Министерство высшего образования и науки Российской Федерации

#### ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

##### Кафедра «Математическое обеспечение и применение ЭВМ»

Пояснительная записка

к курсовому проектированию по дисциплине

«Логическое и функциональное программирование»

на тему: «Преобразование формы записи выражения»

|  |  |
| --- | --- |
| Автор работы: | Подлеснов Я.Е. |
| Направление бакалавриата | 09.03.04 («Программная инженерия») |
| Обозначение курсовой работы | ПГУ 09.03.04 – 05КР161.22 ПЗ |
| Группа | 16ВП1 |
| Руководитель работы | доцент кафедры «МО и ПЭВМ», к.т.н. Михалев А.Г. |
| Работа защищена «\_\_» \_\_\_\_\_\_ 2018 г. | Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Пенза 2018 г.

**Министерство высшего образования и науки Российской Федерации**

**Пензенский государственный университет**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Кафедра «Математическое обеспечение и применение ЭВМ»**

**«УТВЕРЖДАЮ»**

**Зав. кафедрой \_\_\_\_\_\_\_\_**

**«\_\_\_» \_\_\_\_\_\_\_\_\_ 2018 г.**

**ЗАДАНИЕ**

**на курсовое проектирование по курсу**

«Логическое и функциональное программирование»

**Студенту** \_\_\_\_\_\_\_\_Подлеснову Я.Е.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Группа** \_\_\_\_16ВП1\_\_\_\_

**Тема проекта** «Преобразование формы записи выражения»

**Исходные данные (технические требования) на проектирование**

Необходимо разработать программу, осуществляющую проверку правильности записи исходного выражения (с простейшей диагностикой ошибок) и преобразование его из инфиксной формы в постфиксную.

Исходные данные вводить из клавиатуры или из файла, результаты выводить на экран монитора или на принтер. Все стадии работы и диалога с пользователем (ввод, промежуточная обработка, вывод) отображать в отдельных окнах и снабжать необходимыми пояснениями.

Используемый язык – *SWI Prolog*

Используемыетехнические средства *- ОС Windows 10*

**Объём работы по курсу**

**1. Расчетная часть**

1) Разработка структуры данных

2) Краткие спецификации составных частей программы (число аргументов, предикатов, их значения, связь с другими предикатами)

3) Основные используемые приемы и методы разработки

Оформление теоретической части. Тема – «Miranda – язык функционального программирования»

**2. Графическая часть**

Отсутствует

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**3. Экспериментальная часть**

Кодирование программы на языке Пролог, ее отладка, демонстрация ее работоспособности.

**Срок выполнения проекта по разделам**

**1.** Расчетная часть **к** 27.11 **2018**

**2.** Экспериментальная часть **к** 11.12 **2018 г.**

**3.** Оформление пояснительной записки **к** 25.12 **2018 г.**

**4. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_к**  **2018 г.**

**5. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_к**   **2018г.**

**6. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_к** **2018 г.**

**7. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_к \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.**

**8. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_к \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г.**

**Дата выдачи задания** "9" октября **2018 г.**

**Дата защиты проекта** " **"**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **2018 г.**

**Руководитель** \_Михалев А.Г.

**Задание получил "** 9  **"** октября **2018 г.**

**Студент** \_\_\_\_\_\_Подлеснов Я.Е.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Реферат Пояснительная записка содержит 36 листов, 6 рисунков, 1 таблица, 6 использованных источников, 2 приложения.  ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ, ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ, MIRANDA, PROLOG, ФОРМА ЗАПИСИ ВЫРАЖЕНИЯ  Целью курсовой работы является разработка программы, выполняющей преобразование формы записи выражение из инфиксной формы в постфиксную и обзор языка функционального программирования Miranda.  Разработка проводилась на языке программирования SWI Prolog.  Разработка проведена с использованием операционной системы Windows 10. | | | | | | | | | | |
|  |  |  |  |  | ПГУ 09.03.04 - 05КП161.22 ПЗ | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № докум. | Подп. | Дата |
| Разраб. | | Подлеснов Я.Е. |  |  | «Преобразование формы записи выражения»  Пояснительная записка. | Лист. | | | Лист | Листов |
| Пров. | | Михалев А.Г. |  |  |  |  |  | 4 | 36 |
|  | |  |  |  | Группа 16ВП1 | | | | |
| Н. контр. | |  |  |  |
| Утв. | |  |  |  |

Содержание

[Реферат 4](#_Toc532666243)

[Введение 6](#_Toc532666244)

[1. Теоретическая часть 7](#_Toc532666245)

[1.1. Общие сведения о языке Miranda 7](#_Toc532666246)

[1.2. Описание типов 9](#_Toc532666247)

