**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

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**

Горбенко Кирилл Николаевич

(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 № 6

Тема для анализа: Условные операторы;

Шаблон while (имя1 < число) { имя1 += число; имя2+=имя1; }.

**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 Горбенко Кирилл Николаевич

(подпись)

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

(фамилия, инициалы, должность) (подпись)

« 04 » сентября 2018г.

**Содержание**

[ВВЕДЕНИЕ 6](#_Toc532154326)

[1 Сравнительный анализ существующих подходов к реализации условных операторов различных языков программирования 7](#_Toc532154327)

[1.1 Общие сведения об условных операторах 7](#_Toc532154328)

[1.1.1 Условный оператор типа IF 7](#_Toc532154329)

[1.1.2 Условный оператор типа CASE 8](#_Toc532154330)

[1.2 Условные операторы в языке Pascal 9](#_Toc532154331)

[1.3 Условные операторы в С-языках 11](#_Toc532154332)

[1.4 Условные операторы в языке Java 13](#_Toc532154333)

[1.5 Условные операторы в языке Python 14](#_Toc532154334)

[1.6 Условные операторы в языке JavaScript 16](#_Toc532154335)

[1.7 Вывод 17](#_Toc532154336)

[2 Программный проект анализа синтаксической структуры оператора языка программирования 18](#_Toc532154337)

[2.1 Постановка задачи 18](#_Toc532154338)

[2.2 Техническое задание на разработку программы 19](#_Toc532154339)

[2.2.1 Введение 19](#_Toc532154340)

[2.2.2 Требования к программе 19](#_Toc532154341)

[2.2.2.1 Требования к функциональным характеристикам 19](#_Toc532154342)

[2.2.2.2 Список ошибок программы 20](#_Toc532154343)

[2.2.2.3 Требования к надежности 21](#_Toc532154344)

[2.2.3 Условия эксплуатации 22](#_Toc532154345)

[2.2.4 Технико-экономические показатели 23](#_Toc532154346)

[2.2.5 Стадии и этапы разработки 23](#_Toc532154347)

[2.3 Описание программы 25](#_Toc532154348)

[2.3.1 Общие сведения 25](#_Toc532154349)

[2.3.2 Функциональное назначение 25](#_Toc532154350)

[2.3.3 Описание логической структуры 26](#_Toc532154351)

[2.3.4 Используемые технические средства 28](#_Toc532154352)

[2.3.5 Вызов и загрузка 29](#_Toc532154353)

[2.3.6 Входные и выходные данные 29](#_Toc532154354)

[2.4 Руководство программиста 29](#_Toc532154355)

[2.4.1 Назначение и условия применения програмы 29](#_Toc532154356)

[2.4.2 Характеристика программы 30](#_Toc532154357)

[2.4.3 Обращение к программе 31](#_Toc532154358)

[2.4.4 Входные и выходные данные программы 32](#_Toc532154359)

[2.4.5 Сообщения 32](#_Toc532154360)

[2.5 Программа и методика испытаний 33](#_Toc532154361)

[2.5.1 Объект испытаний 33](#_Toc532154362)

[2.5.2 Цель испытаний 33](#_Toc532154363)

[2.5.3 Требования к программе 34](#_Toc532154364)

[2.5.4 Требования к программной документации 35](#_Toc532154365)

[2.5.5 Средства и порядок испытаний 35](#_Toc532154366)

[2.5.6 Методы испытаний 36](#_Toc532154367)

[2.5.6.1 Модульное тестирование 36](#_Toc532154368)

[2.5.6.2 Ручное тестирование 40](#_Toc532154369)

[ЗАКЛЮЧЕНИЕ 42](#_Toc532154370)

[Библиографический список 43](#_Toc532154371)

[Приложение 44](#_Toc532154372)

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.

* 1. Сравнительный анализ существующих подходов к реализации условных операторов различных языков программирования
     1. Общие сведения об условных операторах
        1. Условный оператор типа IF

На практике решение большинства задач не удается описать с помощью программ линейной структуры. При этом после проверки некоторого условия выполняется та или иная последовательность операторов, однако происходит нарушение естественного порядка выполнения операторов. Для этих целей используют управляющие операторы. Условный оператор используется для реализации разветвлений в программе, которые происходят при выполнении некоторого условия и имеет следующую структуру:

Когда выполнение основной ветви программы доходит до условного оператора, происходит вычисление логического выражения, результат которого может быть равен одному из двух значений True или False, в зависимости от полученного результата выполняется один и только один из операторов, входящих в состав условного оператора. При этом допускается использовать вместо операторов операторные блоки, которые оборачиваются в операторные скобки в соответствии с языком программирования и могут содержать один или более операторов.

Поскольку развилка может быть неполной, допускается следующая форма записи условного оператора:

В данном случае, если значение логического выражения равно False, то выполнение программы сразу вернется в ее основную ветвь.

Данное тип условных операторов применяется для ветвления лишь по двум ветвям. Для реализации ветвления по нескольким ветвям можно использовать комбинацию нескольких условных операторов или вложенные условные операторы (допускается внутри операторного блока ветвей условного оператора использовать другие условные операторы, степень вложенности которых не ограничена). После выполнения условного оператора ход программы возвращается в ее основную ветвь.

Однако, ветвление сложных разветвленных программ при необходимости реализовать более двух ветвей вычислительного процесса может стать довольно трудоемкой задачей, использование условного оператора такого типа в данных типах задач не рекомендуется.

* + - 1. Условный оператор типа CASE

Кроме условного оператора в качестве управляющей структуры довольно часто используется оператор выбора CASE. Эта структура позволяет переходить на одну из ветвей в зависимости от значения заданного выражения (селектора выбора). Ее особенность состоит в том, что выбор решения здесь осуществляется не в зависимости от истинности или ложности условия, а является вычислимым. Оператор выбора позволяет заменить несколько операторов развилки (в силу этого его ещё называют оператором множественного ветвления).

