Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Теория автоматов и формальных языков

ЛАБОРАТОРНАЯ РАБОТА № 1-2

Тема: **Дополнительные функции лексического анализатора**

Выполнил:

студент гр. 1ПИб-02-2оп-23

Кринкин Олег Алексеевич

Проверил:

доцент, к.т.н. Ганичева Оксана Георгиевна

ЗАДАНИЕ

1. Написать программный код для своего варианта задания на 20-25 строк (допустимо до 30 строк)

2. Проверить его работоспособность. В качестве доказательства сделать скриншот программы и результатов.  **Результат работы программы должен выводиться в оформленном виде.**

3. Написать **функцию л**ексического анализатора, выполняющую следующие действия: (**работу этой функции проверять на примере написанного в п.1 рабочего кода)**

1) удаление лишних пробелов во входном коде;

2) удаление комментариев из текста программы;

3) подсчет количества строк и нумерация строк в коде после выполнения п1 и 2.

4) Составить блок-схему для этой функции.

4. Протестировать работу функции на других примерах.

ХОД РАБОТЫ

1. Для тестирования работы функций лексического анализатора был написан следующий программный код:

|  |
| --- |
| #include <iostream>  // This file is bigger than 20-30 lines. I can’t get it smaller, but  // functional...  /\*  \* This class represents some sorted array  \* This example uses bubble sorting  \*/  class SortedArray {  private:  int\* array;  int arr\_size;  public:  SortedArray(int\* arr, int size) {  // Yes, array sorted in constructor  array = arr;  this->arr\_size = size;  for (int j = 1; j < size; j++) {  bool isSorted = true;  for (int i = 0; i < size - j; i++) {  if (array[i] > array[i + 1]) {  int tmp = array[i];  array[i] = array[i + 1];  array[i + 1] = tmp;  isSorted = false;  }  }  if ( isSorted ) break;  }  }  int size() const { return arr\_size; }  int\* get\_array() const { return array; }  };  void arr\_print(int\* arr, int size) {  for (int i = 0; i < size; i++) { // Some comment  std::cout << arr[i] << " ";  }  std::cout << std::endl;  }  int main() { // Even more comments  int\* arr = new int[10] { 5, 7, 3, 0, 1, 8, 4, 2, 6, 9 };  std::cout << "Unsorted array: "; arr\_print(arr, 10);  // AHhahahahahahhaahhahaha, it is comment, btw  SortedArray\* s\_arr = new SortedArray(arr, 10);  std::cout << "Sorted array: "; arr\_print( s\_arr->get\_array(), s\_arr->size() );  return 0;  } |

В программный код намерено внесено большое количество переносов и пробелов в абсолютно случайных местах, для того чтобы в полной мере оценить работу анализатора. Программный код состоит из слоя абстракции для массивов – класса, предоставляющего тип данных «отсортированный массив» и функции main() для работы с ним. В классе реализована минимально необходимая структура для выполнения какой-либо работы.

1. Работа программного кода продемонстрирована на рисунке ниже (рис. 1):

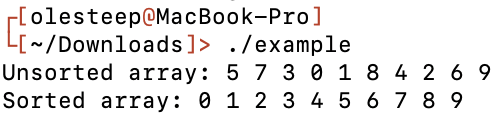


Рис. 1. Работа программы

Программа преобразовала некоторый массив чисел в отсортированный массив.

