

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт компьютерных наук и технологий

Отчет о прохождении ознакомительной практики

Прохоров Максим Андреевич
(Ф.И.О. обучающегося)

2 курс, гр. 23531/3
(номер курса обучения и учебной группы)

09.03.01 «Информатика и вычислительная техника»
(Направление подготовки (код и наименование))

Место прохождения практики: кафедра КСПТ ИКНТ
(указывается наименование профильной организации или наименование структурного подразделения
ФГАОУ ВО «СПбПУ»)

Сроки практики: 25 июня по 21 июля 2018 г.

Руководитель практики: Сидорина Татьяна Леонидовна, ведущий программист
(Ф.И.О., уч.степень, должность)

Оценка:

Руководитель практики / Сидорина Т.Л./

Обучающийся: / Прохоров М.А./

Дата:

УТВЕРЖДАЮ
Зав. кафедрой КСПТ ИКНТ

В.М. Ицыксон
подпись

«__» _____ 2018 г.

ИНДИВИДУАЛЬНЫЙ ПЛАН ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКИ (вид и тип практики)

Ф.И.О. обучающегося: Прохоров Максим Андреевич

Учебное подразделение: кафедра Компьютерных систем и программных технологий ИКНТ.

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Руководитель практики от ФГАОУ ВО «СПбПУ»: Сидорина Татьяна Леонидовна

Сроки проведения: 25 июня по 21 июля 2018 г.

Содержание практики: Симулятор односторонней машины Тьюринга, написанный на языке программирования C++ с возможностью отладки программы. Реализация взаимодействия через чтение входных текстовых файлов и консольного ввода.

Форма итоговой отчетности по практике: зачет

Перечень учебной литературы и методических материалов, в том числе ресурсы сети «Интернет»:

Шилдт Г. – «Самоучитель C++»

c++studio.com

c++preference.com

Материально-техническая база: Среда разработки JetBrains CLion

Обучающийся _____ /Прохоров М.А./

Руководитель практики _____ /Сидорина Т.Л./

Содержание

Введение.....	4
1. Техническое задание.....	5
2. Метод решения.....	8
3. Коды ошибок.....	9
4. Моделирование ошибок. Методика испытаний	11
Заключение	18
Список использованных источников	18
Приложения	19
Main.h.....	19
Main.cpp.....	19
Input.h.....	19
Input.cpp.....	20
Command.h	22
Parser.h	22
Parser.cpp	23
Process.h.....	28
Process.cpp	29
Output.h.....	31
Output.cpp	31
Utility.h.....	32
Utility.cpp.....	32

Введение

В качестве ознакомительной практики предлагалось самостоятельно освоить новый язык программирования. Было принято решение изучить один из самых популярных языков – «C ++».

Мы рассматривали языки «Си», «Java», «Kotlin», однако самый известный язык программирования не был исследован нами ранее. Изучить его самостоятельно будет не так сложно, так как база Си и объектно-ориентированного программирования из Java уже есть.

В качестве практики я решил доработать свой уже существующий проект, написанный ранее на «Си». Итак, темой работы стала машина А. Тьюринга. Работы Алана Тьюринга известны во всем мире, его проекты внесли огромный вклад в развитие ЭВМ и программирования.

1. Техническое задание

Симулятор односторонней машины Тьюринга, выбранный в качестве практики, будет реализован мною на языке программирования C++. Конечный продукт будет схож по функционированию с известными аналогами. Однако предполагается реализовать не графическое, а текстовое представление симулятора. Входные и выходные данные будут представлять собой текстовые файлы. Рассмотрим каждый аспект проекта подробнее.

Односторонность. Машина Тьюринга называется односторонней или машиной Тьюринга с односторонней лентой, если в процессе вычисления ее головка никогда не сдвигается левее начальной ячейки, т.е. располагается только в ячейках с положительными номерами.

В программе будет реализовано ограниченное перемещение каретки в соответствии с изложенным выше правилом. Все задачи будут также предполагать это условие. В случае попытки выхода за левую границу ленты будет выдана ошибка.

Взаимодействие с пользователем. Взаимодействие симулятора с пользователем осуществляется с помощью командной строки и входных файлов. Для каждого объекта – свой файл: для входной ленты, для алфавита, для таблицы состояний конечного автомата. Далее определены правила ввода данных: алфавит, синтаксис, разметка. Выходная лента будет подаваться в соответствующий текстовый файл. Все файлы задаются в командной строке.

Реализация отладки: остановки и пошагового запуска. Для запуска отладчика необходимо использовать ключ [-debug] во время запуска программы. Далее появится возможность пошагово проходить программу. С помощью последовательного нажатия (ввода в командную строку) комбинации клавиш [n + ENTER] программа сможет сделать один шаг. Программа выполнится до конца после ввода слова [RUN] и остановится, если ввести [STOP]. Все эти шаги будут выводиться в выходной файл.

Формат командной строки. Вызов программы из командной строки будет осуществляться следующим образом:

```
turing.exe -a alphabet.txt -i inputtape.txt -q machine.txt -o outputtape.txt [-debug]
```

Формат входных файлов.

В текстовом файле, отвечающем за **алфавит**, данные задаются следующим образом: в строку и без разделителей.

```
ABC123,
```

Файл, содержащий **входную ленту**, выглядит так:

```
..V.....  
__101_xxABC
```

Все символы вводятся без отступов и пробелов. Верхняя строка состоит их символов «.» и одной прописной латинской буквы «V», отвечающей за положение головки на ленте. Следующая строка отвечает за входные данные на ленте, состоящие из набора символов из алфавита. В качестве пустого символа («дырки») на ленте будет использоваться символ «нижнее подчеркивание» («_»).