В конструкции CASE вычисляется выражение K и выбирается ветвь, значение метки которой совпадает со значением K. После выполнения выбранной ветви происходит выход из конструкции CASE. Если в последовательности нет метки со значением, равным K, то управление передается внешнему оператору, следующему за конструкцией CASE (в случае отсутствия альтернативы ELSE; если она есть, то выполняется следующий за ней оператор, а уже затем управление передается внешнему оператору).

Любая из указанных серий операторов может состоять как из единственного оператора, так и нескольких (в этом случае оператор заменяется блоком операторов).

* + 1. Условные операторы в языке Pascal

В языке Pascal условный оператор имеет вид:

Неполная форма записи условного оператора:

В логическом выражении операции имеют следующий приоритет:

1. операция логического отрицания NOT;
2. операция логического умножения AND;
3. операции логического сложения OR и XOR;
4. операции отношения: =, <>, >=, <=, >, <.

В случае, когда вместо одной операции необходимо использовать несколько операций, используются операторные скобки вида begin (начало вычислительного процесса) и end (конец вычислительного процесса), например:

IF (x > y) THEN

Begin

temp:=ABS(x);

z := temp;

End;

ELSE Begin

temp := ABS(y);

z := temp;

End;

В данном случае точка с запятой ставится в конце всего оператора, а также после каждого из операторов внутри операторного блока (между операторными скобками).

Оператор CASE в языке Pascal имеет вид:

**CASE** селектор **of**

значение1: оператор1;

значение2: оператор2;

значение3: оператор3;

...

**ELSE** операторN

**End**;

В заголовке оператора вместо логического выражения фигурирует переменная, которую называют селектором. До оператора CASE в программе необходимо обязательно этой переменной присвоить какое-либо значение. Эта переменная может иметь только перечисляемый тип (например, она не может быть вещественного типа). По ходу выполнения оператора CASE, значение переменной-селектора сравнивается с различными, описанными в нем альтернативами (метками-значениями). Как только совпадение будет найдено, то выполняется блок кода при данной метке и происходит выход в основную ветку программы. Значения-метки являются константами, которые может принимать селектор. Их тип и тип селектора должны быть совместимы по присваиванию.

Если совпадений не будет, то выполняется блок ELSE. Если блок ELSE отсутствует (он является не обязательным), то никакой блок кода в операторе CASE не выполняется.

На использование оператора выбора накладываются следующие ограничения:

* селектор должен иметь какой-либо порядковый тип;
* каждая альтернатива должна быть константой, диапазоном, списком диапазонов, но не переменной или выражением.
  + 1. Условные операторы в С-языках

Основной синтаксис C-языков (C, C++, C#) в случае условных операторов не отличается. Условный оператор в C-языках имеет вид:

Неполная форма записи условного оператора:

Как можно увидеть из вида условного оператора, в его записи пропущено ключевое слово, отвечающее за объявление оператора, соответствующего THEN языка Pascal. Блок в C-языках записывается фигурными скобками: «{оператор1; оператор2;}».

В отличие от языка Pascal, логические операции в C-языках представлены по-другому (операции отношения не приводятся, т.к. они представлены также, как и в Pascal):

* логическое отрицание: !;
* логическое умножение: & или &&;
* логическое сложение: | или ||;
* исключающее или: ^;
* проверка на равенство: ==;
* проверка на неравенство: !=.

В языках C, C++, C# кроме стандартного способа записи условного оператора существует также сокращение в виде тернарного оператора (принимает 3 параметра). Его название связано с тем, что в данных языках больше нет операторов, принимающих три параметра. Его конструкция:

В этом выражении прежде всего вычисляется логическое выражение. В случае, когда оно равно True, вместо всего оператора подставляется значение, стоящее после знака вопроса. В случае, когда логическое выражение равно False, вместо всего оператора подставляется значение, стоящее после двоеточия.

int findMaximum(int a, int b){

return (a > b) ? a : b;

}

Это выражение отличается от стандартной записи условного оператора лишь внешним представлением. Компилятор интерпретирует эти два вида условных оператора идентично.

Место оператора CASE в С и С++ занимает оператор switch, который работает похожим образом. Его вид:

switch (ЦелоеВыражение)

{

case значение1: БлокОпераций1;

case значение2: БлокОпераций2;

. . .

case значениеn: БлокОперацийn;

default: БлокОперацийПоУмолчанию;

}

В данном случае метка default соответствует метке ELSE в блоке CASE языка PASCAL. Отличием является то, что каждый блок операций должен содержать по меньшей мере один оператор передачи управления (чаще всего break). Также, на место селектора может встать переменная целого типа, а не перечисляемого, т.к. в С и С++ такого типа попросту нет.

Отдельное место в классификации условных операторов С-подобных языков занимает оператор switch языка C#. В нем реализована возможность проверять альтернативы на совпадение не только по параметру значения соответствующей переменной, но и по ее типу и другим параметрам. Это дает возможность использования в качестве селектора оператора switch переменных, объявленных динамически (не имеющих статического типа на момент компиляции). Это, в свою очередь, дает возможность добавить ветвь null, т.к. селектор может также быть и ссылочным типом. Пример оператора switch языка С#:

switch (x)

{

case int i:

Console.WriteLine ("It's an int!");

break;

case string s:

Console.WriteLine (s.Length);

break;

case bool b when b == true:

Console.WriteLine ("True");

break;

case null:

Console.WriteLine ("Nothing");

break;

}

В данном примере для проверки дополнительных условий используется ключевое слово when, после которого указываются все необходимые условия (логическое выражение).

* + 1. Условные операторы в языке Java

Конструкция логического оператора языка Java имеет вид:

В качестве оператора, как и в рассмотренных ранее языках, может служить и блок кода, «завернутый» в фигурные скобки.

Неполная форма записи условного оператора:

В данном случае, как можно увидеть из конструкции условного оператора, его запись не отличается от записи условного оператора в языках C, C++, C#. Не изменяются также и логические операторы: ==, !=, <=, >=, <, >, &&, ||, !. Кроме того, форма записи тернарного оператора также идентична таковой в С-языках.

Форма записи оператора switch совпадает форме записи оператора switch в C, C++, C#. Но отличия все же существуют. В Java селектором могут выступать такие типы как byte, short, char, int, Enum, String, Character, Byte, Short, Integer. Блок default, как и в C-языках не является обязательным.

