Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Теория автоматов и формальных языков

ЛАБОРАТОРНАЯ РАБОТА № 3-4

Тема: **Распознавание заданных слов языка программирования**

Выполнил:

студент гр. 1ПИб-02-2оп-23

Кринкин Олег Алексеевич

Проверил:

доцент, к.т.н. Ганичева Оксана Георгиевна

ЗАДАНИЕ

1. Построить синтаксические диаграммы (конечный автомат) для распознавания заданных слов языка программирования для варианта:

Константы с плавающей точкой и ключевые слова;

2. Написать программу для распознавания   слов языка С++ по своему варианту задания (ключевые слова, идентификаторы, знаки операций, знаки пунктуации, целые константы, вещественные константы, числа с фиксированной и плавающей точкой...).  **Правильные слова должны заноситься в таблицу. На словах с ошибкой - выдаваться сообщение об ошибке.**

**Распознавание лексем выполняется по конечному автомату!!!**

3. Составить блок-схему алгоритма решения задачи.

4. Оформить отчет (задание, блок-схема алгоритма, код программы с комментариями, результаты тестирования программы (не менее 10 тестовых заданий), скриншоты результатов)

ХОД РАБОТЫ

1. В рамках лабораторной работы были выбраны следующие ключевые слова и варианты чисел с плавающей точкой:

* auto, bool, break, case, char, class, delete, do, double, else, false, for, if, int, private, public, return, this, true, void
* 15.75, 1.575E1, 1575e-2, -2.5e-3, 25E-4, +3.4, +6.5E+2

1. Для распознавания ключевых слов языка программирования был построен следующий конечный автомат (рис. 1):

Изображение выглядит как текст, снимок экрана, документ, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. . Граф автомата ключевых слов

1. Для определения констант построен следующий конечный автомат (рис. 2):

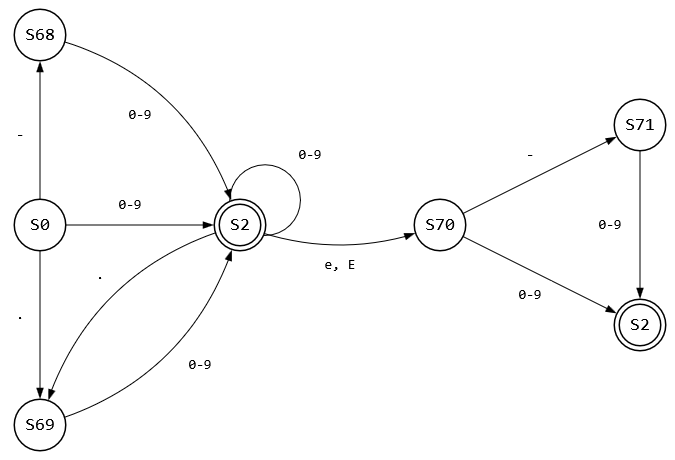


Рис. . Граф автомата констант

1. В автоматах зарезервированы 6 первых состояний для определения класса лексемы:

* 1 – ключевое слово;
* 2 – константа;
* 3 – идентификатор;
* 4 – знак отношения;
* 5 – знак операции;
* 6 – разделитель.

1. Если автомат перешёл в одно из этих состояний, то лексема возвращается и записывается в таблицу для своего класса (для этой лабораторной работы реализовано лишь две таблицы) (рис. 3).

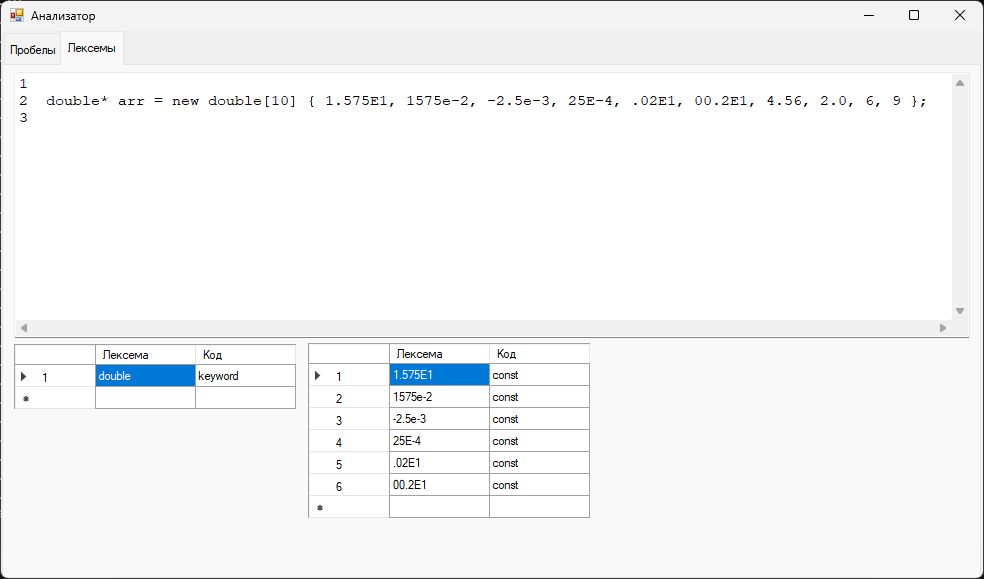


Рис. . Разделение по таблицам

1. Определение границ лексем происходит ожиданием следующим символом разделителя: «если следующий символ – разделитель, то лексема закончилась». Блок-схема такого алгоритма представлена на рис. 4.

