Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра прикладной математики и кибернетики

РАСЧЕТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ

по дисциплине «Функциональное и логическое программирование»

Вариант 8

Выполнил:

студент группы ИВ-121

Ермаков А. В.

Работу проверила:

Сороковых Д. А.

Новосибирск 2023 г.

**Оглавление**

[**Условия задач** 3](#_Toc149595957)

[**Задача 1** 4](#_Toc149595958)

[**Листинг программы** 6](#_Toc149595959)

[**Тесты** 6](#_Toc149595960)

[**Задача 2** 7](#_Toc149595961)

[**Листинг** 11](#_Toc149595962)

[**Тесты** 12](#_Toc149595963)

[**Список используемой литературы** 13](#_Toc149595964)

**Условия задач**

1.8. Подсчитайте число вхождений каждого элемента списка и сформируйте новый список, каждый элемент которого – это список из двух элементов: самого элемента и числа его вхождений в список. Например, [2,4,2,3,2,3]-> [[2,3], [4,1], [3,2]].

2.8. В текстовом файле, состоящем из нескольких строк, определите количество слов в каждой строке и информацию об этом занесите в отдельную строку нового файла в виде: строка № содержит слов.

**Задача 1**

1. Для начала формируем предикат «**вызов**», который вызывается из консоли SWI-Prolog, и делает следующие действия:  
 1)Создает переменную InputList и инициализирует ее списком.

2) Вызывает предикат формируем\_список, передавая в него InputList

и результирующий список.

3) Выводит получившийся список пар на экран.

2. Далее расписываем предикат **формируем\_список**, который имеет базовый случай и основной:

1) формируем\_список([], []), когда входной список пустой, то в результате вернется пустой результирующий список. Это важный случай, т.к. он позволяет завершить построение рекурсивного дерева и начать выполнять обратную рекурсию.

2) Основное правило предиката формируем\_список([Elem|Tail], [[Elem, Count]|Result]), которое гласит, что если входной список имеет в качестве головного элемента первое вхождение его в изначальный список и оставшийся список - хвост, а возвращаемый список представляет собой добавление пары [Elem, Count] к результирующему списку, то он имеет несколько правил:

а) Правило подсчет, которое имеет 3 аргумента: Elem – элемент, для которого мы считаем количество вхождений, [Elem|Tail] - список, который начинается с Elem, после которого совпадает с частью первоначального списка (Tail) и count – количество вхождений Elem в список.

б) Правило delete, который удаляет все вхождения Elem в Tail, и создает новый список без этого элемента.

в) Вызов предиката формируем\_список(NewTail, Result), в который передается новый список и список результата вызова и выполнения предиката.

3. Еще один предикат подсчет, расписанный на 3 правила:

1) Базовый предикат, который получает на вход анонимный элемент и пустой список, возвращая в качестве count ноль.

2) Предикат подсчет(Elem, [Elem|Tail], Count), который имеет следующие правила:  
 а) подсчет(Elem, Tail, TailCount) – рекурсивный вызов для остатка списка Tail, результат подсчета сохраняется в TailCount, иными словами мы подсчитываем количество вхождений элемента Elem в оставшейся части списка.

б) Count is TailCount + 1 – устанавливаем значение count, как количество вхождений в оставшейся части списка, увеличенное на единицу.

3) подсчет(Elem, [\_|Tail], Count) – предикат, обрабатывающий случай, когда Elem не совпадает с головным элементом списка, и имеет одно правило:

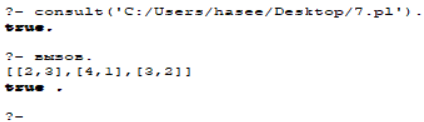
а) подсчет(Elem, Tail, Count) – рекурсивный вызов предиката, для оставшейся части списка.

**Листинг программы**

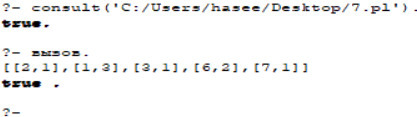
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **подсчет**(**\_**, [], **0**).  **подсчет**(Elem, [Elem|Tail], Count) :-  **подсчет**(Elem, Tail, TailCount),  Count is TailCount + **1**.  **подсчет**(Elem, [**\_**|Tail], Count) :-  **подсчет**(Elem, Tail, Count).  **формируем\_список**([], []).  **формируем\_список**([Elem|Tail], [[Elem, Count]|Result]) :-  **подсчет**(Elem, [Elem|Tail], Count),  **delete**(Tail, Elem, NewTail),  **формируем\_список**(NewTail, Result).  **вызов** :-  InputList = [**2**, **4**, **2**, **3**, **2**, **3**],  **формируем\_список**(InputList, ResultList),  **writeln**(ResultList). |

**Тесты**

1. Зададим входящий список InputList = [2, 4, 2, 3, 2, 3]

****

1. Зададим входящий список InputList = [2, 1, 3, 1, 1, 6, 7, 6]

****

**Задача 2**

**Для начала определим предикат для подсчета количество слов в строке -count\_words\_in\_line(Line, WordCount).** Этот предикат с двумя аргументами: Line – подаваемая строка и WordCount - результат.

