**КУРСОВАЯ РАБОТА**

по дисциплине «Теория автоматов и формальных языков»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выполнил |  |  |  |  |
| студент гр. 33504/4 |  |  |  | Череповский Д.К. |
|  |  |  |  |  |
| Руководитель |  |  |  | Тышкевич А.И. |

**Оглавление**

[**Постановка задачи** 3](#_Toc515186957)

[**Анализ компилятора Milan** 3](#_Toc515186958)

[**Описание языка** 3](#_Toc515186959)

[**Описание компилятора** 4](#_Toc515186960)

[**Компилятор Milan** 4](#_Toc515186961)

[**Ход работы** 5](#_Toc515186962)

[**Грамматика Milan+** 5](#_Toc515186963)

[**Модификации компилятора** 6](#_Toc515186964)

[**Пример работы** 7](#_Toc515186965)

[**Результаты** 9](#_Toc515186966)

[**Приложение** 10](#_Toc515186967)

# **Постановка задачи**

На основе имеющегося исходного кода компилятора MiLan построить компилятор с расширенными возможностями (далее MiLan+). В частности, предлагается реализовать тернарный оператор *<логическое выражение>? <выражение>:<выражение>*.

В общем-то, конкретные требования к тернарному оператору не выдвинуты, поэтому будем ориентироваться на "классические" языки программирования. Сформулируем требования к арифметическому "if" на основе языка Си:

• Опираясь на таблицу приоритетов операций языка Си, построим таблицу приоритетов операций Milan+ с учётом тернарной операции. В приложении представлена таблица приоритета операций языка Си и таблица приоритетов языка Milan+.

• Язык Си поддерживает вложенность тернарных операций. Поставим аналогичное требование к компилятору MiLan+.

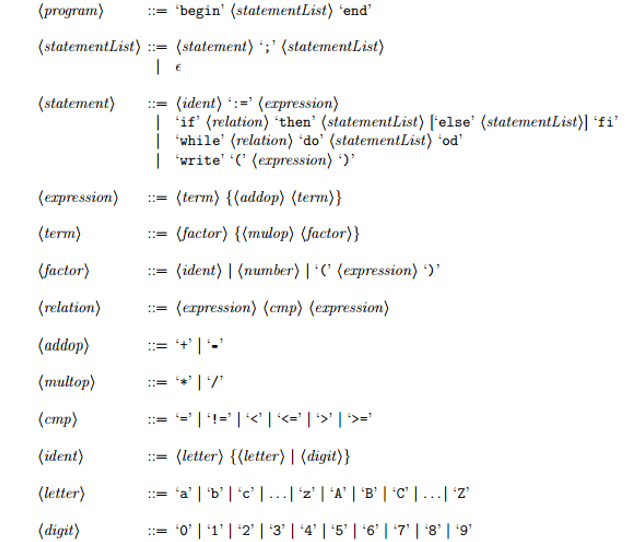
• Получаемый в ходе компиляции код для виртуальной машины MiLan должен оперировать тем же набором команд.

• Программы, написанные на MiLan должны корректно компилироваться с помощью MiLan+.

# **Анализ компилятора Milan**

## **Описание языка**

Студентам предлагается ознакомиться с методическим пособием, которое описывает возможности языка MiLan. Грамматика языка в расширенной форме Бэкуса-Наура приведена ниже:



Вкратце можно сказать, что программа на языке MiLan представляет собой последовательность операторов, заключённых между ключевыми словами ′𝑏𝑒𝑔𝑖𝑛′ и ′𝑒𝑛𝑑′. Операторы отделяются друг от друга символом *′; ′*. Таким образом признаком конца списка операторов является отсутствие точки запятой после последнего оператора. Согласно описанию, язык поддерживает неполный набор арифметических операций (отсутствуют унарные операции сложения и вычитания) и ограниченную версию логических выражений (поддерживаются только внутри оператора ′𝑖𝑓′, не поддерживается арифметика и вложенность операций сравнения). В соответствии с представленной грамматикой логические операции имеют равный приоритет (что отличает компилятор MiLan от компиляторов Си и Си++ - необходимо учесть это различие в соответствии с сформулированными требованиями). Идентификаторы представлены непрерывной последовательностью цифр и символов, начинающейся с символа. Язык поддерживает условный оператор и оператор цикла с предусловием. Язык не обладает полнотой по Тьюрингу[[1]](#footnote-1), так как построен на основе контекстно-свободной грамматики[[2]](#footnote-2).

