Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

C++ остаётся одним из наиболее актуальных и значимых языков программирования благодаря своей мощности, гибкости и эффективности. Он широко используется в разработке системного и прикладного программного обеспечения, включая операционные системы, драйверы, встроенные системы и игры.

Практическая значимость C++ заключается в его производительности и возможности низкоуровневого управления памятью, что делает его незаменимым для ресурно-требовательных приложений. Кроме того, C++ поддерживает парадигмы объектно-ориентированного, процедурного и функционального программирования, что позволяет создавать сложные и многофункциональные программы.

C++ активно используется в таких областях, как финансовые технологии, научные вычисления и графические приложения. Он также имеет большое сообщество разработчиков и обширную библиотеку кода, что способствует быстрому решению задач и обмену опытом.

Важность C++ также подкрепляется его применением в учебных заведениях для обучения базовым и продвинутым концепциям программирования. Его изучение открывает путь к пониманию более высокоуровневых языков и

технологий.

Объектно-ориентированные языки программирования, в частности C++, могут быть использованы для моделирования сложных технических систем и сложных нелинейных процессов.

Таким образом, целями курсовой работы является получение практических навыков в:

- работе с файлами и данными, хранящимися в них;
- построении дерева иерархии объектов;
- взаимодействии между объектами этого дерева иерархии;
- моделировании сложных технических систем с использованием языка программирования C++.

Для достижения поставленных целей в курсовой работе были сформулированы и решены следующие задачи:

1. Организовывать иерархическое построение объектов нескольких классов.
2. Научиться изменять функциональность программы с учётом постановки задачи.
3. Смоделировать расстановку символов на доске с использованием языка программирования C++.
4. Научиться формулировать постановку задачи и подводить итоги выполненной работы на концептуальном уровне.

1 ПОСТАНОВКА ЗАДАЧИ

Дана квадратная доска с клетками 10 x 10. Нумерация позиций по горизонтали и по вертикали начинается с 1, слева на права и сверху вниз. В каждой клетке можно записать символ или стереть. При записи нового символа содержимое клетки стирается. По доске, по клеткам перемещается робот, который может стереть содержимое клетки или записать новый символ.

Исходное положение робота в позиции (1,1).

Действует робот по командам. Команды имеют вид:

```
MOVE «число 1» «число 2»
```

где, «число 1» и «число 2» принимают целочисленные значения.

```
WRITE «символ»
END
```

По команде MOVE робот перемещается над клеткой с координатой («число 1», «число 2»), где «число 1» номер строки, а «число 2» номер столбца.

По команде WRITE стирает содержимое клетки и пишет «символ».

По команде END система завершает работу.

Доска моделируется в файле field.txt и первоначально все позиции (клетки) заполняются значением 8.

На вход подаются последовательно команды. Символ для записи принадлежит латинскому алфавиту.

Построить систему, которая использует объекты:

1. Объект «система».
2. Объект для создания файла и формирования исходного содержимого.
3. Объект для чтения команд. После чтения очередной команды объект выдает сигнал с текстом, содержащим команду. Если команда не распознана, то выдает сигнал об ошибке.

4. Объект для перемещения робота в заданной позиции, моделируется перемещением курсора в файле согласно координате. Если обнаружена ошибка в значениях координат объект выдает сигнал об ошибке с соответствующим текстом.

5. Объект для записи символа в файле в установленную позицию. Объект пишет символ в установленной позиции или выдает сигнал о недопустимости записи. В одну клетку подряд запись символа не производиться. Пишем только символы латинского алфавита.

6. Объект для вывода текста сообщения об ошибке в конце файла. Сообщения выводится с новой строки. Первое сообщение выводится с 11 строки (после поля 10 х 10).

7. Объект для вывода всего содержимого файла на консоль. После вывода выдается сигнал о завершении.

Написать программу, реализующую следующий алгоритм:

1. Вызов метода объекта «система» bild_tree_objects ( ):
    1.1. Построение дерева иерархии объектов.
    1.2. Установка связей сигналов и обработчиков между объектами.
2. Вызов метода объекта «система» exec_app ( ):
    2.1. Приведение всех объектов в состояние готовности.
    2.2. Создание файла и формирование исходного содержания.
    2.3. Если файл не удается создать, то выдает сообщение на консоль:

```
File field.txt not create
```

и программа завершает работу.

    1.1. Цикл для обработки вводимых команд:
        1.1.1. Выдача сигнала объекту для ввода команды.
        1.1.2. Отработка команды.
    1.2. После ввода команды «END» вывод на консоль содержимого

файла.

Все приведенные сигналы и соответствующие обработчики должны быть реализованы.

Все сообщения об ошибках пишутся в файле построчно, с новой строки.

В набор поддерживаемых команд добавить команду «SHOWTREE» и по этой команде вывести дерево иерархии объектов системы с отметкой о готовности и завершить работу программы.

Пример иерархии объектов:

```
Объект «система».
    Объект «для создания и подготовки файла».
    Объект «чтения команд».
    Объект «перемещения робота».
        Объект «записи символа в файл».
    Объект «вывода текста в конце файла».
    Объект «вывод содержимого файла на консоль».
```

1.1 Описание входных данных

Построчно множество команд, в любом количестве и любой последовательности.

```
MOVE «целое число» «целое число»
```

или

```
WRITE «символ»
```

Последняя команда присутствует всегда:

```
END
```

Пример ввода:

```
MOVE 3 3
MOVE 0 3
SEEK 5
WRITE I
WRITE K
MOVE 4 4
WRITE K
MOVE 4 1
```

```
WRITE !
END
```

1.2 Описание выходных данных

Десять строк по десять символов в каждой, согласно сформированному в файле полю.

Если какая-либо координата для команды MOVE не принадлежит интервалу [1, 10] то выводится сообщение:

```
Coordinate is wrong ( «значение номера строки», «значение номера столбца» )
```

Если символ не принадлежит латинскому алфавиту:

```
The letter does not belong to the Latin alphabet: «символ»
```

При попытке подряд, без изменения значения координат текущей клетки, записать букву в клетку, выдается сообщение:

```
Attempt to sequentially write to a cell ( «значение номера строки», «значение номера столбца» )
```

Если команда не распознана, то выдается сообщение:

```
ERROR command: «текст команды с параметрами»
```