Файл, содержащий **таблицу состояний конечного автомата**, будет выглядеть так:

<pre> . .q1. .q2. 0 1>q1 1>q1 1 2>q1 2>q1 2 0>q2 0.q0 _ .q0 _ .q0 </pre> <p>а) Пример таблицы в два состояния</p>	<pre> . .q1. .q2. - .q9. .q10. .q11. 0 1>q1 1>q1 - 2<q2 1 2>q1 2>q1 - 1>q11 2 0>q2 0.q0 - _ .q0 _ .q0 - _ .q0 </pre> <p>б) Пример таблицы для 11 состояний (Символы «-» – разрыв таблицы, допущенный для сокращения примера)</p>
----------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

В первой строке написаны состояния, отделенные друг от друга пробелами и точками, чтобы сохранить упорядоченность в столбцах. В первом слева столбце написаны все символы из алфавита, которые могут встретиться прочтении входной ленты, в том числе пустой символ («_»). Внутри такой таблицы описаны действия, которые происходят при прочтении очередного символа. Первый символ – символ, который следует поместить в эту ячейку. Второй – направление перемещения головки (влево «<», вправо «>», на месте «.»). Оставшиеся символы – номер состояния, в которое необходимо перейти. Для остановки зарезервировано состояние q0.

Формат файла с **выходной лентой**.

<pre> START: ..V..... __1010xxABC 12345678901 q1: 1 -> x q2 ...V..... __x010xxABC 12345678901 q2: 0 <- C q1 ..V..... __xC10xxABC 12345678901 q1: x . C q0 ..V..... __CC10xxABC 12345678901 END. </pre>

В файле содержится начальное состояние ленты и все промежуточные состояния и команды, выполненные в процессе работы. Формат команды (последовательность данных): начальное состояние, исходный символ в ячейке, направление сдвига головки, новый символ, новое состояние. Последняя строка каждого шага – нумерованная последовательность ячеек для удобства. Чтобы сохранить порядок в столбцах, нумерация идет лишь в разряде единиц, десятки и сотни не прописываются.

В связи с ограничением времени и памяти, максимальное число итераций (шагов) программы равно 1000. После этого программа будет принудительно остановлена, и появится соответствующее сообщение вместо слова «END», например, «FORCED STOP»

Предполагаемый метод решения. Программа будет состоять из 5 классов. Один из них – вспомогательный. Этот класс содержит несколько востребованных универсальных команд, не сильно относящихся к теме проекта. Например, перевод строки файла в пригодную для обработки строку или проверка символа на принадлежность к цифрам.

Начинается программа с класса «input». В него передаются аргументы командной строки для дальнейшей обработки. Если строка запуска удовлетворяет заданному формату, файлы и флаг, отвечающий за запуск «дебаггера», передаются в следующий класс.

Следующий этап – разбор и проверка входных данных: алфавита, ленты и команд управляющего автомата. На основе различных правил и соотношений, каждый из элементов получает «оценку пригодности». Если формат данных не удовлетворяет заданным, программа выдает соответствующую ошибку с кодом и пояснением проблемы.

Сохраненные удовлетворяющие данные поступают в управляющий класс – «process». Здесь последовательно выполняются шаги машины Тьюринга. Также здесь выполняется отладка путем взаимодействия с пользователем через командную строку. При достижении определенного результата программа выдает соответствующие данные (печатает их с помощью функций класса «output») и прекращает работу.

Ошибки. В процессе создания продукта будут предусмотрены возможные проблемы взаимодействия пользователя с программой. Все эти случаи будут генерировать определённые ошибки, состоящие из кода, пояснения, степени критичности и возможного решения. По окончании будет составлена таблица этих ошибок.

Тестирование. Для тестирования будут взяты примеры лабораторных работ. Чтобы проверить правильность выполнения будут созданы специальные тестирующие функции, подающие на вход начальные условия и сопоставляющие полученный результат с ожидаемым. Возможно, будут проведены отдельные тестирования конкретных частей программы для наибольшей уверенности в результате, особенно на этапе разработки.

2. Метод решения

Программа состоит из 6 классов. Один из них – вспомогательный – `Utility.cpp`. Этот файл содержит несколько востребованных универсальных команд, не сильно относящихся к теме проекта. Например, функция для открытия файла для чтения или проверка символа на принадлежность к цифрам.

Каждый из классов имеет свой заголовочный файл («header»).

Основным файлом является `main.cpp`, в котором строится вся структура и последовательность программы. В него передаются аргументы командной строки для дальнейшей обработки. Если строка запуска удовлетворяет заданному формату, файлы и флаг, отвечающий за запуск «дебаггера», передаются в следующий класс – `Parser.cpp`.

Следующий этап – разбор и проверка входных данных: алфавита, ленты и команд управляющего автомата. На основе различных правил и соотношений, каждый из элементов получает «оценку пригодности». Если формат данных не удовлетворяет заданным, программа выдает соответствующую ошибку с кодом и пояснением проблемы. Для хранения команд конечного автомата организован класс `Command.cpp`.

Сохраненные удовлетворяющие данные поступают в управляющую функцию класса `Process.cpp`. Здесь последовательно выполняются шаги машины Тьюринга. Также здесь выполняется отладка путем взаимодействия с пользователем через командную строку. При достижении определенного результата программа выдает соответствующие данные (печатает их с помощью функций `Output.cpp`) и прекращает работу.

Файл `Output.cpp` содержит необходимые команды вывода – печати в файл и командную строку.

В программе используются только стандартные подключаемые библиотеки и методы. Все действие основано на обработке строк и числовых массивов. Отсутствуют глобальные переменные, все передачи происходят за счет аргументов функций.

