
		`matrix.getElement(2, 4);`
		`matrix.setElement(2, 4, value);`

3.2.	Множественное число следует использовать для представления наборов (коллекций) объектов	Пример:
		`vector<Point>  points;`
		`int  values[];`
		Улучшает читаемость, поскольку имя даёт пользователю прямую подсказку о типе переменной и операциях, которые могут быть применены к этим элементам.

Таблица Г.1 – Продолжение

№	Правило	Пояснения
3.3.	Префикс n следует использовать для представления числа объектов	Пример: `n_points, n_lines` Обозначение взято из математики, где оно является установившимся соглашением для обозначения числа объектов.
3.4.	Суффикс No следует использовать для обозначения номера сущности	Пример: `table_no, employee_no` Обозначение взято из математики, где оно является установившимся соглашением для обозначения номера сущности.

3.5.	Переменным-итераторам следует давать имена i, j, k и т. д. Или it, чтобы не путать его с целым типом.	Пример:
		`for(int i = 0; i < n_tables); i++)`
		`{`
		`    ...`
		`}`
		Обозначение взято из математики, где оно является установившимся соглашением для обозначения итераторов.
		`for(auto it = list.begin(); it != list.end(); ++it)`
		`{`
		`    Element element = *it;`
		`    ...`
		`}`

3.6.	Префикс is следует использовать только для булевых (логических) переменных и методов	Пример:
		`s_set, is_visible, is_finished, is_found, is_open`
		Использование этого префикса избавляет от таких имён, как status или flag. is_status или is_flag просто не подходят, и программист вынужден выбирать более осмысленные имена.
		В некоторых ситуациях префикс is лучше заменить на другой: has, can или should:
		`bool has_license();`
		`bool can_evaluate();`
		`bool should_sort();`

Таблица Г.1 – Продолжение

№	Правило	Пояснения
4. Файлы исходных кодов		
4.1.	Заголовочным файлам C++ следует давать расширение .h (предпочтительно) либо .hpp. Файлы исходных кодов могут иметь расширения .cpp. Имена файлов записываются в нижнем регистре	Пример: `myclass.cpp, myclass.h` Эти расширения, одобряемые стандартом C++.
4.2.	Класс следует объявлять в заголовочном файле и определять (реализовывать) в файле исходного кода, имена файлов совпадают с именем класса (но пишутся в нижнем регистре)	Пример: `myclass.cpp, myclass.h` Облегчает поиск связанных с классом файлов. Очевидное исключение — шаблонные классы, которые должны быть объявлены и определены в заголовочном файле.
4.3.	Нельзя использовать специальные символы (например, TAB) и разрывы страниц	Такие символы вызывают ряд проблем, связанных с редакторами, эмуляторами терминалов и отладчиками, используемыми в программах для совместной разработки и кроссплатформенных средах.

4.4.	Заголовочные файлы должны содержать защиту от вложенного включения	Пример:
		`#ifndef _COM_COMPANY_MODULE_CLASSNAME_H_`
		`#define _COM_COMPANY_MODULE_CLASSNAME_H_`
		`...`
		`#endif // _COM_COMPANY_MODULE_CLASSNAME_H_`
		Конструкция позволяет избегать ошибок компиляции. Это соглашение позволяет увидеть положение файла в структуре проекта и предотвращает конфликты имён.

Таблица Г.1 – Продолжение

№	Правило	Пояснения

4.5.	Директивы включения следует сортировать (по месту в иерархии системы, ниже уровень — выше позиция) и группировать. Оставляйте пустую строку между группами	Пример:
		`#include <fstream>`
		`#include <iomanip>`
		
		`#include <qt/qbutton.h>`
		`#include <qt/qtextfield.h>`
		
		`#include "com/company/ui/PropertiesDialog.h"`
		`#include "com/company/ui/MainWindow.h"`
		Пути включения не должны быть абсолютными. Вместо этого следует использовать директивы компилятора.