* + 1. Условные операторы в языке Python

В языке Python условный оператор выглядит следующим образом:

В данном случае инструкции представляют собой операторный блок. В языке Python операторный блок синтаксически задается с помощью отступов перед операторами и предшествующему им двоеточию.

Неполная форма записи оператора:

В отличие от ранее рассмотренных языков, синтаксис условного оператора в Python не содержит фигурных скобок для помещения в них логического выражения. Кроме того, внутри операторного блока каждая операция должна располагаться на следующей строке. Это позволяет не использовать разделитель в виде точки с запятой для отделения операций.

Для вложенных условных операторов в языке Python предусмотрено ключевое слово elif, по смыслу означающее комбинацию ключевых слов else if. Используется эта конструкция таким образом:

a = int(input("введите число:"))

if a < 0:

print("Neg")

elif a == 0:

print("Zero")

else:

print("Pos")

Также в языке предусмотрено тернарное выражение несколько иного вида в отличие от рассмотренных ранее:

Данное выражение работает таким же образом, как и тернарный оператор языков C, C++, C# и Java. Сначала вычисляется значение логического выражения и, в зависимости от его значения все выражение заменяется значением левого или правого оператора. Если условие равно True, то выражение заменяется значением левого выражения, иначе – правого.

Список логических операторов языка Python:

* логическое отрицание: not;
* логическое умножение and;
* логическое сложение or;
* исключающее или: xor(bool, bool);
* операции отношения ==, !=, <, >, <=, >=.

Также в Python возможно соединять операции отношения, не используя логические операторы, например:

c = a == b

В этом случае вычисление операции происходит справа налево.

Пожалуй, самым значительным отличием условных операторов Python от условных операторов рассмотренных ранее языков является полное отсутствие оператора switch (case). Сценарии, которые бы покрывались этими операторами в других языках в Python реализуются банальной комбинацией if-elif-else. Но существует и более совершенный способ организации конструкции switch-case – использование словарей с ключом – селектором и значением, представляющим из себя оператор. В данном случае необходимо всего лишь предусмотреть обработку исключения KeyError, появляющегося при попытке получить из словаря значение по ключу, не содержащемуся в данном словаре. Например:

unit\_to\_multiplier = {

'mm': 10\*\*-3,

'cm': 10\*\*-2,

'dm': 10\*\*-1,

'm': 1,

'km': 10\*\*3

}

try:

mult = unit\_to\_multiplier['cm']

except KeyError as e:

raise ValueError('Undefined unit: {}'.format(e.args[0]))

Приведен самый простой способ «синтеза» оператора switch (case). Существуют еще несколько более сложных реализаций этого оператора с помощью словарей, но они опущены в данной курсовой работе в виду необходимости наличия подготовки по теме синтаксиса и устройства языка Python.

* + 1. Условные операторы в языке JavaScript

В языке JavaScript условный оператор имеет следующий вид:

Сокращенная форма:

Также, как и в рассмотренных ранее языках, в условный оператор JS можно вкладывать неограниченное количество условных операторов. Операторный блок представляет из себя фигурные скобки. Логические операции и операции отношения являются идентичными языкам C и Java.

Оператор switch имеет вид:

switch (селектор)

{

case Константа1:

Список операторов1;

break;

case Константа2:

Список операторов2;

break;

...

case Константаn:

Список операторовn;

Break;

default:

Список операторов;

break;

}

Он также имеет вид, совпадающий с C-языками и языком Java.

Тернарная операция имеет вид:

Использование:

var res = x < a ? (x + a) : (x - a);

* + 1. Вывод

В ходе курсовой работы были рассмотрены условные операторы следующих языков программирования: Pascal, C, C++, C#, Java, Python, JavaScript. Наиболее гибкой и универсальной опцией среди условных операторов оказался оператор switch языка C#. Он предоставляет возможность ветвления не только по значению селектора, но и по его типу в ходе выполнения программы и другим параметрам. Также, среди остальных условных операторов дополнительной функциональностью выделился оператор switch языка Pascal. Он предоставляет возможность устанавливать в качестве селектора не константы, а диапазоны констант, что крайне удобно при ветвлении переменной, которая может принимать многократные последовательные целые значения.

Не рекомендуется программировать сложные ветвления на языке Python в виду отсутствия встроенного оператора switch, что значительно усложняет задачу и читаемость написанного кода.

* 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 программы**

Наименование программы – «Simple template checker».

**Назначение и область применения**

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. Требования к программе
         1. Требования к функциональным характеристикам

Программа должна обеспечивать возможность выполнения следующих функций:

1. проверять любой оператор вида «while (имя1 < число) {имя1+=число; имя2=имя1;}» на соответствие правилам записи оператора. Проверяются: числа, имена, баланс скобок, знаки операций. Отсутствие и наличие пробелов программой игнорируются и ошибок, связанных с этим, не должно возникать.
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. Для ввода из файла на форме должна существовать отдельная форма, принимающая путь к файлу, с соответствующей кнопкой. 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 сообщение о правильности оператора;
7. результаты тестирования выводить на форму и во внешний файл.
   * + - 1. Список ошибок программы

В ходе работы программы могут возникнуть следующие ошибки:

1. ошибка при попытке заполнения обоих управляющих полей основного окна программы;
2. ошибка несоответствия введенного оператора шаблону;
3. ошибка преувеличения допустимого размера оператора (введенный оператор имеет длину более 80 символов);
4. ошибка несоответствия имени (идентификатора) языка программирования его синтаксису;
5. ошибка при вводе недопустимого числа или вещественного числа;
6. ошибка при несовпадении соответствующих имен (идентификаторов) в введенном операторе;
7. ошибка при несовпадении количества открывающихся скобок количеству закрывающихся;
8. ошибка в случае, когда файл, указанный в главном окне программы, не существует;
9. ошибка в случае, когда файл, указанный в главном окне программы, содержит несколько операторов while.
   * + - 1. Требования к надежности

**Требования к обеспечению надежного функционирования программы**

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением следующей совокупности организационно-технических мероприятий:

1. организацией бесперебойного питания технических средств;
2. использованием лицензионного программного обеспечения;
3. регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов.