3. Коды ошибок

Код ошибки	Сообщение	Комментарий
0	-	Успешное завершение
1	-	Остановка программы пользователем
3	Memory allocation error	Произошла проблема выделения памяти. Требуется больше свободного места. Рекомендуется ограничить входные данные
5	Wrong state-machine input data format	Необходимо ввести данные файла управляющего автомата в соответствии с заданным в ТЗ форматом
6	Wrong state number	Номер состояния должен быть целым числом и следовать после символа 'q'
7	State-machine must contain all alphabet symbols in the first column	В файле управляющего автомата каждая строка должна начинаться с соответствующего символа из алфавита
8	New tape symbol must be from alphabet	В команде управляющего автомата новый символ должен содержаться в алфавите
9	Unknown move command	Программой распознаются только три вида обозначения перемещения головки: '>', '<', '.'
10	The state columns in the input file must have strong structure separated by spaces	Управляющий автомат задается в виде таблицы согласно формату. Колонки состояний разделяются пробелами, причем первые элементы состояний (точка) и команд должны быть друг под другом
11	Wrong state-machine command format	Необходимо исправить команду в соответствии с заданным форматом
12	State-machine input file cannot have additional line at the end	Необходимо удалить лишние строки после управляющего автомата
13	New state in commands must be present in the first line of state-machine table (it must exist)	Переход должен осуществляться только в существующее (объявленное ранее в первой строке) состояние
14	First symbols of lines in state-machine file must have the same sequence as the sequence of characters in the alphabet	Последовательность символов в первом столбце должна совпадать с последовательностью в алфавите
15	Symbols in alphabet input file cannot have duplicate	В алфавите не должно быть повторений символов
16	Tape must contain only alphabet characters	На ленте не должно быть символов не из алфавита

17	The problem of determining the position of the head	Не удается распознать позицию головки. Исправьте входной файл ленты в соответствии с форматом
18	Tape must contain exactly one head	Лента должна содержать ровно одну позицию головки
20	The command for such combination of state and symbol was not found	Отсутствует команда, которую пытаются выполнить
21	The head went beyond the left border	Головка не должны заходить за левую границу ленты (нулевое значение)
22	Wrong number of arguments in command line launch	Неправильное число аргументов при запуске. Повторите попытку согласно заданному формату
23	Cannot get file	Не удается получить доступ к файлу. Проверьте путь и название файла, а также его атрибуты
24	Wrong format of command line launch	Неверно задана команда запуска программы. Повторите попытку согласно заданному формату
25	RUNTIME ERROR	Было достигнуто максимальное число шагов машины (1000)

4. Моделирование ошибок. Методика испытаний

Если в командной строке просто ввести `turingmachine.exe` будет получена справка о программе:

```
Turing Machine simulator.
The simulator is based on a text representation.
It converts the input tape according to the commands of state-control machine.
A feature of this machine is its one-sidedness.
It means that head-pointer cannot go to the left of the initial position.

To RUN the program type this line:
TuringMachine.exe -a alphabet.txt -i inputtape.txt -q machine.txt -o output.txt [-debug]
Format of alphabet input file:
0123ABC
Format of input tape file:
....V....
___1010_
Format of state-machine input file:
. .q1. .q2.
0 1>q1 1>q1
1 2>q1 2>q1
2 0>q2 0.q0
_ .q0 _ .q0

Symbol '_' stands for empty cell.
Flags '-a', '-i', '-q', '-o' must be placed in front of the corresponding file.
Flag '-debug' runs debug configuration, where:
'N' -> next step
'RUN' -> running to the end
'STOP' -> stops the program

Process finished with exit code 0
```

Далее будет соотнесен формат вводимых данных и полученный результат.

Ошибка с кодом 5: символы ‘_’ вместо ‘.’

Входные данные	Результат						
<div>machine.txt</div> <table><tr><td>1</td><td>__ .q1. .q2.</td></tr><tr><td>2</td><td>0 0>q1 1>q0</td></tr><tr><td>3</td><td>1 1>q1 2>q0</td></tr></table>	1	__ .q1. .q2.	2	0 0>q1 1>q0	3	1 1>q1 2>q0	<pre>Wrong state-machine input data format State-machine input file. Error in line: 1 Process finished with exit code 5</pre>
1	__ .q1. .q2.						
2	0 0>q1 1>q0						
3	1 1>q1 2>q0						

Ошибка с кодом 6: символ ‘k’ вместо номера состояния

Входные данные	Результат						
<div>machine.txt</div> <table><tr><td>1</td><td>. .qk. .q2.</td></tr><tr><td>2</td><td>0 0>q1 1>q0</td></tr><tr><td>3</td><td>1 1>q1 2>q0</td></tr></table>	1	. .qk. .q2.	2	0 0>q1 1>q0	3	1 1>q1 2>q0	<pre>Wrong state number Error in line: 1 Process finished with exit code 6</pre>
1	. .qk. .q2.						
2	0 0>q1 1>q0						
3	1 1>q1 2>q0						

Ошибка с кодом 8: символ 'x' не входит в алфавит

Входные данные	Результат
<pre>machine.txt 1 . .q1. .q2. 2 0 x>q1 1>q0 3 1 1>q1 2>q0 alphabet.txt 1 0123456789</pre>	<pre>New tape symbol must be from alphabet State-machine input file. Error in line: 2 Process finished with exit code 8</pre>

Ошибка с кодом 9: символ 'L' не является командой перемещения

Входные данные	Результат
<pre>machine.txt 1 . .q1. .q2. 2 0 0Lq1 1>q0 3 1 1>q1 2>q0</pre>	<pre>Unknown move command State-machine input file. Error in line: 2 Process finished with exit code 9</pre>

Ошибка с кодом 10: вместо '/' ожидался пробел

Входные данные	Результат
<pre>machine.txt 1 . .q1. .q2. 2 0 /0>q1 1>q0 3 1, 1>q1 2>q0</pre>	<pre>The state columns in the input file must have strong structure separated by spaces State-machine input file. Error in line: 2 Process finished with exit code 10</pre>

Ошибка с кодом 11: лишний символ 'n'

Входные данные	Результат
<pre>machine.txt 1 . .q1. .q2. 2 0 0>q1 1>q0 3 1 1>qn1 2>q0</pre>	<pre>Wrong state-machine command format State-machine input file. Error in line: 3 Process finished with exit code 11</pre>

Ошибка с кодом 12: лишняя строка в конце файла

Входные данные	Результат
<pre>machine.txt 11 9 0>q2 0<q2 12 _<q2 1.q0 13</pre>	<pre>State-machine input file cannot have additional line at the end State-machine input file. Error in line: 12 Process finished with exit code 12</pre>

Ошибка с кодом 13: неправильный номер состояния

Входные данные	Результат
<p>machine.txt</p> <pre> 1 . .q1. .q2. 2 0 0>q1 1>q0 3 1 1>q5 2>q0 </pre>	<pre> New state in commands must be present in the first line of state-machine table State-machine input file. Error in line: 3 Process finished with exit code 13 </pre>