4.6.	Директивы включения должны располагаться только в начале файла	Общая практика. Избегайте нежелательных побочных эффектов, которые может вызвать «скрытое» включение где-то в середине файла исходного кода.
5. Разное		
5.1.	Следует избегать «магических» чисел в коде. Числа, отличные от 0 или 1, следует объявлять, как именованные константы	Если число само по себе не имеет очевидного значения, читаемость улучшается путём введения именованной константы. Другой подход — создание метода, с помощью которого можно было бы осуществлять доступ к константе.
5.2.	Следует использовать null_ptr вместо «NULL»	NULL является частью стандартной библиотеки C и устарело в C++
6. Оформление и комментарии		

6.1.	Основной отступ следует делать в четыре пробела	Пример:
		`for(i = 0; i < n_elements; i++)`
		`    a[i] = 0;`
		Отступ в один или два пробела достаточно мал, чтобы отражать логическую структуру кода. Отступ более 4 пробелов делает глубоко вложенный код нечитаемым и увеличивает вероятность того, что строки придётся разбивать. Широко распространены варианты в 2, 3 или 4 пробела; причём 2 и 4 — более широко.

Таблица Г.1 – Продолжение

№	Правило	Пояснения

6.2.	Блоки кода следует оформлять так, как показано в примере 1 (рекомендуется), но ни в коем случае не так, как показано в примере 2. Оформление функций и классов должно следовать примеру 1	Пример 1:
		`while(!done)`
		`{`
		`  doSomething();`
		`  done = moreToDo();`
		`}`
		
		Пример 2:
		`while(!done)`
		`  {`
		`  doSomething();`
		`  done = moreToDo();`
		`  }`
		Пример 2 использует лишние отступы, что мешает ясному отображению логической структуры кода.

ПРИЛОЖЕНИЕ Г

Примеры модульных тестов

Листов 2

tableFunctionAt-05.sbl

```
import "tf" tf;

// Задание точки на функции
float x [1] = [4];
// Задание таблицы функции
float f [3, 2] = [
    [1, 1],
    [2, 4],
    [3, 9]
];

// Вывод результата вычисления табличной функции
print tf.tableFunctionAt(x, f);
```

tableFunctionAt-06.sbl

```
import "tf" tf;

// Задание точки на функции
float x1 [2] = [0, -1];
// Задание точки на функции
float x2 [2] = [0, -2];
// Задание таблицы функции
float f [4, 3] = [
    [0, 0, 1],
    [0, 1, 2],
    [1, 0, 2],
    [1, 1, 3]
];

// Вывод результата вычисления табличной функции
print tf.tableFunctionAt(x1, f);
print tf.tableFunctionAt(x2, f);
```

writeFileLineE-01.sbl

```
import "io" io;

// Запись строки в файл с удалением старого файла
wait io.writeFileLineE(
    "test/tmp/writeFileLine-01.out", "First", false);
// Добавление строк в конец файла
wait io.writeFileLineE(
    "test/tmp/writeFileLine-01.out", "Second", true);
wait io.writeFileLineE(
    "test/tmp/writeFileLine-01.out", "Third", true);

// Чтение содержимого файла
[file, error] = io.readFileE("test/tmp/writeFileLine-01.out")
// Вывод содержимого файла
wait io.printUndefinedE(file);
```

ode-01.sbl

```
// Задание второстепенных параметров модели
float sigma;
float R;
float b;

float x;
float y;
float z;

// Задание интегрируемых параметров
float __dt_x;
float __dt_y;
float __dt_z;

// Задание СДУ
func __ODE[*]()
{
    __dt_x = sigma*(y-x);
    __dt_y = R*x - y - x*z;
    __dt_z = -b*z + x*y;
}
```

scene-04.sbl

```
import "test/source/sbl/modules/scene/ode-01.sbl" ("sbl") type LS;
// Инициализация модели
LS ls = {
    sigma : 10, b : 8./3, R : 28,
    x : 10, y : 1, z : 1,
};

// Инициализации сцены
import "scene" scene = {
    t : 0.0, tk : 10.0, dt : 0.01,
    callback_period : 100,
    callback : func [scene, ls] () { print scene.t + ", " + ls.x +
", " + ls.y + ", " + ls.z; },
    actors : [ls],
};

// Запуск моделирования
wait scene.start();
```