Пояснительная записка

к курсовой работе по дисциплине «Алгоритмизация вычислительных процессов»

на тему Разработка синтаксического анализатора оператора языка программирования

Выполнил: студент ­ 2 курса, группы ИВТ/б-22о

направления подготовки 09.03.01 – Информатика и вычислительная техника

Черняев Никита Георгиевич

Руководитель Лелеков С.Г., к.ф.-м.н., доцент

(фамилия, инициалы, степень, звание, должность)

Дата допуска к защите «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

Зав. кафедрой Брюховецкий А.А.

(подпись) (инициалы, фамилия)

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и управления в технических системах

Кафедра информационных технологий и компьютерных систем

Направление подготовки/специальность 09.03.01 – Информатика и вычислительная техника (код и название)

Профиль/специализация

Курс 2 Группа ИВТ/б-22о Семестр 3

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ СТУДЕНТА

Черняев Никита Георгиевич

(фамилия, имя, отчество)

1. Тема проекта (работы) Разработка синтаксического анализатора языка программирования

2. Сроки сдачи студентом законченного проекта (работы) 17.12.2018г.

3. Исходные данные к проекту (работе) Вариант № 4

Тема для анализа: Массивы;

Шаблон do \*(имя1+ имя2 ) = число; имя2-=число; while(имя2>0);

4. Содержание пояснительной записки (перечень вопросов, подлежащих разработке)

1. Титульный лист.
2. Лист задания.
3. Содержание.
4. Введение.
5. Сравнительный анализ существующих подходов к реализации…
6. Программный проект анализа синтаксической структуры оператора языка программирования.
   1. Постановка задачи.
   2. Описание программы следующими программными документами:
      1. Техническое задание на выполнение проектных работ (ГОСТ 19.201-78 Единая система программной документации (ЕСПД). Техническое задание. Требования к содержанию и оформлению (с Изменением N 1)).
      2. Описание программы (ГОСТ 19.402-78 Описание программы. Требования к содержанию и оформлению);
      3. Руководство программиста (ГОСТ 19.504-79 Руководство программиста. Требования к содержанию и оформлению);
      4. Программа и методика испытаний (ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению).
7. Выводы(Заключение).
8. Библиографический список.
9. Приложения (тексты программных модулей).

5. Перечень графического материала (с точным указанием обязательных чертежей)

1. Структурная схема программной системы.
2. Блок – схема работы системы.

6. Дата выдачи задания 04.09.2018г.

КАЛЕНДАРНЫЙ ПЛАН

|  |  |
| --- | --- |
| Номер недели | Содержание работы |
| 1-2 | Получение задания. Уяснение его сути |
| 3-4 | Подготовка и утверждение преподавателем технического задания на проектирование программного продукта. Выполнение задания 1 |
| 5-6 | Подготовка документа «Описание программы» |
| 7-8 | Программирование в соответствии с заданием 2 |
| 9-12 | Тестирование и отладка программной системы. |
| 13-14 | Оформление документов «Руководство программиста» и «Программа и методика испытаний» |
| 15-16 | Оформление пояснительной записки |
| 17 | Защита курсовой работы |

Студент Черняев Никита Георгиевич

(подпись)

Руководитель проекта (работы) Лелеков С.Г., доцент

(фамилия, инициалы, должность) (подпись)

« 04 » сентября 2018г.

Содержание

[ВВЕДЕНИЕ 6](#_Toc533013080)

[1. **Сравнительный анализ реализации массивов в различных языках программирования.** 7](#_Toc533013081)

[1.1. **Общие сведения о массивах** 7](#_Toc533013082)

[1.2. **Массивы в языке Pascal** 7](#_Toc533013083)

[1.3. **Массивы в языке JavaScript** 8](#_Toc533013084)

[1.4. **Массивы в языке Java** 9](#_Toc533013085)

[1.5. **Массивы в языке С++** 12](#_Toc533013086)

[1.6. **Массивы в языке PHP** 14](#_Toc533013087)

[1.7. **Массивы в языке C#** 16](#_Toc533013088)

[1.8. **Вывод** 19](#_Toc533013089)

[2. **Программный проект анализа синтаксической структуры оператора языка программирования** 20](#_Toc533013090)

[2.1. **Постановка задачи** 20](#_Toc533013091)

[2.2. **Техническое задание на разработку программы** 21](#_Toc533013092)

[2.2.1. **Введение** 21](#_Toc533013093)

[2.2.2. **Основания для разработки** 21](#_Toc533013094)

[2.2.3. **Назначения разработки** 21](#_Toc533013095)

[2.2.4. **Требование, предъявляемое к программе** 21](#_Toc533013096)

[2.2.4.1. **Требования к функциональным характеристикам:** 21](#_Toc533013097)

[2.2.4.2. **Возможные ошибки программы** 22](#_Toc533013098)

[2.2.4.3. **Требование к надежности** 22](#_Toc533013099)

[2.2.4.4. **Условия эксплуатации** 23](#_Toc533013100)

[2.2.4.5. **Требования к составу и параметрам технических средств** 23](#_Toc533013101)

[2.2.4.6. **Требования к информационной и программной совместимости** 23](#_Toc533013102)

[2.2.5. **Технико-экономические показатели** 24](#_Toc533013103)

[2.2.6. **Стадии и этапы разработки** 24](#_Toc533013104)

[2.2.7. **Порядок контроля и приемки** 25](#_Toc533013105)

[2.3. **Описание программы** 26](#_Toc533013106)

[2.3.1. **Общие сведения** 26](#_Toc533013107)

[2.3.2. **Функциональное назначение** 26](#_Toc533013108)

[2.3.3. **Описание логической структуры** 26](#_Toc533013109)

[2.3.4. **Используемые средства** 30](#_Toc533013110)

[2.3.5. **Вызов и загрузка** 30](#_Toc533013111)

[2.3.6. **Входные и выходные данные** 30](#_Toc533013112)

[2.4. **Руководство программиста** 30](#_Toc533013113)

[2.4.1. **Назначение и условие применения программы** 30](#_Toc533013114)

[2.4.2. **Характеристика программы** 31](#_Toc533013115)

[2.4.3. **Обращение к программе** 32](#_Toc533013116)

[2.4.4. **Входные и выходные данные программы** 33](#_Toc533013117)

[2.4.5. **Сообщения** 33](#_Toc533013118)

[2.5. **Программа и методика испытаний** 34](#_Toc533013119)

[2.5.1. **Объект испытаний** 34](#_Toc533013120)

[2.5.2. **Цель испытаний** 35](#_Toc533013121)

[2.5.3. **Требования к программе** 35](#_Toc533013122)

[2.5.4. **Требования к программной документации** 36](#_Toc533013123)

[2.5.5. **Методы испытаний** 37](#_Toc533013124)

[**Заключение** 42](#_Toc533013125)

[**Библиографический список** 43](#_Toc533013126)

[**Приложение** 44](#_Toc533013127)

ВВЕДЕНИЕ

Цель курсовой работы: выполнение сравнительного анализа существующих синтаксических конструкций языков программирования C, Pascal, PHP, VisualBasic, Python, C# и др. Реализация программного проекта анализа синтаксической структуры языка программирования.

Задачи курсовой работы:

* развитие умения разрабатывать программные модели средней сложности;
* развитие навыков выполнения научно-исследовательских работ в области разработки и стыковки многомодульных программных комплексов;
* развитие навыков работы со стандартами, научно-технической и справочной литературой в области программирования и вычислительной техники.

Предметом работы является автоматизированная система синтаксического анализа оператора языка программирования.

В нынешнее время автоматизированные устройства и ПО повсеместно замещают ручное средства для хранения и работы с информацией. Из – за этого в области информационных технологий существует высокий спрос на специалистов, способных спроектировать и отладить гибкую информационную систему, не только предоставляющую эффективные и удобные способы обработки информации, но и способную к обновлениям и модификациям без значительных временных и материальных затрат.