**Время восстановления после отказа**

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем операционной системы, не должно превышать 5-ти минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

**Отказы из-за некорректных действий оператора**

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.

Отказы программы вследствие некорректных действий пользователя при взаимодействии с программой через интерфейс недопустимы.

* + - 1. Условия эксплуатации

**Климатические условия эксплуатации**

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

**Требования к квалификации и численности персонала**

Использование программы операторами (пользователями) в количестве, большем 1-го возможно, но не предполагается. Для обеспечения работы программы наличие системного администратора не требуется.

**Требования к составу и параметрам технических средств**

Для запуска программы на компьютере необходимо обеспечение следующих условий:

1. операционная система Windows XP с Service Pack 2 или выше;
2. одноядерный процессор с тактовой частотой 800 МГц или лучше;
3. 512 Мбайт ОЗУ;
4. Графический адаптер, поддерживающий DirectX 9.

**Требования к информационным структурам и методам решения**

Требования к информационным структурам не предъявляются.

**Требования к исходным кодам и языкам программирования**

Для запуска программы требуется установленный .NET Framework 3.0 или выше.

**Требования к программным средствам, используемым программой**

Требования к программным средствам не предъявляются.

**Требования к защите информации и программ**

Требования к защите информации и программ не предъявляются.

**Специальные требования**

Специальные требования к данной программе не предъявляются.

**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.
   * + 1. Технико-экономические показатели

Ориентировочная экономическая эффективность не рассчитывается. Аналогия не проводится ввиду уникальности предъявляемых требований к разработке.

* + - 1. Стадии и этапы разработки

**Стадии разработки**

Разработка должна быть проведена в три стадии:

1. разработка технического задания;
2. рабочее проектирование;
3. внедрение.

**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 внедрения должен быть выполнен этап разработки, подготовка и передача программы.

**Содержание работ по этапам**

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

1. постановка задачи;
2. определение и уточнение требований к техническим средствам;
3. определение требований к программе;
4. определение стадий, этапов и сроков разработки программы и документации на нее;
5. согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями к составу документации.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

1. разработка, согласование и утверждение методики испытаний;
2. проведение приемо-сдаточных испытаний;
3. корректировка программы и программной документации по результатам испытаний.

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию на объектах заказчика.

* + 1. Описание программы
       1. Общие сведения

Наименование программы – «Simple template checker».

Программное обеспечение, необходимое для функционирования программы:

1. операционная система Windows XP с Service Pack 2 или выше;
2. одноядерный процессор с тактовой частотой 800 МГц или лучше;
3. 512 Мбайт ОЗУ;
4. Графический адаптер, поддерживающий DirectX 9.

Программа написана на языке объектного программирования C# с использованием системы построения графических интерфейсов WPF (Windows Presentation Platform).

* + - 1. Функциональное назначение

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. Шаблон, по которому проверяется синтаксис оператора, является жестко привязанным к программе и имеет вид: «while (имя1 < число) {имя1+=число; имя2=имя1;}». Для иных целей программа не предназначена.

* + - 1. Описание логической структуры

Словесное описание алгоритма метода TryCheck(string operation, out string message), проверяющего оператор на соответствие шаблону:

1. проверить строку operation на null: если operation = null – бросить исключение;
2. проверить длину строки operation: если длина operation > 80 –бросить исключение;
3. объявить список ошибок, массив, содержащий символы-разделители для извлечения чисел и имен из оператора;
4. удалить пробелы из оператора;
5. проверить строку на общий шаблон оператора: если оператор не соответствует шаблону – поместить текст соответствующей ошибки в список ошибок;
6. извлечь из оператора имена и числа в массив;
7. проитерировать в цикле по всем элементам массива с именами и числами:
   1. если элемент начинается на цифру – проверить его в методе bool IsNumberValid(string number) – если результат проверки – false – добавить соответствующую ошибку в список;
   2. иначе, проверить элемент на шаблон языка программирования: если элемент не соответствует шаблону – добавить соответствующую ошибку в список;
8. проверить, что соответствующие имена равны: если условие не выполняется – добавить соответствующую ошибку в список;
9. проверить, что первый элемент массива с именами и числами – строка «while»: если условие не выполняется – добавить соответствующую ошибку в массив;
10. проверить, что количество открывающихся круглых скобок равно количеству закрывающихся круглых скобок; если условие не выполняется – добавить в список соответствующую ошибку;
11. проверить, что количество открывающихся фигурных скобок равно количеству закрывающихся фигурных скобок; если условие не выполняется – добавить в список соответствующую ошибку;
12. составить строку с ошибкой путем конкатенации каждого элемента списка ошибок;
13. вернуть строку с ошибками.

Используемые методы:

* 1. string.IsNullOrWhiteSpace – проверяет, что строка – null или состоит из пробелов или пустых символов;
  2. string.Length – возвращает количество символов в строке;
  3. string.Replace – заменяет символ или строку другим символом или строкой;
  4. string.Split – возвращает массив подстрок, выделенных из исходной строки с использованием разделителей;
  5. Regex.IsMatch – проверяет, соответствует ли строка данному шаблону;
  6. List<T>.Add – добавляет очередной элемент в конец списка;
  7. IEnumerable<T>.First – возвращает первый член коллекции;
  8. IEnumerable<T>.Count – возвращает число элементов коллекции либо число элементов коллекции, удовлетворяющих данному условию;
  9. int.TryParse – проверяет, возможно ли преобразование строкового десятичного представления числа к типу Int32; если возможно, возвращает также сам результат преобразования;
  10. File.ReadAllText –возвращает все содержимое файла;
  11. Regex.Matches – возвращает все соответствия шаблону из данной строки;
  12. string.IndexOf – возвращает номер первого появления символа или строки в строке;
  13. string.LastIndexOf – возвращает номер последнего появления символа или строки в строке;
  14. string.Substring – извлекает подстроку из строки, начинающуюся с данного символа и длиной в данное количество символов;
  15. StreamWriter.Write – записывает в файл информацию;
  16. MessageBox.Show – выводит на экран окно с данным сообщением.

