Лабораторная работа №2

Рекурсивные структуры данных (списки)

Пролог позволяет определить и использовать рекурсивные типы данных. Примерами рекурсивных типов данных служат списки и деревья.

Список – это объект данных, содержащий конечное число других объектов (элементов списка). Список, содержащий числа 1, 2 и 3, записывается следующим образом: [1, 2, 3]

Для объявления списка используется следующее описание домена:

domains
*integerlist=integer**

Список является рекурсивным составным объектом, он состоит из двух частей:

- головы списка – первого элемента списка;
- хвоста списка – списка, включающего все последующие элементы.

[1| [2, 3]]
голова списка 1
хвост списка [2, 3]

Так как список имеет рекурсивную составную структуру, для работы со списками используется рекурсия.

Пример 1. Вывод элементов списка.

```
domains
integerlist=integer*

predicates
printlist (integerlist)

clauses
printlist ([]):- !.                    % для пустого списка ничего не делать
printlist ([H|T]):- write (H), nl, printlist (T).   % для непустого списка
                                       % отделить голову,
                                       % вывести ее,
                                       % продолжить вывод для
                                       % хвоста списка
```

Пример 2. Необходимо преобразовать список, элементами которого являются целые числа, инвертируя знак элементов списка, то есть положительные числа преобразовать в отрицательные, отрицательные в положительные, для нулевых значений никаких действий не предпринимать.

Придется рассмотреть два случая - для непустого и пустого списков. Преобразование пустого списка дать в результате также пустой список. Если

же список не пуст, то следует рекурсивно выполнять отделение головы списка, ее обработку и рассматривать полученный результат как голову списка-результата.

```
domains
intlist=integer*
predicates
inverting (intlist, intlist)
processing (integer, integer)
clauses
%обработка пустого списка дает, в результате, тоже пустой список
inverting ([ ], [ ]):- !.
%если список непустой, отделить голову, обработать ее,
%и добавить в качестве головы списка-результата
inverting ([H | T], [Inv_H | Inv_T]):-
        processing (H, Inv_H), inverting (T, Inv_T).
%предикат processing выполняет действия по обработке элемента списка в
%зависимости от его знака, предикат имеет два предложения,
%так как нужно рассмотреть два варианта: ненулевое и нулевое значения
processing (0, 0):- !.
processing (H, Inv_H):- Inv_H=-H.
goal
inverting ([-2, -1, 0, 1, 2], Inv_List), write("Inv_List=", Inv_List).
```

Результат работы программы:
Inv_List=[2, 1, 0, -1, -2]

Задание

1. Напишите на языке Visual Prolog программу, реализующую заданные операции над списками в соответствии с индивидуальным вариантом задания.
2. Произведите отладку программы в системе Visual Prolog для запросов на решение прямой и обратной задачи и задачи на перебор вариантов.
3. Постройте трассу программы при выполнении каждого запроса.

Операция 1

Реверс списка. Например: список [1, 2, 3] преобразуется в список [3, 2, 1].

Операция 2

Получение значения n-го элемента списка. Например: в списке [three, one, two] второй элемент равен one.

Операция 3

Удаление из списка всех элементов, равных 0. Например: список [1, 0, 2, 0, 0, 3] преобразуется в список [1, 2, 3].

Операция 4

Циклический сдвиг списка вправо на заданное число элементов. Например: список [6, 5, 4, 3, 2, 1], циклически сдвинутый вправо на 2 элемента, преобразуется в список [2, 1, 6, 5, 4, 3].

Операция 5

Удаление из списка 2-ого, 4-ого и т.д. элементов. Например: список [6, 5, 4, 3, 2, 1] преобразуется в список [6, 4, 2].

Операция 6

Замена в списке всех элементов, равных 0, на -1. Например: список [1, 0, 0] преобразуется в список [1, -1, -1].

Операция 7

Перевод списка арабских чисел (от 1 до 10) в список римских чисел. Например: список [1, 2, 3] преобразуется в список [“I”, “II”, “III”].

Операция 8

Подсчет количества определенных элементов в списке. Например: в списке [1, 2, 1, 3, 1] три единицы.

Операция 9

Подсчет количества элементов списка без какого-либо указываемого элемента. Например: в списке [1, 2, 1, 3, 1] два элемента без учета единиц.

Операция 10

Подсчет количества элементов списка, значения которых лежат в определенном диапазоне. Например: в списке [10, 20, 10, 30, 15] два элемента, значения которых больше 10 и меньше 30.

Операция 11

CROSS(X,Y,Z) – список *Z* является пересечением списков *X* и *Y*. Реализовать с использованием предиката *ENTER(A,X)* – элемент *A* входит в список *X*.

Операции 12

SUFFIX(X,Y) – список *X* является суффиксом списка *Y*.

PREFIX(X,Y) – список *X* является префиксом списка *Y*.

Операция 13

NEAR(A,B,L) – элементы *A* и *B* находятся рядом в списке (на соседних местах) *L*. Реализовать с использованием предиката *AFTER(A,B,L)* – элемент *B* следует непосредственно за элементом *A* в списке *L*.

Операция 14

SUBLIST(X,Y) – список *X* является подсписком *Y*.

Операция 15

NEXT(A,B,Z) – элемент *B* следует за *A* в списке *Z*.

Операция 16

SUBSET(X,Y) – элементы списка *X* являются подмножеством элементов списка *Y*.

Операция 17

SUB(X,Y,Z) – множество элементов списка *Z* является разностью множеств элементов списков *X* и *Y*.

Операции 18

FIRST(A,X) – A – первый элемент списка *X*.

LAST(A,X) – A – последний элемент списка *X*.

Операции 19

ADD(A,X,Y) – в список *X* добавляется элемент *A* с образованием списка *Y*.

DEL(A,X,Y) – из списка *X* удаляется элемент *A* с образованием списка *Y*.

Варианты заданий

№ варианта	Операции	№ варианта	Операции	№ варианта	Операции
1	1, 7, 13	7	1, 10, 11	13	7, 11, 18
2	2, 8, 14	8	2, 12, 13	14	8, 12, 17
3	3, 9, 15	9	3, 14, 15	15	9, 13, 16
4	4, 10, 16	10	4, 16, 17	16	10, 14, 15
5	5, 11, 17	11	5, 18, 19	17	1, 10, 18
6	6, 12, 18	12	6, 10, 19	18	2, 11, 19

Результаты работы и содержание отчета

1. Текст программы с комментариями.
2. Трассы выполнения запросов и объяснение результатов их выполнения.