Курсовая работа является актуальной, так как знания и умения в областях программирования и проектирования ПО, необходимые для выполнения курсовой работы, являются базовыми для любого разработчика автоматизированного программного обеспечения.

Выполнить сравнительный анализ синтаксических конструкций массивов в различных языках программирования: С, Pascal, PHP, Python, Java, C#, VisualBasic, и др.

1. **Сравнительный анализ реализации массивов в различных языках программирования.**
   1. **Общие сведения о массивах**

Массив — [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона. Одномерный массив можно рассматривать как реализацию [абстрактного типа данных](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [вектор](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80).

Размерность массива — это количество индексов, необходимое для однозначной адресации элемента в рамках массива. По количеству используемых индексов массивы делятся на одномерные, двумерные, трёхмерные и т. д.

Форма или структура массива — сведения о количестве размерностей и размере (протяжённости) массива по каждой из размерностей; может быть представлена одномерным массивом.

Особенностью массива как структуры данных (в отличие, например, от [связного списка](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA)) является константная [вычислительная сложность](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) доступа к элементу массива по индексу. Массив относится к структурам данных с [произвольным доступом](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF).

* 1. **Массивы в языке Pascal**

В языке Паскаль тип массива задается с использованием специального слова array (англ. – массив). Для объявления массива необходимо указать число и тип элементов массива. Для обращения к элементу массива необходимо указать имя переменной-массива и в квадратных скобках индекс элемента. Например,

**var**

m, n: **array**[1..100]**of** real;

x:**array**[1..100]**of** char;

Здесь число элементов массива указывается с помощью диапазона, причём минимальное число будет соответствовать минимальному индексу элемента, а максимальное – максимальному индексу элемента.

Массивы могут иметь и несколько размерностей. В этом случае необходимо через запятую указать диапазоны индексов для каждой размерности. В частности, для представления матриц 100х200 можно использовать следующее объявление:

**type**

Matrix = **array**[1..100, 1..200]**of** real;

**var**

m: Matrix;

* 1. **Массивы в языке JavaScript**

Javascript поддерживает два вида структуры "массив":

1. Ассоциативный массив (хеш), где данные хранятся по произвольному ключу.
2. Числовой массив Array, где данные хранятся по номерам.

Javascript - очень гибкий язык, поэтому технически в Array можно хранить произвольные ключи, как в Object. Но лучше использовать типы по назначению.

Для хранения данных по номеру предназначен тип Array. В типе Array есть специальные методы, ориентированные именно на работу с числовыми ключами.

Есть два эквивалентных способа создания массива:

var a = new Array()

var a = []

Или, сразу со значениями

**var** a = **new** Array("a", 1, **true**)

**var** a = ["a", 1, **true**]

Отсчет элементов начинается с нуля.

Массив хранит данные по численным ключам, но внутри он использует точно такой же хэш (ту же структуру данных), как и обычный объект, поэтому можно сделать так:

**var** a = []

a[1] = 1

a[999999] = 2

и массив a будет занимать память, нужную для хранения этих двух соответствий, а не займет длинный непрерывный кусок памяти, как это произошло бы в языке С.

Методы push и pop добавляют или вынимают значение с конца массива; методы shift/unshift делают то же самое, с начала массива.

Также в данном языке программирования предусмотрены множество функций и методов для работы с массивами.

* 1. **Массивы в языке Java**

Массив представляет набор однотипных значений. Объявление массива похоже на объявление обычной переменной, которая хранит одиночное значение, причем есть два способа объявления массива:

|  |
| --- |
| тип\_данных название\_массива[];  // либо  тип\_данных[] название\_массива; |

Например, определим массив чисел:

|  |
| --- |
| int nums[];  int[] nums2; |

После объявления массива мы можем инициализовать его:

|  |
| --- |
| int nums[];  nums = new int[4];  // массив из 4 чисел |

Создание массива производится с помощью следующей конструкции: new тип\_данных[количество\_элементов], где new - ключевое слово, выделяющее память для указанного в скобках количества элементов. Например, nums = new int[4]; - в этом выражении создается массив из четырех элементов int, и каждый элемент будет иметь значение по умолчанию - число 0.

Также можно сразу при объявлении массива инициализировать его:

|  |
| --- |
| int nums[] = new int[4];    // массив из 4 чисел  int[] nums2 = new int[5];   // массив из 5 чисел |

При подобной инициализации все элементы массива имеют значение по умолчанию. Для числовых типов (в том числе для типа char) это число 0, для типа boolean это значение false, а для остальных объектов это значение **null**. Например, для типа int значением по умолчанию является число 0, поэтому выше определенный массив nums будет состоять из четырех нулей.

Однако также можно задать конкретные значения для элементов массива при его создании:

|  |
| --- |
| // эти два способа равноценны  int[] nums = new int[] { 1, 2, 3, 5 };    int[] nums2 = { 1, 2, 3, 5 }; |

Стоит отметить, что в этом случае в квадратных скобках не указывается размер массива, так как он вычисляется по количеству элементов в фигурных скобках.

После создания массива мы можем обратиться к любому его элементу по индексу, который передается в квадратных скобках после названия переменной массива:

|  |
| --- |
| int[] nums = new int[4];  // устанавливаем значения элементов массива  nums[0] = 1;  nums[1] = 2;  nums[2] = 4;  nums[3] = 100;    // получаем значение третьего элемента массива  System.out.println(nums[2]);    // 4 |

Индексация элементов массива начинается с 0, поэтому в данном случае, чтобы обратиться к четвертому элементу в массиве, нам надо использовать выражение nums[3].

И так как у нас массив определен только для 4 элементов, то мы не можем обратиться, например, к шестому элементу: nums[5] = 5;. Если мы так попытаемся сделать, то мы получим ошибку.

Многомерные массивы

Ранее мы рассматривали одномерные массивы, которые можно представить как цепочку или строку однотипных значений. Но кроме одномерных массивов также бывают и многомерными. Наиболее известный многомерный массив - таблица, представляющая двухмерный массив:

|  |
| --- |
| int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };    int[][] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } }; |

* 1. **Массивы в языке С++**

Массив представляет набор однотипных данных. Формальное определение массива выглядит следующим образом:

|  |
| --- |
| тип\_переменной название\_массива [длина\_массива] |

После типа переменной идет название массива, а затем в квадратных скобках его размер. Например, определим массив из 4 чисел:

|  |
| --- |
| **int** numbers[4]; |

Данный массив имеет четыре числа, но все эти числа имеют неопределенное значение. Однако мы можем выполнить инициализацию и присвоить этим числам некоторые начальные значения через фигурные скобки:

|  |
| --- |
| **int** numbers[4] = {1,2,3,4}; |

Значения в фигурных скобках еще называют инициализаторами. Если инициализаторов меньше, чем элементов в массиве, то инициализаторы используются для первых элементов. Если инициализаторов больше, чем элементов в массиве, то при компиляции возникнет ошибка:

|  |
| --- |
| **int** numbers[4] = {1, 2, 3, 4, 5, 6}; |

Здесь массив имеет размер 4, однако ему передается 6 значений.

Если размер массива не указан явно, то он выводится из количества инициализаторов:

|  |
| --- |
| **int** numbers[] = {1, 2, 3, 4, 5, 6}; |

В данном случае в массиве есть 6 элементов.

Свои особенности имеет инициализация символьных массивов. Мы можем передать символьному массиву как набор инициализаторов, так и строку:

|  |
| --- |
| **char** s1[] = {'h', 'e', 'l', 'l', 'o'};  **char** s2[] = "world"; |

Причем во втором случае массив s2 будет иметь не 5 элементов, а 6, поскольку при инициализации строкой в символьный массив автоматически добавляется нулевой символ '\0'.