Изображение выглядит как текст, диаграмма, Шрифт, чек

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. . Определение конца лексемы

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, отделяет в них лексемы, классифицирует их и записывает в таблицу лексем. Функция анализатора реализована в виде графической программы (рис. 5), способной открывать и обрабатывать текстовые файлы.

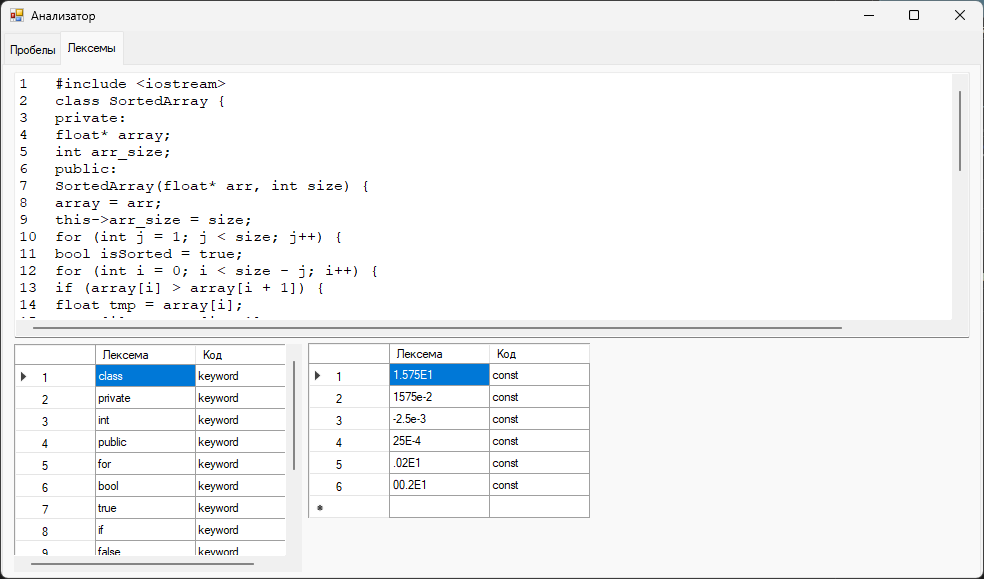


Рис. 5. Окно программы

Тестировалась программа на следующих входных текстах:

|  |
| --- |
| // Тест 1: Сумма и среднее трёх чисел (арифметические операторы)  #include <iostream>  using namespace std;  int main() {      /\* Объявление и инициализация переменных \*/      int a = 5, b = 10, c = 15;      // Вычисляем сумму      int sum = a + b + c;      // Вычисляем среднее (делим на 3.0, чтобы получить float)      float average = sum / 3.0;      cout << "Sum: " << sum << ", Average: " << average << endl;      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 6.

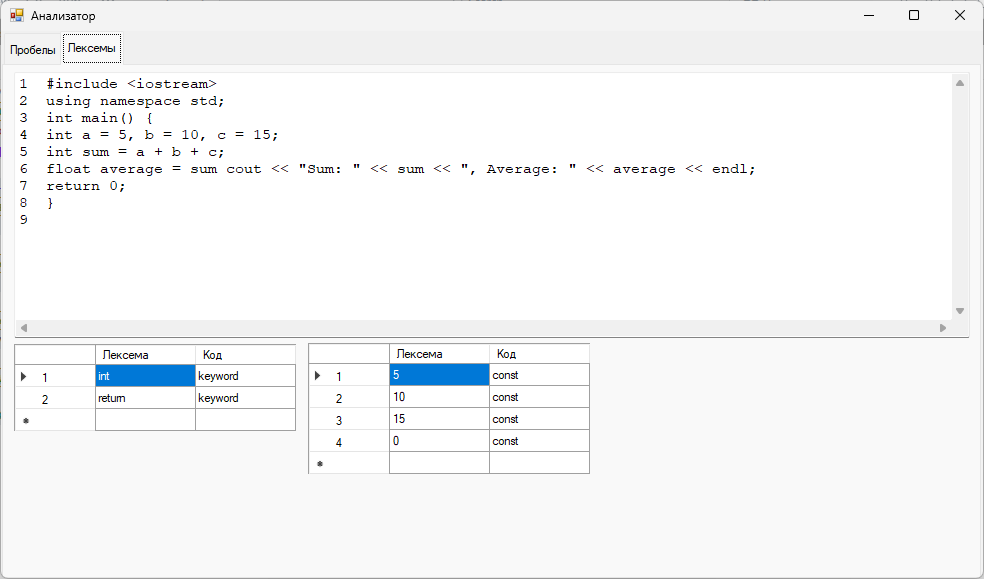


Рис. . Результат работы для теста 1.

|  |
| --- |
| // Тест 2: Определение чётного или нечётного числа (оператор %, условный if)  #include <iostream>  using namespace std;  int main() {      int num;      cout << "Enter a number: ";      cin >> num;  // Ввод числа пользователем      // Проверка на чётность      if (num % 2 == 0)          cout << "Even" << endl;      else          cout << "Odd" << endl;      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 7.