[1.3. Работа со списками 11](#_Toc532666248)

[1.4. Синтаксис языка 13](#_Toc532666249)

[2. Практическая часть 16](#_Toc532666250)

[2.1. Постановка задачи 16](#_Toc532666251)

[2.2. Описание применения программы 17](#_Toc532666252)

[2.2.1. Общии сведения и функциональное назначение 17](#_Toc532666253)

[2.2.2. Проектирование программы 19](#_Toc532666254)

[2.2.3. Схема логической программы 21](#_Toc532666255)

[2.2.4. Входные данные 25](#_Toc532666256)

[2.2.5. Выходные данные 26](#_Toc532666257)

[2.2.6. Используемые технические средства 27](#_Toc532666258)

[2.2.7. Тестирование 28](#_Toc532666259)

[Заключение 29](#_Toc532666260)

[Список используемой литературы 30](#_Toc532666261)

[ПРИЛОЖЕНИЕ A 31](#_Toc532666262)

[ПРИЛОЖЕНИЕ B 35](#_Toc532666263)

# Введение

Математические выражения традиционно за записывают в инфиксной форме, то есть знак операции находится между операндами. Не смотря на свою распространенность данная форма записи имеет свои недостатки, а именно: неопределенные явно приоритеты операций и неудобность вычисления программным способом. В связи с этим были разработаны другие формы записи выражения.

В данной работе будет рассмотрена обратная польская запись, иначе называемая постфиксной. Она имеет ряд преимуществ по сравнению с инфиксной. Во-первых, любая формула в постфиксной записи может быть выражена без скобок. Во-вторых, она [удобна для вычисления формул в машинах со стеками](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C#.D0.92.D1.8B.D1.87.D0.B8.D1.81.D0.BB.D0.B5.D0.BD.D0.B8.D1.8F_.D0.BD.D0.B0_.D1.81.D1.82.D0.B5.D0.BA.D0.B5). В-третьих, инфиксные операторы имеют приоритеты, которые произвольны и нежелательны.

Таким образом, преобразование формы записи выражения является актуальной на сегодняшний день задачей.

В рамках данной работы будет рассмотрена задача преобразования записи выражения из инфиксной записи в постфиксную на диалекте swi prolog языка пролог.

В теоретической части данной работы будет рассмотрен язык функционального программирования Miranda.

# Теоретическая часть

## Общие сведения о языке Miranda

Miranda – [функциональный язык](http://www.wikiwand.com/ru/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) программирования, созданный в [1985 году](http://www.wikiwand.com/ru/1985_%D0%B3%D0%BE%D0%B4) [Дэвидом Тёрнером](http://www.wikiwand.com/ru/%D0%A2%D1%91%D1%80%D0%BD%D0%B5%D1%80,_%D0%94%D1%8D%D0%B2%D0%B8%D0%B4) в качестве стандартного функционального языка. Имеет строгую полиморфную систему типов, поддерживает типы данных пользователя. Как и язык [ML](http://www.wikiwand.com/ru/ML), преподаётся во многих университетах.6

Miranda является чисто функциональным языком – в нем отсутствуют какие-либо побочные эффекты функций. Программа (обычно ее называют «сценарием») – это набор уравнений, определяющих различные функции и структуры данных, которые необходимы в вычислениях. Порядок, в котором даются уравнения, в целом не имеет значения. Например, нет никакого обязательства для определения субъекта предшествовать его первому использованию.

Язык обладает ленивой семантикой, следовательно, не гарантирует последовательность выполнения выражений, так как вычисления выполняются только при необходимости.

Встроенные типы языка:

* числа (целые неограниченной размерности и числа с плавающей точкой двойной точности)
* символы
* списки
* кортежи
* функции.

Пользовательские типы данных произвольной структуры определяются при помощи уравнений типов (алгебраические типы данных).

Miranda – первый коммерчески поддерживаемый чисто функциональный язык программирования.

Первый интерпретатор Miranda для Unix-подобных ОС (написанный на [C](http://progopedia.ru/language/c/)) был выпущен в 1985 году.

Появившийся позже язык [Haskell](http://progopedia.ru/language/haskell/) во многом похож на Miranda.

Название языка происходит от имени героини пьесы «[Буря](http://www.wikiwand.com/ru/%D0%91%D1%83%D1%80%D1%8F_(%D0%BF%D1%8C%D0%B5%D1%81%D0%B0))» [Уильяма Шекспира](http://www.wikiwand.com/ru/%D0%A3%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC_%D0%A8%D0%B5%D0%BA%D1%81%D0%BF%D0%B8%D1%80), Миранды. В ней она произносит фразу «О дивный новый мир!», что, по словам создателей, должно принести «дивный новый мир в функциональное программирование». Также героиня изображена на логотипе языка.6

## Описание типов

Не существует обязательных объявлений типов, хотя (см. Далее) язык строго типизирован

Не смотря на то язык программирования Miranda не требует обязательных объявлений типов, он строго типизирован, – то есть каждое выражение и каждая переменная имеют тип, который может быть выведен путем статического анализа текста программы.