1. Для анализа кода написана функция лексического анализатора, которая является компонентной функцией для класса Analyzer и принимает на вход символ, по которому определяет состояние автомата анализатора. Внутри себя функция проверяет текущее состояние автомата, а затем в зависимости от него проверяет пришедший символ. Если после обратки символа состояние автомата оказывается конечным – функция возвращает этот же символ, в противном случае функция возвращает код -1. Текст функции представлен ниже:

|  |
| --- |
| char filter(char c) {  if (c == ' ') return -1;  switch (state) {  case 0: {  switch (c) {  case '/': { state = 1; break; }  case ' ': { state = 4; break; }  case '\n': { state = 4; break; }  default: { state = 0; break; }  } break;  }  case 1: {  switch (c) {  case '/': { state = 1; break; }  case '\*': { state = 2; break; }  case '\n': { state = 5; break; }  } break;  }  case 2: {  switch (c) {  case '\*': { state = 3; break; }  default: { state = 2; break; }  } break;  }  case 3: {  switch (c) {  case '/': { state = 5; break; }  default: { state = 2; break; }  } break;  }  case 4: {  switch (c) {  case '/': { state = 1; break; }  case '\n': { state = 5; break; }  case ' ': { state = 5; break; }  default: { state = 0; break; }  } break;  }  case 5: {  switch (c) {  case '/': { state = 1; break; }  case ' ': { state = 5; break; }  case '\n': { state = 5; break; }  default: { state = 0; break; }  } break;  }  }  if (state == 0 || state == 4)  return c;  return -1;  } |

Граф автомата, по которому составлена функция, а также её блок-схема представлены на рисунках ниже (рис. 2-3):

