Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе №7

Название: Обработка списков на Prolog.

Дисциплина: Функциональное и логическое программирование.

Студент	ИУ7-64Б		Л.Е.Тартыков
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Н.Б.Толпинская
		(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Ю.В.Строганов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2022

1 Практические задания

1.1 Задание

Используя хвостовую рекурсию, разработать эффективную программу, (комментируя назначение аргументов), позволяющую:

1. найти длину списка (по верхнему уровню);
2. найти сумму элементов числового списка;
3. найти сумму элементов числового списка, стоящих на нечетных позициях исходного списка (нумерация от 0);

Убедиться в правильности результатов. Для одного из вариантов ВОПРОСА и одного из заданий составить таблицу, отражающую конкретный порядок работы системы.

Код программы представлен на листинге 1.1.

Листинг 1.1 – Код программы

```
domains
    list = integer*
predicates
    len_list(list, integer, integer).
    len_list(list, integer).

    sum_list(list, integer, integer).
    sum_list(list, integer).

    sum_odd_index_elems(list, integer, integer).
    sum_odd_index_elems(list, integer).
clauses
    len_list([_|Tail], Old_len, Len) :-
                        New_Len = Old_len + 1, !,
                        len_list(Tail, New_len, Len).
    len_list([], Old_len, Old_len).
    len_list(List, Len) :- len_list(List, 0, Len).

    sum_list([Head|Tail], Old_sum, Sum) :-
                        New_sum = Old_sum + Head, !,
                        sum_list(Tail, New_sum, Sum).
    sum_list([], Old_sum, Old_sum).
    sum_list(List, Sum_elems) :- sum_list(List, 0, Sum_elems).

    sum_odd_index_elems([_, Head|Tail], Old_Sum, Sum) :-
                        New_sum = Old_sum + Head, !,
                        sum_odd_index_elems(Tail, New_sum, Sum).
    sum_odd_index_elems([_], Old_sum, Old_sum) :- !.
    sum_odd_index_elems([], Old_sum, Old_sum).
    sum_odd_index_elems(List, Sum_elems) :- sum_odd_index_elems(List, 0,
        Sum_elems).
goal
    %найти длину списка (по верхнему уровню)
    %len_list([1, 2, 3], Len).

    %найти сумму элементов числового списка
    %sum_list([1, 2, 3], Sum_elems).

    %найти сумму элементов числового списка, стоящих на нечетных позициях
    %исходного списка (нумерация от 0)
    sum_odd_index_elems([1, 2, 3, 4, 5], Sum_elems).
```

Ниже на рисунках 1.1 1.2 приведена таблица порядка поиска ответа для нахождения суммы элементов списка:

№ шага	Состояние резольвенты, и вывод: дальнейшие действия (почему?)	Для каких термов запускается алгоритм унификации: T1=T2 и каков результат (и подстановка)	Дальнейшие действия: прямой ход или откат (почему и к чему приводит?)
…			
6	sum_list([1, 2, 3], Sum_elems)	sum_list([1, 2, 3], Sum_elems) = sum_list(List, Sum_elems) Результат: унификация успешна. Подстановка: {List = [1, 2, 3]}	Прямой ход. Переход к телу правила. Редукция и подстановка в резольвенту.
…			
11	sum_list([1, 2, 3], 0, Sum_elems)	sum_list([1, 2, 3], 0, Sum_elems) = sum_list([Head|Tail], Old_sum, Sum) Результат: унификация успешна. Подстановка: {Head = 1, Tail = [2, 3], Old_sum = 0}	Прямой ход. Редукция и подстановка в резольвенту.
12	New_sum = 0 + 1 ! sum_list([2, 3], New_sum, Sum)	New_sum = 0 + 1 Результат: унификация успешна. Подстановка: {New_sum = 1}	Прямой ход. Переход к следующей цели в резольвенте.
13	! sum_list([2, 3], 1, Sum)	! Результат: да.	Прямой ход. Переход к следующей цели в резольвенте.
…			
18	sum_list([2, 3], 1, Sum)	sum_list([2, 3], 1, Sum) = sum_list([Head|Tail], Old_sum, Sum) Результат: унификация успешна. Подстановка: {Head = 2, Tail = 3, Old_sum = 1}	Прямой ход. Переход к телу правила. Редукция и подстановка в резольвенту.
19	New_sum = 1 + 2 ! sum_list([3], New_sum, Sum)	New_sum = 1 + 2 Результат: унификация успешна. Подстановка: {New_sum = 3}	Прямой ход. Переход к следующей цели в резольвенте.
20	! sum_list([3], 3, Sum)	! Результат: да.	Прямой ход. Переход к следующей цели в резольвенте.
…			
25	sum_list([3], 3, Sum)	sum_list([3], 3, Sum) = sum_list([Head|Tail], Old_sum, Sum) Результат: унификация успешна. Подстановка: {Head = 3, Tail = [], Old_sum = 3}	Прямой ход. Переход к телу правила. Редукция и подстановка в резольвенту.
26	New_sum = 3 + 3 ! sum_list([3], New_sum, Sum)	New_sum = 3 + 3 Результат: унификация успешна. Подстановка: {New_sum = 6}	Прямой ход. Переход к следующей цели в резольвенте.
27	! sum_list([], 6, Sum)	! Результат: да.	Прямой ход. Переход к следующей цели в резольвенте.
…			
33	sum_list([], 6, Sum)	sum_list([3], 6, Sum) = sum_list([], Old_sum, Old_sum). Результат: унификация успешна. Подстановка: {Old_sum = 6, Sum = 6}	Прямой ход. Переход к следующему предложению.
…			
38	sum_list([], 6, 6)	sum_list([], 6, 6) = sum_odd_index_elems(List, Sum_elems) Результат: унификация неуспешна.	Сохранение подстановки {Old_sum = 6, Sum = 6} в памяти. Реконкретизация переменных. Восстановление состояния резольвенты (шаг 33).