Структура программы:

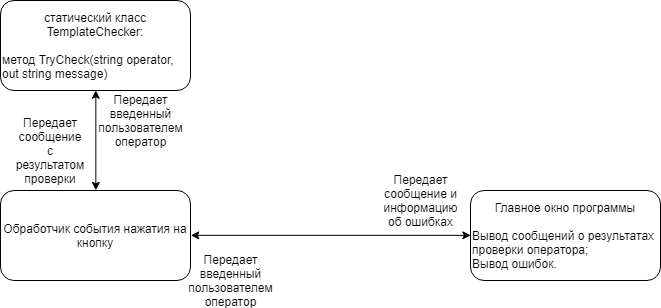


Рис. 1 – Схема структуры программы

Связи данной программы с другими программами нет.

* + - 1. Используемые технические средства

В работе программы должны использоваться вычислительные машины на базе x86 или x64 процессоров с установленной операционной системой Windows не старше Windows XP. Системные требования описаны в п. 2.3. – Описание программы.

* + - 1. Вызов и загрузка

Программа запускается путем запуска выполняемого файла WhileTemplateChecker.exe с диска компьютера или с любого другого носителя данных. Объем программы составляет 3,86 Мбайт.

* + - 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 условия применения програмы

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. Шаблон, по которому проверяется синтаксис оператора, является жестко привязанным к программе и имеет вид: «while (имя1 < число) {имя1+=число; имя2=имя1;}». Для иных целей программа не предназначена.

Функции, выполняемые программой:

1. проверка любого оператора такого вида на соответствие правилам записи оператора. Проверяются: числа, имена, баланс скобок, знаки операций…
2. проверяемый оператор вводится как строка символов длиной не более 80 символов с клавиатуры или из внешнего файла;
3. при обнаружении ошибок выводятся сведения: тип ошибки (ошибка в имени, числе и т.д.), место ошибки в операторе;
4. при отсутствии ошибок выводится сообщение о правильности оператора;
5. результаты тестирования выводятся на форму и во внешний файл.

Условия, необходимые для выполнения программы:

Программное обеспечение, необходимое для функционирования программы:

1. операционная система Windows XP с Service Pack 2 или выше;
2. одноядерный процессор с тактовой частотой 800 МГц или лучше;
3. 512 Мбайт ОЗУ;
4. Графический адаптер, поддерживающий DirectX 9.

Предполагается, что пользователем будет выступать человек без специальной подготовки.

* + - 1. Характеристика программы

Временная характеристика программы: программа работает без каких-либо задержек и моментально реагирует на действия пользователя. Возможные задержки связаны с нарушением работы аппаратной части компьютера или операционной системы.

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

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

Самовосстанавливаемость программы не предусмотрена т.к. в этом отсутствует необходимость. В случае критического завершения программы ее необходимо вручную перезапустить.

* + - 1. Обращение к программе

Вид главного окна программы:

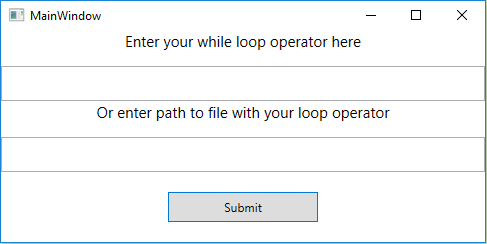
****

Рис. 2 – Вид главного окна программы

Вид сообщения программы:

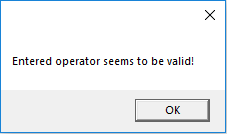


Рис. 3 – Вид окна сообщения программы

Программа запускается путем запуска выполняемого файла WhileTemplateChecker.exe с диска компьютера или с любого другого носителя данных. Объем программы составляет 3,86 Мбайт.

Для выполнения функции проверки оператора непосредственно необходимо ввести в поле под названием «Enter your while loop operator here» главного окна программы и затем нажать на кнопку «Submit».

Для выполнения функции проверки оператора, записанного в отдельный файл необходимо предварительно скопировать путь к данному файлу из проводника Windows и вставить его в поле под названием «Or enter path to file with your loop operator» и затем нажать на кнопку «Submit».

Перед всяким нажатием кнопки «Submit» необходимо убедиться в том, что в главном окне из полей «Enter your while loop operator here» и «Or enter path to file with your loop operator» заполнено только одно.

Во время вывода программой сообщения с результатом выполнения или сообщением об ошибке необходимо нажать кнопку «ОК» чтобы вернуться к главному окну программы.

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

* + - 1. Входные и выходные данные программы

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.

* + - 1. Сообщения

В программе предусмотрены следующие сообщения:

1. "Entered operator seems to be valid" –подтверждение правильности введенного оператора.
2. "Entered operation doesn’t match the required template" – введенный оператор не соответствует шаблону.
3. "Operator length should be less than 80" – длина оператора не может быть больше 80 символов.
4. "number \_\_\_ is not valid" – неправильно введен номер.
5. "Name \_\_\_ is not valid" – неправильно введено имя.
6. "Names \_\_\_, \_\_\_ and \_\_\_ should match" – несоответствие имен.
7. "Operator should be while loop" – несоответствие вила оператора.
8. "Broken parentheses balance" – нарушен баланс скобок.
9. "Broken braces balance" – нарушен баланс фигурных скобок.
10. "Chosen file should contain only one operator" – в выбранном файле содержится более одного оператора.
11. "Only one text box could be filled" – заполненным может быть только одно поле.
12. "The result has been written to a file" – результат был выведен в выбранный файл.
13. Множество ошибок, связанных с работой с файлами. Все ошибки содержат максимально содержательное сообщение, не требующее дополнительных разъяснений.
    * 1. Программа и методика испытаний
         1. Объект испытаний

Наименование программы – «Simple template checker». 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 «Simple template checker» в целом и в работе отдельные ее модулей. Критерием нормальной работы программы является полное соответствие сообщений, возвращаемых программой в ответ на ввод синтаксического оператора определенного вида, ожидаемым сообщениям.