Примитивные типы:

1. num
2. bool
3. char

Значения типа «num» включают в себя как целые числа, так и числа с плавающей запятой, например, 23, 0, -17, 1.26e11.

Они хранятся с использованием разных внутренних представлений, но могут свободно смешиваться в вычислениях и оба имеют тип «num» для проверки типов. При необходимости происходит автоматическое преобразование из целого числа в число с плавающей запятой (но не в обратном направлении). Числа с плавающей точкой удерживаются с двойной точностью, целые числа с неограниченной точностью.1

Значения типа «bool» могут иметь два значения: True и False.

Значения типа «char» являются символами ascii, например, 'a', '%', '1'.

Наиболее часто используемая структура данных - это список, который в языке Miranda объявляется перечислением элементов одного типа через запятую в квадратных скобках, например:

week\_days = ["Пн", "Вт", "Среда", "Чет", "Пт"]

При этом, список символов ['h', 'e', ​​'l', 'l', 'o'] является строкой и равнозначен записи "hello".

Следующим типом данных является кортеж. Его отличие от списка в том, что он может содержать элементы разных типов. Объявление кортежа аналогично объявлению списка, но с применением круглых скобок.

Например, выражение

(1, True, "red")

Является кортежем, который имеет тип (num, bool, [char]). Как список, так и кортеж могут быть пустыми и объявляются соответственно: [], ().

В языке Miranda не существует понятия одиночного кортежа, поэтому использование скобок для включения подвыражений, как, например, a \* (b + c), не противоречит их использованию для формирования кортежа. 1

## Работа со списками

Язык Miranda содержит ряд способов работы со списками.

Во-первых, знак двоеточие осуществляет разбиение списка на голову и хвост, либо обратное действие.

0 : [1, 2, 3] – список из чисел от нуля до трех.

0 : 1 : 2 : [] – список из чисел от нуля до двух.

Во-вторых, инфиксный оператор «++» осуществляет объединение списков.

Выражение

[1,2,3] ++ [6,7]

Будет вычислено как [1,2,3,6,7].

В-третьих, префиксный оператор «#» определяет длину списка.

В-четвертых, инфиксный оператор «!» позволяет получить элемент списка по индексу.

Выражение

[1,2,3] ! 1

Будет вычислено как 2.

В-пятых, оператор «-» осуществляет вычитание списков

Выражение

[1,2,3,4,5] - [2,4]

Будет вычислено как [1,3,5].

Язык Miranda также предоставляет иные способы задания списков, помимо перечисления.

1. Диапазон. Список заданный в виде [1..5] равнозначен списку [1, 2, 3, 4], [1.0, 1.5 .. 6] равнозначен [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0].
2. Бесконечный ряд. [0, 2 ..] – список четных чисел. Благодаря ленивой семантике, элементы списка вычисляются при необходимости.
3. Квалификатор. Выражение [ i \* i | i <- [1..5]] вычисляется как [1,4,9,16,25]
4. Диагонализация списка. Объявление аналогично квалификатору, но вместо вертикальной черты используется два знака слеш. Его отличие заключается в том, что все генераторы имеют одинаковый приоритет. Примером необходимости такого объявления является список треугольников Пифагора, заданный как L = [(а, б, в) // A, B, C <- [1 ..]; а ^ 2 + B ^ 2 = с ^ 2].

## Синтаксис языка

Рассмотрим синтаксис языка Miranda на примере нескольких программ.

fac 0 = 1

fac n = n \* fac (n-1)

Выше приведено определение функции, вычисляющей факториал числа. Как можно увидеть на этом примере объявление представляет собой название объявляемой функции, за которым через пробел записаны аргументы функции. При выполнении программы происходит сопоставление выражения и объявленных функций. Таким образом, выражение

fac 5

будет сопоставлено сначала с объявлением

fac 0 = 1

и, после неудачи, так как fac равен fac, а 5 не равен 0, будет проведено удачное сопоставление с

fac n = n \* fac (n-1)

а выражение будет заменено на вычисленную правую часть, а именно: 120.

По данному примеру также видно, что язык Miranda поддерживает основные арифметические операции: +, -, \*, / и т.д.