Рис. . Результат работы для теста 2.

|  |
| --- |
| // Тест 3: Поиск наибольшего из трёх чисел (оператор if-else)  #include <iostream>  using namespace std;  int main() {      float x = 10.5, y = 25.6, z = 15.3;      float max;      /\* Определяем максимальное значение с помощью условных операторов \*/      if (x > y && x > z)          max = x;      else if (y > z)          max = y;      else          max = z;      cout << "Max: " << max << endl;      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 8.

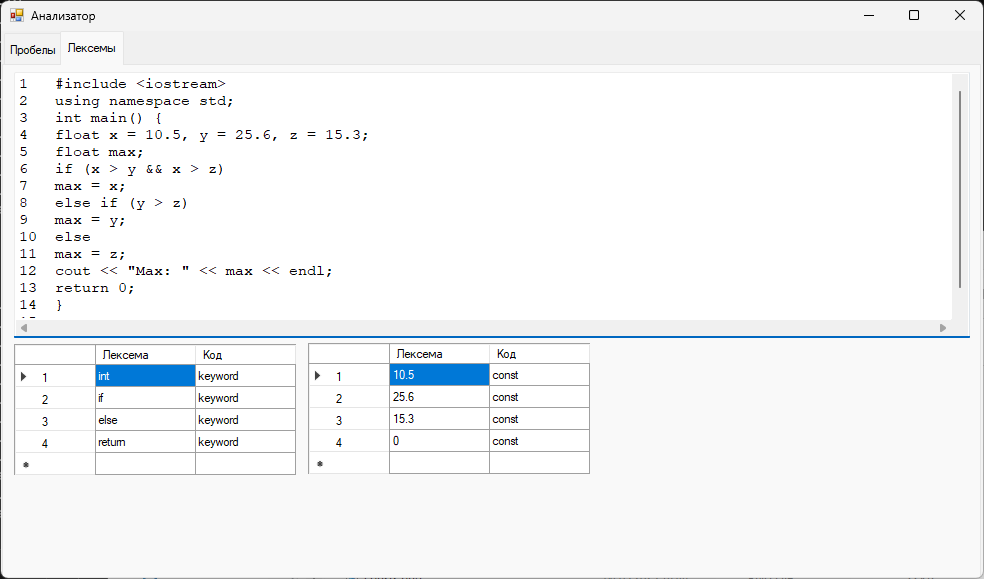


Рис. . Результат работы для теста 3.

|  |
| --- |
| // Тест 4: Простое меню (switch-case)  #include <iostream>  using namespace std;  int main() {      int choice;      cout << "1. Say Hello\n2. Say Bye\nEnter choice: ";      cin >> choice;  // Ввод выбора пользователем      /\* Обработка выбора с помощью оператора switch \*/      switch(choice) {          case 1:              cout << "Hello!" << endl;              break; // Обязательно прерываем выполнение          case 2:              cout << "Bye!" << endl;              break;          default:              // Обработка неверного ввода              cout << "Invalid choice." << endl;      }      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 9.

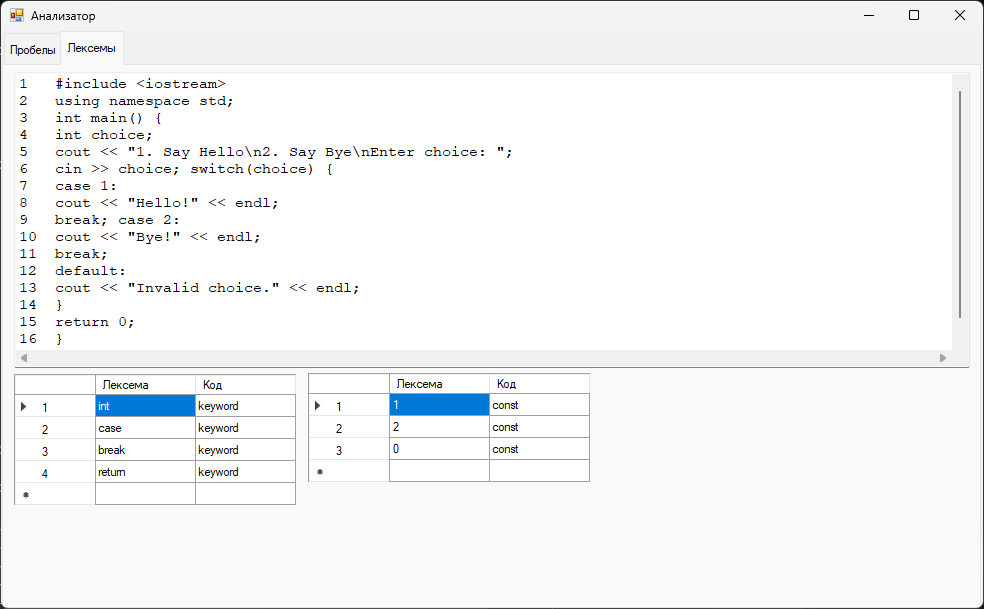


Рис. . Результат работы для теста 4.

|  |
| --- |
| // Тест 5: Таблица умножения (цикл for)  #include <iostream>  using namespace std;  int main() {      int n = 7;  // Таблица умножения для 7      // Цикл от 1 до 10      for (int i = 1; i <= 10; ++i)          cout << n << " \* " << i << " = " << n \* i << endl;      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 10.

