Отчет по лабораторной работе № 1: Обработка списков по курсу Логическое программирование

Выполнила студентка группы 08-307 МАИ *Усачева Елизавета*.

Введение

В логических языках программирования, так же как в императивных или функциональных, списки используются часто. Они важны тем, что позволяют хранить набор данных почти любой длины. В языке Prolog список представляет собой структуру данных, состоящую из узлов, т.е. формируя как бы односвязный список. Списки в Prolog имеют существенные отличия от массивов, используемых в императивных языках. Например, элемент можеть быть легко добавлен или, наоборот, удален из начала списка. Однако, чтобы получить доступ к произвольному (n-му) элементу потребуется n операций перехода по ссылкам вдоль списка, в отличие от массивов в императивных языках, где подобная операция имеет имеет сложность *O(1)*. При работе с односвязными списками необходимо выделять первый узел (называемый головой списка), остальные узлы (составляющие хвост списка) можно получить передвигаясь по указателям вплоть до последнего узла. Хвост списка является таким же списком, как и исходный, поэтому обрабатывается аналогичным образом (рекурсивно).

Задание 1: Реализация стандартных предикатов

- Предикат вычисления длины списка

```
my_length([], 0).
my_length([_|T], N) :-
    my_length(T, N1), N is N1+1.
```

Результат работы:

```
?- my_length([], X).
X = 0.

?- my_length([1, 2, 3], X).
X = 3.
```

- Предикат принадлежности

```
my_member(X, [X|_]).
my_member(X, [_|T]) :-
    my_member(X, T).
```

Результат работы:

```
?- my_member(X, [1, 2, 3]).
X = 1 ;
X = 2 ;
X = 3 .
```

```
?- my_member(99, [1, 2, 99, 3]).
true .
```

- Конкатенация списков

```
my_append([], M, M).
my_append([H|T], M, [H|X]) :-
    my_append(T, M, X).
```

Результат работы:

```
?- my_append([1, 2, 3], 4, X).
X = [1, 2, 3, 4].
```

- Предикат удаления элемента из списка

```
my_remove(M, [M|T], T).
my_remove(M, [H|T], [H|X]) :-
    my_remove(M, T, X).
```

Результат работы:

```
?- my_remove(3, [1, 2, 3, 4], X).
X = [1, 2, 4] .
```

- Перестановки в списке

```
my_sublist([], []).
my_permute(L, [H|T]) :-
    my_remove(H, L, X),
    my_permute(X, T).
```

Результат работы:

```
?- my_permute([1, 2, 3], X).
X = [1, 2, 3] ;
X = [1, 3, 2] ;
X = [2, 1, 3] ;
X = [2, 3, 1] ;
X = [3, 1, 2] ;
X = [3, 2, 1] .
```

- Подсписок списка

```
my_sublist([], []).
my_sublist(X, Y) :-
    my_append(_, T, Y),
    my_append(X, _, T).
```

Результат работы:

```
?- my_sublist(X, [1, 2, 3]).
X = [] ;
X = [1] ;
X = [1, 2] ;
X = [1, 2, 3] ;
X = [] ;
X = [2] ;
X = [2, 3] ;
X = [] ;
X = [3] ;
X = [] .
```

Задание 2: Предикаты обработки списков (индивидуальное задание)

Вариант 1: получение последнего элемента списка.

- Реализация с использованием стандартных предикатов:

```
last_elem(X, L) :-
    append(_, [X], L).
```

Результат работы:

```
?- last_elem(X, [1, 2, 3]).
X = 3.
```

- Реализация без использованием стандартных предикатов:

```
my_last_elem(X, [X]).
my_last_elem(X, [_|T]) :-
    my_last_elem(X, T).
```

Результат работы:

```
?- my_last_elem(X, [1, 2, 3]).
X = 3.
```

Задание 3: Предикаты обработки числовых списков (индивидуальное задание)

Вариант 1: вычисление суммы элементов списка.

- Реализация без использования стандартный предикатов:

```
my_sum([], 0).
my_sum([H|T], X) :-
        my_sum(T, X1),
        X is H + X1.
```

Результат работы:

```
?- my_sum([1, 2, 3], X).
X = 6.

?- my_sum([1, 2, 20], X).
X = 23.

?- my_sum([], X).
X = 0.
```

Задание 4: Пример совместного использования предикатов, реализованных в предыдущих пунктах

- Удаление последнего элемента списка.:

```
my_delete_last(L, X) :-
    my_last_elem(Y, L), my_remove(Y, L, X).
```

Результат работы:

```
?- my_delete_last([1, 2, 3], X).
X = [1, 2].
```

Выводы

В данной лабораторной работе я познакомилась с такой концепцией как логическое программирование и, в частности, с языком программирования Пролог. Данная концепция существенно отличается от привычной для меня концепции императивного программирования. Отличие этих двух концепций заключается в том, что в императивных языках программирования происходит описание алгоритма получения необходимого результата, а в логических - описание необходимого результата. Данный подход оказывается нетривиальным для обычного интуитивного мышления, но в этом и заключается его исключительная польза для развития мышления программистов и математиков.