* + - 1. Требования к программе

Программа должна обеспечивать возможность выполнения следующих функций:

1. проверять любой оператор вида «while (имя1 < число) {имя1+=число; имя2=имя1;}» на соответствие правилам записи оператора. Проверяются: числа, имена, баланс скобок, знаки операций. Отсутствие и наличие пробелов программой игнорируются и ошибок, связанных с этим, не должно возникать.
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. Для ввода из файла на форме должна существовать отдельная форма, принимающая путь к файлу, с соответствующей кнопкой. 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 сообщение о правильности оператора;
7. результаты тестирования выводить на форму и во внешний файл.

В ходе работы программы могут возникнуть следующие ошибки:

1. ошибка при попытке заполнения обоих управляющих полей основного окна программы;
2. ошибка несоответствия введенного оператора шаблону;
3. ошибка преувеличения допустимого размера оператора (введенный оператор имеет длину более 80 символов);
4. ошибка несоответствия имени (идентификатора) языка программирования его синтаксису;
5. ошибка при вводе недопустимого числа или вещественного числа;
6. ошибка при несовпадении соответствующих имен (идентификаторов) в введенном операторе;
7. ошибка при несовпадении количества открывающихся скобок количеству закрывающихся;
8. ошибка в случае, когда файл, указанный в главном окне программы, не существует;
9. ошибка в случае, когда файл, указанный в главном окне программы, содержит несколько операторов while.
   * + 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. Средства и порядок испытаний

Для проведения испытаний программы «Simple template checker», написанной на платформе .NET, будут использоваться такие методы, как ручное тестирование по тестовым примерам и модульное тестирование. Модульное тестирование будет применяться для тестирования и отладки логической части программы, непосредственно осуществляющей синтаксический анализ оператора языка программирования. Ручное тестирование будет производиться над той областью программы, которая не может быть протестирована модульно – над интерфейсом программы, то есть средствами обмена информацией с пользователем и обработки его действий.

* + - 1. Методы испытаний
         1. Модульное тестирование

В данном разделе будут последовательно приведены тесты, использовавшиеся для отладки программы. Данный вид тестирования применяется только для тестирования метода TryCheck(), выполняющего исключительно анализ синтаксической структуры вводимого оператора. Каждый тест будет содержать код теста на языке C# с использованием тестового фреймворка NUnit, краткое описание и результат выполнения теста.

1. Тест, проверяющий, что программа при вводе оператора правильного вида, введенного с пробелами, программа покажет верный результат. Тест пройден.

[Test]

public void ManagesOperatorWithWhitespaces()

{

var operation = "while ( name1 < 50 ) { name1 += 10; name2 := name1; }";

TryCheck(operation, out string message);

Assert.That(message.Contains("Entered operator seems to be valid"));

}

1. Тест, проверяющий, что программа при вводе оператора правильного вида, введенного без пробелов, программа покажет верный результат. Тест пройден.

[Test]

public void ManagesOperatorWithoutWhitespaces()

{

var operation = "while(name1<50){name1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("Entered operator seems to be valid"));

}

1. Тест, проверяющий, что при вводе оператора длиной 80 и более в программе возникает ошибка. Тест пройден.

[Test]

public void ThrowsExceptionWhenGivingTooLongOperator()

{

var operation = new string('a', 81);

Assert.Throws<ArgumentException>(() => TryCheck(operation, out string message));

}

1. Тест, проверяющий, что при вводе на место оператора пустой строки или строки, состоящей из пробелов, в программе возникнет ошибка. Тест пройден.

[Test]

public void ThrowsExceptionWhenGivingWhiteSpaceOperator()

{

var operation = new string(' ', 81);

Assert.Throws<ArgumentNullException>(() => TryCheck(operation, out string message));

}

1. Тест, проверяющий, что при вводе на место оператора null в программе возникнет ошибка. Тест пройден.

[Test]

public void ThrowsExceptionWhenGivingNullOperator()

{

Assert.Throws<ArgumentNullException>(() => TryCheck(null, out string message));

}

1. Тест, проверяющий, что при вводе в программу оператора вида, отличного от while(), программа выведет соответствующее сообщение. Тест пройден.

[Test]

public void GivesAMessageWhenNotAWhileLoop()

{

var operation = "for(name1<50){name1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Operator should be while loop"));

}

1. Тест, проверяющий, что при вводе в оператора, в котором имя2 не соответствует стандартному синтаксису языка программирования, программа выведет соответствующую ошибку. Тест пройден.

[Test]

public void GivesAMessageWhenName2IsNotValid()

{

var operation = "while(name1<50){name1+=10;nam^e2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Name nam^e2 is not valid"));

}

1. Тест, проверяющий, что при вводе в оператора, в котором имя1 в первой позиции не соответствует стандартному синтаксису языка программирования, программа выведет соответствующую ошибку. Тест пройден.

[Test]

public void GivesMessageWhenName1OnFirstPositionIsNotValid()

{

var operation = "while(na@me1<50){name1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Name na@me1 is not valid"));

}

1. Тест, проверяющий, что при вводе в оператора, в котором имя1 во второй позиции не соответствует стандартному синтаксису языка программирования, программа выведет соответствующую ошибку. Тест пройден.

[Test]

public void GivesMessageWhenName1OnSecondPositionIsNotValid()

{

var operation = "while(name1<50){na@me1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Name na@me1 is not valid"));

}

1. Тест, проверяющий, что при вводе в оператора, в котором имя1 в третьей позиции не соответствует стандартному синтаксису языка программирования, программа выведет соответствующую ошибку. Тест пройден.

[Test]

public void GivesMessageWhenName1OnThirdPositionIsNotValid()

{

var operation = "while(name1<50){name1+=10;name2:=na@me1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Name na@me1 is not valid"));

}

1. Тест, проверяющий, что при вводе оператора, в котором соответствующие имена не равны, в программе возникнет ошибка. Тест пройден.

[Test]

public void GivesAMessageWhenNames1DoesntMatch()

{

var operation = "while(name1<50){notname1+=10;name2:=notname1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Names name1, notname1 and notname1 should match"));

}

1. Тест, проверяющий, что при вводе оператора, в котором нарушен баланс скобок, в программе возникнет ошибка. Тест пройден.