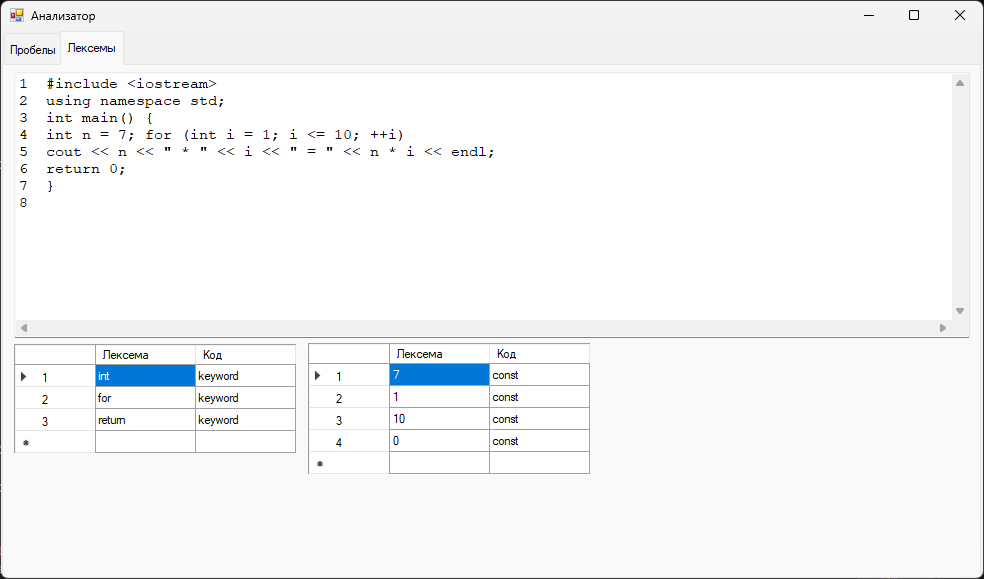


Рис. . Результат работы для теста 5.

|  |
| --- |
| // Тест 6: Факториал числа (цикл while)  #include <iostream>  using namespace std;  int main() {      int n = 5;      int result = 1;      // Вычисляем факториал от 5      while (n > 0) {          result \*= n;          --n;  // Уменьшаем значение n      }      cout << "Factorial: " << result << endl;      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 11.

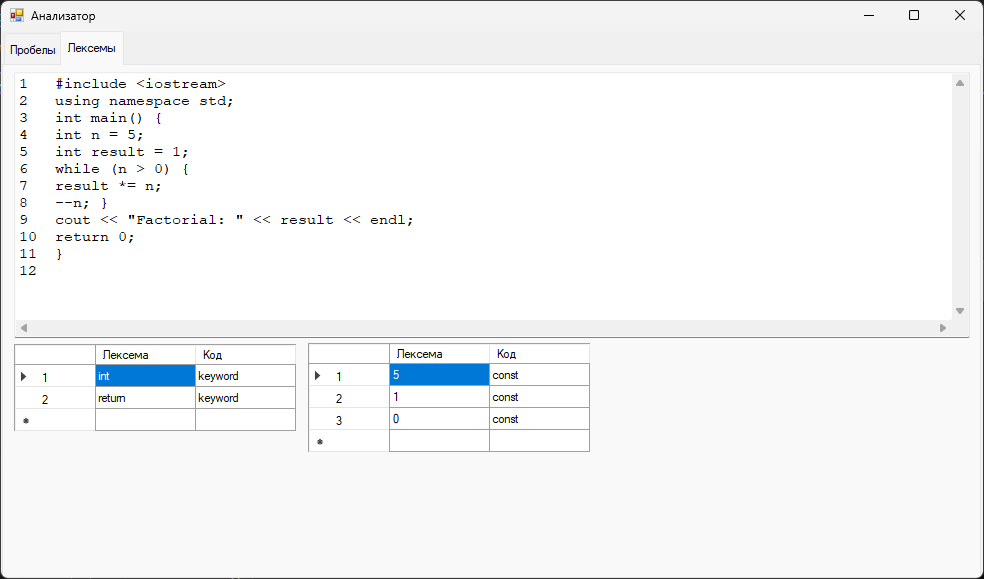


Рис. . Результат работы для теста 6.

|  |
| --- |
| // Тест 7: Обратный отсчёт с пропуском (цикл for, continue)  #include <iostream>  using namespace std;  int main() {      // Обратный отсчёт от 10 до 0      for (int i = 10; i >= 0; --i) {          if (i == 5)              continue;  // Пропустить число 5          cout << i << " ";      }      cout << endl;      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 12.