При этом не допускается присвоение одному массиву другого массива:

|  |
| --- |
| **int** nums1[] = {1,2,3,4,5};  **int** nums2[] = nums1;    // ошибка  nums2 = nums1;          // ошибка |

После определения массива мы можем обратиться к его отдельным элементам по индексу. Индексы начинаются с нуля, поэтому для обращения к первому элементу необходимо использовать индекс 0. Обратившись к элементу по индексу, мы можем получить его значение, либо изменить его

Число элементов массива также можно определять через константу

Кроме одномерных массивов в C++ есть многомерные. Элементы таких массивов сами в свою очередь являются массивами, в которых также элементы могут быть массивами. Например, определим двухмерный массив чисел:

**int** numbers[3][2];

Такой массив состоит из трех элементов, при этом каждый элемент представляет массив из двух элементов. Инициализируем подобный массив:

**int** numbers[3][2] = { {1, 2}, {4, 5}, {7, 8} };

Также при инициализации можно опускать фигурные скобки:

**int** numbers[3][2] = { 1, 2, 4, 5, 7, 8 };

Возможна также инициализация не всех элементов, а только некоторых:

**int** numbers[3][2] = { {1, 2}, {}, {7} };

И чтобы обратиться к элементам вложенного массива, потребуется два индекса.

Также в данном языке программирования предусмотрены множество функций и методов для работы с массивами.

* 1. **Массивы в языке PHP**

Массив - это набор данных, которые объединены под одним именем. Массив состоит из нескольких элементов, которые имеют свой определенный индекс. Массивы создаются при помощи [оператора присвоения](http://php720.com/lesson/9), также как и переменная.   
Имена массивов начинаются со знака **$**, после которого следует произвольный идентификатор, далее идут квадратные скобки: **$**arr[0] = "php";

Данная конструкция создает массив и присваивает его элементу с индексом 0 значение "php", после чего мы можем обращаться к этому элементу как к обычной переменной: echo $arr[0]. В результате мы увидим слово php.

Также, мы можем добавить еще элементы к массиву:

<?php

$arr[1] = "html";

$arr[2] = "css";

?>

В качестве индекса элементов массива мы можем использовать не только числа:

<?php

$arr["Kiev"] = 3000000;

$arr["Paris"] = 5000000;

$arr["LA"] = 15000000;

?>

В качестве значений индексов элементов и самих элементов мы можем использовать одинаковые [типы данных](http://php720.com/lesson/6) **одновременно**.

Также существует сокращенная запись для индексирования:

<?php

$arr[] = 3000000;

$arr[] = 5000000;

$arr[] = 15000000;

?>

В этом случае первый элемент (3000000) получит индекс 0.

Для создания массива мы можем использовать функцию array:

<?php

$arr = array("php", "html", "css");

?>

В этом случае первый элемент получит индекс 0. Если нужно присвоить какой-то другой номер, то можно воспользоваться конструкцией =>:

<?php

$arr = array(1 => "php", "html", "css");

?>

Теперь элемент под номером 1 это "php", а не "html". Также, можно создать массив со строковым индексом:

<?php

$arr = array("first" => "php", "second" => "html", "third" => "css")

?>

Начиная с версии PHP 5.4, массивы можно создать через квадратные скобки:

<?php

$arr = ["php", "laravel", "yii", "zend", "cakephp"];

?>

Многомерные массивы записываются аналогично.

Также в данном языке программирования предусмотрены множество функций и методов для работы с массивами.

* 1. **Массивы в языке C#**

Массив представляет набор однотипных данных. Объявление массива похоже на объявление переменной за тем исключением, что после указания типа ставятся квадратные скобки:

|  |
| --- |
| тип\_переменной[] название\_массива; |

Например, определим массив целых чисел:

|  |
| --- |
| int[] numbers; |

После определения переменной массива мы можем присвоить ей определенное значение:

|  |
| --- |
| int[] nums = new int[4]; |

Здесь вначале мы объявили массив nums, который будет хранить данные типа int. Далее используя операцию new, мы выделили память для 4 элементов массива: new int[4]. Число 4 еще называется **длиной массива**. При таком определении все элементы получают значение по умолчанию, которое предусмотренно для их типа. Для типа int значение по умолчанию - 0.

Также мы сразу можем указать значения для этих элементов:

|  |
| --- |
| int[] nums2 = new int[4] { 1, 2, 3, 5 };    int[] nums3 = new int[] { 1, 2, 3, 5 };    int[] nums4 = new[] { 1, 2, 3, 5 };    int[] nums5 = { 1, 2, 3, 5 }; |

Все перечисленные выше способы будут равноценны.

Для обращения к элементам массива используются **индексы**. Индекс представляет номер элемента в массиве, при этом нумерация начинается с нуля, поэтому индекс первого элемента будет равен 0. А чтобы обратиться к четвертому элементу в массиве, нам надо использовать индекс 3, к примеру: nums[3]. Используем индексы для получения и установки значений элементов массива:

|  |
| --- |
| int[] nums = new int[4];  nums[0] = 1;  nums[1] = 2;  nums[2] = 3;  nums[3] = 5;  Console.WriteLine(nums[3]);  // 5 |

И так как у нас массив определен только для 4 элементов, то мы не можем обратиться, например, к шестому элементу: nums[5] = 5;. Если мы так попытаемся сделать, то мы получим исключение IndexOutOfRangeException.

Многомерные массивы

Массивы характеризуются таким понятием как **ранг** или количество измерений. Выше мы рассматривали массивы, которые имеют одно измерение (то есть их ранг равен 1) - такие массивы можно представлять в виде горзонтального ряда элемента. Но массивы также бывают многомерными. У таких массивов количество измерений (то есть ранг) больше 1.

Массивы которые имеют два измерения (ранг равен 2) называют двухмерными. Например, создадим одномерный и двухмерный массивы, которые имеют одинаковые элементы:

|  |
| --- |
| int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };    int[,] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } }; |

Также в данном языке программирования предусмотрены множество функций и методов для работы с массивами.

* 1. **Вывод**

В ходе курсовой работы было рассмотрено устройство массивов в следующих языках программирования: Pascal, C++, C#, Java, PHP, JavaScript.

В языках C++, C#, Java, JavaScript объявление массива, создание его элементов и обращение к ним, практически совпадают. Языки Java и C# располагают большим количеством функций и методов для работы с массивами.

Язык PHP сильно отличается от остальных языков описанием массивов, но при этом схож с этими языками в работе с массивами и их элементами.

Наиболее сложное описание массивов в языке Pascal, так как там требуется описывать массивы в начале программы с указанием номера первого и последнего элемента, что является неудобным при программировании. Также в языке Pascal очень мало функций для работы с массивами.

Лучшие языки для работы с массивами – Java, C#.

Худший язык для работы с массивами – Pascal.

1. **Программный проект анализа синтаксической структуры оператора языка программирования**
   1. **Постановка задачи**

Задан вид оператора некоторого языка программирования. Отдельные части оператора разделяются группой пробелов или соответствующим разделителем (например, знаком операции, скобками и т.д.).

Разработанная программа должна:

1) проверять любой оператор такого вида на соответствие правилам записи оператора. Проверяются: числа, имена, баланс скобок, знаки операций….

2) быть разбита на подпрограммы (выделены классы), среди которых обязательно должны быть подпрограммы(методы) проверки имени и проверки числа. Параметры передавать через параметры подпрограмм.

3) иметь графический интерфейс.

4) использовать средства ввода как с клавиатуры, так и из внешнего файла. Проверяемый оператор вводится как строка символов длиной не более 80 символов.

5) при обнаружении ошибок выводить сведения: тип ошибки (ошибка в имени, числе и т.д.), место ошибки в операторе.

6) при отсутствии ошибок выводить сообщении о правильности оператора.

7) результаты тестирования выводить на форму и во внешний файл.

Среда программирования – по выбору.