[Test]

public void GivesAMessageWhenBrokenParenthesesBalance()

{

var operation = "while((name1<50){name1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Broken parentheses balance"));

}

1. Тест, проверяющий, что при вводе оператора, в котором нарушен баланс фигурных скобок, в программе возникнет ошибка. Тест пройден.

[Test]

public void GivesAMessageWhenBrokenBracesBalance()

{

var operation = "while(name1<50){{{name1+=10;name2:=name1;}}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

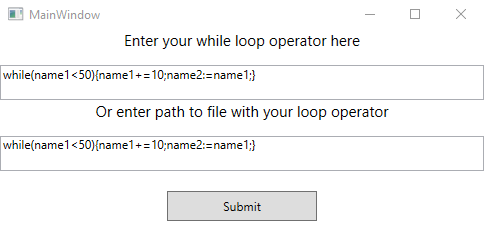
Assert.That(message.Contains("Broken braces balance"));

}

* + - * 1. Ручное тестирование

В данном разделе будут выписаны тестовые примеры, использовавшиеся при тестировании программы. Каждый тест включает в себя ожидаемый и действительный результат.

1. Тестовый пример, проверяющий, что при заполнении двух полей в главном окне программы, произойдет ошибка.



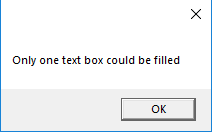
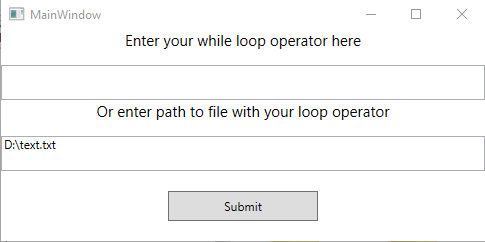


Рис. 4 – Тестовый пример № 1

1. Тестовый пример, проверяющий, что при указании в главном окне файла, содержащего несколько операторов, произойдет ошибка.



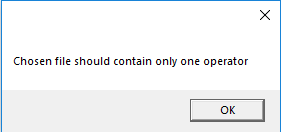
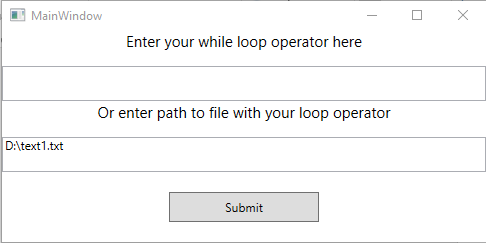


Рис. 5 – Тестовый пример № 2

1. Тестовый пример, проверяющий, что при вводе в главное окно программы пути к несуществующему файлу, произойдет ошибка.



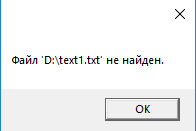


Рис. 6 – Тестовый пример № 3

1. ЗАКЛЮЧЕНИЕ

В ходе курсовой работы был реализован проект программной реализации синтаксического анализатора языка программирования по заданному шаблону. Спроектированная программа под названием «Simple template checker» имеет графический интерфейс в виде окна операционной системы Windows с двумя текстовыми полями и одной кнопкой для обмена информацией с пользователем.

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

В программе реализована возможность вводить оператор как в главное окно программы, так и из внешнего файла. Программные модули были отлажены средствами модульного и ручного тестирования: для модульного тестирования были разработаны 13 тестовых методов, для ручного тестирования – 3 тестовых примера.

На реализованный проект была оформлена следующая документация:

1. техническое задание;
2. описание программы;
3. руководство программиста;
4. 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.

1. Библиографический список
2. Павловская Т.А. C/C++. Программирование на языке высокого уровня. Учебник для ВУЗов. – СПб.: Питер, 2005. – 464 с.
3. Глинский Я.Н. Turbo Pascal 7.0 и Delphi. Учебное пособие. – СПб.: ДиаСофт, 2003. – 208 с.
4. Albahari John, Albahari Ben. C# 7.0 in a Nutshell – The definitive reference. – O’Reilly Media, Inc., 2017. – 1071 c.
5. Шилдт, Герберт Java 8. Руководство для начинающих / Герберт Шилдт. - М.: Вильямс, 2015. - 720 c.
6. Гарнаев, Андрей WEB-программирование на Java и JavaScript / Андрей Гарнаев , Сергей Гарнаев. - Москва: СПб. [и др.] : Питер, 2017. - 718 c.
7. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.

Приложение

Код класса Template Checker:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

namespace WhileSyntaxChecker

{

public static class TemplateChecker

{

static string pattern = @"[A-Za-z\_][\w\_]\*\([A-Za-z\_][\w\_]\*(<=|>=|==|>|<)[0-9].\*\)\{[A-Za-z\_][\w\_]\*\+=[0-9].\*;[A-Za-z\_][\w\_]\*:=[A-Za-z\_][\w\_]\*;\}";

public static bool TryCheck(string operation, out string message)

{

if (string.IsNullOrWhiteSpace(operation)) throw new ArgumentNullException(nameof(operation));

if (operation.Length > 80) throw new ArgumentException("Operator length should be less than 80");

var errors = new List<string>();

operation = operation.Replace(" ", string.Empty);

var namesAndNumbers = operation.Split(new[] { " ", "(", ")", "{", "}", ";", "<", ">", "=", "+", "-", "/", "\*", ":" },

StringSplitOptions.RemoveEmptyEntries);

if (!Regex.IsMatch(operation, pattern))

errors.Add("Entered operation doesn't match the required template");

foreach (var identifier in namesAndNumbers)

{

if (Regex.IsMatch(identifier, @"^[0-9].\*$"))

{

if (!IsNumberValid(identifier))

errors.Add($"number {identifier} is not valid");

}

else if (!Regex.IsMatch(identifier, @"^[A-Za-z\_][A-Za-z0-9\_]\*$"))

errors.Add($"Name {identifier} is not valid");