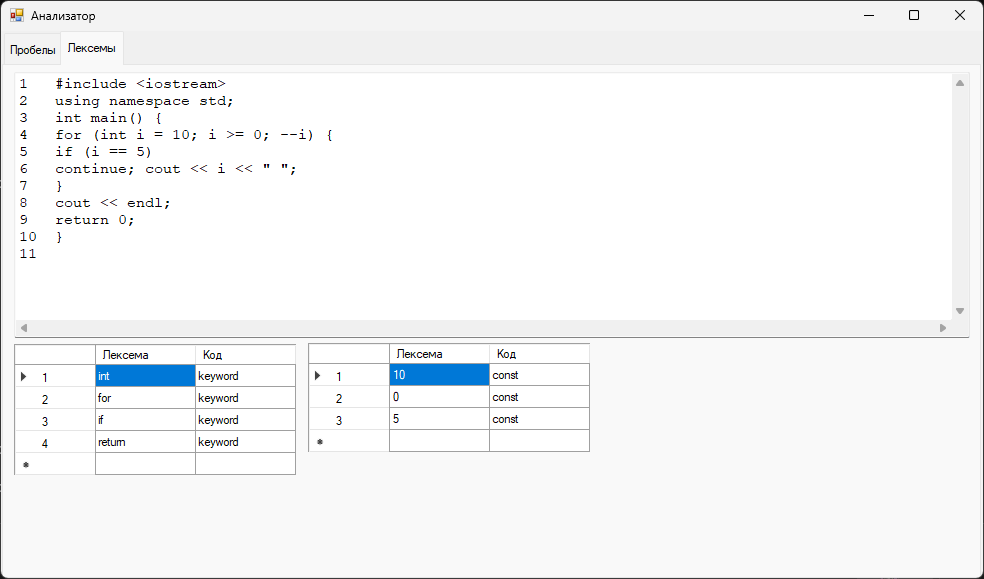


Рис. . Результат работы для теста 7.

|  |
| --- |
| // Тест 8: Проверка числа на простоту (операторы деления и логики)  #include <iostream>  using namespace std;  int main() {      int num = 29;      bool isPrime = true;      /\* Проверка делимости от 2 до sqrt(num) \*/      for (int i = 2; i \* i <= num; ++i) {          if (num % i == 0) {              isPrime = false;  // Нашли делитель — не простое              break;          }      }      // Выводим результат с тернарным оператором      cout << num << (isPrime ? " is prime" : " is not prime") << endl;      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 13.

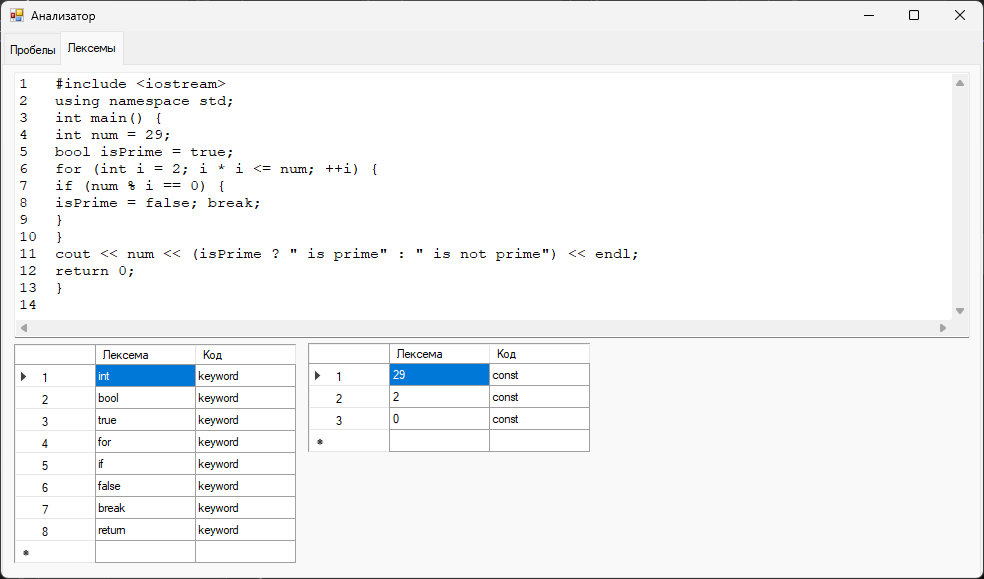


Рис. . Результат работы для теста 8.

|  |
| --- |
| // Тест 9: Определение високосного года (логические операторы)  #include <iostream>  using namespace std;  int main() {      int year = 2024;      /\* Високосный год:         - делится на 4 и не делится на 100,         - либо делится на 400 \*/      bool leap = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);      cout << year << (leap ? " is a leap year." : " is not a leap year.") << endl;      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 14.

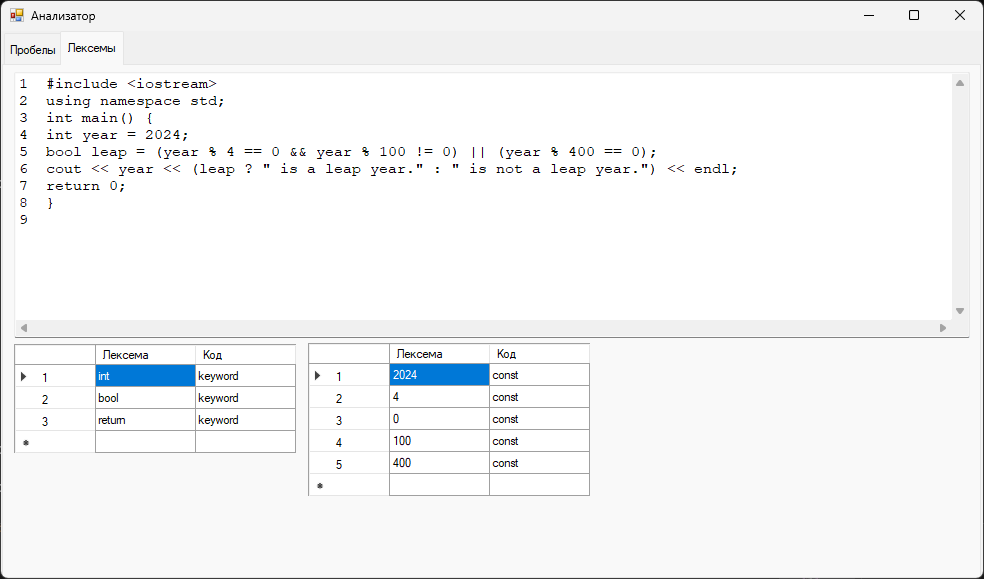


Рис. . Результат работы для теста 9.