* 1. **Техническое задание на разработку программы**
     1. **Введение**

Наименование: «Анализатор строки»

**Назначение и область применения:**

Программа разрабатывается для анализа синтаксических структур оператора некоторого языка программирования.

* + 1. **Основания для разработки**

Учебный план по направлению «Информатика и вычислительная техника»

* + 1. **Назначения разработки**

Программа предназначена для анализа синтаксических структур оператора некоторого языка программирования и проверки любого оператора такого вида на соответствие правилам записи.

* + 1. **Требование, предъявляемое к программе**
       1. **Требования к функциональным характеристикам:**

Разработанная программа должна:

1) проверять любой оператор вида do \*(имя1+ имя2 ) = число; имя2-=число; while(имя2>0); на соответствие правилам записи оператора. Проверяются: числа, имена, знаки операций….

2) быть разбита на подпрограммы (выделены классы), среди которых обязательно должны быть подпрограммы(методы) проверки имени и проверки числа. Параметры передавать через параметры подпрограмм.

3) иметь графический интерфейс.

4) использовать средства ввода как с клавиатуры, так и из внешнего файла. Проверяемый оператор вводится как строка символов длиной не более 80 символов.

5) при обнаружении ошибок выводить сведения: тип ошибки (ошибка в имени, числе и т.д.), место ошибки в операторе.

6) при отсутствии ошибок выводить сообщении о правильности оператора.

7) результаты тестирования выводить на форму и во внешний файл.

* + - 1. **Возможные ошибки программы**

В ходе работы программы могут возникнуть следующие ошибки:

1. ошибка несоответствия введенного оператора шаблону;
2. ошибка преувеличения допустимого размера строки (введенный оператор имеет длину более 80 символов);
3. ошибка несоответствия имени (идентификатора) языка программирования его синтаксису;
4. ошибка при вводе недопустимого числа;
5. ошибка при несовпадении соответствующих имен (идентификаторов) в введенном операторе;
6. ошибка в случае, когда файл, указанный в главном окне программы, не существует;
   * + 1. **Требование к надежности**

Организацией бесперебойного питания технических средств;

Использованием лицензионного программного обеспечения;

Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой.   
Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу конечного пользователя без предоставления ему административных привилегий.

* + - 1. **Условия эксплуатации**

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям,   
предъявляемым к техническим средствам в общем случае.

Минимальное количество персонала, требуемого для работы программы, должно составлять не менее 2 штатных единиц — преподаватель и конечный пользователь программы — оператор.

* + - 1. **Требования к составу и параметрам технических средств**

В состав технических средств должен входить персональный компьютер (ПЭВМ), выполняющий роль сервера, включающий в себя: оперативную память объемом, 1Гигабайт, не менее; процессор Pentium-2.0Hz, не менее; операционную систему Windows, версию не ранее Windows 2007, графический адаптер, поддерживающий DirectX 9.

* + - 1. **Требования к информационной и программной совместимости**

Среда программирования – JAVA.

Интегрированная среда разработки – NetBeans, не ранее версии 8.0.2.

Входные данные: строка, вводимая с клавиатуры или файла, выходными данными является информация, выводимая на экран и в файл.

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы.

**Требования к маркировке и упаковке не предъявляются**

**Требования к транспортированию и хранению не предъявляются**

**Требования к программной документации**

**Требования к программной документации:**

Состав программной документации должен включать:

* техническое задание;
* описание программы;
* руководство программиста;
* стадии и этапы разработки.
  + 1. **Технико-экономические показатели**

Требования к технико-экономическим показателям не предъявляются, ориентировочная экономическая эффективность не рассчитывается.

* + 1. **Стадии и этапы разработки**

**Стадии разработки**

Разработка должна быть проведена в три стадии:

1. разработка технического задания;
2. рабочее проектирование;
3. сдача преподавателю.

**Этапы разработки**

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания. На стадии рабочего проектирования должны быть выполнены следующие этапы работ:

1. разработка программы;
2. разработка программной документации;
3. испытания программы.

На стадии сдачи преподавателю должен быть выполнен этап разработки, подготовка и передача программы.

* + 1. **Порядок контроля и приемки**

Приемо-сдаточные испытания проводятся согласно указаниям преподавателя.

* 1. **Описание программы**
     1. **Общие сведения**

Наименование программы:

«Анализатор строки»

Программное обеспечение, необходимое для функционирования программы:

1. Операционная система Windows, версия не ранее Windows 2007;
2. Оперативная память объемом, 1Гигабайт, не менее;
3. Процессор Pentium-2.0Hz, не менее;
4. Графический адаптер, поддерживающий DirectX 9.

Программа написана на языке объектного программирования JAVA с использованием пакета графического интерфейса Java Swing.

* + 1. **Функциональное назначение**

Система представляет собой программный комплекс и предназначена для проверки оператора алгоритмического языка программирования на соответствие заранее заданному шаблону. Шаблон, по которому проверяется синтаксис оператора, заранее установлен в программе и имеет вид: «do \*(имя1+ имя2) = число; имя2-=число; while(имя2>0);»

Результатом является строка, сообщающая о наличие ошибок или их отсутствии. Эти данные выводятся в диалоговое окно и в файл.

Для иных целей программа не предназначена.

* + 1. **Описание логической структуры**

Метод, проверяющий оператор на соответствие шаблону – BtnCheck(String arg). Описание данного метода:

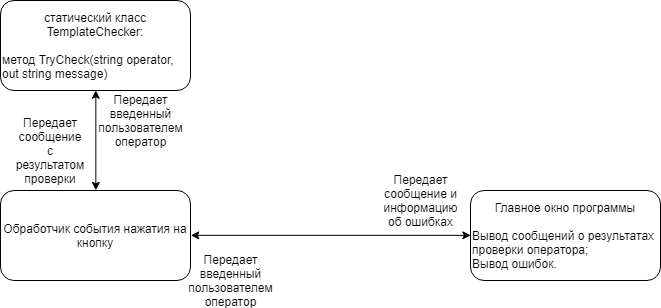
* проверить длину строки arg: если длина arg> 80 – вывести соответствующую ошибку;
* удалить пробелы из оператора с помощью метода ReplaceAll(String a, String b);
* создать строку, в которую будет помещаться выводимая информация;
* объявить массив, состоящий из имен и чисел оператора, полученных разбиением строки при помощи метода StringUtils.split(String, a) ;
* проверить совпадает ли количество имен и чисел в объявленном массиве и шаблоне, если нет - вывести соответствующую ошибку;
* проверить строку на общий шаблон оператора: если оператор не соответствует шаблону – поместить текст соответствующей ошибки в строку ошибок;
* создать цикл по каждому элементу массива с числами и именами, и в цикле проверять – является ли первый символ элемента цифрой:
  + если элемент начинается на цифру, проверить его в методе CheckNumber(String Number) – если результат false – добавить соответствующую ошибку в строку ошибок;
  + иначе, проверить его в методе CheckName(String Name) – если результат false – добавить соответствующую ошибку в строку ошибок;
* проверить, что первый элемент массива с именами и числами – строка «do»: если условие не выполняется – добавить соответствующую ошибку в строку ошибок;
* проверить, что седьмой элемент массива с именами и числами – строка «while»: если условие не выполняется – добавить соответствующую ошибку в строку ошибок;
* проверить, что девятый элемент массива с именами и числами – число «0»: если условие не выполняется – добавить соответствующую ошибку в строку ошибок;
* проверить, что второй элемент массива с именами и числами –переменная: если условие не выполняется – добавить соответствующую ошибку в строку ошибок;
* проверить, что четвертый и шестой элемент массива с именами и числами – числа: если условие не выполняется – добавить соответствующую ошибку в строку ошибок;
* проверить, совпадают ли первая переменная (второй элемент массива с именами и числами) и вторая (третий, пятый и восьмой элемент массива с именами и числами): если условие не выполняется – добавить соответствующую ошибку в строку ошибок;
* проверить, равны ли между собой вторые переменные (третий, пятый и восьмой элемент массива с именами и числами): если условие не выполняется – добавить соответствующую ошибку в строку ошибок;
* проверить, пустая ли строка ошибок
  + если да, то вернуть строку, что ошибок нет;
  + иначе проверить были ли найдены какие-то ошибки:
    - если да, то вернуть строку ошибок;
    - иначе добавить в строку ошибок, что ошибка в синтаксисе (скобки, знаки операций и др.) и вернуть строку ошибок.