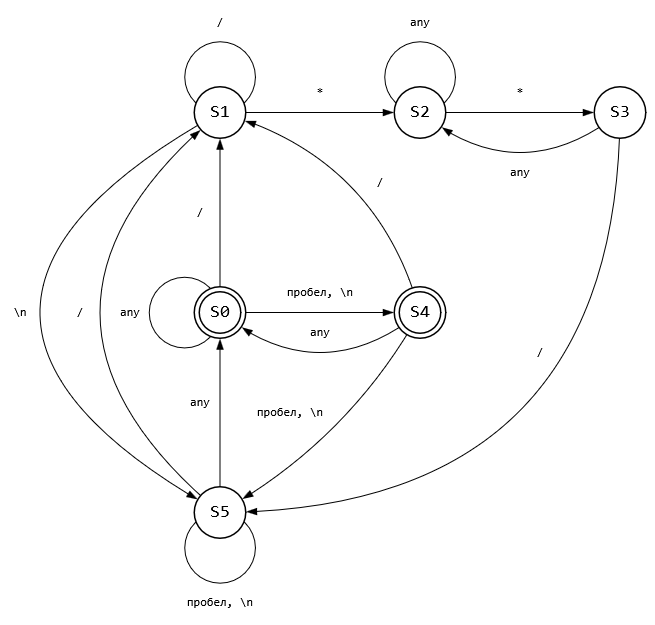


Рис. 2. Граф автомата

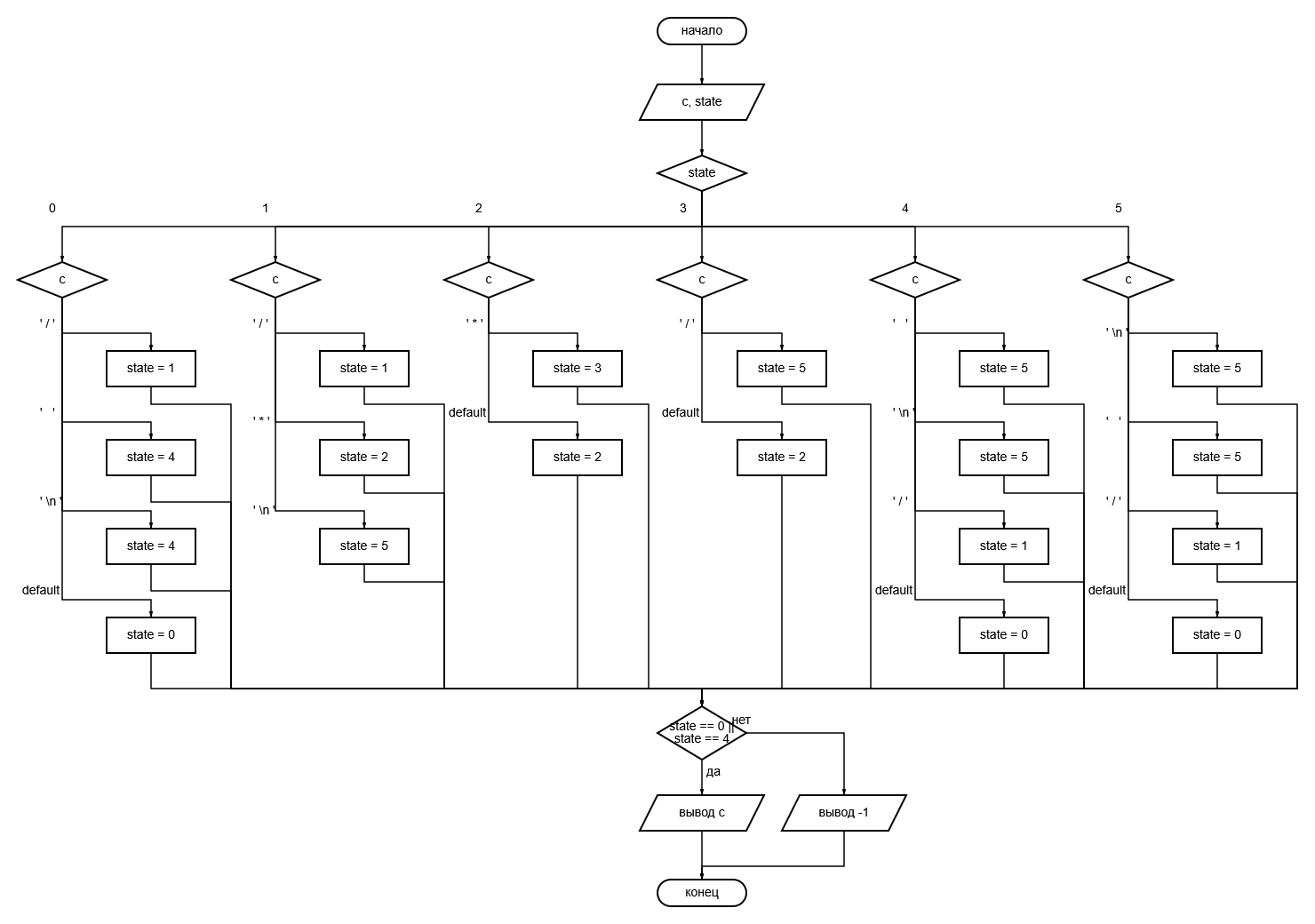


Рис. 3. Блок-схема функции

1. Взаимодействие с функцией лексического анализатора оформлено в виде графического интерфейса, в котором пользователь выбирает файл с исходным кодом, который затем анализируется, преобразуется и выводится на экран. Для примера кода с работой с классом, представленным выше, программа выдала следующий результат (рис. 4):