Пример вывода:

```
8888888888
8888888888
88I8888888
888K888888
8888888888
8888888888
8888888888
8888888888
8888888888
8888888888
Coordinate is wrong ( 0, 3 )
ERROR command: SEEK 5
Attempt to sequentially write to a cell ( 3, 3 )
The letter does not belong to the Latin alphabet: !
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект app класса application предназначен для взаимодействия с программой;
- объект fc класса file_creator предназначен для использования в дереве иерархии объектов;
- объект cr класса command_reader предназначен для использования в дереве иерархии объектов;
- объект rm класса robot_mover предназначен для использования в дереве иерархии объектов;
- объект cw класса character_writer предназначен для использования в дереве иерархии объектов;
- объект fw класса file_writer предназначен для использования в дереве иерархии объектов;
- объект co класса console_outputer предназначен для использования в дереве иерархии объектов;
- объект creator класса file_creator предназначен для использования в методе класса application;
- объект mover класса robot_mover предназначен для использования в методах классов robot_mover и character_writer;
- объект writer класса character_writer предназначен для использования в методах классов robot_mover и character_writer;
- объект creator класса file_creator предназначен для использования в методах классов character_writer, console_outputer и file_writer;
- функция main для основная функция программы;
- std::ofstream - класс стандартного потока вывода, который

используетсядля открытия файлов и записи данных в них;

- std::ifstream - класс стандартного потока ввода, который используется для открытия файлов и чтения данных из них;
- std::fstream - класс стандартного потока ввода-вывода, который используется для открытия файлов и выполнения операция чтения и записи данных в них;
- std::ios::app - флаг, используемый при открытии файла, чтобы данные добавлялись в конец файла без удаления его текущего содержимого;
- seekp - метод, который устанавливает позицию указателя записи в потоке;
- put - метод, который записывает один символ в поток;
- is_open - метод, который проверяет успешно ли открыт файл;
- close - метод, который закрывает файловый поток, очищая буферы и освобождая ресурсы, связанные с файлом;
- std::to_string - функция, которая преобразует числовое значение в строку;
- std::ostringstream - класс для создания объектов, которые могут использоваться как потоки вывода для форматирования строк в стиле потоков;
- str - метод класса std::ostringstream, который возвращает накопленное содержимое потока в виде строки;
- std::chrono::system_clock::time_point - тип, представляющий точку времени в системных часах;
- std::chrono::system_clock::now() - функция, возвращающая текущий момент времени в виде объекта std::chrono::system_clock::time_point;
- std::chrono::seconds - тип, представляющий длительность в секундах;
- std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch()) - функция, преобразующая длительность времени с начала эпохи (1 января 1970 года) до текущего момента в секунды;

- std::chrono::milliseconds - тип, представляющий длительность в миллисекундах;
- std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()) - функция, преобразующая длительность времени с начала эпохи (1 января 1970 года) до текущего момента в миллисекунды;
- count - метод, который возвращает количество единиц в объекте длительности.

Класс application:

- свойства/поля:
  - поле позволяет программе работать, по умолчанию равно ИСТИНА:
    - наименование — working;
    - тип — bool;
    - модификатор доступа — private;
- функционал:
  - метод application — параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 1;
  - метод build_tree_objects — создаёт дерево иерархии объектов;
  - метод exec_app — запускает программу;
  - метод end — завершает работу программы.

Класс file_creator:

- свойства/поля:
  - поле имя файла:
    - наименование — filename;
    - тип — std::string;
    - модификатор доступа — private;
- функционал:

- метод file_creator — параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 2;
- метод create_file — создаёт файл и подготавливает его к работе;
- метод get_filename — возврат поля имени объекта.

Класс command_reader:

- функционал:
  - метод command_reader — параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 2;
  - метод read_command — читает команды.

Класс robot_mover:

- свойства/поля:
  - поле номер строки, в которой находится робот, по умолчанию равно 1:
    - наименование — line;
    - тип — int;
    - модификатор доступа — private;
  - поле номер столбца, в котором находится робот, по умолчанию равно 1:
    - наименование — column;
    - тип — int;
    - модификатор доступа — private;
- функционал:
  - метод robot_mover — параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 2;

- метод move_robot — перемещает робота по полю;
- метод get_line — возвращает значение поля line;
- метод get_column — возвращает значение поля column.

Класс character_writer:

- свойства/поля:
  - поле номер строки, в которой находился робот в предыдущий раз, по умолчанию равно 0:
    - наименование — line;
    - тип — int;
    - модификатор доступа — private;
  - поле номер столбца, в котором находился робот в предыдущий раз, по умолчанию равно 0:
    - наименование — column;