Далее рассмотрим программу, выполняющую переворот списка.

myreverse (head : tail) = myreverse tail ++ [head]

myreverse [] = []

Здесь объявляется функция myreverse, в первой альтернативе которой происходит разбиение списка на голову и хвост, с последующим рекурсивным вызовом функции для хвоста и присоединением к результату вызова текущей головы списка справа. Вторая альтернатива предусматривает, что разворотом пустого списка является пустой список.

Язык Miranda предусматривает введение пользовательских типов данных. Рассмотрим программу задающую тип бинарного дерева и работающую с ним.

tree ::= Nilt | Node num tree tree

t = Node 7 (Node 3 Nilt Nilt) (Node 4 Nilt Nilt)

mirror Nilt = Nilt

mirror (Node a x y) = Node a (mirror y) (mirror x)

treeToList (Node a x y) = (treeToList x) ++ [a] ++ (treeToList y)

treeToList Nilt = []

Первая строка об вводит три новых идентификатора - «tree», который является именем типа, «Nilt» и «Node», которые являются конструкторами для деревьев, следует обратить внимание, что конструкторы должны начинаться с заглавной буквы. Nilt является атомарным конструктором, в то время как Node принимает три аргумента указанных типов.

Вторая строка задает дерево с использованием заданных конструкторов.

Далее объявляется функция mirror, осуществляющая зеркальное отображение дерева.

Последние две строки представляют собой объявление функции, выполняющей обход дерева слева направо и формирующей из элементов дерева список.

У уравнения может быть несколько альтернативных правых частей, выбираемых с помощью условия, записанного справа после запятой. Например, для решения квадратных уравнений может быть записана:

quadsolve abc

= ошибка "комплексные корни", if delta <0

= [-b / (2 \* a)], если delta = 0

= [-b / (2 \* a) + radix / (2 \* a),

-b / (2 \* a) - radix / (2 \* a)], if delta> 0,

where

delta = b \* b - 4 \* a \* c

radix = sqrt delta

В данной функции также применяется блок where в котором определяются локальные определения.

Последнее условие может быть записано как «else» вместо «if» чтобы блок выполняемый по умолчанию.

# Практическая часть

## Постановка задачи

В рамках данной работы необходимо написать программу на языке prolog, осуществляющую преобразование формы записи выражения из инфиксной в постфиксную.

Программа должна предусматривать работу с выражениями, содержащими основные арифметические операции: «+», «-», «\*», «/», круглые скобки, числа и переменные.

Интерфейс программы должен представлять собой консоль, взаимодействующую с пользователем в формате «запрос-ответ». При запуске, программа должна отображать приглашение для ввода выражения. После ввода выражение программа должна выводить результат своей работы, снабженный пояснением, либо сообщать об ошибке.

При получении выражения, включающего в себя посторонние символы, программа должна сообщать об ошибке.

При неправильном порядке операций, операндов или скобок программа должна сообщать об ошибке.

## Описание применения программы

## Общие сведения и функциональное назначение

Разрабатываемая программа ориентирована на прямое взаимодействие с пользователем через консольный интерфейс. Программа не предусматривает специальных интерфейсов для включения в иные программные системы, но, тем не менее, не исключает возможность подобного применения.

Рассмотрим более подробно действия, которые должна выполнять программа:

1. При запуске программа должна в консоли сообщение с приглашением ввода выражения для преобразования.
2. Программа должна ожидать завершения ввода выражения пользователем.
3. После ввода выражения, программа должна прочитать из консоли строку, содержащую выражение.
4. Программа должна сформировать новую строку, содержащую выражение в постфиксной форме, идентичное введенному.
5. Программа должна отобразить сообщение с пояснением и полученную строку.
6. Программа должна отобразить в консоль информацию об успешной, либо не успешной работе (при неудаче пункта 4 вследствие некорректного заданного выражения пункт 5 не выполняется).

Приведем формальное определение преобразования, которое предполагает пункт 4.

Обозначим выражения в инфиксной нотации E, E1, E2, эквивалентные им выражения в постфиксной нотации P, P1, P2 соответственно; o – произвольный бинарный оператор, тогда:

1. Если E — переменная или константа, то P есть E.

2. Если E — выражение вида E1 o E2, то P есть P1 P2 o.

3. Если E — выражение вида (E1), то P есть P1.3

## Проектирование программы

Наиболее важной частью программы является преобразование формы записи выражения. Рассмотрим проектирование данной части программы.

В выражении можно выделить два основных элемента: операнд и операция. Операция представляет собой один символ арифметической операции «+», «-», «\*» или «/». Операнд может представлять собой число, переменную, другое выражение, либо другое выражение в круглых скобках. При этом корректное выражение всегда представляет собой либо одиночный операнд, либо сочетание двух операндов и операции между ними. Исходя из этого можно рассчитывать, что, рассматривая выражение посимвольно слева направо после операнда всегда находится либо операция, либо конец выражения, а после операции – операнд.