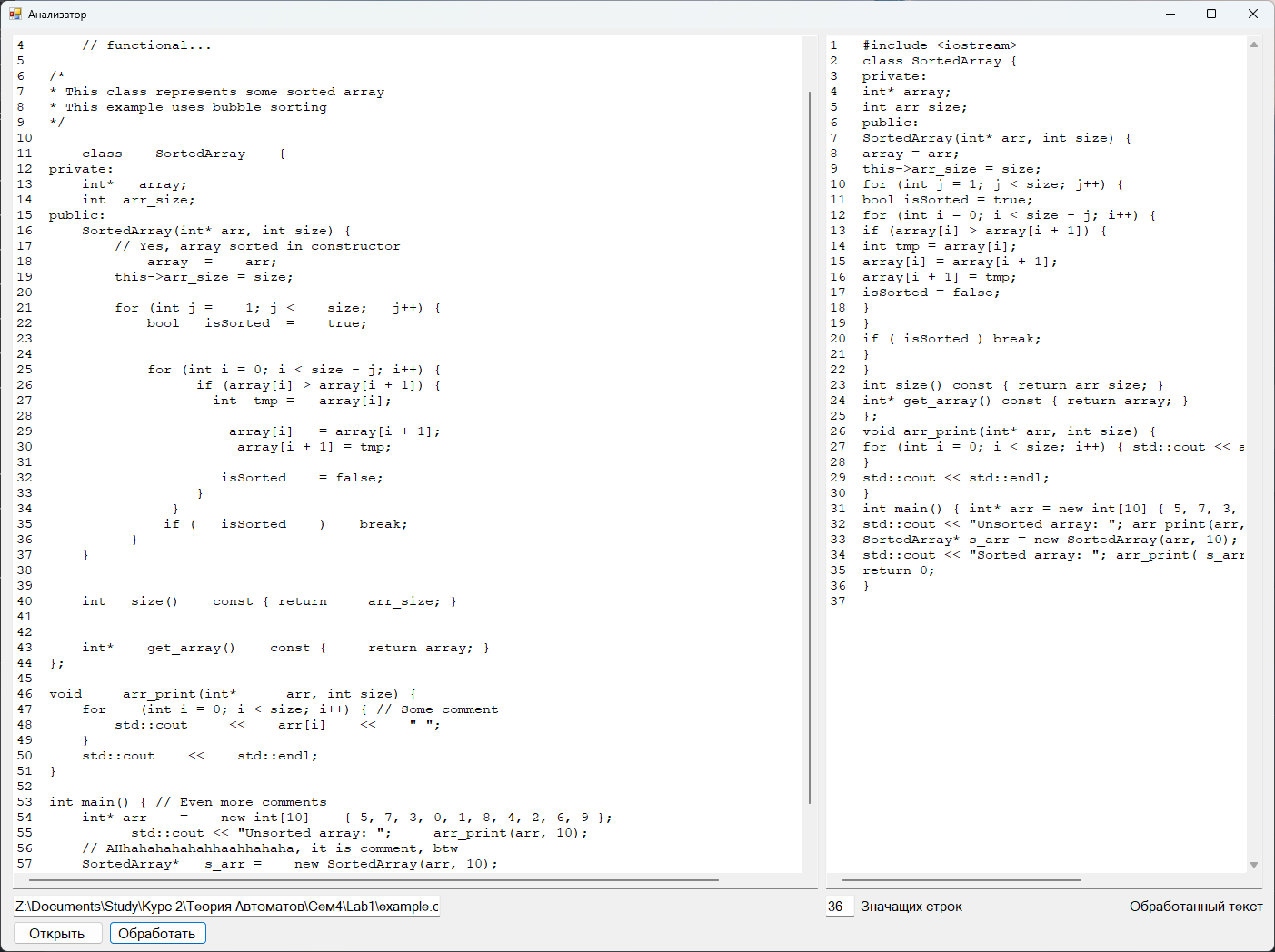


Рис. 4. Результат работы анализатора

|  |
| --- |
| #include <iostream>  class SortedArray {  private:  int\* array;  int arr\_size;  public:  SortedArray(int\* arr, int size) {  array = arr;  this->arr\_size = size;  for (int j = 1; j < size; j++) {  bool isSorted = true;  for (int i = 0; i < size - j; i++) {  if (array[i] > array[i + 1]) {  int tmp = array[i];  array[i] = array[i + 1];  array[i + 1] = tmp;  isSorted = false;  }  }  if ( isSorted ) break;  }  }  int size() const { return arr\_size; }  int\* get\_array() const { return array; }  };  void arr\_print(int\* arr, int size) {  for (int i = 0; i < size; i++) { std::cout << arr[i] << " ";  }  std::cout << std::endl;  }  int main() { int\* arr = new int[10] { 5, 7, 3, 0, 1, 8, 4, 2, 6, 9 };  std::cout << "Unsorted array: "; arr\_print(arr, 10);  SortedArray\* s\_arr = new SortedArray(arr, 10);  std::cout << "Sorted array: "; arr\_print( s\_arr->get\_array(), s\_arr->size() )  return 0;  } |

Результатом работы функции анализатора стал текст программы, избавленный от лишних пробелов, от переносов строк и комментариев.

РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ РАБОТЫ

В результате выполнения лабораторной работы получено представление одной из частей лексического анализатора. Для её реализации был составлен конечный автомат, который обрабатывает символы текстов программы. Функция анализатора реализована в виде графической программы (рис. 5), способной открывать и обрабатывать текстовые файлы.

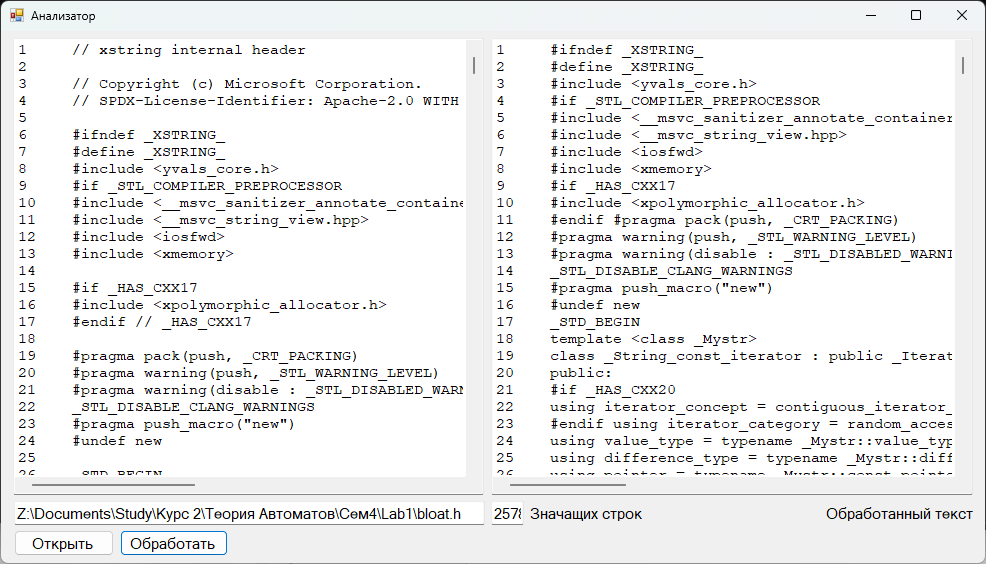


Рис. 5. Окно программы

Часть программного кода спроектированной программы:

private:

// Функция возвращающая некоторое количество пробелов

String^ spaces(int count) {

String^ spaces = gcnew String("");

for (int i = 0; i < count; i++)

spaces += " ";

return spaces;

}

// Функция-счётчик строк

int line = 0;

int getLine() {

line++;

return line;

}

// Функция добавляющая в ноер строки в исходный текст

String^ addLineNum(String^ target, int count) {

line = 0;

count = (count+1).ToString()->Length;

target = getLine() + spaces(count-1) + " " + target;

String^ out = gcnew String("");

for (int i = 0; i < target->Length; i++) {

if (target[i] == '\n') {

int line = getLine();

out += "\n" + line + spaces(count-(line.ToString()->Length)) + " ";

continue;

}

out += target[i];

}

target = out;

return target;

}

// Функция преобразующая System::String в std::string

std::string to\_string(String^ string) {

msclr::interop::marshal\_context context;

return context.marshal\_as<std::string>(string);

}

// Обработка нажатия на кнопку "Отрыть файл"

System::Void openFileButton\_Click(System::Object^ sender, System::EventArgs^ e) {

this->openFileDialog1->FileName = "";

this->openFileDialog1->ShowDialog();

this->fileNameBox->Text = openFileDialog1->FileName;

std::string& source = analyser->readSource(to\_string(openFileDialog1->FileName));

this->sourceBox->Text = gcnew String(source.c\_str());

this->sourceBox->Text = addLineNum(this->sourceBox->Text, analyser->getLineCount(source));

this->sourceBox->Text = this->sourceBox->Text->Replace("\n", "\r\n");

this->sourceBox->BackColor = System::Drawing::SystemColors::Window;

