Лабораторная работа 6

1. Загрузить среду программирования.

2. Выполнить задачи по варианту. Номер варианта равен номеру рабочего места.

3. Представить результат преподавателю.

Для получения полной оценки не забывайте про тесты. Два варианта тестирования:

1. Определите предикат test без аргументов, который выполняется, если все тесты проходят. Этот подход достаточно ограничен по возможностям.

Пример:

```
test_my_last :-
  my_last(1, [3,2,1]),
  \+ my_last(1, []). %% \+ означает "не".

test :-
  test_my_last, ...
```

2. (Предпочтительно) используйте стандартную библиотеку PlUnit (http://www.swi-prolog.org/pldoc/package/plunit.html).

Можно использовать предикаты из стандартной библиотеки.

Рекомендуется также прочитать http://www.cse.unsw.edu.au/~billw/testing.html.

Варианты:

1	1. Постройте деревья вызова для запросов: ?- предок_потомок(лиза, X). %% family.pl ?- my_last2(X, a). %% lists.pl 2. Задайте предикат zip(List1, List2, ZippedList), успешный, если List1 и List2 имеют одинаковую длину и каждый элемент ZippedList -- список из двух элементов, взятых соответственно из List1 и List2. ?- zip([a,b,c], [1,2,3], X) => X = [[a,1], [b,2], [c,3]] Проверьте поведение, если переменные на других местах. 3. Задайте предикат contains_duplicates(List), успешный, если List содержит равные между собой элементы. ?- contains_duplicates([a,b,a,b,c,c,a]) => Yes ?- contains_duplicates([a,b,c]) => No Проверьте поведение, если List содержит переменные.
2	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, сергей). %% family.pl

	?- my_member(List, 1). %% lists.pl 2. Задайте предикат ordered(List), который успешен, если элементы List -- числа, расположенные в порядке возрастания: ?- ordered([1,2,3]) => Yes ?- ordered([3,2]) => No ?- ordered([1,a]) => No 3. Задайте предикат sublist(List1, List2), успешный, если List2 -- отрезок List1 ?- sublist([a,b,c], X) => X = []; X = [a]; X = [b]; X = [c]; X = [a,b]; X = [b,c]; X = [a,b,c] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах.
3	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, катя). %% family.pl ?- my_append(List1, List2, [a,b]). %% lists.pl 2. Задайте предикат stutter(List, DoubledList), успешный, если DoubledList содержит те же элементы, что List, но по два раза подряд. ?- stutter([a,b,c], X) => X = [a,a,b,b,c,c]. Проверьте поведение, если переменные на других местах. 3. Задайте предикат remove_duplicates(List, ListWithoutDuplicates), успешный, если ListWithoutDuplicates содержит те же элементы, что List, в том же порядке, но с удалением всех повторений. ?- remove_duplicates([a,b,a,b,c,c,a], X) => X = [a,b,c] Проверьте поведение, если переменные на других местах.
4	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, X). %% family.pl ?- my_append([a,b,c], List, [Head | Tail]). %% lists.pl 2. Задайте предикат starts_with(List1, List2), который успешен, если List2 -- начальная часть List1. ?- starts_with([a,b,c], X) => X = [a, b]; X = [a]; X = [] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах. 3. Задайте предикат replace(List, Member, Replacement, ResultList). ResultList должен быть результатом замены всех вхождений терма Member в List на Replacement. ?- replace([1,2,3,1,2,3], 1, 5, X) => X = [1, 2, 5, 1, 2, 5]. Проверьте поведение, если переменные на других местах.
5	1. Постройте деревья вызова для запросов: ?- предок_потомок(лиза, X). %% family.pl ?- my_last2(X, a). %% lists.pl 2. Задайте предикат zip(List1, List2, ZippedList), успешный, если List1 и List2 имеют одинаковую длину и каждый элемент ZippedList -- список из двух элементов, взятых соответственно из List1 и List2. ?- zip([a,b,c], [1,2,3], X) => X = [[a,1], [b,2], [c,3]] Проверьте поведение, если переменные на других местах. 3. Задайте предикат contains_duplicates(List), успешный, если List содержит равные между собой элементы.