## **Описание компилятора**

Методические пособие также содержит описание архитектуры компилятора MiLan. По задумке автора компилятор представлен совокупностью трёх компонент: лексический анализатор, синтаксически анализатор и генератор кода для виртуальной машины MiLan, - объединённых неким драйвером, управляющего процессом трансляции программы в язык команд виртуальной машины.

# **Компилятор Milan**

В ходе анализа компилятора MiLan можно сформулировать следующие тезисы относительно его программной архитектуры:

• Класс Scanner занимается лексическим анализом и, как следствие, формированием потока токенов[[3]](#footnote-3), отражающего содержимое файла с исходным кодом на языке MiLan.

• Класс CodeGen отвечает за формирование отдельных команд, которые способна распознать виртуальная машина MiLan. Класс не отвечает за корректность составленной последовательности команд

• Класс Parser реализован посредством классов Scanner и CodeGen. Класс выполняет функции синтаксического анализатора и отвечает за корректность генерируемой последовательности команд. Как следствие синтаксического анализа, выявляются синтаксические ошибки.

• Функция main обеспечивает минимальный интерфейс командной строки - входным параметром является имя файла с исходным кодом. main отвечает за запуск класса Parser.

# **Ход работы**

## **Грамматика Milan+**

Согласно сформулированным требованиям, основные изменения компилятор MiLan+ претерпит в классе Scanner и Parser. Необходимо выполнить следующее:

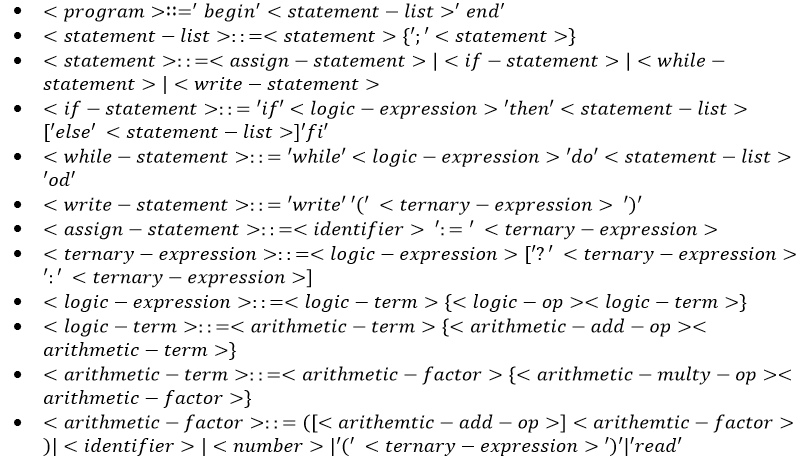
• Обеспечить корректную идентификацию токенов тернарного оператора: ′? ′ и ′: ′ (класс Scanner)

• Построить корректную продукцию для тернарного оператора

• Реализовать новую грамматику, при которой поддерживается обратная совместимость языка MiLan и MiLan+. Кроме того, несоответствие заявленной грамматики языка MiLan и конкретной реализации компилятора ставит перед нами следующие задачи:

• Обеспечить поддержку унарных операций (′+′и ′ − ′). Требование выдвигается на основе наличия в реализации по крайней мере одной унарной операции. Расширение функционала требует минимальных усилий.

• Обеспечить поддержку скобочных конструкций для логических выражений. Требования выдвигается с опорой на "классические" языки программирования. С учётом дополненных требований, ниже представлена грамматика языка MiLan+ в расширенной форме Бэкуса-Наура[[4]](#footnote-4):



## **Модификации компилятора**

Для распознавания элементов тернарного оператора выполним следующие действия:

• Добавим новый тип лексем в общий список.

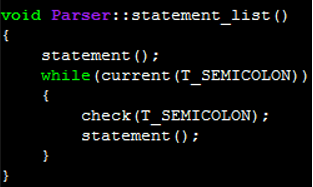
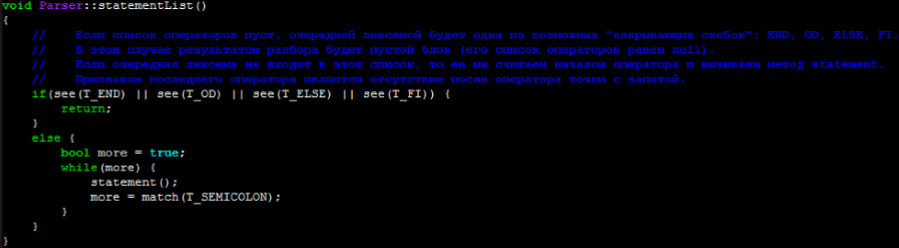
• Модифицируем функцию идентификации операций: добавим идентификацию ′? ′ и внесём изменения в идентификацию токена ′ ≔ ′ с учётом необходимости идентификации ′: ′.