Описание метода CheckNumber: данный метод получает как аргумент строку, использует метод Double.parseDouble(String a), и если используемый метод возвращает результат преобразования, CheckNumber возвращает true, иначе false.

Описание метода CheckName: данный метод получает как аргумент строку, создает регулярное выражение на проверку того, является ли данная строка переменной и возвращает значение метода Pattern.compile(regex).matcher(String).matches.

Методы, используемые в программе:

1. String.ReplaceAll(String a, String b) – заменяет в строке String все символы a на b;
2. String.length() – возвращает количество символов строки String;
3. StringUtils.split(String,char a) – возвращает массив подстрок, выделенных из исходной строки String с использованием разделителей a;
4. Pattern.compile(regex) – создает шаблон с регулярным выражением;
5. Pattern.compile(regex).matcher(String).matches – возвращает true, если String соответствует регулярному выражению regex;
6. Character.isDigit(String.charAt(int a)) – возвращает true, если в строке String а-ый элемент – цифра;
7. String.equals(String a) – возвращает true, если строка String совпадает со строкой a;
8. Paths.get(String a)) – открывает файл по введенному пути a;
9. Files.readAllBytes(Path path) – считывает весь файл, находящийся по пути path
10. Double.parseDouble(String a) - проверяет, возможно ли преобразование строкового десятичного представления числа к типу Double; если возможно, возвращает также сам результат преобразования;
11. String.indexOf(String a) – возвращает номер первого появления строки a в строке String;



метод BtnCheck

Класс Main

Рис. 1 – Схема структуры программы

* + 1. **Используемые средства**

Требования описаны в п. 2.3.1 – общие сведения о программе.

* + 1. **Вызов и загрузка**

Программа запускается путем запуска выполняемого файла StringAnalyst.exe с диска компьютера или с любого другого носителя данных. Объем программы составляет 37 Кбайт.

* + 1. **Входные и выходные данные**

Программа не требует входных данных при запуске. Во время работы программа способна принимать информацию текстового типа - проверяемый оператор языка программирования, путь к файлу с данным оператором и текстовая информация, считанная из файла. Входные данные не требуют предварительной подготовки.

Выходными данными являются сообщения текстового типа. Они могут выводиться как в файл, так и в главное окно программы.

* 1. **Руководство программиста**
     1. **Назначение и условие применения программы**

Система представляет собой программный комплекс и предназначена для проверки оператора алгоритмического языка программирования на соответствие заранее заданному шаблону. Программное обеспечение должно быть установлено на персональный компьютер.

Функции, выполняемые программой: проверка любого оператора такого вида на соответствие правилам записи оператора. Проверяются: числа, имена, знаки операций, проверяемый оператор вводится как строка символов длиной не более 80 символов с клавиатуры или из внешнего файла; при обнаружении ошибок выводятся сведения: тип ошибки (ошибка в имени, числе и т.д.), место ошибки в операторе; при отсутствии ошибок выводится сообщение о правильности оператора;

Результаты тестирования выводятся на форму и во внешний файл.

Условия, необходимые для выполнения программы описаны в п. 2.3.1 – общие сведения о программе.

* + 1. **Характеристика программы**

При корректной работе программы, она работает без задержек, моментально реагируя на действия пользователя. Возможные задержки связаны с нарушением работы аппаратной части компьютера или операционной системы.

Средства контроля правильности выполнения программы: программа выполняется без автоматических средств контроля. Контроль осуществляется пользователем, производящим работу над программой.

Режим работы программы: программа работает на главном экране операционной системы Windows под непосредственным управлением пользователя.

В случае критического завершения программы ее необходимо вручную перезапустить, так как самовосстановление не предусмотрено из-за ненадобности.

* + 1. **Обращение к программе**

Вид главного окна программы:

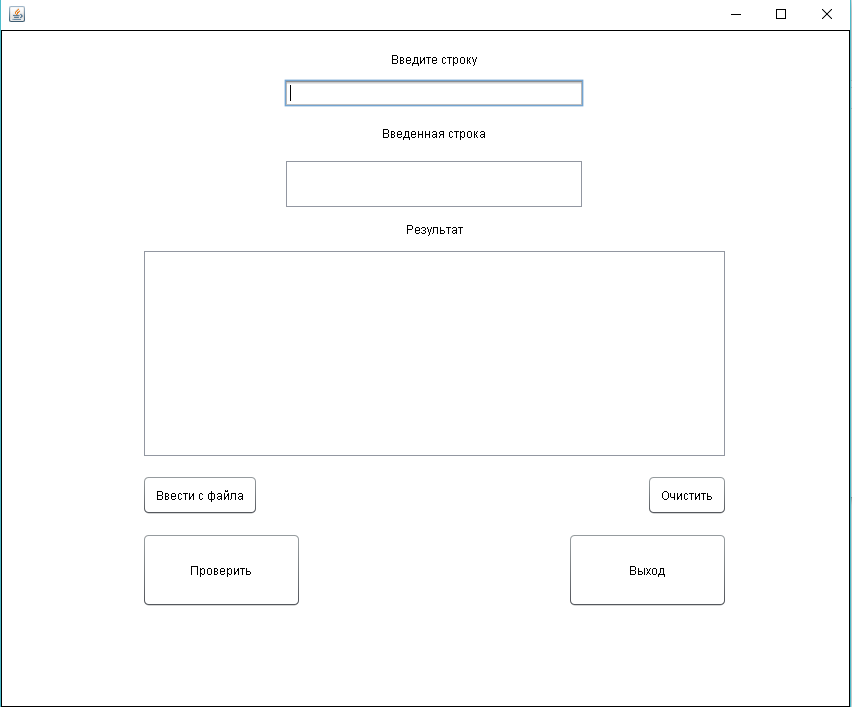


Рис. 2 – Вид главного окна программы

Вид сообщения, выводимого программой при запросе ввода пути к файлу:

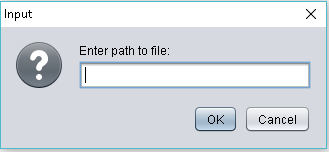


Рис. 3 – Вид окна сообщения программы

Программа запускается путем запуска выполняемого файла StringAnalyst.exe с диска компьютера или с любого другого носителя данных. Объем программы составляет 37 Кбайт.

Для выполнения функции проверки оператора непосредственно необходимо ввести в поле под названием «Введите строку» главного окна программы и затем нажать на кнопку «Проверить».

Для выполнения функции проверки оператора, записанного в отдельный файл необходимо нажать на кнопку «Ввести с файла» и в появившемся окне ввести путь к файлу в поле под названием «Enter path to file:» и затем нажать на кнопку «OK».

Чтобы очистить все поля необходимо нажать на кнопку «Очистить»

Для завершения работы программы необходимо воспользоваться стандартным средством завершения работы программы в Windows – нажать на крестик в правом верхнем углу главного окна программы или нажать на кнопку «Выход».

* + 1. **Входные и выходные данные программы**

Программа не требует входных данных при запуске. Во время работы программа может принимать текстовую информацию – оператор, который необходимо проверить, путь к файлу и информация из этого файла, а также путь к файлу, в который будет осуществлена запись. Входные данные не требуют предварительной подготовки.