В своем теле он содержит несколько предикатов:

1) trim\_whitespace(Line, TrimmedLine) – предикат, который удаляет пробелы в начале и конце строки Line и сохраняет результат в TrimmedLine:  
 было: « Я не хочу », стало: «Я не хочу».

2) split\_string(TrimmedLine, " ", " ", Words) – предикат, который разделяет строку TrimmedLine на слова, используя пробел в качестве разделителя, сохраняя результат в список Words:  
 было: « Я не хочу », стало: [“ ”, “ Я”, “ не”, “ хочу”, “ ”].

3) exclude(=(''), Words, NonEmptyWords) – предикат, который удаляет элементы являющиеся пустыми строками из списка, записывая результат в список NonEmptyWords: а было: [“ ”, “ Я”, “ не”, “ хочу”, “ ”], стало: [“ Я”, “ не”, “ хочу”].

4) length(NonEmptyWords, WordCount) – предикат, который подсчитывает количество элементов в списке NonEmptyWords, сохраняя результат WordCount: а length([1, 2, 3, 4, 5], L) на выходе L=5.

**Далее определим предикат для удаления пробелов в начале и конце строки trim\_whitespace(Input, Output).** Этот предикат используется для удаления пробелов из начала и конца строки.

В своем теле он содержит несколько предикатов:

1) atom\_codes(Input, InputCodes) - предикат, который преобразует Input в список кодов символов и сохраняет его в переменной InputCodes, получая внутреннее представление строки в виде списка чисел:

Input = 'Привет', InputCodes = [1055, 1088, 1071, 1074, 1077, 1090].

2) trim\_whitespace\_codes(InputCodes, TrimmedCodes) – предикат, имеющий два аргумента:

InputCodes - входной аргумент и представляет собой список кодов символов (исходную строку).

TrimmedCodes - выходной аргумент, представляющий исходную строку без пробелов в начале и конце.

3) atom\_codes(Output, TrimmedCodes) – предикат преобразования списка кодов символов TrimmedCodes (полученный после удаления пробелов) обратно в атом Output (строка без пробелов в начале и конце).

Теперь определим предикат **trim\_whitespace\_codes(InputCodes, TrimmedCodes)**, который удаляет пробелы из начала и конца списка кодов символов InputCodes и сохраняет результат в виде списка кодов символов TrimmedCodes.

В своем теле он содержит несколько предикатов:

1) drop\_while(space, InputCodes, TrimmedStart) – предикат, который удаляет элементы из начала списка InputCodes, пока они соответствуют условию, заданному предикатом space (предикат, который проверяет, является ли символ пробелом), сохраняя результат выполнения в TrimmedStart:

InputCodes = [32, 32, 32, 66, 67, 32, 32, 68, 69].

В этом списке пробелы представлены кодом символа 32, а буквы - кодами символов 66, 67, 68, 69.

TrimmedStart = [66, 67, 32, 32, 68, 69].

2) reverse(TrimmedStart, Reversed) – предикат, который переворачивает список TrimmedStart, чтобы в дальнейшем удалить пробелы, который были в конце списка:

TrimmedStart = [1, 2, 3, 4, 5], Reversed = [5, 4, 3, 2, 1].

3) drop\_while(space, Reversed, TrimmedEnd) – аналогично (см. 1 пункт).

4) reverse(TrimmedEnd, TrimmedCodes) – аналогично (см. 2 пункт).

Далее предикат **drop\_while:**

**drop\_while(\_, [], [])** - это базовый случай для предиката drop\_while вызывается только если список, представленный вторым аргументом, пуст, то результатом будет пустой список (3 аргумент).

**drop\_while(Pred, [H|T], Rest)** - это часть предиката drop\_while, которая работает следующим образом:

1) Pred – это аргумент предиката, который представляет условие или проверку, выполняемую для элементов списка.

2) [H|T] - это список, который разделен на голову H (первый элемент) и хвост T (остальные элементы).

Rest – это результат выполнения предиката Pred.

3) call(Pred, H) – предикат, позволяющий проверить элемент H на соответствие условию Pred, позволяя либо удалить элемент H из списка (соответствие условию) либо оставить его в списке.

4) drop\_while(Pred, T, Rest) – рекурсивный вызов предиката в том случае, когда элемент H был проверен и удален, для оставшихся элементов в списке T. Результат этого вызова сохраняется в переменной Rest.

5) drop\_while(\_, List, List) - базовый случай предиката drop\_while, вызывающийся тогда, когда больше нет элементов в списке, которые нужно было бы удалить согласно условию в Pred, в качестве результата является пустой список List.

space(Code) **–** предикат, который возвращает true или false, в зависимости от того, соответствует ли переданный аргумент Code коду символа пробела (кода 32 в Unicode) или нет. Если вызвать space(32), то предикат вернет true, так как код 32 соответствует символу пробела, space(75) вернет false.

process\_file – предикат, который используется для обработки текстового файла:

1) open(InputFile, read, InputStream) - открывает файл InputFile для чтения в файловый поток (InputStream), который позволяет читать содержимое файла.