Из-за неоправданных сложностей в реализации синтаксического анализатора в языке MiLan и его несоответствие концепции "рекурсивного спуска" задача по модификации имеющегося кода чрезвычайно трудоёмкая. Выдвинем к синтаксическому анализатору MiLan+ следующие требования:

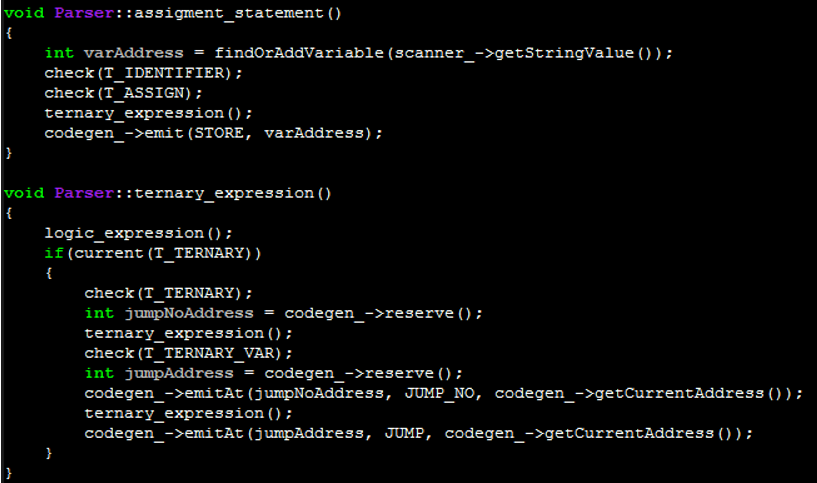
• Исходный код синтаксического анализатора должен быть удобочитаемым и соответствовать заявленной грамматике языка MiLan+

• Реализация должна соответствовать концепции метода рекурсивного спуска и сентенциальной диаграмме

На основе сформулированных требований был построен новый синтаксический анализатор MiLan+: более компактный, простой и лаконичный по сравнению с синтаксическим анализатором языка MiLan. В частности, реализация продукции <𝑠𝑡𝑎𝑡𝑒𝑚𝑒𝑛𝑡𝐿𝑖𝑠𝑡>, не соответствующая грамматике и сентенциальной диаграмме, в языке MiLan+ реализована следующим образом (продукция <𝑠𝑡𝑎𝑡𝑒𝑚𝑒𝑛𝑡 − 𝑙𝑖𝑠𝑡>) ... На первом рисунке код компилятора MiLan, на втором - MiLan+:



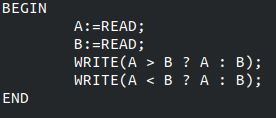
Что касается непосредственно продукций тернарного оператора, то реализация в языке MiLan+ выглядит следующим образом (рядом представлена реализация продукции оператора присваивания):



Проверка осуществлялась на основе примеров, представленных в описании к компилятору MiLan. Генерируемые последовательности команд компилятором MiLan и MiLan+ эквиваленты.

