|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Теоретической и прикладной математики | | |
|  | | |
| Лабораторная работа № 4 | | |
| по дисциплине «Логическое программирование» | | |
| ***Рекурсивные структуры данных - списки*** | | |
|  | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-02 |
| Вариант: | 7 |
| Студент: | Сидоров Даниил, |
|  | Дюков Богдан |
| Преподаватель: | Авдеенко Т. В., |
|  | Целебровская М. Ю.  . |
|
|  |  |
|  | | |
| Новосибирск | | |
| 2022 | | |

1. **Цель работы**

Изучить основные принципы работы с универсальной рекурсивной структурой в логическом программировании, правостороннюю и лево­стороннюю рекурсию, алгоритм унификации.

1. **Упражнение**

Проанализируйте работу одного из основных преди­катов обработки списков, выполнив его в режиме трассировки. Какие задачи можно решать, используя данный предикат? Придумайте цели для решения каждой задачи и начертите деревья поиска решений.

Соединение списков:

append([], L, L).

append([X|L1], L2, [X|L3]): -append(L1, L2, L3).

Способы использования (цели):

- конкатенация списков

append([a, b], [c, d], List)

**List = [a, b, c, d]**

append([a, b], [c, d], List)

append([b], [c, d], List)

append([], [c, d], List)

L1=[b] L2=[c, d] X=a

L1=[] L2=[c, d] X=b

List=[c,d]

append([], [c, d], [c,d])

**List = [a, b, c, d]**

- проверка того, что третий список является конкатенацией первых двух

append([a, b], [c, d], [a, b, c])

**Ответ: No**

append([a, b], [c, d], [a, b, c])

append([b], [c, d], [b, c])

append([], [c, d], [c])

L1=[b] L2=[c, d] L3=[b,с] X=a

L1=[] L2=[c, d] L3=[с] X=b

List=[c,d]

append([], [c, d], [c,d])

**[a, b, c, d]!= [a, b, c]**

**no**

-нахождение разности двух списков, например неизвестен первый аргумент

append(List1, [c, d], [a, c, d])

**List1 = [a]**

append(List1, [c, d], [a, c, d])

append(List1, [c, d], [c, d])

append(List1, [c, d], [d])

L2=[c, d] L3=[c, d] X=a

L2=[c, d] L3=[d] X=c

L2=[c, d] L3=[] X=d

append(List1, [c, d], [])

**List = [a]**

-нахождение всевозможных разбиений списка (когда неизвестны первые два аргумента)

append(List1, List2, [a, b, c])

**List1=[] List2=[a, b, c]**

**List1=[a] List2=[b, c]**

**List1=[a, b] List2=[c]**

**List1=[a, b, c] List2=[]**

append(List1, List2, [a, b, c])

append(List1, List2, [b, c])

L3=[b, c] X=a

L3=[c] X=b

append(List1, List2, [c])

L3=[] X=c

append(List1, List2, [])

**List1=[a]**

**List2=[b, c]**

**List1=[a,b]**

**List2=[c]**

**List1=[]**

**List2=[a, b, c]**

**List1=[a,b,c]**

**List2=[]**

1. **Задание**

Написать программу, которая:

а) переносит в конец непустого списка *L* его первый элемент;

б) переносит в начало непустого списка *L* его последний эле­мент;

в) проверяет, есть ли в списке *L* хотя бы два одинаковых элемента;

г) вставляет в список *L* за первым вхождением элемента *X* все элементы списка *L1*, если *X* входит в *L*;

д) в списке *L* из каждой группы подряд идущих равных элементов оставляет только один;

ж) подсчитывает число вхождений каждого элемента списка L и формирует новый список *L1*, в котором каждый элемент - список, состоящий из элемента и числа его вхождений в список L.

Например, если *L*= [*f, d, f, g, a, f, d*], то *L1*= [[*f, 3*], [*d, 2*], [*g, 1*], [*a, 1*]].

1. **Листинг программы**

domains

type = integer

list = type\*

listlist = list\*

predicates

nondeterm member(type,list)

nondeterm dublicates(list)

nondeterm listWithNumbersOfEachSymbols(list,listlist)

nondeterm numberOfSymbol(integer,integer,list,list,list)

nondeterm deleteRepeat(list,list)

nondeterm insertListAfterX(list,type,list,list)

nondeterm append(list,list,list)

nondeterm lastInHead(list,list)

nondeterm last(list,type)

nondeterm deleteLast(list, list)

nondeterm deleteFirst(type,list, list)

nondeterm firstInTail(list,list)

clauses

member(Elem, [Elem|\_Tail]).%member Of List%

member(Elem, [\_Head|Tail]):-member(Elem, Tail).

listWithNumbersOfEachSymbols([Elem|L],[Result|M]):-numberOfSymbol(1,Elem,L,L1,Result),!,listWithNumbersOfEachSymbols(L1,M).%List With Numbers Of Each Symbols%

listWithNumbersOfEachSymbols([],[]).

numberOfSymbol(N,Elem,[Elem|L],L1,Result):-N1=N+1,!,numberOfSymbol(N1,Elem,L,L1,Result).%Number Of Symbols%

numberOfSymbol(N,Elem,[U|L],[U|L1],Result):-numberOfSymbol(N,Elem,L,L1,Result).

numberOfSymbol(N,Elem,[],[],[Elem,N]).

deleteRepeat([Head, Head | Tail], Result) :-!,deleteRepeat([Head | Tail], Result).%delete repeat symbols%

deleteRepeat([Head | Tail], [Head | Result]) :-!,deleteRepeat(Tail, Result).

deleteRepeat(Result, Result).