	?- contains_duplicates([a,b,a,b,c,c,a]) => Yes ?- contains_duplicates([a,b,c]) => No Проверьте поведение, если List содержит переменные.
6	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, сергей). %% family.pl ?- my_member(List, 1). %% lists.pl 2. Задайте предикат ordered(List), который успешен, если элементы List -- числа, расположенные в порядке возрастания: ?- ordered([1,2,3]) => Yes ?- ordered([3,2]) => No ?- ordered([1,a]) => No 3. Задайте предикат sublist(List1, List2), успешный, если List2 -- отрезок List1 ?- sublist([a,b,c], X) => X = []; X = [a]; X = [b]; X = [c]; X = [a,b]; X = [b,c]; X = [a,b,c] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах.
7	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, катя). %% family.pl ?- my_append(List1, List2, [a,b]). %% lists.pl 2. Задайте предикат stutter(List, DoubledList), успешный, если DoubledList содержит те же элементы, что List, но по два раза подряд. ?- stutter([a,b,c], X) => X = [a,a,b,b,c,c]. Проверьте поведение, если переменные на других местах. 3. Задайте предикат remove_duplicates(List, ListWithoutDuplicates), успешный, если ListWithoutDuplicates содержит те же элементы, что List, в том же порядке, но с удалением всех повторений. ?- remove_duplicates([a,b,a,b,c,c,a], X) => X = [a,b,c] Проверьте поведение, если переменные на других местах.
8	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, X). %% family.pl ?- my_append([a,b,c], List, [Head | Tail]). %% lists.pl 2. Задайте предикат starts_with(List1, List2), который успешен, если List2 -- начальная часть List1. ?- starts_with([a,b,c], X) => X = [a, b]; X = [a]; X = [] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах. 3. Задайте предикат replace(List, Member, Replacement, ResultList). ResultList должен быть результатом замены всех вхождений терма Member в List на Replacement. ?- replace([1,2,3,1,2,3], 1, 5, X) => X = [1, 2, 5, 1, 2, 5]. Проверьте поведение, если переменные на других местах.
9	1. Постройте деревья вызова для запросов: ?- предок_потомок(лиза, X). %% family.pl ?- my_last2(X, a). %% lists.pl 2. Задайте предикат zip(List1, List2, ZippedList), успешный, если List1 и List2 имеют одинаковую длину и каждый элемент ZippedList -- список из двух элементов, взятых соответственно из List1 и List2.

	?- zip([a,b,c], [1,2,3], X) => X = [[a,1], [b,2], [c,3]] Проверьте поведение, если переменные на других местах. 3. Задайте предикат contains_duplicates(List), успешный, если List содержит равные между собой элементы. ?- contains_duplicates([a,b,a,b,c,c,a]) => Yes ?- contains_duplicates([a,b,c]) => No Проверьте поведение, если List содержит переменные.
10	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, сергей). %% family.pl ?- my_member(List, 1). %% lists.pl 2. Задайте предикат ordered(List), который успешен, если элементы List -- числа, расположенные в порядке возрастания: ?- ordered([1,2,3]) => Yes ?- ordered([3,2]) => No ?- ordered([1,a]) => No 3. Задайте предикат sublist(List1, List2), успешный, если List2 -- отрезок List1 ?- sublist([a,b,c], X) => X = []; X = [a]; X = [b]; X = [c]; X = [a,b]; X = [b,c]; X = [a,b,c] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах.
11	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, катя). %% family.pl ?- my_append(List1, List2, [a,b]). %% lists.pl 2. Задайте предикат stutter(List, DoubledList), успешный, если DoubledList содержит те же элементы, что List, но по два раза подряд. ?- stutter([a,b,c], X) => X = [a,a,b,b,c,c]. Проверьте поведение, если переменные на других местах. 3. Задайте предикат remove_duplicates(List, ListWithoutDuplicates), успешный, если ListWithoutDuplicates содержит те же элементы, что List, в том же порядке, но с удалением всех повторений. ?- remove_duplicates([a,b,a,b,c,c,a], X) => X = [a,b,c] Проверьте поведение, если переменные на других местах.
12	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, X). %% family.pl ?- my_append([a,b,c], List, [Head | Tail]). %% lists.pl 2. Задайте предикат starts_with(List1, List2), который успешен, если List2 -- начальная часть List1. ?- starts_with([a,b,c], X) => X = [a, b]; X = [a]; X = [] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах. 3. Задайте предикат replace(List, Member, Replacement, ResultList). ResultList должен быть результатом замены всех вхождений терма Member в List на Replacement. ?- replace([1,2,3,1,2,3], 1, 5, X) => X = [1, 2, 5, 1, 2, 5]. Проверьте поведение, если переменные на других местах.
13	1. Постройте деревья вызова для запросов: ?- предок_потомок(лиза, X). %% family.pl