# **Пример работы**

1. Тест на проверку корректной работы термального оператора



Сгенерированный код для стековой машины

0: INPUT

1: STORE 0

2: INPUT

3: STORE 1

4: LOAD 0

5: LOAD 1

6: COMPARE 3

7: JUMP\_NO 10

8: LOAD 0

9: JUMP 11

10: LOAD 1

11: PRINT

12: LOAD 0

13: LOAD 1

14: COMPARE 2

15: JUMP\_NO 18

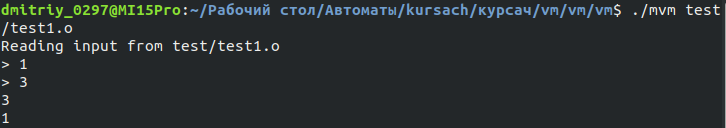
16: LOAD 0

17: JUMP 19

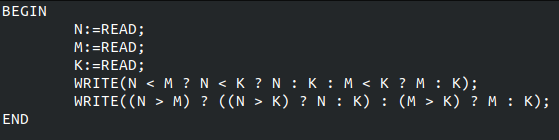
18: LOAD 1

19: PRINT

20: STOP



1. Пример программы со вложенностью тернарных операторов



Сгенерированный код для стековой машины

0: INPUT

1: STORE 0

2: INPUT

3: STORE 1

4: INPUT

5: STORE 2

6: LOAD 0

7: LOAD 1

8: COMPARE 2

9: JUMP\_NO 18

10: LOAD 0

11: LOAD 2

12: COMPARE 2

13: JUMP\_NO 16

14: LOAD 0

15: JUMP 17

16: LOAD 2

17: JUMP 25

18: LOAD 1

19: LOAD 2

20: COMPARE 2

21: JUMP\_NO 24

22: LOAD 1

23: JUMP 25

24: LOAD 2

25: PRINT

26: LOAD 0

27: LOAD 1

28: COMPARE 3

29: JUMP\_NO 38

30: LOAD 0

31: LOAD 2

32: COMPARE 3

33: JUMP\_NO 36

34: LOAD 0

35: JUMP 37

36: LOAD 2

37: JUMP 45

38: LOAD 1

39: LOAD 2

40: COMPARE 3

41: JUMP\_NO 44

42: LOAD 1

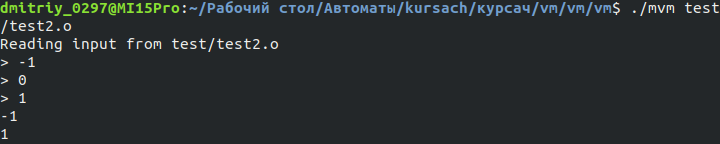
43: JUMP 45

44: LOAD 2

45: PRINT

46: STOP

Программа выводит минимальное и максимальное значение среди трёх чисел:



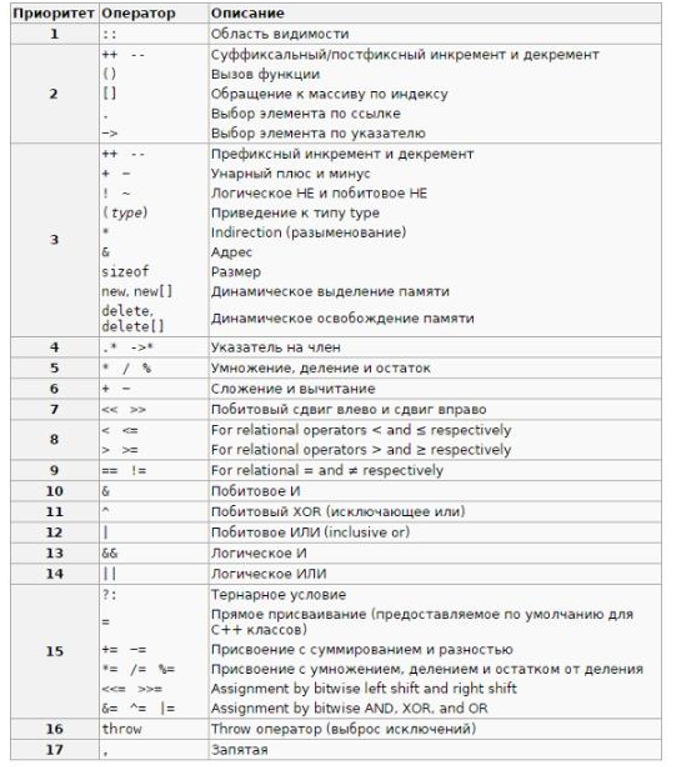
# **Результаты**

В ходе работы на основе задания был сформулирован набор требований для модифицированной версии компилятора MiLan - MiLan+. На основе требований, в частности требования об обратной совместимости, была построена новая грамматика, представленная в отчёте к курсовому проекту в расширенной форме Бэкуса-Наура.

В процессе построения компилятора MiLan+ кардинально модифицирован синтаксический анализатор: его код интуитивно понятный и полностью соответствует заявленной грамматике. Результатом модификации стал расширенный функционал: полная поддержка базовых арифметических операций, "арифметика" логических выражений и сложные логические условия (без операторов && и ||, хотя расширение MiLan+ позволяет легко это сделать), "скобочность" логических выражений, тернарный оператор.

# **Приложение**

Приоритеты операций языка С++:



Приоритет операций языка MiLan+:



1. Полными по Тьюрингу являются такие языки, как C++, Pascal, Haskell и т.д. [↑](#footnote-ref-1)
2. Левые части всех продукций представлены одиночными нетерминалами [↑](#footnote-ref-2)
3. Токен - лексема [↑](#footnote-ref-3)
4. Нетерминальные символы <𝑖𝑑𝑒𝑛𝑡𝑖𝑓𝑖𝑒𝑟> и <𝑛𝑢𝑚𝑏𝑒𝑟> имеют аналогичные грамматике MiLan продукции [↑](#footnote-ref-4)