    - тип — int;
    - модификатор доступа — private;
- функционал:
  - метод character_writer — параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 3;
  - метод write_character — записывает символ в файл;
  - метод set_line_column — устанавливает значения полям line и column по значениям своих параметров соответственно.

Класс file_writer:

- функционал:
  - метод file_writer — параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 2;

- метод write_in_file — записывает ошибки в файл.

Класс console_outputer:

- функционал:
  - метод console_outputer — параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 2;
  - метод output_in_console — выводит содержимое файла в консоль.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	application			"система"	
2	file_creator			создаёт файл и подготавливает его к работе	
3	command_reader			читает команды	
4	robot_mover			перемещает робота по полю	
5	character_writer			записывает символ в файл	
6	file_writer			записывает ошибки в файл	
7	console_outputer			выводит содержимое файла в консоль	

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм функции main

Функционал: основная функция программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта app класса application с помощью параметризированного конструктора с параметром nullptr	2
2		вызов метода build_tree_objects от объекта app	3
3		возврат результата метода exec_app от объекта app	∅

3.2 Алгоритм конструктора класса application

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 1.

Параметры: base_class* parent - указатель на головной объект, std::string name - имя данного объекта, по умолчанию равно "system".

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса application

№	Предикат	Действия	№ перехода
1			∅

3.3 Алгоритм метода build_tree_objects класса application

Функционал: создаёт дерево иерархии объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода build_tree_objects класса application

№	Предикат	Действия	№ перехода
1		создание указателя fc на объект класса file_creator с помощью оператора new и параметризированного конструктора file_creator данного класса с параметром указателем на данный объект this	2
2		создание указателя cr на объект класса command_reader с помощью оператора new и параметризированного конструктора command_reader данного класса с параметром указателем на данный объект this	3
3		создание указателя rm на объект класса robot_mover с помощью оператора new и параметризированного конструктора robot_mover данного класса с параметром указателем на данный объект this	4
4		создание указателя cw на объект класса character_writer с помощью оператора new и параметризированного конструктора character_writer данного класса с параметром указателем rm	5
5		создание указателя fw на объект класса file_writer с помощью оператора new и параметризированного конструктора file_writer данного класса с параметром указателем на данный объект this	6
6		создание указателя co на объект класса console_outputer с помощью	7

№	Предикат	Действия	№ перехода
		оператора new и параметризированного конструктора console_outputer данного класса с параметром указателем на данный объект this	
7		вызов метода set_connect от указателя cr с параметрами: сигнал метода read_command класса command_reader, указатель rm, обработчик метода move_robot класса robot_mover	8
8		вызов метода set_connect от указателя cr с параметрами: сигнал метода read_command класса command_reader, указатель cw, обработчик метода write_charachter класса character_writer	9
9		вызов метода set_connect от указателя cr с параметрами: сигнал метода read_command класса command_reader, указатель co, обработчик метода output_in_console класса console_outputer	10
10		вызов метода set_connect от указателя cr с параметрами: сигнал метода read_command класса command_reader, указатель на данный объект this, обработчик метода print_branch_status класса base_class	11
11		вызов метода set_connect от указателя cr с параметрами: сигнал метода read_command класса command_reader, указатель fw, обработчик метода write_in_file класса file_writer	12
12		вызов метода set_connect от указателя rm с параметрами: сигнал метода move_robot класса robot_mover, указатель fw, обработчик метода write_in_file класса file_writer	13
13		вызов метода set_connect от указателя cw с параметрами: сигнал метода write_character класса character_writer, указатель fw, обработчик метода write_in_file класса file_writer	14
14		вызов метода set_connect от указателя co с параметрами: сигнал метода output_in_console класса console_outputer, указатель на данный объект this, обработчик метода end класса application	∅

3.4 Алгоритм метода exec_app класса application

Функционал: запускает программу.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода exec_app класса application

№	Предикат	Действия	№ перехода
1		вызов метода set_status_on_branch с параметром 1	2
2		инициализация указателя creator на объект класса file_creator путём приведения результата функции get_child с параметром "file_creator" к типу file_creator*	3
3	результат метода create_file от указателя creator равен ЛОЖЬ?	вывод на экран сообщения "\nFile field.txt not create"	4
			5
4		завершение работы программы	∅
5	значение поля working равно ИСТИНА?	создание переменной command строкового типа	6
		возврат значения 0	∅