	?- my_last2(X, a). %% lists.pl 2. Задайте предикат zip(List1, List2, ZippedList), успешный, если List1 и List2 имеют одинаковую длину и каждый элемент ZippedList -- список из двух элементов, взятых соответственно из List1 и List2. ?- zip([a,b,c], [1,2,3], X) => X = [[a,1], [b,2], [c,3]] Проверьте поведение, если переменные на других местах. 3. Задайте предикат contains_duplicates(List), успешный, если List содержит равные между собой элементы. ?- contains_duplicates([a,b,a,b,c,c,a]) => Yes ?- contains_duplicates([a,b,c]) => No Проверьте поведение, если List содержит переменные.
14	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, сергей). %% family.pl ?- my_member(List, 1). %% lists.pl 2. Задайте предикат ordered(List), который успешен, если элементы List -- числа, расположенные в порядке возрастания: ?- ordered([1,2,3]) => Yes ?- ordered([3,2]) => No ?- ordered([1,a]) => No 3. Задайте предикат sublist(List1, List2), успешный, если List2 -- отрезок List1 ?- sublist([a,b,c], X) => X = []; X = [a]; X = [b]; X = [c]; X = [a,b]; X = [b,c]; X = [a,b,c] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах.
15	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, катя). %% family.pl ?- my_append(List1, List2, [a,b]). %% lists.pl 2. Задайте предикат stutter(List, DoubledList), успешный, если DoubledList содержит те же элементы, что List, но по два раза подряд. ?- stutter([a,b,c], X) => X = [a,a,b,b,c,c]. Проверьте поведение, если переменные на других местах. 3. Задайте предикат remove_duplicates(List, ListWithoutDuplicates), успешный, если ListWithoutDuplicates содержит те же элементы, что List, в том же порядке, но с удалением всех повторений. ?- remove_duplicates([a,b,a,b,c,c,a], X) => X = [a,b,c] Проверьте поведение, если переменные на других местах.
16	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, X). %% family.pl ?- my_append([a,b,c], List, [Head | Tail]). %% lists.pl 2. Задайте предикат starts_with(List1, List2), который успешен, если List2 -- начальная часть List1. ?- starts_with([a,b,c], X) => X = [a, b]; X = [a]; X = [] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах.