Ошибка с кодом 14: лишняя строка

Входные данные	Результат
<p>machine.txt</p> <pre> 1 . .q1. .q2. 2 0 0>q1 1>q0 3 4 1 1>q1 2>q0 5 2 2>q1 3.q0 </pre>	<pre> Empty line in state-machine file if forbidden First symbols of lines in state-machine file must have the same sequence as the sequence of characters in the alphabet State-machine input file. Error in line: 3 Process finished with exit code 14 </pre>

Ошибка с кодом 15: повторение символа '5'

Входные данные	Результат
<p>alphabet.txt</p> <pre> 1 012345556789 </pre>	<pre> Symbols in alphabet input file cannot have duplicate Process finished with exit code 15 </pre>

Ошибка с кодом 16: неверный символ на ленте

Входные данные	Результат
<p>alphabet.txt</p> <pre> 1 0123456789 </pre> <p>inputtape.txt</p> <pre> 1 ..V.... 2 __99_xx </pre>	<pre> Tape must contain only alphabet characters Input tape file. Error in line: 2 Process finished with exit code 16 </pre>

Ошибка с кодом 17: неверный символ вместо '.'

Входные данные	Результат
<p>inputtape.txt</p> <pre> 1 ..V.x... 2 __99__ </pre>	<pre> The problem of determining the position of the head Input tape file. Error in line: 1 Process finished with exit code 17 </pre>

Ошибка с кодом 18: две головки

Входные данные	Результат				
<p>inputtape.txt</p> <table border="1"> <tr> <td>1</td><td>..v.v.. </td></tr> <tr> <td>2</td><td>__99__</td></tr> </table>	1	..v.v..	2	__99__	<pre>Tape must contain exactly one head Input tape file. Error in line: 1 Process finished with exit code 18</pre>
1	..v.v..				
2	__99__				

Ошибка с кодом 20: пропущена команда

Входные данные	Результат												
<p>machine.txt</p> <table border="1"> <tr> <td>1</td><td>. .q1. .q2.</td></tr> <tr> <td>2</td><td>0 0>q1 1>q0</td></tr> <tr> <td>3</td><td>1 2>q0</td></tr> <tr> <td>4</td><td>2 2>q1 3.q0</td></tr> </table> <p>inputtape.txt</p> <table border="1"> <tr> <td>1</td><td>..v....</td></tr> <tr> <td>2</td><td>__19__ </td></tr> </table>	1	. .q1. .q2.	2	0 0>q1 1>q0	3	1 2>q0	4	2 2>q1 3.q0	1	..v....	2	__19__	<pre>START ..V.... 19___ 0123456 The command for such combination of state and symbol was not found Process finished with exit code 20</pre>
1	. .q1. .q2.												
2	0 0>q1 1>q0												
3	1 2>q0												
4	2 2>q1 3.q0												
1	..v....												
2	__19__												

Ошибка с кодом 21: головка ушла влево

Входные данные	Результат												
<p>machine.txt</p> <table border="1"> <tr> <td>1</td><td>. .q1. .q2.</td></tr> <tr> <td>2</td><td>0 0>q1 1>q0</td></tr> <tr> <td>3</td><td>1 1<q1 2>q0</td></tr> <tr> <td>4</td><td>2 2>q1 3.q0</td></tr> </table> <p>inputtape.txt</p> <table border="1"> <tr> <td>1</td><td>v....</td></tr> <tr> <td>2</td><td>19__ </td></tr> </table>	1	. .q1. .q2.	2	0 0>q1 1>q0	3	1 1<q1 2>q0	4	2 2>q1 3.q0	1	v....	2	19__	<pre>START V.... 19___ 01234 q1 1 < 1 q1 19___ 01234 The head went beyond the left border Process finished with exit code 21</pre>
1	. .q1. .q2.												
2	0 0>q1 1>q0												
3	1 1<q1 2>q0												
4	2 2>q1 3.q0												
1	v....												
2	19__												

Ошибка с кодом 22: неверная входная строка

Входные данные	Результат
<pre>turingmachine.exe -a alphabet.txt -i inputtape.txt -q machine.txt -o</pre>	<pre>Wrong number of arguments in command line launch Process finished with exit code 22</pre>

Ошибка с кодом 23: отсутствие данного файла

Входные данные
<pre>turingmachine.exe -a alphabet.txt -i input_tape.txt -q machine.txt -o output.txt</pre>
Результат
<pre>Cannot get file input_tape.txt Process finished with exit code 23</pre>

Теперь продемонстрируем работающую программу: прибавления единицы к десятичному числу (увеличение числа на 1).

alphabet.txt:

1	0123456789
---	------------

inputtape.txt:

1	..V....
2	__99__

machine.txt:

1	. .q1. .q2.
2	0 0>q1 1>q0
3	1 1>q1 2>q0
4	2 2>q1 3.q0
5	3 3>q1 4.q0
6	4 4>q1 5.q0
7	5 5>q1 6.q0
8	6 6>q1 7.q0
9	7 7>q1 8.q0
10	8 8>q1 9.q0
11	9 0>q2 0<q2
12	__<q2 1.q0

Результат работы в cmd.exe:

```
..V....  
  99  
-----  
0123456
```

q1 9 > 9 q2

```
...V...  
  99  
-----  
0123456
```

q2 9 < 0 q2

```
..V....  
  90  
-----  
0123456
```

q2 9 < 0 q2

```
.V.....  
  00  
-----  
0123456
```

q2 _ . 1 q0

```
.V.....  
 100  
-----  
0123456
```

END

Process finished with exit code 0

Результат работы в output.txt:

```
1  START
2
3  ..V....
4  __99__
5  0123456
6
7  q1: 9 > 9 q2
8  ...V...
9  __99__
10 0123456
11
12 q2: 9 < 0 q2
13 ..V....
14 __90__
15 0123456
16
17 q2: 9 < 0 q2
18 .V.....
19 __00__
20 0123456
21
22 q2: _ . 1 q0
23 .V.....
24 __100__
25 0123456
26
27 END
28 |
```

Заключение

В результате проделанной работы была разработана программа на языке программирования «C++», реализующая работу симулятора машины Тьюринга. Как и предполагалось, взаимодействие основано на чтении входных текстовых файлов и наборе в консольной строке. Симулятор удовлетворяет заданным требованиям и полностью работоспособен.