Ориентируясь на формальное определение, приведенное выше, можно сказать, что для преобразования формы записи выражения необходимо переместить операцию за второй операнд и исключить скобки, сохраняя приоритет операций.

Рассмотрим данный подход на практике, взяв в качестве примера тривиальное выражение «1 + 2 + 3».

Получив строку, рассматриваем ее как выражение, в начале которого находится операнд. «1» – это число, следовательно, операнд. Первый операнд переносится в новую строку, а остаток исходной рассматривается как сочетание операции и второго операнда, либо как окончание выражения.

После встречи «+» версия конца выражения отсекается, операция запоминается для вставки после второго операнда, а остаток исходной строки снова рассматривается как операнд.

После рассмотрения второго операнда при встрече операции, наличие запомненной операции говорит о том, что первым операндом новой операции являлось выражение, и перед запоминанием новой операции, необходимо полностью записать его в новую строку, приписав к ней запомненную операцию. Таким образом на данном этапе в новую строку записано выражение, являющееся первым операндом: «1 2 +», а остаток исходной: « + 3» снова рассматривается как сочетание операции и второго операнда, либо как окончание выражения.

После анализа «3», будет подтверждена версия о конце выражения, следовательно, если есть запомненная операция, выражение состояло в двух операндов и необходимо дописать операцию.

Так как операции выражения могут иметь разный приоритет необходимо внести изменения в подход. Если встреченная операция имеет более высокий приоритет, чем запомненная, то выражение, включающее встреченную операцию является вторым операндом для запомненной операции, следовательно, предыдущее выражение не закончилось и перед запоминанием новой операции не нужно записывать предыдущую.

Приведем пример: «1 + 2 \* 3».

После рассмотрения «+» в новой строке находится первый «1», запомнено «+», а в остаток «2 \* 3» является операндом. «2» также воспринимается как операнд и переносится в новую, а « \* 3» снова рассматривается как сочетание операции и второго операнда, либо как окончание выражения. Однако, встреча «\*», имеющей более высокий приоритет, чем «+» говорит о том, что вторым операндом для «+» является не «2», а «2 \* 3» и нужно запомнить и «\*», и «+».

Операнд «3» переносится в новую строку, а конец выражения говорит о том, что все выражения закончились, причем сначала закончится более приоритетное «\*», получив в новой строке «1 2 3 \*», а затем после записи второго операнда для «+», записывается и он: «1 2 3 \* +».

Выражение, заключенное в скобки, не зависит от остальных выражений и рассматривается как цельный операнд, поэтому при встрече открывающей скобки необходимо, запомнить ее и распознавать новое независимое выражение, а закрывающую считать концом этого выражения.

## Схема логической программы

Было принято решение осуществлять преобразование выражения работая с ним как со списком символов. Исходя из этого необходимо составить предикат, одним из аргументов которого является список символов исходного выражения. Принимая во внимание, сказанное в выше, логично применить два разных предиката для распознавания операции и операнда.

Рассмотрим общие особенности двух сформулированных предикатов. Они принимают первым аргументом список символов, выполняют действия, зависящие от символа в голове списка и вызывают следующий предикат, передавая ему хвост списка.

Для преобразования необходимо запоминать операции. Из этого следует, что предикатам необходим второй аргумент – список сохраненных операций.

Результирующую строку необходимо вернуть из предикатов, поэтому она является третьим аргументом предикатов.

Сформулируем поведение предиката для распознавания операнда. Для удобства назовем этот предикат parseOperand. Выше было сказано, что операнд может иметь четыре формы.

Если операндом является число, предикат обнаружит, что символ является цифрой и добавит ее к результирующей строке, после чего снова вызовет предикат parseOperand.

Если операндом является переменная, предикат обнаружит, что символ является буквой и добавит ее к результирующей строке, после чего снова вызовет предикат parseOperand.

Если операндом является выражение в скобках, предикат обнаружит, что символ является открывающей скобкой и сохранит эту скобку, после чего снова вызовет предикат parseOperand для разбора операнда внутри скобок.

В процессе разбора числа или переменной предикат может встретить символ не являющейся буквой или цифрой, это будет означать, что операнд закончился и необходимо вызвать предикат для разбора операции. Назовем его parseOperation.

К предыдущем рассуждениям есть одно замечание: если при разборе числа или переменной будет встречена открывающая скобка, ошибка не будет обнаружена. С целью исключения этой ошибки введем предикаты для разбора чисел и переменных. При этом следует заметить, что реакция на число и букву одинакова, следовательно, достаточно введения одного предиката. Назовем его parseAtom. Этот предикат будет вызываться предикатом parseOperand при встрече числа или буквы.