|  |
| --- |
| // Тест 10: Замена местами двух переменных без третьей (арифметика)  #include <iostream>  using namespace std;  int main() {      int a = 10, b = 20;      // Меняем местами значения без третьей переменной      a = a + b; // a = 30      b = a - b; // b = 10      a = a - b; // a = 20      cout << "a: " << a << ", b: " << b << endl;      return 0;  } |

Результат работы этого теста продемонстрирован на рис. 15.

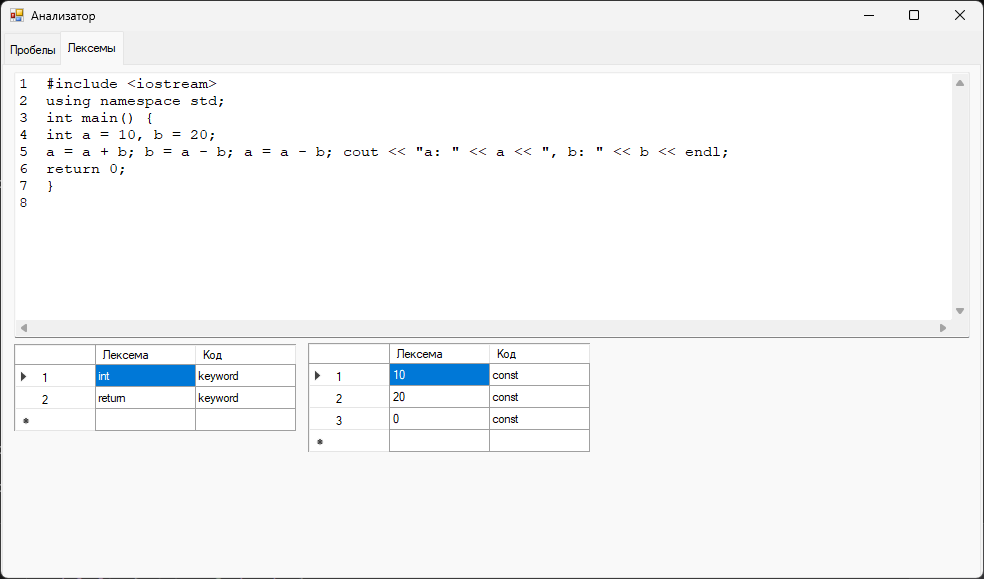


Рис. . Результат работы для теста 10.

Часть программного кода спроектированной программы:

public:

// 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);

}

// Функция возвращающая некоторое количество пробелов

String^ spaces(int count) {

String^ spaces = gcnew String("");

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

spaces += " ";

return spaces;

}

bool findInTable(String^ value, DataGridView^ gridView) {

for (int i = 0; i < gridView->RowCount; i++) {

if (gridView->Rows[i]->Cells[0]->Value != nullptr)

if (gridView->Rows[i]->Cells[0]->Value->ToString() == value)

return true;

}

return false;

}

// Функция разделяющая лексемы по таблицам

System::Void addToTable(int state, String^ lexem) {

switch (state) {

case 1: {

if (!findInTable(lexem, dataGridViewKeys)) {

int k\_count = this->dataGridViewKeys->RowCount++;

dataGridViewKeys->Rows[k\_count - 1]->HeaderCell->Value = k\_count.ToString();

dataGridViewKeys->Rows[k\_count - 1]->Cells[0]->Value = lexem;

dataGridViewKeys->Rows[k\_count - 1]->Cells[1]->Value = "keyword";

}

break;

}

case 2: {

if (!findInTable(lexem, dataGridViewConst)) {

int c\_count = this->dataGridViewConst->RowCount++;

dataGridViewConst->Rows[c\_count - 1]->HeaderCell->Value = c\_count.ToString();

dataGridViewConst->Rows[c\_count - 1]->Cells[0]->Value = lexem;

dataGridViewConst->Rows[c\_count - 1]->Cells[1]->Value = "const";

}

break;

}

}

}

private:

// 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;

}

// 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) {

// Удаление пробелов и переносов

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

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

const char\* stripped = analyser->stripSource().c\_str();

this->outBox->Text = gcnew String(stripped);

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

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

this->outBoxL->Text = this->outBox->Text;

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

// Лексический анализ

const std::string text = analyser->getStripped();

analyser->clearState();

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

std::string lexem = analyser->lexem\_filter(text[i]);

if (!lexem.empty()) {

this->addToTable(analyser->getState(), gcnew String(lexem.c\_str()));

analyser->clearState();

}

}

}

System::Void GUI\_Load(System::Object^ sender, System::EventArgs^ e) {

this->dataGridViewKeys->ColumnCount = 2;

this->dataGridViewConst->ColumnCount = 2;

this->dataGridViewKeys->Columns[0]->HeaderCell->Value = "Лексема";

this->dataGridViewKeys->Columns[1]->HeaderCell->Value = "Код";

this->dataGridViewConst->Columns[0]->HeaderCell->Value = "Лексема";

this->dataGridViewConst->Columns[1]->HeaderCell->Value = "Код";