Рисунок 1.1 – Таблица порядка поиска ответов для нахождения суммы элементов списка.

№ шага	Состояние резольвенты, и вывод: дальнейшие действия (почему?)	Для каких термов запускается алгоритм унификации: T1=T2 и каков результат (и подстановка)	Дальнейшие действия: прямой ход или откат (почему и к чему приводит?)
39	sum_list([3], 3, Sum) (пустая резольвента)	Пусто.	Обратный ход (резольвента пуста, БЗ просмотрена вся). Восстановление состояния резольвенты (шаг 27).
40	! sum_list([3], 3, Sum)	! Результат: неудача.	Запрет выполнения sum_list([3], 3, Sum). Обратный ход. Реконкретизация переменных. Восстановление состояния резольвенты (шаг 20).
41	! sum_list([2, 3], 1, Sum)	! Результат: неудача.	Запрет выполнения sum_list([2, 3], 1, Sum). Обратный ход. Реконкретизация переменных. Восстановление состояния резольвенты (шаг 11).
42	sum_list([1, 2, 3], 0, Sum_elems) (резольвента пуста)	Тело правила пусто (прямой ход).	Обратный ход. Реконкретизация переменных. Восстановление состояния резольвенты (шаг 6).
…			
47	sum_list([1, 2, 3], Sum_elems)	sum_list([1, 2, 3], Sum_elems) = sum_odd_index_elems(List, Sum_elems) Результат: унификация неуспешна.	Обратный ход. Резольвента пуста. БЗ пуста. Вывод на экран подстановки {Sum = 6}.

Рисунок 1.2 – Таблица порядка поиска ответов для нахождения суммы элементов списка (продолжение).

2 Контрольные вопросы

1. **Что такое рекурсия? Как организуется хвостовая рекурсия в Prolog? Как организовать выход из рекурсии?**
   Рекурия – ссылка на описываемый объект при описании объекта. Хвостовая рекурсия организовывается следующим образом: сначала выполняются необходимые вычисления, и последним шагом такой «функции» является вызов того же самого объекта. При этом вычисления собираются по мере выхода из рекурсии. Для того, чтобы система не выполняла лишних действий и при этом правильно отрабатывала, необходимо ставить условия выхода из рекурсии вначале.

2. **Какое первое состояние резольвенты?**
   В резольвенте изначально хранится конъюнкция вопросов.

3. **В каких пределах программы переменные уникальны?**
   Именованные переменные уникальны в пределах одного предложения, анонимные переменные – уникальные всегда.

4. **В какой момент, и каким способом системе удается получить доступ к голове списка?**
   Система использует разделитель (если указано это явно), которая делит исходный список на «голову» и «хвост». Например, такая конструкция языка [Head|Tail] позволит конкретизировать переменной Head голову списка.

5. **Каково назначение использования алгоритма унификации?**
   Назначение алгоритма унификации – подбор знаний.

6. **Как формируется новое состояние резольвенты?**
   Резольвента меняется в два этапа.

   (a) Новое состояние приобретается в результате алгоритма редукции.

   (b) К полученному состоянию применятся подстановка.

7. **Как применяется подстановка, полученная с помощью алгоритма унификации?**
   Подстановка применяется путем конкретизации переменных.

8. **В каких случаях применяется механизм отката?**
   Механизм отката применяется в случае тупиковой ситуации – в ситуации, когда нельзя перейти и данного состояния в новое, и при этом резольвента непуста.

9. **Когда останавливается работа системы? Как это определяется на формальном уровне?**
   Работа системы останавливается, когда ей [системе] не удалось подобрать факт для ответа на вопрос. На формальном уровне резольвента должна быть пуста , а также просмотрена вся БЗ.