Предикат parseAtom должен действовать следующим образом.

При встрече числа или буквы добавляет ее к результирующей строке, после чего снова вызовет предикат parseAtom.

При встрече символа, не являющегося буквой или цифрой, вызывает предикат parseOperation.

Сформулируем поведение предиката parseOperation. Данный предикат вызывается предикатом parseOperand, когда заканчивается разбор операнда. За операндом может следовать либо операция, либо закрывающая скобка, либо конец выражения.

Поскольку при завершении выражения необходимо записать все запомненные операции, необходим новый предикат, который будет выполнять эти действия. Назовем его parseEnd. Этот предикат будет вызываться предикатом parseOperation при неудачи других альтернатив.

Действия предиката parseOperation зависят от того является ли символ операцией, а также имеет ли операция низкий или высокий приоритет. Для упрощения введем предикаты isOperation, isOperHighPrior и isOperLowPrior, которые выполняют соответствующие проверки.

При встрече операции, имеющей более высокий приоритет, чем последняя запомненная предикат parseOperation запоминает операцию и вызывает предикат parseOperand.

Если прочитанная операция равнозначна или менее приоритетна, чем последняя запомненная предикат parseOperation записывает запомненную операцию в строку результата, запоминает новую операцию и вызывает предикат parseOperand.

Если нет запомненных операций или запомнена скобка, предикат parseOperation запоминает новую операцию и вызывает предикат parseOperand.

Предикат parseEnd будет вызываться предикатом parseOperation при неудачи других альтернатив. Задачей parseEnd является обработать ситуации встречи закрывающей скобки и конца выражения.

Предикат parseEnd содержит граничные условия: во-первых, возвращается из рекурсии при опустошении входной строки и списка запомненных операций, во-вторых, вызывает parseOperation при чтении закрывающей скобки и встрече открывающей в голове списка запомненных.

Если прочитана закрывающая скобка предикат parseEnd переносит в строку результата все запомненную операцию вызывает parseEnd.

Если выражение закончилось предикат parseEnd переносит в строку результата все запомненные выражения, после чего снова вызывает parseOperation.

Предикат parseEnd должен проверять парность скобок. Если при закончившемся выражении осталась запомненная открывающая скобка, значит скобки неправильные и выражение некорректно.

Выражение может включать в себя пробелы между операндами и операциями, поэтому в предикаты parseOperation и parseOperand необходимо включить поведение, вызывающее тот же самый предикат, пропуская пробел.

В целях облегчения использования необходимо составить предикат, читающий из консоли строку, вызывающий предикат parseOperand и отображающий результирующую строку. Назовем этот предикат run.

В соответствии с данными рассуждениями была реализована программа. Прокомментированный исходный код программы представлен в приложении A.

## Входные данные

Для запуска программы необходимо вызвать предикат «run», не имеющий параметров.

После запуска программы и вывода приглашения ко вводу, программы ожидает, что пользователь введет в консоль выражение и нажмет клавишу «Enter».

Сформулируем, с какими выражениями может успешно работать программа.

1. Выражение может включать 4 основных арифметических операции: «+», «-», «\*», «/».
2. Выражение может включать парные круглые скобки любого уровня вложенности.
3. Выражение может включать числа любой длины.
4. Выражение может включать переменные, представляющие собой произвольное сочетание букв латинского алфавита и цифр любой длины.
5. Выражение может включать пробелы, разделяющие его элементы.
6. Выражение должно иметь корректную структуру: выражение может являться числом, переменной, двумя выражениями с операцией между ними, выражением в круглых скобках.

Программа корректно обрабатывает нарушение данных условий.

## Выходные данные

При успешном завершении работы программа выводит пояснение: «Выражение в постфиксной форме», после чего на следующей строке отображает преобразованное выражение, эквивалентное заданному.

Преобразованное выражение имеет следующие характеристики.

1. Выражение может включать 4 основных арифметических операции: «+», «-», «\*», «/».
2. Выражение может включать числа любой длины.
3. Выражение может включать переменные, представляющие собой произвольное сочетание букв латинского алфавита и цифр любой длины.
4. Операнды разделены пробелом.
5. Выражение имеет следующую структуру: выражение может являться числом, переменной, двумя выражениями с операцией после них.

В конце программа сообщает об успешности своей работы. В случае успеха она отображает «true.», в случае некорректных входных данных – «false.».

## Используемые технические средства

При разработке программы была использована среда SWI-Prolog («SWI-Prolog version 7.6.4 by Jan Wielemaker»).

Программа разработана с использованием операционной системы Windows 10.