}

if (!AreNamesMatch(namesAndNumbers))

errors.Add($"Names {namesAndNumbers[1]}, {namesAndNumbers[3]} and {namesAndNumbers[6]} should match");

if (namesAndNumbers.First() != "while")

errors.Add("Operator should be while loop");

if (operation.Count(x => x == '(') != operation.Count(x => x == ')'))

errors.Add("Broken parentheses balance");

if (operation.Count(x => x == '{') != operation.Count(x => x == '}'))

errors.Add("Broken braces balance");

message = GetMessage(errors);

return errors.Count() == 0;

bool AreNamesMatch(string[] names)

{

return !(names[1] != names[3] ||

names[3] != names[6] ||

names[1] != names[6]);

}

bool IsNumberValid(string number)

{

return int.TryParse(number, out \_);

}

string GetMessage(List<string> partsOfMessage)

{

if (errors.Count() == 0) return "Entered operator seems to be valid!";

var line = "There has been some issues:\n";

foreach (var part in partsOfMessage) line += $"{part};\n";

return line;

}

}

}

}

Код класса TemplateCheckerTests

using System;

using static WhileSyntaxChecker.TemplateChecker;

using NUnit.Framework;

namespace WhileSyntaxChecker

{

class TemplateCheckerTests

{

[Test]

public void ManagesOperatorWithWhitespaces()

{

var operation = "while ( name1 < 50 ) { name1 += 10; name2 := name1; }";

TryCheck(operation, out string message);

Assert.That(message.Contains("Entered operator seems to be valid"));

}

[Test]

public void ManagesOperatorWithoutWhitespaces()

{

var operation = "while(name1<50){name1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("Entered operator seems to be valid"));

}

[Test]

public void ThrowsExceptionWhenGivingTooLongOperator()

{

var operation = new string('a', 81);

Assert.Throws<ArgumentException>(() => TryCheck(operation, out string message));

}

[Test]

public void ThrowsExceptionWhenGivingWhiteSpaceOperator()

{

var operation = new string(' ', 81);

Assert.Throws<ArgumentNullException>(() => TryCheck(operation, out string message));

}

[Test]

public void ThrowsExceptionWhenGivingNullOperator()

{

Assert.Throws<ArgumentNullException>(() => TryCheck(null, out string message));

}

[Test]

public void GivesAMessageWhenNotAWhileLoop()

{

var operation = "for(name1<50){name1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Operator should be while loop"));

}

[Test]

public void GivesAMessageWhenName2IsNotValid()

{

var operation = "while(name1<50){name1+=10;nam^e2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Name nam^e2 is not valid"));

}

[Test]

public void GivesMessageWhenName1OnFirstPositionIsNotValid()

{

var operation = "while(na@me1<50){name1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Name na@me1 is not valid"));

}

[Test]

public void GivesMessageWhenName1OnSecondPositionIsNotValid()

{

var operation = "while(name1<50){na@me1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Name na@me1 is not valid"));

}

[Test]

public void GivesMessageWhenName1OnThirdPositionIsNotValid()

{

var operation = "while(name1<50){name1+=10;name2:=na@me1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Name na@me1 is not valid"));

}

[Test]

public void GivesAMessageWhenNames1DoesntMatch()

{

var operation = "while(name1<50){notname1+=10;name2:=notname1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Names name1, notname1 and notname1 should match"));

}

[Test]

public void GivesAMessageWhenBrokenParenthesesBalance()

{

var operation = "while((name1<50){name1+=10;name2:=name1;}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Broken parentheses balance"));

}

[Test]

public void GivesAMessageWhenBrokenBracesBalance()

{

var operation = "while(name1<50){{{name1+=10;name2:=name1;}}";

TryCheck(operation, out string message);

Assert.That(message.Contains("There has been some issues"));

Assert.That(message.Contains("Broken braces balance"));

}

}

}

Разметка графического интерфейса главного окна программы:

<Window x:Class="WhileSyntaxChecker.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WhileSyntaxChecker"

mc:Ignorable="d"

Title="MainWindow" Height="250" Width="500">

<Grid>

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<Grid Grid.Row="0">

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<TextBlock Grid.Row="0" HorizontalAlignment="Center" VerticalAlignment="Top" FontSize="15">Enter your while loop operator here</TextBlock>

<TextBox Grid.Row="1" Name="textBox" HorizontalAlignment="Stretch"></TextBox>

</Grid>

<Grid Grid.Row="1">

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<TextBlock Grid.Row="0" HorizontalAlignment="Center" VerticalAlignment="Top" FontSize="15">Or enter path to file with your loop operator</TextBlock>

<TextBox Grid.Row="1" Name="pathBox" HorizontalAlignment="Stretch" ></TextBox>

</Grid>

<Button Grid.Row="2" IsDefault="True" Click="Button\_Click" Height="30" Width="150" VerticalAlignment="Center" Content="Submit"/>

</Grid>

</Window>

Код класса MainWindow:

using System;

using System.Windows;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;

using static WhileSyntaxChecker.TemplateChecker;

namespace WhileSyntaxChecker

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

try

{

if (!(string.IsNullOrWhiteSpace(textBox.Text) ^ string.IsNullOrWhiteSpace(pathBox.Text)))

throw new InvalidOperationException("Only one text box could be filled");

if (!string.IsNullOrWhiteSpace(textBox.Text))

{

TryCheck(textBox.Text, out string message);

MessageBox.Show(message);

}

if (!string.IsNullOrWhiteSpace(pathBox.Text))

{

var allText = File.ReadAllText(pathBox.Text);

if (Regex.Matches(allText, "while").Count > 1)

throw new InvalidOperationException("Chosen file should contain only one operator");

var start = allText.IndexOf("while");

var finish = allText.LastIndexOf("}");

var operation = allText.Substring(start, finish - start + "while".Length);

TryCheck(operation, out string message);

using (var writer = new StreamWriter(pathBox.Text, append: true))

{

writer.Write(message);

}

MessageBox.Show("The result has been written to a file");

}

}

catch (InvalidOperationException exception) when (exception.Message == "Only one text box could be filled")

{

MessageBox.Show(exception.Message);

textBox.Text = string.Empty;

pathBox.Text = string.Empty;

}

catch (Exception exception)

{

MessageBox.Show(exception.Message);

}

}

}

}