За счет написания программы, мною был изучен язык программирования «C++», закреплены знания основ языка «Си». Во время разработки использовались официальные рекомендации из интернет-справочников, а также электронное пособие Г. Шилдта.

Список использованных источников

1. Шилдт Г. Самоучитель C++: Пер. с англ. — 3-е изд. — СПб.: БХВ-Петербург, 2005. — 688 с. ISBN 5-7791-0086-1
2. Основы программирования на языках Си и C++ для начинающих [Электронный ресурс] <http://cppstudio.com/>
3. C++ reference [Электронный ресурс] <https://cppreference.com>

Приложения

Main.h

```
#ifndef TURINGMACHINE_MAIN_H
#define TURINGMACHINE_MAIN_H

#include <string>
#include <iostream>
#include <fstream>
#include "Utility.h"
#include "Input.h"
#include "Parser.h"
#include "Process.h"

#endif //TURINGMACHINE_MAIN_H
```

Main.cpp

```
#ifndef TURINGMACHINE_MAIN_H
#define TURINGMACHINE_MAIN_H

#include <string>
#include <iostream>
#include <fstream>
#include "Utility.h"
#include "Parser.h"
#include "Process.h"

void information();

#endif //TURINGMACHINE_MAIN_H
```

Input.h

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

#ifndef TURINGMACHINE_INPUT_H
#define TURINGMACHINE_INPUT_H

using namespace std;

class Input {
private:
    ifstream inputTapeFile;
    ifstream alphabetFile;
    ifstream machineFile;
    ofstream outputFile;
    int flag;

    void readFiles(int argc, char *argv[]);

    void fileOpening(ifstream &fileName, string arg, string nextArg, string flag);

    void information();

public:
    Input(int argc, char *argv[]) {
```

```

        readFiles(argc, argv);
    }

    ifstream &getInputTapeFile();

    ifstream &getAlphabetFile();

    ifstream &getMachineFile();

    ofstream &getOutputFile();

    int &getFlag();
};

#endif //TURINGMACHINE_INPUT_H

```

Input.cpp

```

#include "Input.h"

void Input::readFiles(int argc, char *argv[]) {
    if (argc == 1) {
        information();
        cout << "\nProcess finished with exit code 0\n";
        exit(0);
    }

    if ((argc != 9) && (argc != 10)) {
        cout << "Wrong number of arguments in command line launch\n";
        cout << "\nProcess finished with exit code 22\n";
        exit(22);
    }

    for (int i = 1; i < 8; i++) {
        fileOpening(alphabetFile, argv[i], argv[i + 1], "-a");
        fileOpening(inputTapeFile, argv[i], argv[i + 1], "-i");
        fileOpening(machineFile, argv[i], argv[i + 1], "-q");
        if (argv[i] == "-o") {
            outputFile.open(argv[i + 1]);
            if (!outputFile) {
                cout << "Cannot get file " << argv[i + 1] << "\n";
                cout << "\nProcess finished with exit code 23\n";
                exit(23);
            }
        }
    }

    if ((!alphabetFile) || (!inputTapeFile) || (!machineFile) || (!outputFile))
    {
        cout << "Wrong format of command line launch\n";
        cout << "\nProcess finished with exit code 24\n";
        exit(24);
    }

    if ((argc == 10) && (strcmp(argv[9], "-debug") == 0)) {
        flag = 1;
    } else {
        flag = 0;
    }
}

void Input::fileOpening(ifstream &fileName, string arg, string nextArg, string

```

```

flag) {
    if (arg == flag) {
        fileName.open(nextArg);
        if (!fileName) {
            cout << "Cannot get file " << nextArg << endl;
            cout << "\nProcess finished with exit code 23\n";
            exit(23);
        }
    }
}

void Input::information() {
    cout << ("Turing Machine simulator.\n"
        "The simulator is based on a text representation.\n"
        "It converts the input tape according to the commands of state-
control machine.\n"
        "A feature of this machine is its one-sidedness.\n"
        "It means that head-pointer cannot go to the left of the initial
position.\n\n"
        "To RUN the program type this line:\n"
        "TuringMachine__.exe -a alphabet.txt -i inputtape.txt -q machine.txt
-o output.txt [-debug]\n"
        "Format of alphabet input file:\n"
        "0123ABC\n"
        "Format of input tape file:\n"
        "....V....\n"
        "___1010_\n"
        "Format of state-machine input file:\n"
        ". .q1. .q2.\n"
        "0 1>q1 1>q1\n"
        "1 2>q1 2>q1\n"
        "2 0>q2 0.q0\n"
        "__.q0 __.q0\n"
        "\n"
        "Symbol '_' stands for empty cell.\n"
        "Flags '-a', '-i', '-q', '-o' must be placed in front of the
corresponding file.\n"
        "Flag '-debug' runs debug configuration, where:\n"
        "'N' -> next step\n"
        "'RUN' -> running to the end\n"
        "'STOP' -> stops the program\n");
}

ifstream &Input::getInputTapeFile() {
    return inputTapeFile;
}

ifstream &Input::getAlphabetFile() {
    return alphabetFile;
}

ifstream &Input::getMachineFile() {
    return machineFile;
}

ofstream &Input::getOutputFile() {
    return outputFile;
}

int &Input::getFlag() {
    return flag;
}

```

Command.h

```
#ifndef TURINGMACHINE_COMMAND_H
#define TURINGMACHINE_COMMAND_H

class Command {
public:
    char newSymbol;
    char move;
    int newState;

    Command() = default;

    Command(char symbol, char move, int state) {
        this->newSymbol = symbol;
        this->newState = state;
        this->move = move;
    }
};