Программа поставляется в виде исходного кода.

Для использования программы необходимо наличие установленной среды SWI-Prolog.

## Тестирование

Приведем тестирование программы в табличной форме в таблице 1. Скриншоты программы, подтверждающие результаты тестов приведены в приложении B. Номер скриншота соответствует номеру теста.

Таблица – Тестирование программы

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходное выражение | Полученное выражение | Результат |
| 1 | 2 + 2 | 2 2+ | Успешно |
| 2 | 1\*3-4/(5+6) | 1 3\*4 5 6+/- | Успешно |
| 3 | ((ab-c)/abcd) | ab c- abcd/ | Успешно |
| 4 | (c-(b+1) |  | Ошибка |
| 5 | Adbd + () |  | Ошибка |
| 6 | 1\*d+c-a/s\*c | 1 d\*c+a s/c\*- | Успешно |

# Заключение

В рамках теоретической части данной работы был рассмотрен функциональный язык программирования Miranda, названы его основные особенности, рассмотрен его синтаксис, приведены и изучены примеры программ на данном языке.

В рамках практической части на диалекте swi prolog языка prolog была реализована программа преобразующая выражение из инфиксной формы в постфиксную. Был описан процесс анализа задачи, проектирования и реализации программы. Приведен протокол тестирования и подтверждающие его скриншоты.

# Список используемой литературы

1. Clack, С. Programming with Miranda: учебное пособие / C. Clack, C. Myers и E. Poon. – 2011. – 307 с.
2. Wielemaker, J. SWI-Prolog 5.0: Reference Manual / Jan Wielemaker – University of Amsterdam, 2002. – 253 с.
3. Ахо, А. В. Компиляторы: принципы, технологии и инструменты / А. В. Ахо, Р. Сети, Д. Д. Ульман. М.: «Вильямс», 2003. С. 51.
4. Хабаров, С.П. prolog – язык разработки интеллектуальных и экспертных систем: учебное пособие / С.П.Хабаров. – Санкт-Петербург, 2013. – 138 с.
5. swi-prolog [Электронный ресурс]. – Режим доступа: <http://www.swi-prolog.org/> – Заглавие с экрана. – (Дата обращения: 1.12.2018).
6. miranda [Электронный ресурс]. – Режим доступа: <http://miranda.org.uk/> – Заглавие с экрана. – (Дата обращения: 3.12.2018).

# ПРИЛОЖЕНИЕ A

/\*

\* Предикат run предоставляет функционал программы

\* parseOperand, parseAtom, parseOperation, parseEnd принимают

\* первым аргументом список символов исходной формулы

\* второй аргумент используется для промежуточного хранения операций

\* третий - возвращаемое значение

\*/

run:-

writeln('Введите выражение в инфиксной форме'),

read\_line\_to\_codes(user\_input, A),

string\_codes(S,A),

string\_chars(S,C),

parseOperand(C,[],Res),

string\_chars(ResString, Res),

writeln('Выражение в постфиксной форме'),

writeln(ResString).

% Пропуск пробела

parseOperand([' '|Tail], Stack, Result):-!,

parseOperand(Tail, Stack, Result).

% Открывающая скобка сохраняется

parseOperand(['('|Tail], Stack, Result):-!,

parseOperand(Tail, ['('|Stack], Result).

% Переписывает операнд

parseOperand([Char|Tail], Stack, [Char|Result]):-

char\_type(Char, alnum),

parseAtom(Tail, Stack, Result).

% Если выражение закончилось а сохранена скобка - в выражении ошибка

parseOperand([], ['('|\_], \_):-!,

fail.

% Читает символ атома - цифру или букву

parseAtom([Char|Tail], Stack, [Char|Result]):-

char\_type(Char, alnum),

parseAtom(Tail, Stack, Result).

% Атом кончился - ожидается операция

parseAtom(Input, Stack, Result):-

parseOperation(Input, Stack, Result),!.

% Операции нет - конец выражения или закрывющая скобка

parseAtom(Input, Stack, Result):-

parseEnd(Input, Stack, Result).

% Пропуск пробела

parseOperation([' '|Tail], Stack, Result):-!,

parseOperation(Tail, Stack, Result).

% Если читается операция с приоритетом выше, чем хранимая, то сохраняем

% ее тоже

parseOperation([Char|Tail], [StackHead|StackTail], [' '|Result]):-

isOperHighPrior(Char),

isOperLowPrior(StackHead),!,

parseOperand(Tail, [Char|[StackHead|StackTail]], Result).

% Если ничего не хранится - сохраняем операцию

parseOperation([Char|Tail], [], [' '|Result]):-

isOperation(Char),!,

parseOperand(Tail, [Char], Result).