}

};

class Analyser {

private:

std::string source;

std::string stripped;

std::vector<char> delimers = {' ', '.', ',', ';', ':', '\0', '\n', '(', ')', '[', ']', '{', '}'};

std::string buffer = "";

int state = 0;

public:

void clearState() {

this->state = 0;

}

int getState() const {

return this->state;

}

const std::string& getStripped() const {

return this->stripped;

}

// 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->state = 0;

this->stripped = "";

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

char out = space\_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;

}

// Функция анализатора

char space\_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: { switch (c) {

default: { state = 0; break; }

} break; 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: { switch (c) {

default: { state = 0; break; }

} break; break; }

} break;

}

case 5: {

switch (c) {

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

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

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

default: { switch (c) {

default: { state = 0; break; }

} break; break; }

} break;

}

}

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

return c;

return -1;

}

std::string lexem\_filter(char c) {

if (state > 0 && state <= 6 && std::count(delimers.begin(), delimers.end(), c)) {

if (state != 2 || c != '.') {

std::string r\_buff = buffer;

buffer = "";

return r\_buff;

}

}switch (state) {

case 0: {

switch (c) {

case 'a': { state = 7; break; }

case 'b': { state = 10; break; }

case 'c': { state = 16; break; }

case 'd': { state = 24; break; }

case 'e': { state = 33; break; }

case 'f': { state = 36; break; }

case 'i': { state = 41; break; }

case 'p': { state = 43; break; }

case 'r': { state = 53; break; }

case 't': { state = 58; break; }

case 'v': { state = 61; break; }

case '0': { state = 64; break; }

case '1': { state = 64; break; }

case '2': { state = 64; break; }

case '3': { state = 64; break; }

case '4': { state = 64; break; }

case '5': { state = 64; break; }

case '6': { state = 64; break; }

case '7': { state = 64; break; }

case '8': { state = 64; break; }

case '9': { state = 64; break; }

case '-': { state = 65; break; }

case '+': { state = 65; break; }

case '.': { state = 66; break; }

default: { state = 0; break; }

} break;

}

case 1: {

switch (c) {

default: { state = 0; break; }

} break;

}

case 2: {

switch (c) {

default: { state = 0; break; }

} break;

}

case 3: {

switch (c) {

default: { state = 0; break; }

} break;

}

case 4: {

switch (c) {

default: { state = 0; break; }

} break;

}

case 5: {

switch (c) {

default: { state = 0; break; }

} break;

}

case 6: {

switch (c) {

default: { state = 0; break; }

} break;

}

case 7: {

switch (c) {

case 'u': { state = 8; break; }

default: { state = 0; break; }

} break;

}

case 8: {

switch (c) {

case 't': { state = 9; break; }

default: { state = 0; break; }

} break;

}

case 9: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 10: {

switch (c) {

case 'r': { state = 11; break; }

case 'o': { state = 14; break; }

default: { state = 0; break; }

} break;

}

case 11: {

switch (c) {

case 'e': { state = 12; break; }

default: { state = 0; break; }

} break;

}

case 12: {

switch (c) {

case 'a': { state = 13; break; }

default: { state = 0; break; }

} break;

}

case 13: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 14: {

switch (c) {

case 'o': { state = 15; break; }

default: { state = 0; break; }

} break;

}

case 15: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 16: {

switch (c) {

case 'l': { state = 17; break; }

case 'h': { state = 20; break; }

case 'a': { state = 22; break; }

default: { state = 0; break; }

} break;

}

case 17: {

switch (c) {

case 'a': { state = 18; break; }

default: { state = 0; break; }

} break;

}

case 18: {

switch (c) {

case 's': { state = 19; break; }

default: { state = 0; break; }

} break;

}

case 19: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 20: {

switch (c) {

case 'a': { state = 21; break; }

default: { state = 0; break; }

} break;

}

case 21: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 22: {

switch (c) {

case 's': { state = 23; break; }

default: { state = 0; break; }

} break;

}

case 23: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 24: {

switch (c) {

case 'o': { state = 25; break; }

case 'e': { state = 29; break; }

default: { state = 0; break; }

} break;

}

case 25: {

switch (c) {

case 'u': { state = 26; break; }

default: { state = 0; break; }

} break;

}

case 26: {

switch (c) {

case 'b': { state = 27; break; }

default: { state = 0; break; }

} break;

}

case 27: {

switch (c) {

case 'l': { state = 28; break; }

default: { state = 0; break; }

} break;

}

case 28: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 29: {

switch (c) {

case 'l': { state = 30; break; }

default: { state = 0; break; }

} break;

}

case 30: {

switch (c) {

case 'e': { state = 31; break; }

default: { state = 0; break; }

} break;

}

case 31: {

switch (c) {

case 't': { state = 32; break; }

default: { state = 0; break; }

} break;

}

case 32: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 33: {

switch (c) {

case 'l': { state = 34; break; }

default: { state = 0; break; }

} break;

}

case 34: {

switch (c) {

case 's': { state = 35; break; }

default: { state = 0; break; }

} break;

}

case 35: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 36: {

switch (c) {

case 'a': { state = 37; break; }

case 'o': { state = 40; break; }

default: { state = 0; break; }

} break;