#endif //TURINGMACHINE_COMMAND_H
```

Parser.h

```
#ifndef TURINGMACHINE_PARSER_H
#define TURINGMACHINE_PARSER_H

#include <string>
#include <fstream>
#include <iostream>
#include <vector>
#include "Command.h"
#include "Utility.h"

using namespace std;

class Parser {
public:
    void parseAlphabet(ifstream &alphabetFile);
    void parseTape(ifstream &inputTapeFile);
    void parseMachine(ifstream &machineFile);

    string &getAlphabet();
    string &getTape();
    int &getHeadPos();
    int &getNumberOfStates();
    vector<int> &getStateNumbers();
    vector<vector<Command>> &getCommands();

private:
```

```

    string alphabet;
    string tape;
    int headPos;
    int numberOfStates;
    vector<int> stateNumbers;
    vector<vector<Command>> commands;

    Command parseCommand(string str, int index, int line);

    int parseState(string str, int index);

    void setCommands(vector<vector<Command>> &commands_temp, int n, int m);
};

#endif //TURINGMACHINE_PARSER_H

```

Parser.cpp

```

#include "Parser.h"

using namespace std;

// Alphabet
void Parser::parseAlphabet(ifstream &alphabetFile) {
    string alphabetString;
    getline(alphabetFile, alphabetString); // Stores alphabet characters
    int alphabetLength = alphabetString.length();
    for (int i = 0; i < alphabetLength; i++) {
        if (alphabetString.find_first_of(alphabetString[i]) !=
            alphabetString.find_last_of(alphabetString[i])) {
            cout << "Symbols in alphabet input file cannot have duplicate\n";
            cout << "\nProcess finished with exit code 15\n";
            exit(15);
        }
    }
    if (alphabetString.find('_') == std::string::npos) {
        alphabetString.append("_");
    }
    alphabet = alphabetString;
}

// Tape
void Parser::parseTape(ifstream &inputTapeFile) {
    // Head
    string headString;
    getline(inputTapeFile, headString);
    headPos = -1; // Stores position of the head, counts from 0
    for (int i = 0; i < headString.length(); i++) {
        if ((headString[i] != '.') && (headString[i] != 'V')) {
            cout << "The problem of determining the position of the head\n";
            cout << "Input tape file. Error in line: 1\n";
            cout << "\nProcess finished with exit code 17\n";
            exit(17);
        } else if ((headString[i] == 'V') && (headPos > 0)) {
            cout << "Tape must contain exactly one head\n";
            cout << "Input tape file. Error in line: 1\n";
            cout << "\nProcess finished with exit code 18\n", 18;

```

```

        exit(18);
    } else if (headString[i] == 'V') {
        headPos = i;
    }
}

if (headPos < 0) {
    cout << "Tape must contain exactly one head\n";
    cout << "Input tape file. Error in line: 1\n";
    cout << "\nProcess finished with exit code 18\n";
    exit(18);
}

// Initial tape
getline(inputTapeFile, tape);
for (char ch : tape) {
    if (alphabet.find(ch) == std::string::npos) {
        cout << "Tape must contain only alphabet characters\n";
        cout << "Input tape file. Error in line: 2\n";
        cout << "\nProcess finished with exit code 16\n";
        exit(16);
    }
}

}

// State-machine
void Parser::parseMachine(ifstream &machineFile) {
    numberOfStates = 0;
    int alphabetLength = alphabet.length();
    stateNumbers.reserve(10);
    vector<int> stateIndexes; // Start line index of each state column
    stateIndexes.reserve(10);

    // Parsing first line of state-machine input data file
    string firstMachineLine;
    getline(machineFile, firstMachineLine);
    if ((firstMachineLine[0] != '.') && (firstMachineLine[1] != ' ')) {
        cout << "Wrong state-machine input data format\n";
        cout << "State-machine input file. Error in line: 1\n";
        cout << "\nProcess finished with exit code 5\n";
        exit(5);
    }

    for (int i = 2; i < firstMachineLine.length() - 1; i++) {
        if (((isNumber(firstMachineLine[i]) == -1) && (firstMachineLine[i] !=
'q') && (firstMachineLine[i] != '.') &&
            (firstMachineLine[i] != ' ')) ||
            ((firstMachineLine[i] == '.') && (firstMachineLine[i + 1] != ' ') &&
(firstMachineLine[i + 1] != 'q') &&
            (firstMachineLine[i + 1] != '\0')) ||
            ((firstMachineLine[i] == 'q') && (isNumber(firstMachineLine[i + 1])
== -1)) ||
            ((isNumber(firstMachineLine[i]) != -1) &&
(isNumber(firstMachineLine[i + 1]) == -1) &&
            (firstMachineLine[i + 1] != '.')) ||
            ((firstMachineLine[i] == ' ') && (firstMachineLine[i + 1] != ' ') &&
(firstMachineLine[i + 1] != '.') &&
            (firstMachineLine[i + 1] != '\0')))) {
            cout << "Wrong state-machine input data format\n";
            cout << "State-machine input file. Error in line: 1\n";
            cout << "\nProcess finished with exit code 5\n";
            exit(5);
        } else if ((firstMachineLine[i] == '.') && (firstMachineLine[i + 1] ==
'q')) {
            stateNumbers[numberOfStates] = parseState(firstMachineLine, i + 2);

```



```

        stateIndexes[numberOfStates] = i;
        numberOfStates++;
    }
}

vector<vector<Command>> commands_temp(static_cast<unsigned
int>(alphabetLength),
                                     vector<Command>(static_cast<unsigned
int>(numberOfStates)));