6		вызов метода emit_signal с параметрами: сигнал метода read_command класса command_reader, значение переменной command	5

3.5 Алгоритм метода end класса application

Функционал: завершает работу программы.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода end класса application

№	Предикат	Действия	№ перехода
1		присваивание полю working данного объекта значение ЛОЖЬ	∅

3.6 Алгоритм конструктора класса file_creator

Функционал: параметризированный конструктор.

Параметры: base_class* parent - указатель на головной объект, std::string name - имя данного объекта, по умолчанию равно "file_creator".

Алгоритм конструктора представлен в таблице 7.

Таблица 7 – Алгоритм конструктора класса file_creator

№	Предикат	Действия	№ перехода
1			∅

3.7 Алгоритм метода create_file класса file_creator

Функционал: создаёт файл и подготавливает его к работе.

Параметры: нет.

Возвращаемое значение: bool - результат создания файла.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода create_file класса file_creator

№	Предикат	Действия	№ перехода
1		создание переменной now типа std::chrono::system_clock::time_point, которая представляет текущий момент времени и	2

№	Предикат	Действия	№ перехода
		присваиванеи ей результата функции std::chrono:system_clock::now, возвращающей текущее время в системных часах	
2		создание переменной s типа std::chrono::seconds, которая представляет количество прошедших секунд с начала эпохи до текущего момента времени, значение которого вычисляется из now с помощью функции time_since_epoch, которое представляет длительность времени с начала эпохи, и затем приводится к типу std::chrono::seconds, затем остаток от деления на 60 берется для получения секунд текущей минуты	3
3		создание переменной ms типа std::chrono::milliseconds, которая представляет количество прошедших миллисекунд с начала эпохи до текущего момента времени, значение которого вычисляется из now с помощью функции time_since_epoch, которое представляет длительность времени с начала эпохи, и затем приводится к типу std::chrono::milliseconds, затем остаток от деления на 1000 берется для получения миллисекунд текущей секунды	4
4		создание объекта oss класса std::ostringstream	5
5		добавление в поток вывода oss значения строки "field", количества секунд, количества миллисекунд и строки ".txt"	6
6		присваивание значению поля filename значение переменной oss, представленной в виде строки	7
7		открытие файла с именем значения поля filename	8

№	Предикат	Действия	№ перехода
		для записи данных в него	
8	файл не открыт?	возврат значения ЛОЖЬ	∅
		инициализация переменной line строкового типа значением "8888888888"	9
9		внесение значения переменной line в файл	10
10		добавление к значению переменной line строки "\n" в начало	11
11		инициализация переменной i целочисленного типа значением 1	12
12	значение переменной i меньше 10	внесение значения переменной line в файл	13
		закрытие файла	14
13		увеличение значения переменной i на 1	12
14		возврат значения ИСТИНА	∅

3.8 Алгоритм конструктора класса command_reader

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 2.

Параметры: base_class* parent - указатель на головной объект, std::string name - имя данного объекта, по умолчанию равно "command_reader".

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса command_reader

№	Предикат	Действия	№ перехода
1			∅

3.9 Алгоритм метода read_command класса command_reader

Функционал: читает команды.

Параметры: std::string command - переменная строкового типа.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода read_command класса command_reader

№	Предикат	Действия	№ перехода
1		ввод значения параметра command с клавиатуры	2
2	значение переменной command равно строке "MOVE"?	вызов метода emit_signal с параметрами: сигнал метода move_robot класса robot_mover, значение переменной command	∅
	значение переменной command равно строке "WRITE"?	вызов метода emit_signal с параметрами: сигнал метода write_character класса character_writer, значение переменной command	∅
	значение переменной command равно строке "END"?	вызов метода emit_signal с параметрами: сигнал метода output_in_console класса console_outputer, значение переменной command	∅
	значение переменной command равно строке "SHOWTREE"?	вызов метода emit_signal с параметрами: сигнал метода print_branch_status класса base_class, значение переменной command	3
		создание переменной error строкового типа	4
3		завершение работы программы	∅
4		занесение оставшихся данных из потока ввода в значение переменной error	5
5		добавление к значению переменной error строки "\nERROR command: " и значение переменной command в начало	6
6		вызов метода emit_signal с параметрами: сигнал метода write_in_file класса file_writer, значение	∅

№	Предикат	Действия	№ перехода
		переменной error	

3.10 Алгоритм конструктора класса robot_mover

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 2.

Параметры: base_class* parent - указатель на головной объект, std::string name - имя данного объекта, по умолчанию равно "robot_mover".

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса robot_mover

№	Предикат	Действия	№ перехода
1			∅

3.11 Алгоритм метода move_robot класса robot_mover

Функционал: перемещает робота по полю.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода move_robot класса robot_mover