}

case 37: {

switch (c) {

case 'l': { state = 38; break; }

default: { state = 0; break; }

} break;

}

case 38: {

switch (c) {

case 's': { state = 39; break; }

default: { state = 0; break; }

} break;

}

case 39: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 40: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 41: {

switch (c) {

case 'n': { state = 42; break; }

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

default: { state = 0; break; }

} break;

}

case 42: {

switch (c) {

case 't': { state = 1; break; }

default: { state = 0; break; }

} break;

}

case 43: {

switch (c) {

case 'r': { state = 44; break; }

case 'u': { state = 49; break; }

default: { state = 0; break; }

} break;

}

case 44: {

switch (c) {

case 'i': { state = 45; break; }

default: { state = 0; break; }

} break;

}

case 45: {

switch (c) {

case 'v': { state = 46; break; }

default: { state = 0; break; }

} break;

}

case 46: {

switch (c) {

case 'a': { state = 47; break; }

default: { state = 0; break; }

} break;

}

case 47: {

switch (c) {

case 't': { state = 48; break; }

default: { state = 0; break; }

} break;

}

case 48: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 49: {

switch (c) {

case 'b': { state = 50; break; }

default: { state = 0; break; }

} break;

}

case 50: {

switch (c) {

case 'l': { state = 51; break; }

default: { state = 0; break; }

} break;

}

case 51: {

switch (c) {

case 'i': { state = 52; break; }

default: { state = 0; break; }

} break;

}

case 52: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 53: {

switch (c) {

case 'e': { state = 54; break; }

default: { state = 0; break; }

} break;

}

case 54: {

switch (c) {

case 't': { state = 55; break; }

default: { state = 0; break; }

} break;

}

case 55: {

switch (c) {

case 'u': { state = 56; break; }

default: { state = 0; break; }

} break;

}

case 56: {

switch (c) {

case 'r': { state = 57; break; }

default: { state = 0; break; }

} break;

}

case 57: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 58: {

switch (c) {

case 'r': { state = 59; break; }

default: { state = 0; break; }

} break;

}

case 59: {

switch (c) {

case 'u': { state = 60; break; }

default: { state = 0; break; }

} break;

}

case 60: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 61: {

switch (c) {

case 'o': { state = 62; break; }

default: { state = 0; break; }

} break;

}

case 62: {

switch (c) {

case 'i': { state = 63; break; }

default: { state = 0; break; }

} break;

}

case 63: {

switch (c) {

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

default: { state = 0; break; }

} break;

}

case 64: {

switch (c) {

case '0': { state = 64; break; }

case '1': { state = 64; break; }

case '2': { state = 64; break; }

case '3': { state = 64; break; }

case '4': { state = 64; break; }

case '5': { state = 64; break; }

case '6': { state = 64; break; }

case '7': { state = 64; break; }

case '8': { state = 64; break; }

case '9': { state = 64; break; }

case 'e': { state = 67; break; }

case 'E': { state = 67; break; }

case '.': { state = 66; break; }

default: { state = 0; break; }

} break;

}

case 65: {

switch (c) {

case '0': { state = 64; break; }

case '1': { state = 64; break; }

case '2': { state = 64; break; }

case '3': { state = 64; break; }

case '4': { state = 64; break; }

case '5': { state = 64; break; }

case '6': { state = 64; break; }

case '7': { state = 64; break; }

case '8': { state = 64; break; }

case '9': { state = 64; break; }

default: { state = 0; break; }

} break;

}

case 66: {

switch (c) {

case '0': { state = 64; break; }

case '1': { state = 64; break; }

case '2': { state = 64; break; }

case '3': { state = 64; break; }

case '4': { state = 64; break; }

case '5': { state = 64; break; }

case '6': { state = 64; break; }

case '7': { state = 64; break; }

case '8': { state = 64; break; }

case '9': { state = 64; break; }

default: { state = 0; break; }

} break;

}

case 67: {

switch (c) {

case '0': { state = 2; break; }

case '1': { state = 2; break; }

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

case '3': { state = 2; break; }

case '4': { state = 2; break; }

case '5': { state = 2; break; }

case '6': { state = 2; break; }

case '7': { state = 2; break; }

case '8': { state = 2; break; }

case '9': { state = 2; break; }

case '-': { state = 68; break; }

case '+': { state = 68; break; }

default: { state = 0; break; }

} break;

}

case 68: {

switch (c) {

case '0': { state = 2; break; }

case '1': { state = 2; break; }

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

case '3': { state = 2; break; }

case '4': { state = 2; break; }

case '5': { state = 2; break; }

case '6': { state = 2; break; }

case '7': { state = 2; break; }

case '8': { state = 2; break; }

case '9': { state = 2; break; }

default: { state = 0; break; }

} break;

}

}

buffer += c;

return "";

}

};

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы было выполнено построение конечного автомата, на основе которого была спроектирована функция лексического анализатора, которая разделяет лексемы и классифицирует их, добавляя в таблицу лексем.

Функция лексического анализатора оформлена в виде графического приложения, и её работа протестирована на некотором тексте программы. Для тестирования распознавания слов был выбран набор из 10 кратких текстов программ на языке C++, которые имели различный набор ключевых слов и имели разный размер.

Конечный автомат, на котором была построена функция лексического анализатора состоял из 63-х состояний, среди которых 6 конечных для определения класса лексем.