// Parsing state-machine commands
for (int i = 0; i < alphabetLength; i++) {
    string line;
    getline(machineFile, line);
    if (line[0] == '\\0') {
        cout << "State-machine must contain all alphabet symbols in the
first column\\n";
        cout << "State-machine input file. Error in line: " << i + 2 <<
"\\n";
        cout << "\\nProcess finished with exit code 7\\n";
        exit(7);
    }
    if ((i == alphabetLength - 1) && (line[line.length() - 1]) == '\\n') {
        cout << "State-machine input file cannot have additional line at the
end\\n";
        cout << "State-machine input file. Error in line: " << i + 2 <<
"\\n";
        cout << "\\nProcess finished with exit code 12\\n";
        exit(12);
    }
    if (line[0] != alphabet[i]) {
        if (line.length() == 0) {
            cout << "Empty line in state-machine file if forbidden\\n";
        }
        cout << "First symbols of lines in state-machine file must have the
same sequence "
               "as the sequence of characters in the alphabet\\n";
        cout << "State-machine input file. Error in line: " << i + 2 <<
"\\n";
        cout << "\\nProcess finished with exit code 14\\n", 14;
        exit(14);
    }

    int prev = 1;
    for (int j = 0; j < numberOfStates; j++) {
        int n = stateIndexes[j];
        if ((line[n] == NULL) || (line[n] == '\\0')) {
            Command command('?', '?', -1);
            commands_temp[i][j] = command;
            continue;
        }
        for (int k = prev + 1; k < n; k++) {
            if (line[k] != ' ') {
                cout << "The state columns in the input file must have
strong structure separated by spaces\\n";
                cout << "State-machine input file. Error in line: " << i + 2
<< "\\n";
                cout << "\\nProcess finished with exit code 10\\n";
                exit(10);
            }
        }
        commands_temp[i][j] = parseCommand(line, n, i + 2);
        if ((commands_temp[i][j].newSymbol == '?') &&
(commands_temp[i][j].move == '?') &&

```

```

        (commands_temp[i][j].newState == -1)) {
            continue;
        }
        if (alphabet.find(commands_temp[i][j].newSymbol) ==
std::string::npos) {
            cout << "New tape symbol must be from alphabet\n";
            cout << "State-machine input file. Error in line: " << i + 2 <<
"\n";

            cout << "\nProcess finished with exit code 8\n";
            exit(8);
        }
        if ((commands_temp[i][j].move != '.') && (commands_temp[i][j].move
!= '>') &&
            (commands_temp[i][j].move != '<')) {
            cout << "Unknown move command\n";
            cout << "State-machine input file. Error in line: " << i + 2 <<
"\n";

            cout << "\nProcess finished with exit code 9\n";
            exit(9);
        }
        int exist = 0;
        for (int k = 0; k < numberOfStates; k++) {
            if (stateNumbers[k] == commands_temp[i][j].newState) {
                exist = 1;
                break;
            }
        }
        if ((commands_temp[i][j].newState != 0) && (exist == 0)) {
            cout
                << "New state in commands must be present in the first
line of state-machine table (it must exist)\n";
            cout << "State-machine input file. Error in line: " << i + 2 <<
"\n";

            cout << "\nProcess finished with exit code 13\n";
            exit(13);
        }
        string stateNumStr;
        stateNumStr = to_string(commands_temp[i][j].newState);
        prev = n + 2 + stateNumStr.length();
    }
}
setCommands(commands_temp, alphabetLength, numberOfStates);
stateIndexes.clear();
}

// Determination of state number
int Parser::parseState(string str, int index) {
    int stateNumber = 0;
    int i = index;
    int rank = 0;
    while (isNumber(str[i]) != -1) {
        rank++;
        i++;
    }
    i = index;
    while (rank > 0) {
        stateNumber += (str[i] - '0') * pow(10, rank - 1);
        rank--;
        i++;
    }
    if (stateNumber > 0) {
        return stateNumber;
    } else {

```

```

        cout << "Wrong state number\n";
        cout << "Error in line: 1\n";
        cout << "\nProcess finished with exit code 6\n";
        exit(6);
    }
}

// Making command from string
Command Parser::parseCommand(string str, int index, int line) {
    Command command{};
    if ((str[index] == ' ') && (str[index + 1] == ' ') && (str[index + 2] == ' ') && (str[index + 3] == ' ')) {
        command.newSymbol = '?';
        command.move = '?';
        command.newState = -1;
        return command;
    }
    command.newSymbol = str[index];
    command.move = str[index + 1];
    if (str[index + 2] != 'q') {
        cout << "Wrong state-machine command format\n";
        cout << "State-machine input file. Error in line: " << line << "\n";
        cout << "\nProcess finished with exit code 11\n";
        exit(11);
    }
    int i = index + 3;
    int rank = 0;
    int num = 0;
    if (isNumber(str[i]) == -1) {
        cout << "Wrong state-machine command format\n";
        cout << "State-machine input file. Error in line: " << line << "\n";
        cout << "\nProcess finished with exit code 11\n";
        exit(11);
    }
    while ((str[i] != ' ') && (str[i] != '\0')) {
        if (isNumber(str[i]) != -1) {
            rank++;
            i++;
        } else {
            cout << "Wrong state number\n";
            cout << "State-machine input file. Error in line: " << line << "\n";
            cout << "\nProcess finished with exit code 6\n";
            exit(6);
        }
    }
    i = index + 3;
    while (rank > 0) {
        num += (str[i] - '0') * pow(10, rank - 1);
        rank--;
        i++;
    }
    command.newState = num;
    return command;
}

string &Parser::getAlphabet() {
    return alphabet;
}

string &Parser::getTape() {
    return tape;
}

int &Parser::getHeadPos() {

```

```

        return headPos;
    }

    int &Parser::getNumberOfStates() {
        return numberOfStates;
    }

    vector<int> &Parser::getStateNumbers() {
        return stateNumbers;
    }

    vector<vector<Command>> &Parser::getCommands() {
        return commands;
    }

    void Parser::setCommands(vector<vector<Command>> &commands_temp, int n, int m) {
        commands.reserve(static_cast<unsigned int>(n));
        for (int i = 0; i < n; ++i) {
            commands[i].reserve(static_cast<unsigned int>(m));
            for (int j = 0; j < m; ++j) {
                commands[i][j] = commands_temp[i][j];
            }
        }
    }
}

```

Process.h