№	Предикат	Действия	№ перехода
1		создание переменных line и column целочисленного типа	2
2		ввод значений переменных line и column с клавиатуры	3
3	значения переменных line и column не принадлежат	инициализация переменной error строкового типа значениями строки "\nCoordinate is wrong (",	4

№	Предикат	Действия	№ перехода
	отрезку [1, 10]?	переменной line в виде строки, строки ", ", переменной column в виде строки и строки ")"	
		инициализация указателя mover на объект класса robot_mover путём приведения результата функции get_child с параметром "robot_mover" к типу robot_mover*	5
4		вызов метода emit_signal с параметрами: сигнал метода write_in_file класса file_writer, значение переменной error	∅
5		инициализация указателя writer на объект класса character_writer путём приведения результата функции get_child от указателя mover с параметром "character_writer" к типу character_writer*	6
6	значение поля line указателя mover не равен значению переменной line ИЛИ значение поля line указателя mover не равен значению переменной line	вызов метода set_line_column от указателя writer с параметрами: значение поля line указателя mover и значение поля column указателя mover	7
			∅
7		присваивание полю line указателя mover значение переменной line и полю column указателя mover значение переменной column	∅

3.12 Алгоритм конструктора класса character_writer

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 3.

Параметры: base_class* parent - указатель на головной объект, std::string name - имя данного объекта, по умолчанию равно "character_writer".

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса character_writer

№	Предикат	Действия	№ перехода
1			∅

3.13 Алгоритм метода write_character класса character_writer

Функционал: записывает символ в файл.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода write_character класса character_writer

№	Предикат	Действия	№ перехода
1		создание переменной character символьного типа	2
2		ввод значения переменной character с клавитуры	3
3	значение переменной character в таблице ASCII не находится между символами 'A' и 'Z' И не находится между символами 'a' и 'z'	инициализация переменной error строкового типа значениями строки "\nThe letter does not belong to the Latin alphabet: " и символа character в виде строки	4
		инициализация указателя mover на объект класса robot_mover путём приведения результата функции get_child с параметром "robot_mover" к типу robot_mover*	5

№	Предикат	Действия	№ перехода
4		вызов метода emit_signal с параметрами: сигнал метода write_in_file класса file_writer, значение переменной error	∅
5		инициализация указателя writer на объект класса character_writer путём приведения результата функции get_child от указателя mover с параметром "character_writer" к типу character_writer*	6
6	значение поля line указателя writer равно результату метода get_line от указателя mover И значение поля column указателя writer равно результату метода get_column от указателя mover?	инициализация переменной error строкового типа значениями строки "\nAttempt to sequentially write to a cell (", поля line указателя writer в виде строки, строки ", ", поля column указателя writer в виде строки и строки ")"	7
		присваивание полю line указателя writer значение результата метода get_line от указателя mover и полю column указателя writer значение результата метода get_column от указателя mover	8
7		вызов метода emit_signal с параметрами: сигнал метода write_in_file класса file_writer, значение переменной error	∅
8		инициализация указателя creator на объект класса file_creator путём приведения результата функции get_child с параметром "file_creator" к типу file_creator*	9
9		открытие файла с именем значения результата метода get_filename от указателя creator для записи	10

№	Предикат	Действия	№ перехода
		и чтения данных	
10		перемещение указателя на позицию: строка с индексом значения поля line указателя writer - 1, столбец с индексом значения поля column указателя writer - 1	11
11		записать символ character вместо символа, стоящего на позиции указателя	12
12		закрытие файла	∅

3.14 Алгоритм метода set_line_column класса character_writer

Функционал: устанавливает значения полям line и column по значениям своих параметров соответственно.

Параметры: int x - номер строки, int y - номер столбца.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода set_line_column класса character_writer

№	Предикат	Действия	№ перехода
1		присваивание полю line значение параметра x и полю column значение параметра y	∅

3.15 Алгоритм конструктора класса file_writer

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 2.

Параметры: base_class* parent - указатель на головной объект, std::string name - имя данного объекта, по умолчанию равно "file_writer".

Алгоритм конструктора представлен в таблице 16.

Таблица 16 – Алгоритм конструктора класса file_writer

№	Предикат	Действия	№ перехода
1			∅

3.16 Алгоритм метода write_in_file класса file_writer

Функционал: записывает ошибки в файл.

Параметры: std::string& error - текст с ошибкой.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода write_in_file класса file_writer

№	Предикат	Действия	№ перехода
1		инициализация указателя creator на объект класса file_creator путём приведения результата функции get_child с параметром "file_creator" к типу file_creator*	2