}

// Обработка нажатия на кнопку "Обработать"

System::Void processButton\_Click(System::Object^ sender, System::EventArgs^ e) {

this->outBox->BackColor = System::Drawing::SystemColors::Window;

this->outBox->Text = gcnew String(analyser->stripSource().c\_str());

this->outBox->Text = addLineNum(this->outBox->Text, analyser->getLineCount(analyser->stripped));

this->outBox->Text = this->outBox->Text->Replace("\n", "\r\n");

this->linesNum->Text = analyser->getLineCount(analyser->stripped).ToString();

}

};

class Analyser {

public:

std::string source;

std::string stripped;

int state;

Analyser() {

state = 0;

}

// Чтение файла

std::string& readSource(std::string fileName) {

std::ifstream file(fileName);

std::string line;

this->source = "";

while (std::getline(file, line)) {

this->source += line + '\n';

}

file.close();

return this->source;

}

// Функция, обрабатывающая текст и оправляющая символы анализатору

std::string& stripSource() {

this->stripped = "";

for (int i = 0; i < source.size(); i++) {

char out = filter(source[i]);

if (out != -1) this->stripped += out;

}

return this->stripped;

}

// Функция, возвращающая количество строк в тексте

int getLineCount(std::string text) {

int res = 0;

for (int i = 0; i < text.length() ;i++)

if (text[i] == '\n')

res++;

return res;

}

private:

// Функция анализатора

char filter(char c) {

if (c == ' ') return -1;

switch (state) {

case 0: {

switch (c) {

case '/': { state = 1; break; }

case ' ': { state = 4; break; }

case '\n': { state = 4; break; }

default: { state = 0; break; }

} break;

}

case 1: {

switch (c) {

case '/': { state = 1; break; }

case '\*': { state = 2; break; }

case '\n': { state = 5; break; }

} break;

}

case 2: {

switch (c) {

case '\*': { state = 3; break; }

default: { state = 2; break; }

} break;

}

case 3: {

switch (c) {

case '/': { state = 5; break; }

default: { state = 2; break; }

} break;

}

case 4: {

switch (c) {

case '/': { state = 1; break; }

case '\n': { state = 5; break; }

case ' ': { state = 5; break; }

default: { state = 0; break; }

} break;

}

case 5: {

switch (c) {

case '/': { state = 1; break; }

case ' ': { state = 5; break; }

case '\n': { state = 5; break; }

default: { state = 0; break; }

} break;

}

}

// Печать обрабатываемых символов в консоль (отладка)

#ifdef \_DEBUG

if (c == '\n') std::cout << "\\n" << " " << state;

else if (c == '\r') std::cout << "\\r" << " " << state;

else std::cout << c << " " << state;

if (state == 0 || state == 4) {

std::cout << " - Valid";

};

std::cout << std::endl;

#endif

if (state == 0 || state == 4)

return c;

return -1;

}

};

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы было выполнено построение конечного автомата, на основе которого была спроектирована функция лексического анализатора, которая исключает из программного текста незначащие строки и символы.

Функция лексического анализатора оформлена в виде графического приложения, и её работа протестирована на некотором тексте программы. Входной текст программы представлял собой описание класса сортированного массива, а также работу с ним.

Текст был размером 60 строк и имел в себе большое количество незначащих символов и блоков, а после обработки уменьшился до 36 значащих строк и избавился от комментариев, множественных переносов и пробелов.

Конечный автомат, на котором была построена функция лексического анализатора состоял из 6-ти состояний (S0, S1, S2, S3, S4, S5), среди которых 2 конечных. Автомат обрабатывал текст как последовательность символом, исключая из неё незначащие символы, путём перехода в отличные от конечного состояния.