```

#ifndef TURINGMACHINE_PROCESS_H
#define TURINGMACHINE_PROCESS_H

#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include "Command.h"
#include "Output.h"

using namespace std;

class Process {
private:
    string alphabet;
    int headPos;
    string tape;
    vector<int> states;
    int statesNumber;
    vector<vector<Command>> commands;
    int flag;

public:
    Process(string &alphabet, int headPos, string &tape, vector<int> &states_,
int statesNumber,
        vector<vector<Command>> &commands_, int flag) {
        this->alphabet = alphabet;
        this->headPos = headPos;
        this->tape = tape;
        this->statesNumber = statesNumber;
        this->flag = flag;

        states.reserve(static_cast<unsigned int>(statesNumber));
    }
};

```

```

        for (int i = 0; i < statesNumber; ++i) {
            states[i] = states_[i];
        }

        commands.reserve(alphabet.length());
        for (int i = 0; i < alphabet.length(); ++i) {
            commands[i].reserve(static_cast<unsigned int>(statesNumber));
            for (int j = 0; j < statesNumber; ++j) {
                commands[i][j] = commands_[i][j];
            }
        }
    }

    int run(ofstream &outputFile);
};

#endif //TURINGMACHINE_PROCESS_H

```

Process.cpp

```

#include "Process.h"

int Process::run(ofstream &outputFile) {
    printStart(outputFile);
    printTape(headPos, tape, outputFile);
    int step = 0;
    int currentState = 1;
    while (step != 1000) {
        step++;
        char ch = tape[headPos];
        int i = alphabet.find_first_of(ch) -
alphabet.find_first_of(alphabet[0]);
        int j = 0;
        for (int k = 0; k < statesNumber; k++) {
            if (currentState == states[k]) {
                j = k;
                break;
            }
        }
        if ((commands[i][j].newSymbol == '?') && (commands[i][j].move == '?') &&
(commands[i][j].newState) == -1) {
            cout << "The command for such combination of state and symbol was
not found\n";
            cout << "\nProcess finished with exit code 20\n";
            outputFile.close();
            exit(20);
        }
        printCommand(currentState, tape[headPos], commands[i][j], outputFile);
        currentState = commands[i][j].newState;
        tape[headPos] = commands[i][j].newSymbol;
        if (commands[i][j].move == '>') {
            headPos++;
        }
        if (commands[i][j].move == '<') {
            headPos--;
        }
        printTape(headPos, tape, outputFile);
        if (currentState == 0) {
            printEnd(outputFile);
            outputFile.close();
            states.clear();
        }
    }
}

```

```

        for (int l = 0; l < alphabet.length(); l++) {
            commands[l].clear();
        }
        commands.clear();
        cout << "\nProcess finished with exit code 0\n";
        exit(0);
    }
    if (headPos < 0) {
        cout << "The head went beyond the left border\n";
        cout << "\nProcess finished with exit code 21\n";
        exit(21);
    }

    if (flag == 1) {
        string action;
        while (true) {
            cin >> action;
            cin.get();
            fflush(stdin);
            if ((action == "n") || (action == "N")) {
                break;
            }
            if ((action == "run") || (action == "RUN")) {
                flag = 0;
                break;
            }
            if ((action == "stop") || (action == "STOP")) {
                printStop(outputFile);
                outputFile.close();
                states.clear();
                for (int l = 0; l < alphabet.length(); l++) {
                    commands[l].clear();
                }
                commands.clear();
                cout << "\nProcess finished with exit code 1\n";
                exit(1);
            } else {
                cout << "Unknown action. Try this: 'N'; 'RUN'; 'STOP';\n";
            }
        }
    }

    }

    printStop(outputFile);
    cout << "RUNTIME ERROR\n";
    outputFile.close();
    states.clear();
    for (int l = 0; l < alphabet.length(); l++) {
        commands[l].clear();
    }
    commands.clear();
    cout << "\nProcess finished with exit code 25\n";
    exit(25);
}

```

Output.h

```
#ifndef TURINGMACHINE_OUTPUT_H
#define TURINGMACHINE_OUTPUT_H

#include <iostream>
#include <fstream>
#include <string>
#include "Command.h"

using namespace std;

void printTape(int headPos, string &tape, ofstream &output);

void printCommand(int currentState, char currentSymbol, Command command,
ofstream &output);

void printStart(ofstream &output);

void printEnd(ofstream &output);

void printStop(ofstream &output);

#endif //TURINGMACHINE_OUTPUT_H
```

Output.cpp

```
#include "Output.h"

void printTape(int headPos, string &tape, ofstream &output) {
    int length = tape.length();
    // Head line
    for (int i = 0; i < length; i++) {
        if (i == headPos) {
            cout << "▼";
            output << "▼";
        } else {
            cout << ".";
            output << ".";
        }
    }
    cout << "\n";
    output << "\n";

    // Tape
    cout << tape << "\n";
    output << tape << "\n";

    // Numbers
    int i = 0;
    int cond = 0;
    while (cond != length) {
        string str = to_string(i);
        cout << str;
        output << str;
        if (i < 9) {
            i++;
        } else {
            i = 0;
        }
        cond++;
    }
}
```

```

        cout << "\n\n";
        output << "\n\n";
    }

    void printCommand(int currentState, char currentSymbol, Command command,
ofstream &output) {
        cout << "q" << currentState << ": " << currentSymbol << " " << command.move
<< " " << command.newSymbol
            << " q" << command.newState << "\n";
        output << "q" << currentState << ": " << currentSymbol << " " <<
command.move << " " << command.newSymbol
            << " q" << command.newState << "\n";
    }

    void printStart(ofstream &output) {
        cout << "START\n\n";
        output << "START\n\n";
    }

    void printEnd(ofstream &output) {
        cout << "END\n";
        output << "END\n";
    }

    void printStop(ofstream &output) {
        cout << "FORCED STOP\n";
        output << "FORCED STOP\n";
    }
}

```

Utility.h

```

#ifndef TURINGMACHINE_UTILITY_H
#define TURINGMACHINE_UTILITY_H

#include <string>

using namespace std;

int isNumber(char ch);

int pow(int x, int y);

#endif //TURINGMACHINE_UTILITY_H

```

Utility.cpp

```

#include "Utility.h"

using namespace std;

int isNumber(char ch) {
    string num = "0123456789";
    if (num.find(ch) != std::string::npos) {
        return ch - '0';
    } else {
        return -1;
    }
}

```



```
int pow(int x, int y) {  
    int res = 1;  
    for (int i = 0; i < y; i++) {  
        res = res * x;  
    }  
    return res;  
}
```