2		открытие файла с именем значения результата метода get_filename от указателя creator для записи данных в конец файла	3
3		внесение значения переменной error в файл	4
4		закрытие файла	∅

3.17 Алгоритм конструктора класса console_outputer

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с передачей в него параметров parent, name и 2.

Параметры: base_class* parent - указатель на головной объект, std::string name - имя данного объекта, по умолчанию равно "console_outputer".

Алгоритм конструктора представлен в таблице 18.

Таблица 18 – Алгоритм конструктора класса console_outputer

№	Предикат	Действия	№ перехода
1			∅

3.18 Алгоритм метода output_in_console класса console_outputer

Функционал: выводит содержимое файла в консоль.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода output_in_console класса console_outputer

№	Предикат	Действия	№ перехода
1		инициализация указателя creator на объект класса file_creator путём приведения результата функции get_child с параметром "file_creator" к типу file_creator*	2
2		открытие файла с именем значения результата метода get_filename от указателя creator для чтения данных из него	3
3		создание переменной line строкового типа	4
4	из файла можно получить строку?	вывод этой строки на экран и переход на новую строку	4
		закрытие файла	5
5		вызов метода emit_signal с параметрами: сигнал метода end класса application, значение переменной line	∅

3.19 Алгоритм метода get_line класса robot_mover

Функционал: возвращает значение поля line.

Параметры: нет.

Возвращаемое значение: int - значение поля line.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода get_line класса robot_mover

№	Предикат	Действия	№ перехода
1		возврат значения поля line	∅

3.20 Алгоритм метода get_column класса robot_mover

Функционал: возвращает значение поля column.

Параметры: нет.

Возвращаемое значение: int - значение поля column.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода get_column класса robot_mover

№	Предикат	Действия	№ перехода
1		возврат значения поля column	∅

3.21 Алгоритм метода get_filename класса file_creator

Функционал: возврат поля имени объекта.

Параметры: нет.

Возвращаемое значение: std::string - имя файла.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода get_filename класса file_creator

№	Предикат	Действия	№ перехода
1		возврат значения поля filename	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-13.

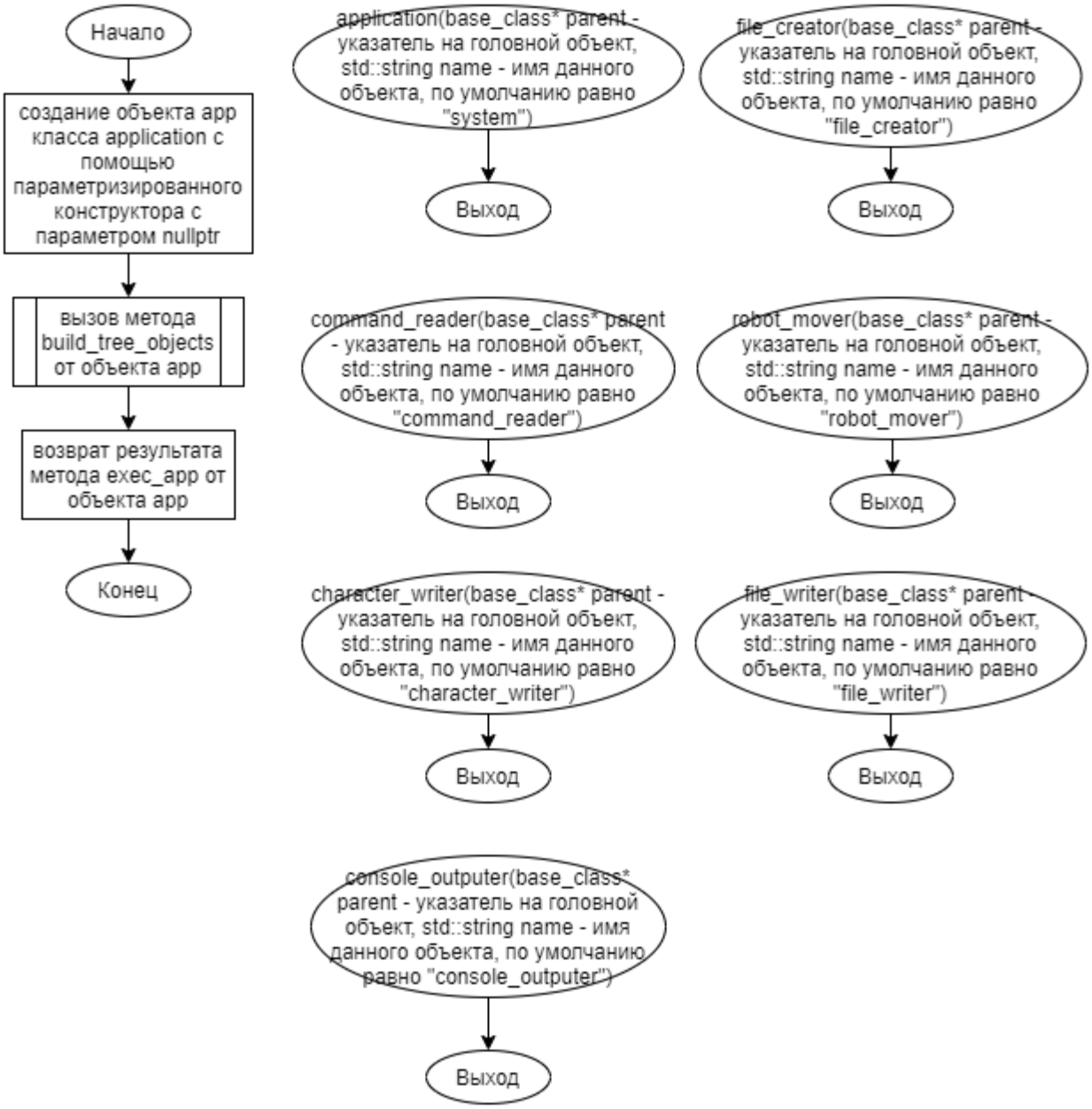

Рисунок 1 – Блок-схема алгоритма

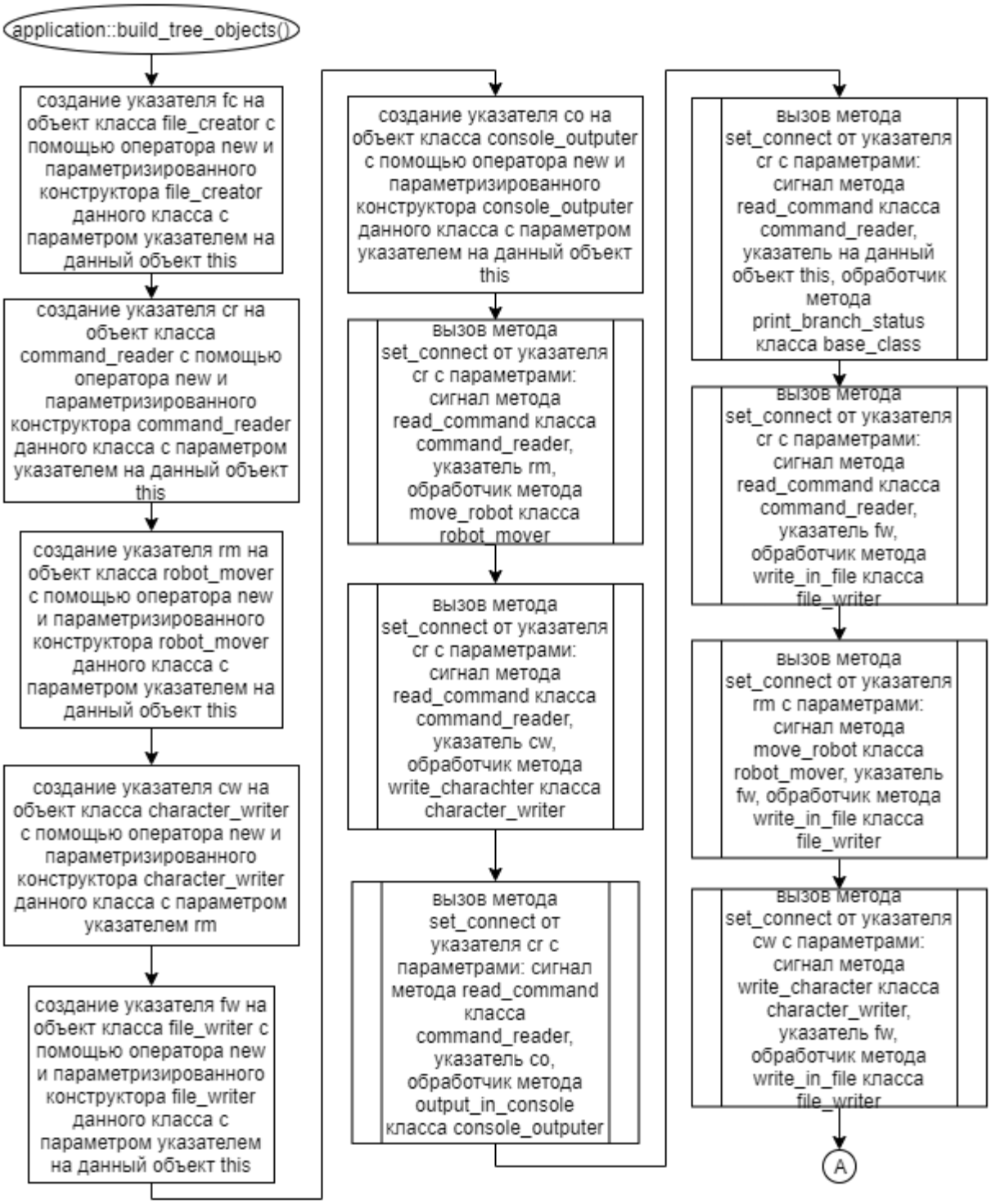

Рисунок 2 – Блок-схема алгоритма

Рисунок 3 – Блок-схема алгоритма

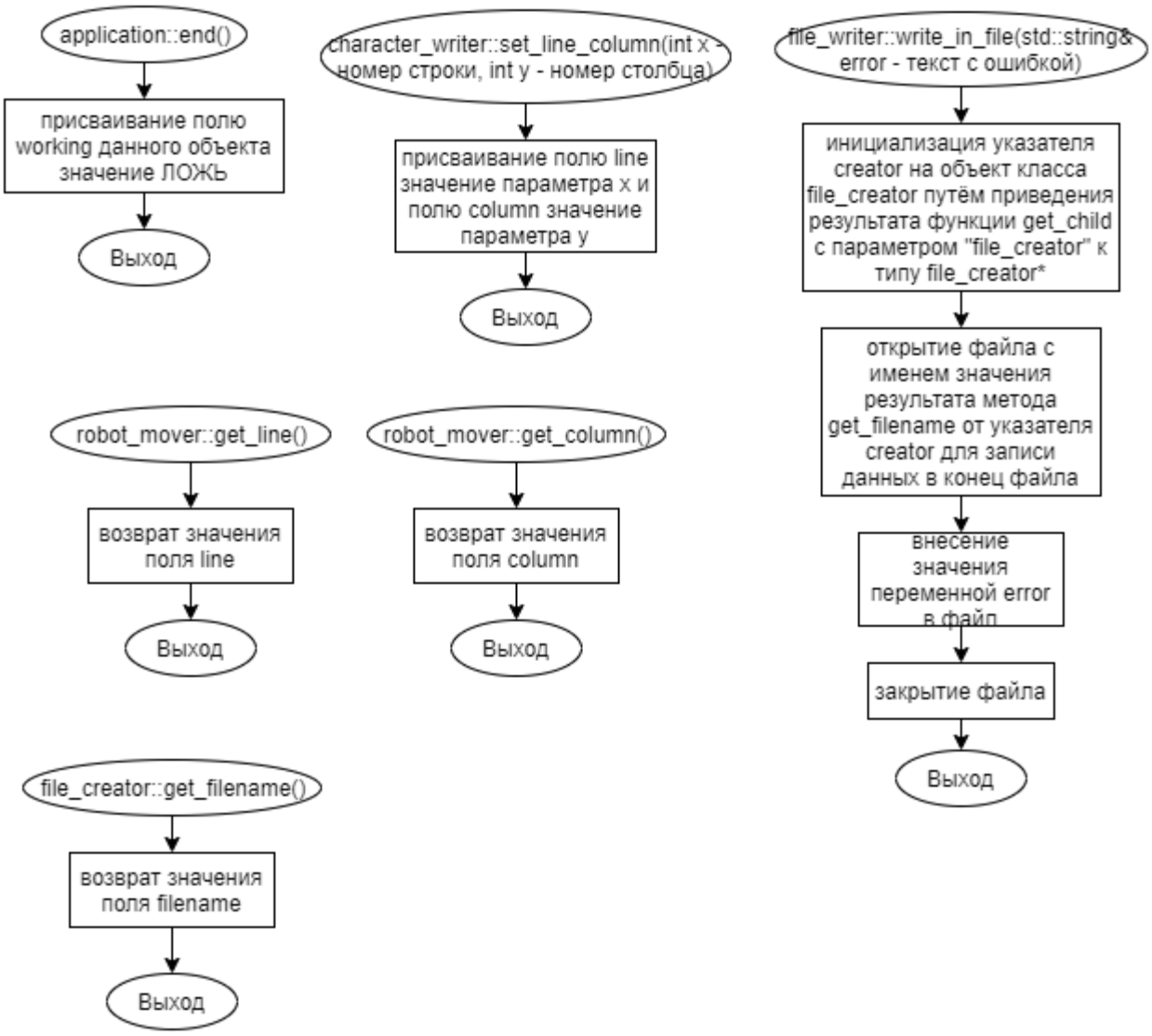

Рисунок 4 – Блок-схема алгоритма

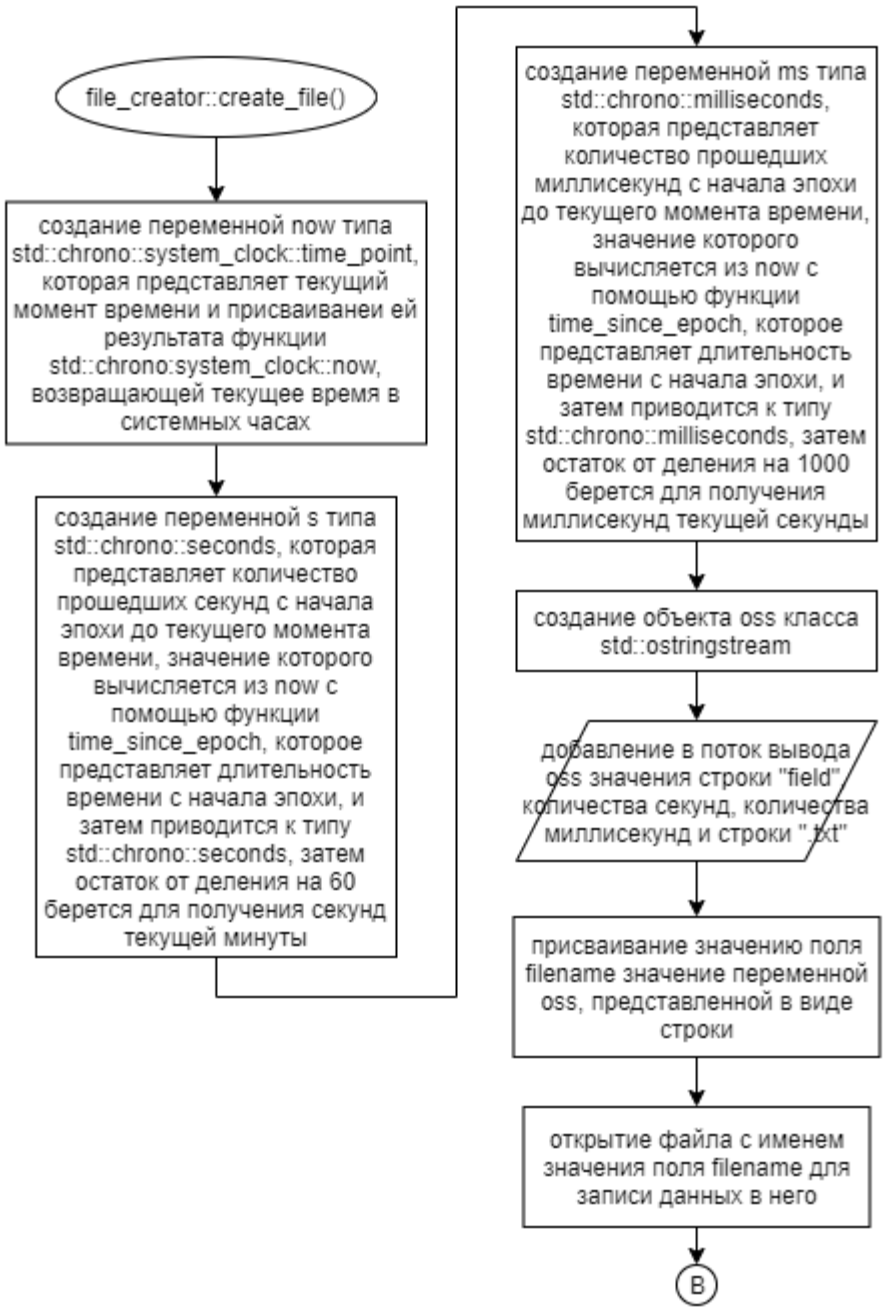

Рисунок 5 – Блок-схема алгоритма

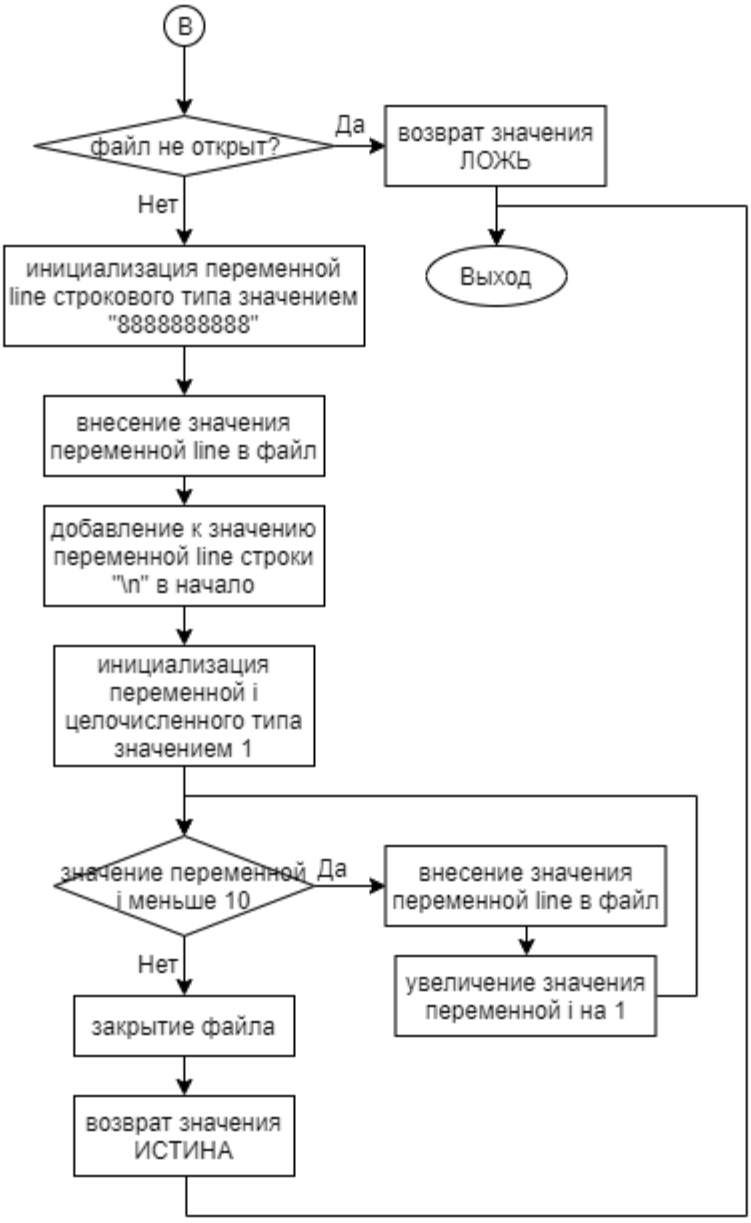

Рисунок 6 – Блок-схема алгоритма

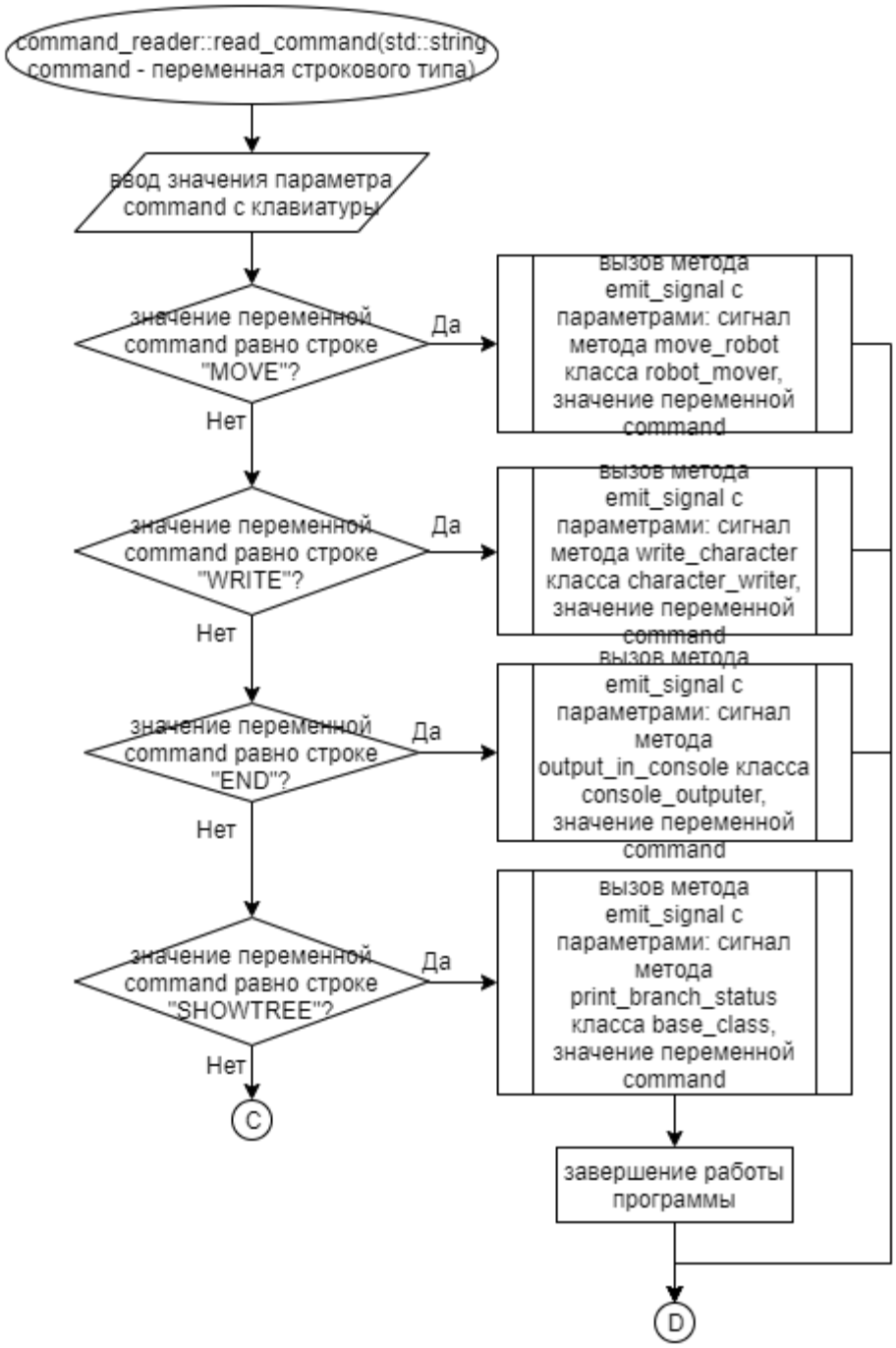

Рисунок 7 – Блок-схема алгоритма

Рисунок 8 – Блок-схема алгоритма

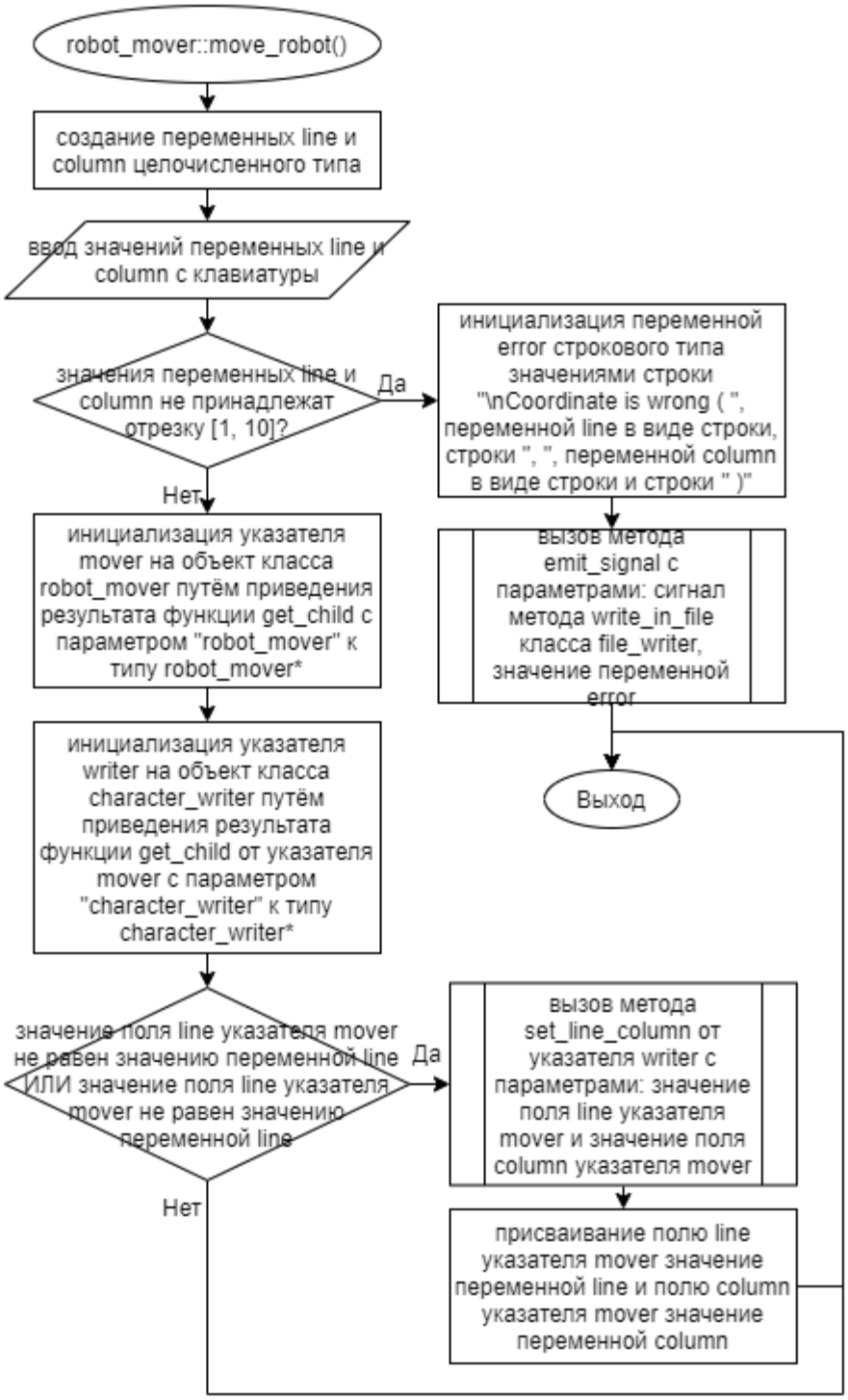

Рисунок 9 – Блок-схема алгоритма

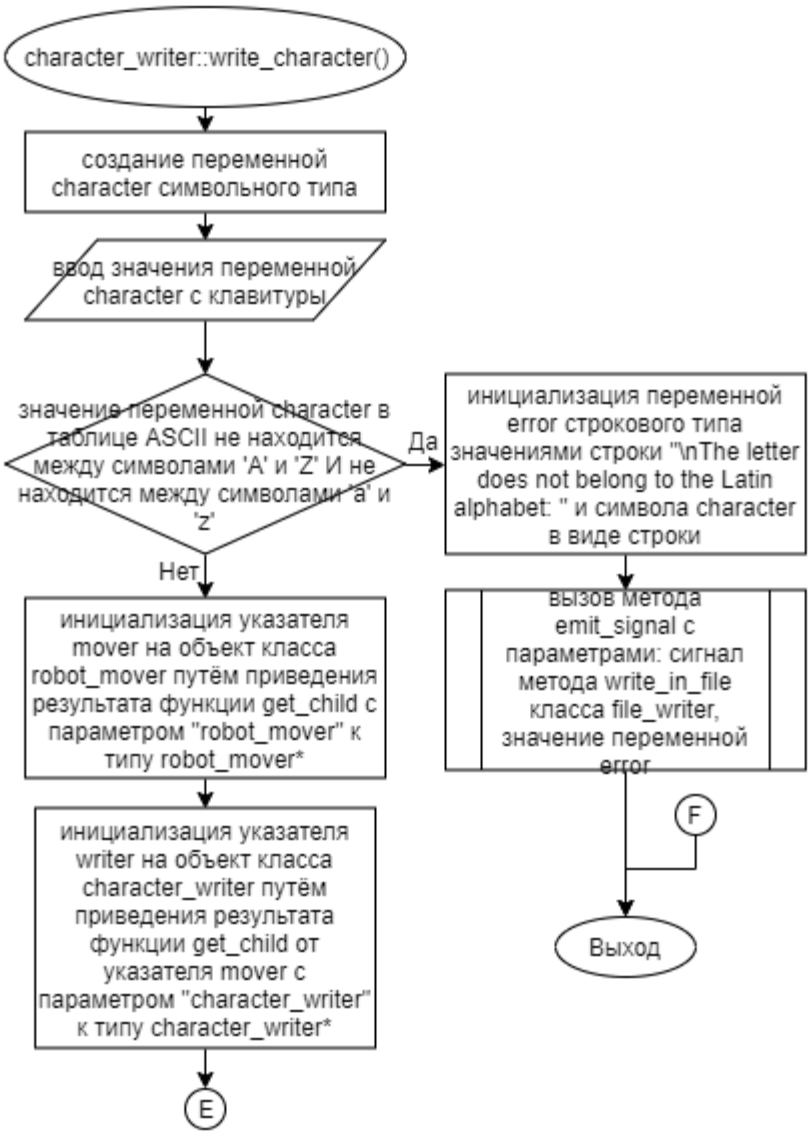

Рисунок 10 – Блок-схема алгоритма

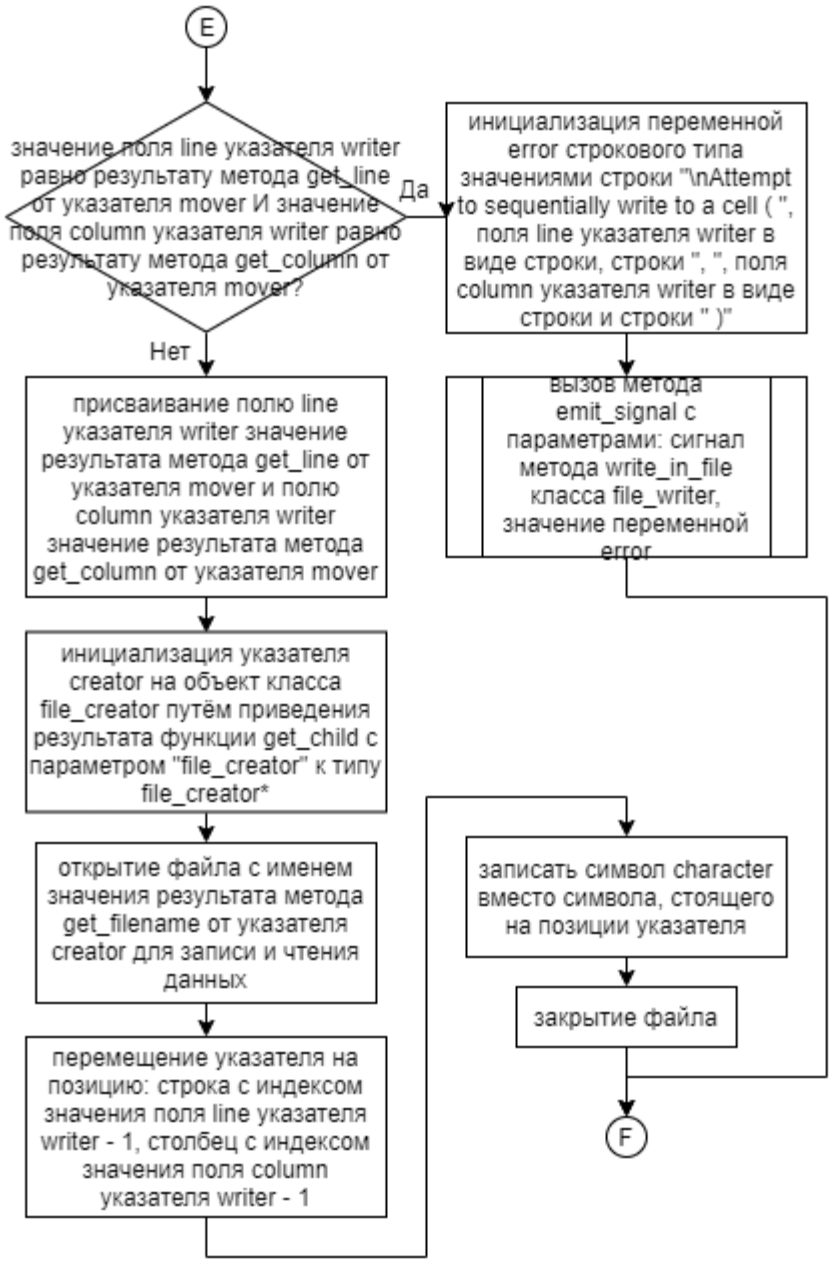

Рисунок 11 – Блок-схема алгоритма

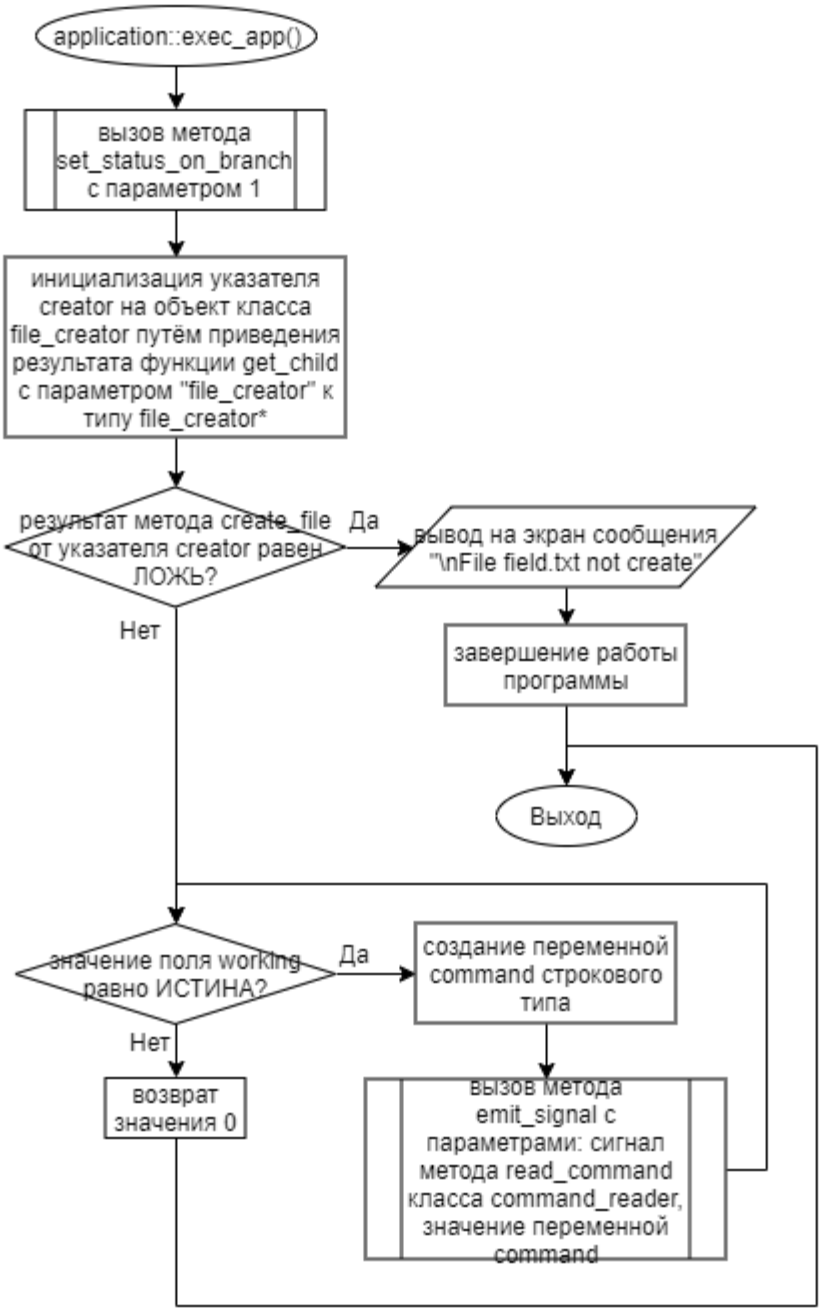

Рисунок 12 – Блок-схема алгоритма

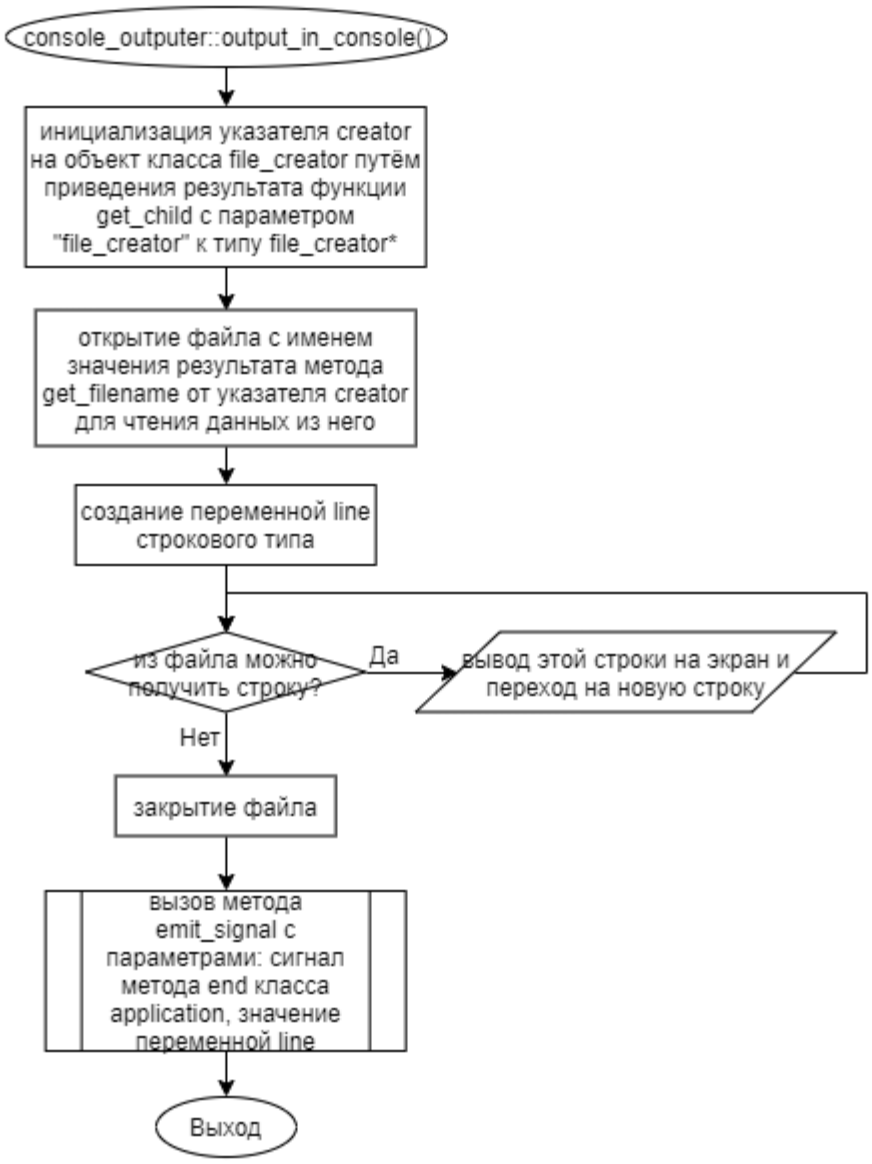

Рисунок 13 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"
#include "file_creator.h"
#include "command_reader.h"
#include "robot_mover.h"
#include "character_writer.h"
#include "file_writer.h"
#include "console_outputer.h"

application::application(base_class*    parent,    std::string    name)    :
base_class(parent, name, 1) {}

void application::build_tree_objects()
{
   file_creator* fc = new file_creator(this);
   command_reader* cr = new command_reader(this);
   robot_mover* rm = new robot_mover(this);
   character_writer* cw = new character_writer(rm);
   file_writer* fw = new file_writer(this);
   console_outputer* co = new console_outputer(this);
   cr->set_connect(SIGNAL(command_reader::read_command),                  rm,
HANDLER(robot_mover::move_robot));
   cr->set_connect(SIGNAL(command_reader::read_command),                  cw,
HANDLER(character_writer::write_character));
   cr->set_connect(SIGNAL(command_reader::read_command),                  co,
HANDLER(console_outputer::output_in_console));
   cr->set_connect(SIGNAL(command_reader::read_command),                this,
HANDLER(base_class::print_branch_status));
   cr->set_connect(SIGNAL(command_reader::read_command),                  fw,
HANDLER(file_writer::write_in_file));
   rm->set_connect(SIGNAL(robot_mover::move_robot),                       fw,
HANDLER(file_writer::write_in_file));
   cw->set_connect(SIGNAL(character_writer::write_character),             fw,
HANDLER(file_writer::write_in_file));
   co->set_connect(SIGNAL(console_outputer::output_in_console),         this,
HANDLER(application::end));
}

int application::exec_app()
{
```

```
   set_status_on_branch(1);
   file_creator* creator = (file_creator*)(get_child("file_creator"));
   if (!creator->create_file())
   {
      std::cout << "\nFile field.txt not create";
      exit(1);
   }
   while (working)
   {
      std::string command;
      emit_signal(SIGNAL(command_reader::read_command), command);
   }
   return 0;
}

void application::end()
{ working = false; }
```