	3. Задайте предикат replace(List, Member, Replacement, ResultList). ResultList должен быть результатом замены всех вхождений терма Member в List на Replacement. ?- replace([1,2,3,1,2,3], 1, 5, X) => X = [1, 2, 5, 1, 2, 5]. Проверьте поведение, если переменные на других местах.
17	1. Постройте деревья вызова для запросов: ?- предок_потомок(лиза, X). %% family.pl ?- my_last2(X, a). %% lists.pl 2. Задайте предикат zip(List1, List2, ZippedList), успешный, если List1 и List2 имеют одинаковую длину и каждый элемент ZippedList -- список из двух элементов, взятых соответственно из List1 и List2. ?- zip([a,b,c], [1,2,3], X) => X = [[a,1], [b,2], [c,3]] Проверьте поведение, если переменные на других местах. 3. Задайте предикат contains_duplicates(List), успешный, если List содержит равные между собой элементы. ?- contains_duplicates([a,b,a,b,c,c,a]) => Yes ?- contains_duplicates([a,b,c]) => No Проверьте поведение, если List содержит переменные.
18	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, сергей). %% family.pl ?- my_member(List, 1). %% lists.pl 2. Задайте предикат ordered(List), который успешен, если элементы List -- числа, расположенные в порядке возрастания: ?- ordered([1,2,3]) => Yes ?- ordered([3,2]) => No ?- ordered([1,a]) => No 3. Задайте предикат sublist(List1, List2), успешный, если List2 -- отрезок List1 ?- sublist([a,b,c], X) => X = []; X = [a]; X = [b]; X = [c]; X = [a,b]; X = [b,c]; X = [a,b,c] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах.
19	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, катя). %% family.pl ?- my_append(List1, List2, [a,b]). %% lists.pl 2. Задайте предикат stutter(List, DoubledList), успешный, если DoubledList содержит те же элементы, что List, но по два раза подряд. ?- stutter([a,b,c], X) => X = [a,a,b,b,c,c]. Проверьте поведение, если переменные на других местах. 3. Задайте предикат remove_duplicates(List, ListWithoutDuplicates), успешный, если ListWithoutDuplicates содержит те же элементы, что List, в том же порядке, но с удалением всех повторений. ?- remove_duplicates([a,b,a,b,c,c,a], X) => X = [a,b,c] Проверьте поведение, если переменные на других местах.
20	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, X). %% family.pl ?- my_append([a,b,c], List, [Head | Tail]). %% lists.pl

	2. Задайте предикат starts_with(List1, List2), который успешен, если List2 -- начальная часть List1. ?- starts_with([a,b,c], X) => X = [a, b]; X = [a]; X = [] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах. 3. Задайте предикат replace(List, Member, Replacement, ResultList). ResultList должен быть результатом замены всех вхождений терма Member в List на Replacement. ?- replace([1,2,3,1,2,3], 1, 5, X) => X = [1, 2, 5, 1, 2, 5]. Проверьте поведение, если переменные на других местах.
21	1. Постройте деревья вызова для запросов: ?- предок_потомок(лиза, X). %% family.pl ?- my_last2(X, a). %% lists.pl 2. Задайте предикат zip(List1, List2, ZippedList), успешный, если List1 и List2 имеют одинаковую длину и каждый элемент ZippedList -- список из двух элементов, взятых соответственно из List1 и List2. ?- zip([a,b,c], [1,2,3], X) => X = [[a,1], [b,2], [c,3]] Проверьте поведение, если переменные на других местах. 3. Задайте предикат contains_duplicates(List), успешный, если List содержит равные между собой элементы. ?- contains_duplicates([a,b,a,b,c,c,a]) => Yes ?- contains_duplicates([a,b,c]) => No Проверьте поведение, если List содержит переменные.
22	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, сергей). %% family.pl ?- my_member(List, 1). %% lists.pl 2. Задайте предикат ordered(List), который успешен, если элементы List -- числа, расположенные в порядке возрастания: ?- ordered([1,2,3]) => Yes ?- ordered([3,2]) => No ?- ordered([1,a]) => No 3. Задайте предикат sublist(List1, List2), успешный, если List2 -- отрезок List1 ?- sublist([a,b,c], X) => X = []; X = [a]; X = [b]; X = [c]; X = [a,b]; X = [b,c]; X = [a,b,c] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах.
23	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, катя). %% family.pl ?- my_append(List1, List2, [a,b]). %% lists.pl 2. Задайте предикат stutter(List, DoubledList), успешный, если DoubledList содержит те же элементы, что List, но по два раза подряд. ?- stutter([a,b,c], X) => X = [a,a,b,b,c,c]. Проверьте поведение, если переменные на других местах. 3. Задайте предикат remove_duplicates(List, ListWithoutDuplicates), успешный, если ListWithoutDuplicates содержит те же элементы, что List, в том же порядке, но с удалением всех повторений. ?- remove_duplicates([a,b,a,b,c,c,a], X) => X = [a,b,c]