% Если сохранена скобка - сохраняем операцию (аналогичен предыдущему)

parseOperation([Char|Tail], ['('|StackTail], [' '|Result]):-

isOperation(Char),!,

parseOperand(Tail, [Char|['('|StackTail]], Result).

% Если прочитанная операция равнозначна или менее приоритетна чем хранимая, записывает

% хранимую в результат и сохраняет новую

parseOperation([Char|Tail], [StackHead|StackTail], [StackHead|Result]):-

isOperation(Char),!,

parseOperand(Tail, [Char|StackTail], Result).

% Если символ не является операцией

parseOperation(Input, Stack, Result):-

parseEnd(Input, Stack, Result).

% Если выражение закончилось а сохранена скобка - в выражении ошибка

parseEnd([], ['('|\_], \_):-!,

fail.

% При окончании формулы, записывает все хранящиеся операции

parseEnd([], [StackHead|StackTail], [StackHead|Result]):-!,

parseEnd([], StackTail, Result).

% Граничное условие для всей программы

parseEnd([], [], []).

% Граничное условие для окончания выражения в скобках

parseEnd([')'|Tail], ['('|StackTail], Result):-!,

parseOperation(Tail, StackTail, Result).

% При окончании выражения в скобках, записывает все хранящиеся операции

% (аналогичен вышестоящему)

parseEnd([')'|Tail], [StackHead|StackTail], [StackHead|Result]):-!,

parseEnd([')'|Tail], StackTail, Result).

% Операции с высоким приоритетом

isOperHighPrior('\*').

isOperHighPrior('/').

% Операции с низким приоритетом

isOperLowPrior('+').

isOperLowPrior('-').

% Операция

isOperation(Char):-

isOperHighPrior(Char);isOperLowPrior(Char).

# ПРИЛОЖЕНИЕ B

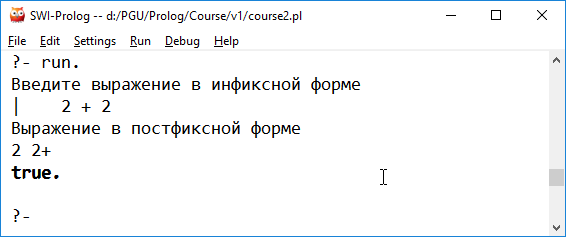


Рисунок – Тест №1

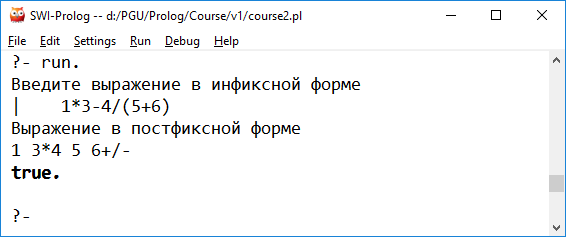


Рисунок – Тест №2

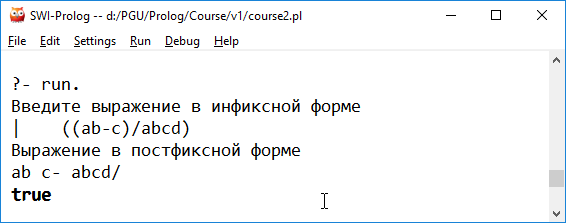


Рисунок – Тест №3

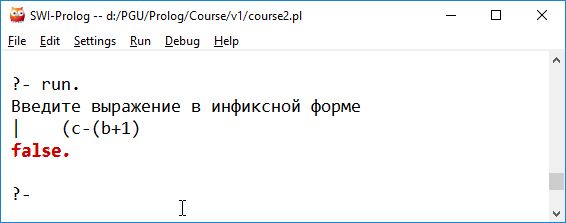


Рисунок – Тест №4

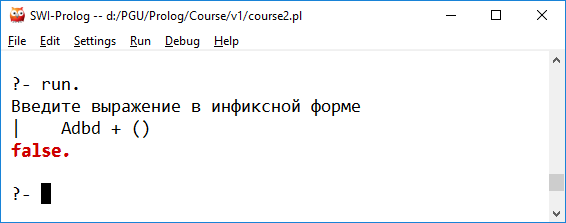


Рисунок – Тест №5

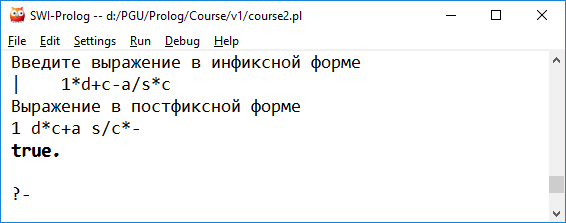


Рисунок – Тест №6