Выходными данными также является текстовая информация, выводимая в главное окно программы в поле под названием «Результат» и в файл, путь к которому необходимо ввести в появляющемся окне.

* + 1. **Сообщения**

Программы может выводить следующие сообщения:

* «String contains no errors» - сообщение о том, что строка не содержит ошибок;
* «Operator length should be less than 80» - сообщение о том, что длина строки превышает 80 символов;
* «Input line doesn't match the pattern» - сообщение о том, что строка сильно не соответствует шаблону;
* «The entered string contains errors:» - сообщение о том, что введенная строка содержит ошибки;
* «Error in the number: … position …» - сообщение о том, что ошибка в числе … позиция …;
* «Error in the variable: … position …» - сообщение о том, что ошибка в переменной … позиция …;
* «Error in position 4, there must be variable» - - сообщение о том, что ошибка в позиции 4, здесь должна быть переменная;
* «Error in position 9 or 14, there must be numbers» - сообщение о том, что ошибка в позициях 9,14, здесь должны быть числа;
* «Error in service word do, position 1» - сообщение о том, что ошибка в слове do, позиция 1;
* «Error in service word while, position 16» - сообщение о том, что ошибка в слове while, позиция 16;
* «The 1st variable(positon 4) is equal to the 2nd(positions 6,11,18)» - сообщение о том, что первая переменная соответствует второй;
* «The second variable must be the same in positions 6,11 and 18» - сообщение о том, что вторая переменная не совпадает;
* «Error in position 20» - сообщение о том, что ошибка в позиции 20;
* «Errors in syntax(parentheses,braces,signs and others)» - сообщение о том, что ошибка в синтаксисе(скобки, фигурные скобки, знаки и др.)
  1. **Программа и методика испытаний**
     1. **Объект испытаний**

Наименование программы – «String Analyst ». Система представляет собой программный комплекс и предназначена для проверки оператора алгоритмического языка программирования на соответствие заранее заданному шаблону. Программное обеспечение должно быть установлено на персональный компьютер.

* + 1. **Цель испытаний**

Целью проведения испытаний является обнаружение и исправление возможных ошибок в работе программы в целом и в работе отдельных ее модулей.

* + 1. **Требования к программе**

Программа должна обеспечивать возможность выполнения следующих функций:

1) проверять любой оператор вида do \*(имя1+ имя2 ) = число; имя2-=число; while(имя2>0); на соответствие правилам записи оператора. Проверяются: числа, имена, знаки операций….

2) быть разбита на подпрограммы (выделены классы), среди которых обязательно должны быть подпрограммы(методы) проверки имени и проверки числа. Параметры передавать через параметры подпрограмм.

3) иметь графический интерфейс.

4) использовать средства ввода как с клавиатуры, так и из внешнего файла. Проверяемый оператор вводится как строка символов длиной не более 80 символов.

5) при обнаружении ошибок выводить сведения: тип ошибки (ошибка в имени, числе и т.д.), место ошибки в операторе.

6) при отсутствии ошибок выводить сообщении о правильности оператора.

7) результаты тестирования выводить на форму и во внешний файл.

Возможные ошибки при работе программы:

1. ошибка несоответствия введенного оператора шаблону;
2. ошибка преувеличения допустимого размера строки (введенный оператор имеет длину более 80 символов);
3. ошибка несоответствия имени (идентификатора) языка программирования его синтаксису;
4. ошибка при вводе недопустимого числа;
5. ошибка при несовпадении соответствующих имен (идентификаторов) в введенном операторе;
6. ошибка в случае, когда файл, указанный в главном окне программы, не существует;
   * 1. **Требования к программной документации**

Состав программной документации должен включать в себя:

1. техническое задание;
2. описание программы;
3. руководство программиста;
4. программа и методика испытаний.

Специальных требований к программе не предъявляется.

* + 1. **Методы испытаний**

В данном разделе будут последовательно приведены тесты, использовавшиеся для отладки программы. В тестировании будут приводится скриншоты окна программы. Скриншоты файла приводится не будут, так как содержимое его и поля в окне одинаково.

* В данном тесте была введена строка, которая не содержит ошибок. Результат работы программы:

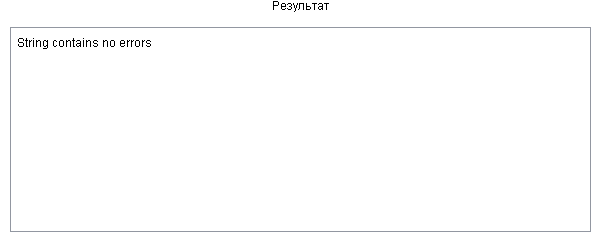


Рис. 4 – тестовый пример №1

* В данном тесте была введена строка, длиной более 80 символов. Результат работы программы:

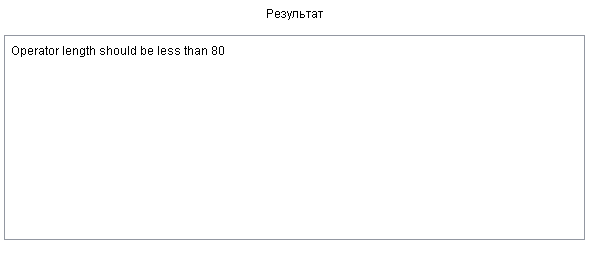


Рис. 5 – тестовый пример №2

* В данном тесте была введена строка, содержащая ошибку в слове do. Результат работы программы:

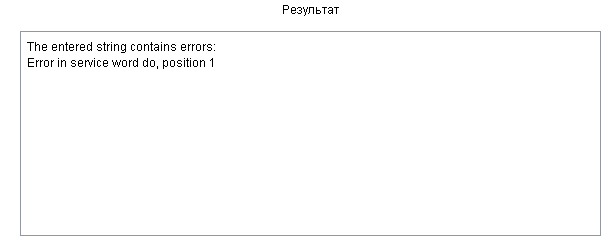


Рис. 6 – тестовый пример №3

* В данном тесте была введена строка, содержащая ошибку в слове while. Результат работы программы:

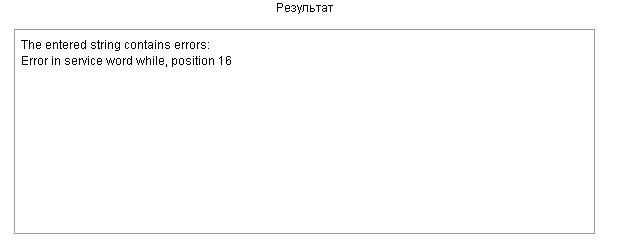


Рис. 7 – тестовый пример №4

* В данном тесте была введена строка, содержащая все одинаковые переменные. Результат работы программы:

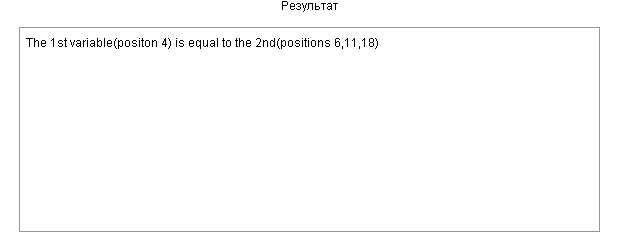


Рис. 8 – тестовый пример №5

* В данном тесте была введена строка, содержащая разные вторые переменные. Результат работы программы:

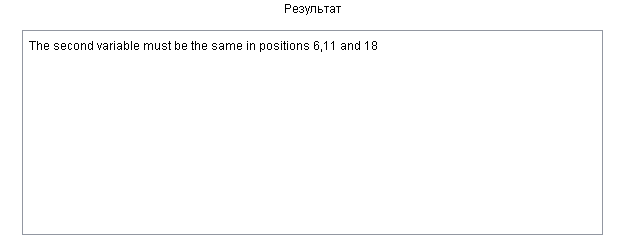


Рис. 9 – тестовый пример №6

* В данном тесте была введена строка, содержащая ошибки в числе на позиции 9. Результат работы программы:

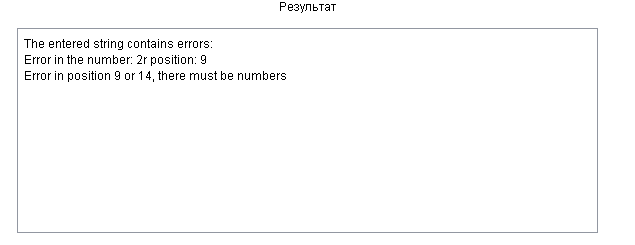


Рис. 10 – тестовый пример №7

* В данном тесте была введена строка, содержащая ошибки в числе на позиции 14. Результат работы программы:

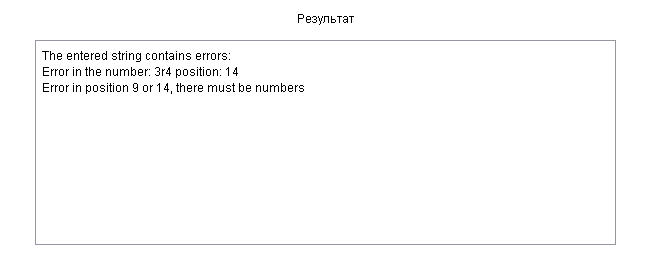


Рис. 11 – тестовый пример №8

* В данном тесте была введена строка, содержащая ошибки знаках. Результат работы программы:

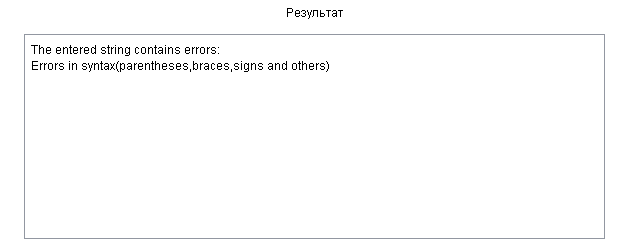


Рис. 12 – тестовый пример №9

* В данном тесте была введена строка, в которой на позиции 20 стоял не «0». Результат работы программы:

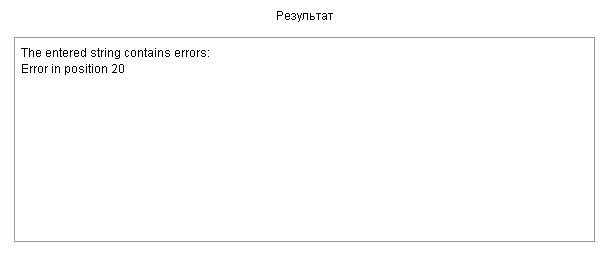


Рис. 13 – тестовый пример №10

* В данном тесте была введена строка, содержащая ошибку в вводе переменной на позиции 4. Результат работы программы:

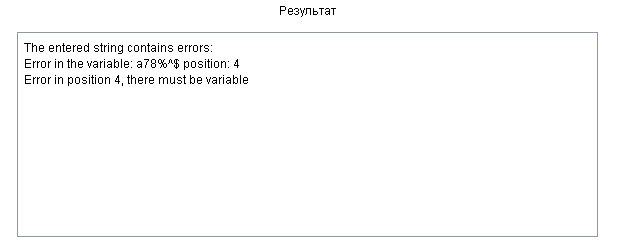


Рис. 14 – тестовый пример №11

# **Заключение**

В ходе курсовой работы был реализован проект программной реализации синтаксического анализатора языка программирования по заданному шаблону. Спроектированная программа под названием «String Analyst» имеет графический интерфейс в виде окна операционной системы Windows с тремя текстовыми полями и четырьмя й кнопками для обмена информацией с пользователем.

В программе реализована возможность вводить оператор как в главное окно программы, так и из внешнего файла. Вывод результата осуществляется в окно программы и во внешний файл. Программные модули были отлажены средствами ручного тестирования: было проведено 11 тестовых испытаний, все из которых программа прошла верно.

Реализация программы включает в себя два взаимосвязанных модуля: модуль логики, непосредственно осуществляющий проверку синтаксиса оператора, и модуль графического интерфейса, содержащий обработчики событий-действий пользователя.

На реализованный проект была оформлена следующая документация:

* техническое задание;
* описание программы;
* руководство программиста;
* программа и методика испытаний.

В результате выполнения курсовой работы были получены навыки объединения в единую систему многомодульных проектов и навыки работы со стандартами программной документации.

**Библиографический список**

1. Демидов, Д. В. - Основы программирования в примерах на языке Pascal: учебное пособие. - Москва: НИЯУ МИФИ, 2010. – 172 с.
2. Э. Фримен, Э. Робсон - Изучаем программирование на JavaScript – перевод Е. Матвеев – СПб: «Питер Пресс», 2014. – 640 с.
3. Брюс Эккель - Философия Java – СПб: Издательство «Питер», 2016. – 1168 с.
4. Бьерн Страуструп - Программирование: принципы и практика с использованием C++ - Издательство «Диалектика», 2011. – 1248 с.
5. Д. Албахари, Б. Албахари - C# 7.0. Справочник. Полное описание языка – Издательсвто «Диалектика», 2016. – 1024 с.
6. Дэвид Скляр - Изучаем РНР 7 - издательство «Диалектика»,2017. - 383c.

**Приложение**

Код класса CheckElement:

package com.main;

import java.util.regex.Pattern;

public class CheckElement {

public static boolean CheckNumber(String Number){

try{

Double.parseDouble(Number);

return true;

}

catch(NumberFormatException ex){

return false;

}

}

public static boolean CheckName(String Name){

Pattern Pat1;

Pat1 = Pattern.compile("^[\_a-zA-Z][\_a-zA-Z0-9]\*$");

return Pat1.matcher(Name).matches();

}

}

Код класса Main:

package com.main;

import java.util.regex.\*;

import java.io.\*;

import java.nio.file.Files;

import java.nio.file.Paths;

import org.apache.commons.lang3.\*;

public class Main {

public static String BtnCheck(String arg){

String str;

str = "";

arg.replaceAll("\\s","");

if (arg.length() > 80){

return str+= "Operator length should be less than 80\n";

}

String[] tokens = StringUtils.split(arg, "{}()\*+=-:;></ ");

if (tokens.length != 9){

return str += "Input line doesn't match the pattern\n";

}

Pattern patMain;

patMain = Pattern.compile("do\\\*\\([\_a-zA-Z][\_a-zA-Z0-9]\*\\+[\_a-zA-Z][\_a-zA-Z0-9]\*\\)=[0-9]\*[.,]?[0-9]+\\;[\_a-zA-Z][\_a-zA-Z0-9]\*-=[0-9]\*[.,]?[0-9]+\\;while\\([\_a-zA-Z][\_a-zA-Z0-9]\*>0\\)\\;");

Matcher matcher;

matcher = patMain.matcher(arg);

if (!matcher.matches()){

str += "The entered string contains errors:\n";

}

for(String token : tokens){

if (Character.isDigit(token.charAt(0))){

if (!CheckElement.CheckNumber(token)){

str += "Error in the number: " + token + " position: " + arg.indexOf(token) + "\n";

}

} else{

if (!CheckElement.CheckName(token)){

str += "Error in the variable: " + token + " position: " + arg.indexOf(token) + "\n";

}

}

}

if (!CheckElement.CheckName(tokens[1])){

str += "Error in position 4, there must be variable\n";

}

if ((!CheckElement.CheckNumber(tokens[3])) || (!CheckElement.CheckNumber(tokens[5]))){

str += "Error in position 9 or 14, there must be numbers\n";

}

if(!tokens[0].equals("do")){

str += "Error in service word do, position 1\n";

}

if(!tokens[6].equals("while")){

str += "Error in service word while, position 16\n";