	Проверьте поведение, если переменные на других местах.
24	1. Постройте деревья вызова для запросов: ?- предок_потомок(алексей, X). %% family.pl ?- my_append([a,b,c], List, [Head | Tail]). %% lists.pl 2. Задайте предикат starts_with(List1, List2), который успешен, если List2 -- начальная часть List1. ?- starts_with([a,b,c], X) => X = [a, b]; X = [a]; X = [] (возможно, в другом порядке) Проверьте поведение, если переменные на других местах. 3. Задайте предикат replace(List, Member, Replacement, ResultList). ResultList должен быть результатом замены всех вхождений терма Member в List на Replacement. ?- replace([1,2,3,1,2,3], 1, 5, X) => X = [1, 2, 5, 1, 2, 5]. Проверьте поведение, если переменные на других местах.
25	1. Постройте деревья вызова для запросов: ?- предок_потомок(лиза, X). %% family.pl ?- my_last2(X, a). %% lists.pl 2. Задайте предикат zip(List1, List2, ZippedList), успешный, если List1 и List2 имеют одинаковую длину и каждый элемент ZippedList -- список из двух элементов, взятых соответственно из List1 и List2. ?- zip([a,b,c], [1,2,3], X) => X = [[a,1], [b,2], [c,3]] Проверьте поведение, если переменные на других местах. 3. Задайте предикат contains_duplicates(List), успешный, если List содержит равные между собой элементы. ?- contains_duplicates([a,b,a,b,c,c,a]) => Yes ?- contains_duplicates([a,b,c]) => No Проверьте поведение, если List содержит переменные.

Дополнительные задания:

4. Определите предикат my_flatten(NestedList, FlattenedList), "расплющивающий" вложенный список NestedList.

?- my_flatten([a, [[b], c], [[d]]], X). => X = [a, b, c, d]

5. Код Грея для n бит -- список всех возможных списков 0 и 1 длины n, два соседних списка в котором отличаются только в одном месте.

Он может быть рекурсивно построен на основе кода для n–1 бит следующим алгоритмом:

1. переворачиваем исходный список.

2. дописываем 0 в начало каждого кода в исходном списке

3. дописываем 1 в начало кодов в перевёрнутом списке.

4. объединяем оба полученных списка.

Например, для генерации списка для 3 бит:

0. Коды для 2 бит: 00, 01, 11, 10

1. Перевёрнутый список кодов: 10, 11, 01, 00

2. К начальному списку дописаны нули: 000, 001, 011, 010

3. К перевёрнутому списку дописаны единицы: 110, 111, 101, 100

4. Объединённый список: 000, 001, 011, 010, 110, 111, 101, 100

Задайте предикат gray(L, Code), возвращающий в Code код Грея для N бит, где N -- длина списка L.

(аргумент L вместо N используется потому, что мы ещё не знакомы с арифметикой в Прологе; какие конкретно элементы содержит L, неважно)

?- gray([0], Code). => Code = [[0], [1]] %% 1 бит

?- gray([0,0], Code). => Code = [[0,0], [0,1], [1,1], [1,0]] %% 2 бит

Критерии оценивания

	Задание сдано в срок	**Задание сдано позже**
Задача 1 выполнена верно		
Задача 2 выполнена верно		
Задача 3 выполнена верно		
Верно выполнено дополнительное задание 4		
Верно выполнено дополнительное задание 5		
Итого	7	3,5