dublicates([]):-!,fail. %two or more same symbols%

dublicates([Head|Tail]):-member(Head, Tail); dublicates(Tail).

insertListAfterX(\_,\_,[],[]):-!.%Insert list after given number%

insertListAfterX(L,Head,[Head|Z],Result):- !, insertListAfterX(L,Head,Z,Tail), append([Head|L],Tail,Result).

insertListAfterX(L,Y,[Head|Z],[Head|Result]):- insertListAfterX(L,Y,Z,Result).

append([],L,L).%Insert elements in list%

append([Head|Tail],L,[Head|Result]):-append(Tail,L,Result).

deleteLast([X,\_],[X]).%Delete last element%

deleteLast([Head|Tail],[Head|Z]) :- deleteLast(Tail,Z).

deleteFirst(\_, [], []). %Delete first occurrence of element%

deleteFirst(Elem, [Elem | L], L) :- !.

deleteFirst(Elem, [Y | L], [Y | L1]) :-deleteFirst(Elem, L, L1).

last([X], X):-!.%Get last element%

last([\_Head|Tail], Element):-last(Tail, Element).

firstInTail(X,Result):-member(Elem, X),!,append(X,[Elem],HalfResult),deleteFirst(Elem,HalfResult,Result).%Move first element to end of list%

lastInHead(X,Result):-last(X, Elem),append([Elem],X,HalfResult),deleteLast(HalfResult,Result).%Move the last element of the list to the beginning of the list%

1. **Результаты выполнения**

firstInTail([1,3,5,7,9,11], L).



lastInHead([1,3,5,7,9,11], L).



insertListAfterX([0,0,0],1, [1,2,1,3,4],L).



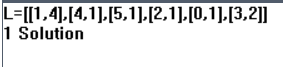
dublicates([1,2,3,4,5,6,1]).



deleteRepeat([0,1,1,2,2,3,3,3,4,4,4,4,5], L).



listWithNumbersOfEachSymbols([1,1,1,4,5,2,1,0,3,3], L).



1. **Описание задач, решаемых данной программой**

**а) переносит в конец непустого списка *L* его первый элемент;**

firstInTail(X,Result):-member(Elem,X),!,append(X,[Elem],HalfResult),deleteFirst(Elem,HalfResult,Result).

Берем первый элемент списка X, вставляем его в конец cписка HalfResult, удаляем первый элемент списка HalfResult и получаем список Result.

**б) переносит в начало непустого списка *L* его последний эле­мент;**

lastInHead(X,Result):-last(X,Elem),append([Elem],X,HalfResult),deleteLast(HalfResult,Result).

Находим последний элемент списка X, вставляем его в начало списка HalfResult, удаляем последний элемент списка HalfResult и получаем список Result.

**в) проверяет, есть ли в списке *L* хотя бы два одинаковых элемента;**

dublicates([]):-!,fail.

dublicates([Head|Tail]):-member(Head, Tail); dublicates(Tail).

Если список пустой в нем нет дубликатов.

Если первый элемент списка повторяется в хвосте списка или в хвосте есть повторяющиеся элементы, то список имеет повторяющиеся элементы.

**г) вставляет в список *L* за первым вхождением элемента *X* все элементы списка *L1*, если *X* входит в *L*;**

insertListAfterX(\_,\_,[],[]):-!.

insertListAfterX(L,Head,[Head|Z],Result):-!,insertListAfterX(L,Head,Z,Tail), append([Head|L],Tail,Result).

insertListAfterX(L,Y,[Head|Z],[Head|Result]):- insertListAfterX(L,Y,Z,Result).

Если список пустой ничего не делаем.

Если текущий элемент не Head, то идем дальше по списку.

Если текущий элемент Head, то вставляем L после X. Так делаем с каждым элементом, равным X.

**д) в списке *L* из каждой группы подряд идущих равных элементов оставляет только один;**

deleteRepeat([Head, Head | Tail], Result) :-!,deleteRepeat([Head | Tail], Result).

deleteRepeat([Head | Tail], [Head | Result]) :-!,deleteRepeat(Tail, Result).

deleteRepeat(Result, Result).

Если текущий элемент не повторяется, идем дальше.

Если текущий элемент повторяется, то убираем один элемент и идем дальше. В конце записываем результат в список Result.

**ж) подсчитывает число вхождений каждого элемента списка L и формирует новый список *L1*, в котором каждый элемент - список, состоящий из элемента и числа его вхождений в список L.**

listWithNumbersOfEachSymbols([Elem|L],[Result|M]):-numberOfSymbol(1,Elem,L,L1,Result),!,listWithNumbersOfEachSymbols(L1,M).

listWithNumbersOfEachSymbols([],[]).

numberOfSymbol(N,Elem,[Elem|L],L1,Result):-N1=N+1,!,numberOfSymbol(N1,Elem,L,L1,Result).

numberOfSymbol(N,Elem,[U|L],[U|L1],Result):-numberOfSymbol(N,Elem,L,L1,Result).

numberOfSymbol(N,Elem,[],[],[Elem,N]).

Если два списка пусты, ничего не делаем.

Считаем количество каждого символа и формируем список в списке, в котором содержится элемент и количество его вхождений.

1. **Дерево поиска решений**

duplicates([1,2,3,4,5,6,1])

[Head, Tail] = {1, [2,3,4,5,6,1]}

member(1, [ 2,3,4,5,6,1]); dublicates(Tail).

YES

1. **Унификация**

**Шаг 1:**

**Cтек {**duplicates([1,2,3,4,5,6,1])| member(1, [ 2,3,4,5,6,1])**}**

**O={Yes}**

**Y={1}**