}

if((tokens[1].equals(tokens[2])) || (tokens[1].equals(tokens[4])) || (tokens[1].equals(tokens[7]))){

str += "The 1st variable(positon 4) is equal to the 2nd(positions 6,11,18)\n";

}

if ((!tokens[2].equals(tokens[4])) || (!tokens[2].equals(tokens[7])) || (!tokens[4].equals(tokens[7]))){

str += "The second variable must be the same in positions 6,11 and 18\n";

}

if(!tokens[8].equals("0")){

str += "Error in position 20\n";

}

if (str.length() > 0){

if(str.equals("The entered string contains errors:\n")){

return str+= "Errors in syntax(parentheses,braces,signs and others)\n";

}

return str;

} else

{

return str += "String contains no errors\n";

}

}

public static String BtnFile(String PathToFile) throws IOException{

String strInFile;

String str;

strInFile = Main.GetStrInFile(PathToFile);

str = Main.BtnCheck(strInFile);

return str;

}

public static String GetStrInFile(String PathToFile) throws IOException{

String str;

str = new String(Files.readAllBytes(Paths.get(PathToFile)));

return str;

}

}

Код класса GraphInterface:

package com.main;

import java.io.IOException;

import javax.swing.JOptionPane;

public class GraphInterface extends javax.swing.JFrame {

public GraphInterface() {

initComponents();

}

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

MainPanel = new javax.swing.JPanel();

LabelTextField1 = new javax.swing.JLabel();

LabelTextArea1 = new javax.swing.JLabel();

LabelTextArea2 = new javax.swing.JLabel();

TextField1 = new javax.swing.JTextField();

ButtonFile = new javax.swing.JButton();

ButtonClear = new javax.swing.JButton();

ButtonExit = new javax.swing.JButton();

ButtonCheck = new javax.swing.JButton();

ScrollPane1 = new javax.swing.JScrollPane();

TextArea1 = new javax.swing.JTextArea();

ScrollPane2 = new javax.swing.JScrollPane();

TextArea2 = new javax.swing.JTextArea();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

MainPanel.setBackground(new java.awt.Color(255, 255, 255));

MainPanel.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0)));

MainPanel.setMinimumSize(new java.awt.Dimension(500, 500));

MainPanel.setPreferredSize(new java.awt.Dimension(800, 600));

LabelTextField1.setLabelFor(TextField1);

LabelTextField1.setText("Введите строку");

LabelTextArea1.setText("Введенная строка");

LabelTextArea2.setText("Результат");

TextField1.setHorizontalAlignment(javax.swing.JTextField.LEFT);

ButtonFile.setBackground(new java.awt.Color(255, 255, 255));

ButtonFile.setText("Ввести с файла");

ButtonFile.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

ButtonFileActionPerformed(evt);

}

});

ButtonClear.setBackground(new java.awt.Color(255, 255, 255));

ButtonClear.setText("Очистить");

ButtonClear.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

ButtonClearActionPerformed(evt);

}

});

ButtonExit.setBackground(new java.awt.Color(255, 255, 255));

ButtonExit.setText("Выход");

ButtonExit.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

ButtonExitActionPerformed(evt);

}

});

ButtonCheck.setBackground(new java.awt.Color(255, 255, 255));

ButtonCheck.setText("Проверить\n");

ButtonCheck.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

ButtonCheckActionPerformed(evt);

}

});

TextArea1.setEditable(false);

TextArea1.setColumns(20);

TextArea1.setRows(2);

TextArea1.setTabSize(10);

ScrollPane1.setViewportView(TextArea1);

TextArea2.setEditable(false);

TextArea2.setColumns(20);

TextArea2.setRows(5);

ScrollPane2.setViewportView(TextArea2);

javax.swing.GroupLayout MainPanelLayout = new javax.swing.GroupLayout(MainPanel);

MainPanel.setLayout(MainPanelLayout);

MainPanelLayout.setHorizontalGroup(

MainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(MainPanelLayout.createSequentialGroup()

.addGap(140, 140, 140)

.addGroup(MainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)

.addComponent(LabelTextField1)

.addComponent(TextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, 300, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(LabelTextArea1)

.addComponent(ScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 300, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(LabelTextArea2)

.addComponent(ScrollPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 585, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGroup(MainPanelLayout.createSequentialGroup()

.addComponent(ButtonCheck, javax.swing.GroupLayout.PREFERRED\_SIZE, 159, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(ButtonExit, javax.swing.GroupLayout.PREFERRED\_SIZE, 159, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(MainPanelLayout.createSequentialGroup()

.addComponent(ButtonFile)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(ButtonClear)))

.addGap(229, 229, 229))

);

MainPanelLayout.setVerticalGroup(

MainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(MainPanelLayout.createSequentialGroup()

.addGap(20, 20, 20)

.addComponent(LabelTextField1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(TextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addComponent(LabelTextArea1)

.addGap(18, 18, 18)

.addComponent(ScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(LabelTextArea2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(ScrollPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 209, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 31, Short.MAX\_VALUE)

.addGroup(MainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(ButtonFile, javax.swing.GroupLayout.PREFERRED\_SIZE, 40, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ButtonClear, javax.swing.GroupLayout.PREFERRED\_SIZE, 40, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(18, 18, 18)

.addGroup(MainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(ButtonCheck, javax.swing.GroupLayout.PREFERRED\_SIZE, 74, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ButtonExit, javax.swing.GroupLayout.PREFERRED\_SIZE, 74, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(99, 99, 99))

);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(MainPanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(MainPanel, javax.swing.GroupLayout.DEFAULT\_SIZE, 674, Short.MAX\_VALUE)

);

pack();

setLocationRelativeTo(null);

}// </editor-fold>

private void ButtonCheckActionPerformed(java.awt.event.ActionEvent evt) {

String arg;

String str;

String PathToFile;

arg = TextField1.getText();

TextArea1.setText(arg);

str = Main.BtnCheck(arg);

TextArea2.setText(str);

PathToFile = JOptionPane.showInputDialog("Enter path to file:");

try (FileWriter writer = new FileWriter(PathToFile)) {

writer.write(TextArea2.getText());

writer.flush();

writer.close();

}

catch(IOException ex){

TextArea2.setText("Can't open the file - Invalid path to file");

}

}

private void ButtonClearActionPerformed(java.awt.event.ActionEvent evt) {

TextField1.setText("");

TextArea1.setText("");

TextArea2.setText("");

}

private void ButtonExitActionPerformed(java.awt.event.ActionEvent evt) {

System.exit(0);

}

private void ButtonFileActionPerformed(java.awt.event.ActionEvent evt) {

try{

String PathToFile;

PathToFile = JOptionPane.showInputDialog("Enter path to file:");

TextArea1.setText(Main.GetStrInFile(PathToFile));

TextArea2.setText(Main.BtnFile(PathToFile));

try (FileWriter writer = new FileWriter(PathToFile)) {

writer.write(TextArea2.getText());

writer.flush();

writer.close();

}

}

catch(IOException ex){

TextArea2.setText("Invalid path to file");

}

}

public static void main(String args[]) {

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(GraphInterface.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(GraphInterface.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(GraphInterface.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(GraphInterface.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new GraphInterface().setVisible(true);

}

});

}

private javax.swing.JButton ButtonCheck;

private javax.swing.JButton ButtonClear;

private javax.swing.JButton ButtonExit;

private javax.swing.JButton ButtonFile;

private javax.swing.JLabel LabelTextArea1;

private javax.swing.JLabel LabelTextArea2;

private javax.swing.JLabel LabelTextField1;

private javax.swing.JPanel MainPanel;

private javax.swing.JScrollPane ScrollPane1;

private javax.swing.JScrollPane ScrollPane2;

private javax.swing.JTextArea TextArea1;

private javax.swing.JTextArea TextArea2;

private javax.swing.JTextField TextField1;

}