2) open(OutputFile, write, OutputStream) - открывает файл OutputFile, для записи в файловый поток OutputStream, который позволяет записывать информацию в файл.

process\_lines(InputStream, OutputStream – предикат, обрабатывающий содержимое файла, построчно читая строки из InputStream, подсчитывает количество слов в каждой строке и записывает результат в OutputStream.

close(InputStream) закрывает файловый поток ввода (InputStream).

close(OutputStream) закрывает файловый поток вывода (OutputStream).

**process\_lines(InputStream, OutputStream, LineNumber)** – предикат, который читает строки из входного потока (InputStream), подсчитывает количество слов в каждой строке и записывает результат в выходной поток (OutputStream):

1) read\_line\_to\_string(InputStream, Line) - встроенный предикат, который используется для чтения строки из входного потока (InputStream) и сохранения в строку Line.

2) Условное выражение ( Line = end\_of\_file -> true ; ... ), в котором проверяется, равна ли переменная Line значению end\_of\_file. Если это условие выполнилось, то выполняется true, в противном же случае выполнится код, указанный после «;»:

1) Если Line не является концом файла, вызывается предикат count\_words\_in\_line для подсчета количества слов в строке Line и сохранения результата в переменную WordCount.

2) format(OutputStream, 'строка ~d содержит ~d слов.~n', [LineNumber, WordCount]) - формируем строку, которая будет записана в выходной поток (OutputStream):

~d - форматный спецификатор, используемый в строках для того чтобы указать, что на этом месте должно быть целое число (LineNumber и WordCount).

3) NextLineNumber is LineNumber + 1 – вычисление значения следующего номера строки.

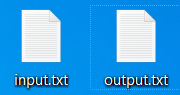
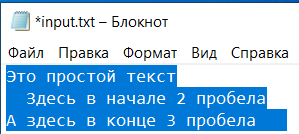
4) process\_lines(InputStream, OutputStream, NextLineNumber) - рекурсивный вызов с номером следующей строки NextLineNumber.

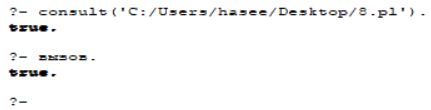
**Листинг**

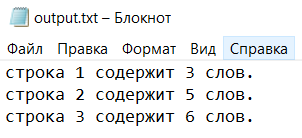
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | **count\_words\_in\_line**(Line, WordCount) :-  **trim\_whitespace**(Line, TrimmedLine),  **split\_string**(TrimmedLine, " ", " ", Words),  **exclude**(=(''), Words, NonEmptyWords),  **length**(NonEmptyWords, WordCount).  **trim\_whitespace**(Input, Output) :-  **atom\_codes**(Input, InputCodes),  **trim\_whitespace\_codes**(InputCodes, TrimmedCodes),  **atom\_codes**(Output, TrimmedCodes).  **trim\_whitespace\_codes**(InputCodes, TrimmedCodes) :-  **drop\_while**(space, InputCodes, TrimmedStart),  **reverse**(TrimmedStart, Reversed),  **drop\_while**(space, Reversed, TrimmedEnd),  **reverse**(TrimmedEnd, TrimmedCodes).  **drop\_while**(**\_**, [], []).  **drop\_while**(Pred, [H|T], Rest) :-  **call**(Pred, H),  **drop\_while**(Pred, T, Rest).  **drop\_while**(**\_**, List, List).  **space**(Code) :-  Code =:= **32**.  **process\_file**(InputFile, OutputFile) :-  **open**(InputFile, read, InputStream),  **open**(OutputFile, write, OutputStream),  **process\_lines**(InputStream, OutputStream, **1**),  **close**(InputStream),  **close**(OutputStream).  **process\_lines**(InputStream, OutputStream, LineNumber) :-  **read\_line\_to\_string**(InputStream, Line),  ( Line = end\_of\_file ->  true  ;  **count\_words\_in\_line**(Line, WordCount),  **format**(OutputStream, 'количество слов в ~d строке равно ~d.~n', [LineNumber, WordCount]),  NextLineNumber is LineNumber + **1**,  **process\_lines**(InputStream, OutputStream, NextLineNumber)  ).  **вызов** :-  **process\_file**('C:/Users/hasee/Desktop/input.txt', 'C:/Users/hasee/Desktop/output.txt'). |

**Тесты**

Для начала создадим файл input.txt и output.txt:

** **

****

****

Здесь можно увидеть, как работает программа, избегая лишних слов из-за пробелов.

**Список используемой литературы**

1. <https://eios.sibsutis.ru/course/view.php?id=672>
2. <https://habr.com/ru/articles/552318/>
3. <https://shawnpang.github.io/files/Prolog.html>
4. <https://www.cs.mtsu.edu/~rbutler/courses/sam/prolog_tutorial>