5.2 Файл application.h

Листинг 2 – application.h

```
#ifndef __APPLICATION__H
#define __APPLICATION__H

#include "base_class.h"

class application : public base_class
{
private:
   bool working = true;
public:
   application(base_class* parent, std::string name = "system");
   void build_tree_objects();
   int exec_app();
   void end();
};

#endif
```

5.3 Файл base_class.cpp

Листинг 3 – base_class.cpp

```
#include "base_class.h"
```

```
base_class::base_class(base_class*    parent,    std::string    name,    int
number):number(number)
{
   this->parent = parent;
   this->name = name;
   if (parent) parent->children.push_back(this);
}
base_class::~base_class()
{ for (base_class* child: children) delete child; }

bool base_class::set_name(std::string name)
{
   if (parent && parent->get_child(name))
      return false;
   this->name = name;
   return true;
}
std::string base_class::get_name()
{ return name; }
base_class* base_class::get_parent()
{ return parent; }
base_class* base_class::get_child(std::string name)
{
   for (base_class* child: children)
      if (child->get_name() == name)
         return child;
   return nullptr;
}

base_class* base_class::search_in_branch(std::string name)
{
   std::function <int(std::string name, base_class* object)> count = [&count]
(std::string name, base_class* object)
   {
      int c = 0;
      if (object->get_name() == name) c++;
      for (int i = 0; i < object->children.size(); i++)
         c += count(name, object->children[i]);
      return c;
   };
   if (count(name, this) != 1) return nullptr;
   if (get_name() == name) return this;
   else
      for (base_class* child: children)
      {
         base_class* object = child->search_in_branch(name);
         if (object) return object;
      }
   return nullptr;
}
base_class* base_class::search_in_tree(std::string name)
{
   base_class* parent = this;
   while (parent->get_parent())
```

```
      parent = parent->get_parent();
   return parent->search_in_branch(name);
}

void base_class::print_branch()
{
   base_class* parent = get_parent();
   while (parent)
   {
      std::cout << "    ";
      parent = parent->get_parent();
   }
   std::cout << name << std::endl;
   for (base_class* child: children)
      child->print_branch();
}
void base_class::print_branch_status()
{
   base_class* parent = get_parent();
   while (parent)
   {
      std::cout << "    ";
      parent = parent->get_parent();
   }
   std::cout << name << " is ";
   if (!status)
      std::cout << "not ";
   std::cout << "ready\n";
   for (base_class* child: children)
      child->print_branch_status();
}

void base_class::set_status(int status)
{
   if (status == 0)
   {
      this->status = 0;
      for (base_class* child: children)
         child->set_status(0);
   }
   else
   {
      if (get_parent() && get_parent()->status == 0)
         this->status = 0;
      else
         this->status = status;
   }
}
void base_class::set_status_on_branch(int status)
{
   set_status(status);
   if (status == 0) return;
   for (base_class* child: children)
      child->set_status_on_branch(status);
}
```

```
base_class* base_class::find(std::string coords)
{
   if (coords.empty()) return nullptr;
   base_class* parent = this;
   if (coords[0] == '.')
   {
      if (coords == ".")
         return this;
      coords.erase(0, 1);
      return this->search_in_branch(coords);
   }
   else if (coords[0] == '/')
   {
      while (parent->get_parent())
         parent = parent->get_parent();
      if (coords == "/")
         return parent;
      if (coords[1] == '/')
      {
         coords.erase(0, 2);
         return parent->search_in_tree(coords);
      }
      coords.erase(0, 1);
   }
   std::string current = "";
   for (int i = 0; i < coords.length(); i++)
   {
      if (coords[i] == '/')
      {
         parent = parent->get_child(current);
         if (!parent) return parent;
         current = "";
      }
      else current += coords[i];
   }
   return parent->get_child(current);
}
void base_class::delete_child(std::string name)
{
   base_class* child = get_child(name);
   if (child)
      for (int i = 0; i < children.size(); i++)
         if (children[i] == child)
         {
            children.erase(children.begin() + i);
            delete child;
            break;
         }
}
bool base_class::set_parent(base_class* object)
{
   if  (!object  ||  !this->parent  ||  object->get_child(this->get_name()))
return false;
   base_class* parent = object;
```

```
   while (parent)
   {
      if (parent == this) return false;
      parent = parent->get_parent();
   }
   base_class* current = this->parent;
   for (int i = 0; i < current->children.size(); i++)
      if (current->children[i] == this)
      {
         current->children.erase(current->children.begin() + i);
         break;
      }
   this->parent = object;
   object->children.push_back(this);
   return true;
}

base_class::connect::connect(TYPE_SIGNAL    signal,    base_class*    object,
TYPE_HANDLER handler)
{
   this->signal = signal;
   this->object = object;
   this->handler = handler;
}
void   base_class::set_connect(TYPE_SIGNAL   signal,   base_class*   object,
TYPE_HANDLER handler)
{
   for (int i = 0; i < connects.size(); i++)
      if (connects[i]->signal == signal && connects[i]->object == object &&
connects[i]->handler == handler)
         return;
   connects.push_back(new connect(signal, object, handler));
}
void   base_class::del_connect(TYPE_SIGNAL   signal,   base_class*   object,
TYPE_HANDLER handler)
{
   for (int i = 0; i < connects.size(); i++)
      if (connects[i]->signal == signal && connects[i]->object == object &&
connects[i]->handler == handler)
      {
         connects.erase(connects.begin() + i);
         return;
      }
}
void base_class::emit_signal(TYPE_SIGNAL signal, std::string& command)
{
   if (status == 0) return;
   TYPE_HANDLER handler;
   base_class* object;
   (this->*signal) (command);
   for (int i = 0; i < connects.size(); i++)
      if (connects[i]->signal == signal)
      {
         handler = connects[i]->handler;
         object = connects[i]->object;
```

```
            if (object->status != 0)
               (object->*handler) (command);
         }
      }
      std::string base_class::get_path()
      {
         base_class* current = get_parent();
         if (!current) return "/";
         std::string path = '/' + get_name();
         while (current->get_parent())
         {
            path = '/' + current->get_name() + path;
            current = current->get_parent();
         }
         return path;
      }
```

5.4 Файл base_class.h

Листинг 4 – base_class.h

```
      #ifndef __BASE_CLASS__H
      #define __BASE_CLASS__H

      #include <string>
      #include <vector>
      #include <iostream>
      #include <functional>
      #include <fstream>

      class base_class
      {
      public:
         const int number;
         base_class(base_class* parent, std::string name = "default", int number =
      0);
         ~base_class();
         bool set_name(std::string name);
         std::string get_name();
         base_class* get_parent();
         base_class* get_child(std::string name);
         base_class* search_in_branch(std::string name);
         base_class* search_in_tree(std::string name);
         void print_branch();
         void print_branch_status();
         void set_status(int status);
         base_class* find(std::string coords);
         void delete_child(std::string name);
         bool set_parent(base_class* object);
         typedef void (base_class::*TYPE_SIGNAL) (std::string&);
```

```
   typedef void (base_class::*TYPE_HANDLER) (std::string);
   void  set_connect(TYPE_SIGNAL  signal,  base_class*  object,  TYPE_HANDLER
handler);
   void  del_connect(TYPE_SIGNAL  signal,  base_class*  object,  TYPE_HANDLER
handler);
   void emit_signal(TYPE_SIGNAL signal, std::string& command);
   std::string get_path();
   void set_status_on_branch(int status);
private:
   std::string name;
   base_class* parent;
   std::vector <base_class*> children;
   int status = 0;
   struct connect
   {
      TYPE_SIGNAL signal;
      base_class* object;
      TYPE_HANDLER handler;
      connect(TYPE_SIGNAL signal, base_class* object, TYPE_HANDLER handler);
   };
   std::vector <connect*> connects;
};
#define SIGNAL(signal_func) (TYPE_SIGNAL) (&signal_func)
#define HANDLER(handler_func) (TYPE_HANDLER) (&handler_func)

#endif
```

5.5 Файл character_writer.cpp

Листинг 5 – character_writer.cpp

```
#include "character_writer.h"
#include "robot_mover.h"
#include "file_writer.h"
#include "file_creator.h"

character_writer::character_writer(base_class*  parent,  std::string  name)  :
base_class(parent, name, 3) {}

void character_writer::write_character()
{
   char character;
   std::cin >> character;
   if  (!(character  >=  'A'  &&  character  <=  'Z'  ||  character  >=  'a'  &&
character <= 'z'))
   {
      std::string  error  =  "\nThe  letter  does  not  belong  to  the  Latin
alphabet: " + std::string(1, character);
      emit_signal(SIGNAL(file_writer::write_in_file), error);
   }
```

```
   else
   {
      robot_mover* mover = (robot_mover*)(get_child("robot_mover"));
      character_writer*       writer       =       (character_writer*)(mover-
>get_child("character_writer"));
      if  (writer->line  ==  mover->get_line()  &&  writer->column  ==  mover-
>get_column())
      {
         std::string error = "\nAttempt to sequentially write to a cell ( " +
std::to_string(writer->line) + ", " + std::to_string(writer->column) + " )";
         emit_signal(SIGNAL(file_writer::write_in_file), error);
      }
      else
      {
         writer->line   =   mover->get_line(),   writer->column   =   mover-
>get_column();
         file_creator* creator = (file_creator*)(get_child("file_creator"));
         std::fstream file(creator->get_filename());
         file.seekp((writer->line - 1) * 11 + (writer->column - 1));
         file.put(character);
         file.close();
      }
   }
}

void character_writer::set_line_column(int x, int y)
{ line = x, column = y; }
```

5.6 Файл character_writer.h

Листинг 6 – character_writer.h

```
#ifndef __CHARACTER_WRITER__H
#define __CHARACTER_WRITER__H

#include "base_class.h"

class character_writer : public base_class
{
private:
   int line = 0, column = 0;
public:
   character_writer(base_class*      parent,      std::string      name      =
"character_writer");
   void write_character();
   void set_line_column(int x, int y);
};

#endif
```

5.7 Файл command_reader.cpp

Листинг 7 – command_reader.cpp

```
#include "command_reader.h"
#include "robot_mover.h"
#include "character_writer.h"
#include "console_outputer.h"
#include "file_writer.h"

command_reader::command_reader(base_class*  parent,  std::string  name)  :
base_class(parent, name, 2) {}

void command_reader::read_command(std::string command)
{
   std::cin >> command;
   if (command == "MOVE")
      emit_signal(SIGNAL(robot_mover::move_robot), command);
   else if (command == "WRITE")
      emit_signal(SIGNAL(character_writer::write_character), command);
   else if (command == "END")
      emit_signal(SIGNAL(console_outputer::output_in_console), command);
   else if (command == "SHOWTREE")
   {
      emit_signal(SIGNAL(base_class::print_branch_status), command);
      exit(1);
   }
   else
   {
      std::string error;
      std::getline(std::cin, error);
      error = "\nERROR command: " + command + error;
      emit_signal(SIGNAL(file_writer::write_in_file), error);
   }
}
```

5.8 Файл command_reader.h

Листинг 8 – command_reader.h

```
#ifndef __COMMAND_READER__H
#define __COMMAND_READER__H

#include "base_class.h"

class command_reader : public base_class
{
public:
```

```
   command_reader(base_class* parent, std::string name = "command_reader");
   void read_command(std::string command);
};

#endif
```

5.9 Файл console_outputer.cpp

Листинг 9 – console_outputer.cpp

```
#include "console_outputer.h"
#include "application.h"
#include "file_creator.h"

console_outputer::console_outputer(base_class* parent, std::string name) :
base_class (parent, name, 2) {}

void console_outputer::output_in_console()
{
   file_creator* creator = (file_creator*)(get_child("file_creator"));
   std::ifstream file(creator->get_filename());
   std::string line;
   while (std::getline(file, line))
      std::cout << line << std::endl;
   file.close();
   emit_signal(SIGNAL(application::end), line);
}
```

5.10 Файл console_outputer.h

Листинг 10 – console_outputer.h

```
#ifndef __CONSOLE_OUTPUTER__H
#define __CONSOLE_OUTPUTER__H

#include "base_class.h"

class console_outputer : public base_class
{
public:
   console_outputer(base_class*      parent,      std::string      name      =
"console_outputer");
   void output_in_console();
};
```

```
#endif
```

5.11 Файл file_creator.cpp

Листинг 11 – file_creator.cpp

```
#include "file_creator.h"
#include <chrono>
#include <sstream>

file_creator::file_creator(base_class*    parent,    std::string    name)    :
base_class(parent, name, 2) {}

bool file_creator::create_file()
{
   std::chrono::system_clock::time_point                now                =
std::chrono::system_clock::now();
   std::chrono::seconds                             s                          =
std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch())    %
60;
   std::chrono::milliseconds                         ms                        =
std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()
) % 1000;
   std::ostringstream oss;
   oss << "field" << s.count() << ms.count() << ".txt";
   filename = oss.str();
   std::ofstream file(filename);
   if (!file.is_open()) return false;
   std::string line = "8888888888";
   file << line;
   line = "\n" + line;
   for (int i = 1; i < 10; i++)
      file << line;
   file.close();
   return true;
}

std::string file_creator::get_filename()
{ return filename; }
```

5.12 Файл file_creator.h

Листинг 12 – file_creator.h

```
#ifndef __FILE_CREATOR__H
```

```
#define __FILE_CREATOR__H

#include "base_class.h"

class file_creator : public base_class
{
private:
   std::string filename;
public:
   file_creator(base_class* parent, std::string name = "file_creator");
   bool create_file();
   std::string get_filename();
};

#endif
```

5.13 Файл file_writer.cpp

Листинг 13 – file_writer.cpp

```
#include "file_writer.h"
#include "file_creator.h"

file_writer::file_writer(base_class*    parent,    std::string    name)    :
base_class(parent, name, 2) {}

void file_writer::write_in_file(std::string& error)
{
   file_creator* creator = (file_creator*)(get_child("file_creator"));
   std::fstream file(creator->get_filename(), std::ios::app);
   file << error;
   file.close();
}
```

5.14 Файл file_writer.h

Листинг 14 – file_writer.h

```
#ifndef __FILE_WRITER__H
#define __FILE_WRITER__H

#include "base_class.h"

class file_writer : public base_class
{
```

```
public:
   file_writer(base_class* parent, std::string name = "file_writer");
   void write_in_file(std::string& error);
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "application.h"

int main()
{
   application app(nullptr);
   app.build_tree_objects();
   return app.exec_app();
}
```

5.16 Файл robot_mover.cpp

Листинг 16 – robot_mover.cpp

```
#include "robot_mover.h"
#include "character_writer.h"
#include "file_writer.h"

robot_mover::robot_mover(base_class*    parent,    std::string    name)    :
base_class(parent, name, 2) {}

void robot_mover::move_robot()
{
   int line, column;
   std::cin >> line >> column;
   if (!(line >= 1 && line <= 10 && column >= 1 && column <= 10))
   {
      std::string error = "\nCoordinate is wrong ( " + std::to_string(line) +
", " + std::to_string(column) + " )";
      emit_signal(SIGNAL(file_writer::write_in_file), error);
   }
   else
   {
      robot_mover* mover = (robot_mover*)(get_child("robot_mover"));
      character_writer*       writer       =       (character_writer*)(mover-
```

```
>get_child("character_writer"));
      if (!(mover->line == line && mover->column == column))
      {
         writer->set_line_column(mover->line, mover->column);
         mover->line = line, mover->column = column;
      }
   }
}

int robot_mover::get_line()
{ return line; }

int robot_mover::get_column()
{ return column; }
```

5.17 Файл robot_mover.h

Листинг 17 – robot_mover.h

```
#ifndef __ROBOT_MOVER__H
#define __ROBOT_MOVER__H

#include "base_class.h"

class robot_mover : public base_class
{
private:
   int line = 1, column = 1;
public:
   robot_mover(base_class* parent, std::string name = "robot_mover");
   void move_robot();
   int get_line(), get_column();
};

#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 23.

Таблица 23 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
MOVE 3 3 MOVE 0 3 SEEK 5 WRITE I WRITE K MOVE 4 4 WRITE K MOVE 4 1 WRITE ! END	8888888888 8888888888 88I8888888 888K888888 8888888888 8888888888 8888888888 8888888888 8888888888 8888888888 Coordinate is wrong (0, 3) ERROR command: SEEK 5 Attempt to sequentially write to a cell (3, 3) The letter does not belong to the Latin alphabet: !	8888888888 8888888888 88I8888888 888K888888 8888888888 8888888888 8888888888 8888888888 8888888888 8888888888 Coordinate is wrong (0, 3) ERROR command: SEEK 5 Attempt to sequentially write to a cell (3, 3) The letter does not belong to the Latin alphabet: !
WRITE A MOVE 2 2 WRITE B MOVE 3 3 WRITE C MOVE 4 4 WRITE D MOVE 5 5 WRITE E MOVE 6 6 WRITE F MOVE 7 7 WRITE G MOVE 8 8 WRITE H MOVE 9 9 WRITE I MOVE 10 10 WRITE J END	A888888888 8B88888888 88C8888888 888D888888 8888E88888 88888F8888 888888G888 8888888H88 88888888I8 888888888J	A888888888 8B88888888 88C8888888 888D888888 8888E88888 88888F8888 888888G888 8888888H88 88888888I8 888888888J
MOVE 3 3 MOVE 0 3	8888888888 8888888888	8888888888 8888888888

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
WRITE I WRITE K MOVE 14 5 WRITE K MOVE 3 3 WRITE K SEEK 5 WRITE K MOVE 3 4 MOVE 3 3 WRITE K MOVE 7 5 WRITE ! WRITE I MOVE 5 5 MOVE 5 5 WRITE L GOTO 3 4 START DRAW Apple END	88K8888888 8888888888 8888L88888 8888888888 8888I88888 8888888888 8888888888 8888888888 Coordinate is wrong (0, 3) Attempt to sequentially write to a cell (3, 3) Coordinate is wrong (14, 5) Attempt to sequentially write to a cell (3, 3) Attempt to sequentially write to a cell (3, 3) ERROR command: SEEK 5 Attempt to sequentially write to a cell (3, 3) The letter does not belong to the Latin alphabet: ! ERROR command: GOTO 3 4 ERROR command: START ERROR command: DRAW Apple	88K8888888 8888888888 8888L88888 8888888888 8888I88888 8888888888 8888888888 8888888888 Coordinate is wrong (0, 3) Attempt to sequentially write to a cell (3, 3) Coordinate is wrong (14, 5) Attempt to sequentially write to a cell (3, 3) Attempt to sequentially write to a cell (3, 3) ERROR command: SEEK 5 Attempt to sequentially write to a cell (3, 3) The letter does not belong to the Latin alphabet: ! ERROR command: GOTO 3 4 ERROR command: START ERROR command: DRAW Apple
SHOWTREE	system is ready file_creator is ready command_reader is ready robot_mover is ready character_writer is ready file_writer is ready console_outputer is ready	system is ready file_creator is ready command_reader is ready robot_mover is ready character_writer is ready file_writer is ready console_outputer is ready

ЗАКЛЮЧЕНИЕ

В ходе данной работе были тщательно изучены основные принципы разработки систем с использованием объектно-ориентированного программирования. ООП предоставляет средства для создания модульных и гибких систем, основанных на взаимодействии объектов.

С опорой на изученные принципы ООП была успешно разработана система, представляющая собой модель расстановки символов на доске. Эта система предназначена для эмуляции перемещения робота по доске и написания им символов в его текущей позиции.

В процессе разработки был найден эффективный метод решения задачи моделирования расстановки символов на доске. Были разработаны алгоритмы, описывающие взаимодействие различных компонентов системы. Эти алгоритмы управляют перемещением робота, написанием символа, созданием файла и его изменением.

В разработанной системе использовались:

1. Объект "Система", который управляет работой программы;
2. Объект "Создатель файла", который создаёт файл с уникальным названием и вносит в него стартовые данные;
3. Объект "Читатель команд", который считывает команды из консоли и посылает сигналы в другие объекты;
4. Объект "Перемещатель робота", который имеет закрытые поля с координатами текущего положения робота, а старые заносит в поля объекта "Писатель символов";
5. Объект "Писатель символов", который имеет закрытые поля с координатами предыдущего положения робота и пишет символ из латинского алфавита по координатам текущего положения робота, после

чего изменяет значения полей с предыдущих на текущие;

6. Объект "Писатель файла", который заносит данные обо всех ошибках в файл;

7. Объект "Выводитель в консоль", который выводит все данные из файла в консоль.

Таким образом разработанная система обладает способностью моделировать расстановку символов на доске и обрабатывать все ошибки, введённые пользователем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).