**Рекурсивное программирование и управление на языке Пролог.**

**Задание №1.** Разработать рекурсивный вариант программы для решения предложенной ниже задачи.

**Задание №2.** Разработать вариант программы с использованием средств управления в Прологе.

В программе нельзя использовать стандартные процедуры для работы со списками. Все используемые процедуры должны быть определены в исходном тексте.

**Конкретное задание, доставшееся мне по варианту:**

|  |  |
| --- | --- |
| Формулировка задачи(действия, которые должна выполнять процедура «name») | Пример обращения к процедуре |
| Формировать список, состоящий из элементов, которые встречаются в обоих исходных списках | ?-proc([a, s, d, f, g, h], [q, a, w, s, e, d, r],L).L=[a, s, d] |

**Пример выполнения задания, вдруг поможет:**

**Задание №1.** Разработать рекурсивный вариант программы для решения предложенной ниже задачи.

33**Текст задания.**

|  |  |
| --- | --- |
| Формулировка задачи(действия, которые должна выполнять процедура «proc») | Пример обращения к процедуре |
| Формировать список, являющийся слиянием двух исходных, состоящий только из совпадающих эле-ментов, а каждый элемент является подсписком уровня, соответствующего его месту, исключая повторы элементов | ?-proc([1,2,3,4,2,5],[3,77,4,23,2],Res).L = [23,[3],[[4]],[[[6]]]] ;no |

**Введение.** Разрабатываемая процедура должна иметь арность 3. Все параметры процедуры – списки. Первый и второй параметры предназначены для передачи исходных списков, третий соответствует формируемому списку.

**Декларативное описание решения.** Решение обладает следующими свойствами:

• результирующий список - объединение общих элементов двух исходных списков;

• каждый элемент результирующего списка является под-списком;

• уровень каждого подсписка соответствует его месту в результирующем списке;

• повторы элементов результирующего списка исключаются.

**Комментированный текст программы**

/\* Разрабатываемая процедура \*/

proc(List,List1,Result):-

/\* искомый список Result как слияние списков List и List1, если \*/

concatх(List,List1,R\_prom),

/\* R\_prom – слияние общих элементов списков List,List1 и \*/

spi(R\_prom,Result).

/\* Result – список подуровней списка R\_prom \*/

/\* Вспомогательные процедуры \*/

concatx([],Z,[]). /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* Результат слияния пустого списка со списком Z – пустой спи-сок\*/

concatx([X1|Y1],X2,Z1):-

/\* первый список состоит из головы X1 и хвоста X2, а результи-рующий - Z1\* **если** \*/

nomemb(X1,X2),

/\* элемент X1 не встречается в списке X2 **и** \*/

concatx(Y1,X2,Z1);

/\* Z1- объединение списков Y1 и X2 **или** \*/

concatx(Y1,X2,Z),

/\* Z – объединение списков Y1 и X2 **и** \*/

add(X1,Z,Z1).

/\* Z1 состоит из головы X и хвоста Z\*/

add(X,T,[X|T]).

/\* добавление элемента X в голову списка T \*/

nomemb(N,X):-

/\* элемент N не встречается в списке X **если** \*/

not memb(N,X).

/\* нет принадлежности элемента N списку X \*/

memb(N,[N|X]). /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* элемент N совпадает с головой списка \*/

memb(N,[Y|T]):-

/\* элемент N не совпадает с головой Y, следовательно\*/

memb(N,T).

/\* проверить принадлежность элемента N хвосту T \*/

spo([],[]). /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* для пустого исходного списка и список уровней тоже пустой \*/

spo([X1|Y1],[X1|Y3]):-

/\* голова исходного списка X1 становится головой результи-рующего **если** \*/

udal(X1,Y1,T),

/\* удалить повторы X1 в хвосте Y1 **и** \*/

spo([X1|T],[X1|Y3]);

/\* перенести элемент X1 в результирующий список **или \*/**

spisok([X1|Y1],[\_|Y2]),

/\* сформировать подсписок уровня из элемента **и** \*/

spo(Y2,Y3).

/\* сформировать список уровней \*/

spisok([],[]). /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* для пустого исходного и список уровней пустой \*/

spisok([X|Y],[[X]|Z]):-

/\* голова исходного списка становится подсписком [X] в резуль-тирующем списке если \*/

spisok(Y,Z).

/\* списки уровней хвост Y и Z \*/

udal(X,[X|T],T). /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* если элемент совпадает с головой, то результат удаления хвост \*/

udal(X,[Y|T],[Y|T1]):-udal(X,T,T1).

/\* в противном случае удалить элемент из хвоста \*/

**Комментированное тестирование программы**

Основная процедура

?- proc([1,2,3,4,2],[3,7,4,2],Res).

Res = [2,[3],[[4]]] ;

Res = [2,[3],[[4]],[[[2]]]] ;

Res = [1,[2],[[3]],[[[4]]]] ;

Res = [1,[2],[[3]],[[[4]]],[[[[2]]]]] ;

no

Вспомогательные процедуры

?- concatx([1,2,3,4],[2,6,4],Res).

Res = [2,4] ;

Искомое решение

Res = [2,3,4] ;

Res = [1,2,4] ;

Res = [1,2,3,4] ;

no

После обнаружения требуемого решения процедура может генерировать другие.

?- nomemb(3,[1,2,4,5]).

yes

?- memb(3,[1,2,4,5]).

no

?- add(1,[2,3,4],Res).

Res = [1,2,3,4] ;

no

?- spisok([[2]],Res).

Res = [[[2]]] ;

no

?- spo([1,2,3,4,2,5,4],Res).

Res = [1,[2],[[3]],[[[4]]],[[[[5]]]]] ;

Искомое решение

Res = [1,[2],[[3]],[[[4]]],[[[[5]]]],[[[[[4]]]]]] ;

Res = [1,[2],[[3]],[[[4]]],[[[[2]]]],[[[[[5]]]]]] ;

Res = [1,[2],[[3]],[[[4]]],[[[[2]]]],[[[[[5]]]]],[[[[[[4]]]]]]] ;

no

После обнаружения требуемого решения процедура может генери-ровать другие

?- udal(2,[3,2,1,4,2,5],Res).

Res = [3,1,4,2,5] ;

Искомое решение

Res = [3,2,1,4,5] ;

После обнаружения требуемого решения процедура может генерировать другие

no

**Задание №2.** Разработать вариант программы с использованием средств управления в Прологе.

**Текст задания.**

|  |  |
| --- | --- |
| Формулировка задачи(действия, которые должна выполнять процедура «proc») | Пример обращения к процедуре |
| Формировать список являющийся слиянием двух исходных, состоящий только из совпадающих элементов, а каждый элемент является подсписком уровня, соответствующего его месту, исключая повторы элементов. | ?-proc([1,2,3,4,2,5],[3,77,4,23,2],Res).L= [23,[3],[[4]],[[[6]]]] ;no |

**Введение.** Под средствами управления в Пролог - программе понимается отсечение. Отсечение позволяет прекратить перебор, когда искомое решение найдено и дальнейший поиск не нужен или исключить варианты, которые не приведут к требуемому решению. В решении варианта №2 требуется применить механизм отсечения в процедурах с целью повышения эффективности решения. Разрабатываемая процедура должна иметь арность 3. Все параметры процедуры – списки. Первый и второй параметры предназначены для передачи исходных списков, третий соответствует формируемому списку.

**Декларативное описание решения.** Аналогично предыдущему варианту. В ряде вспомогательных процедур, которые были разработаны в задании №1, можно использовать отсечение для предотвращения ненужного поиска. Такие процедуры после нахождения требуемого решения продолжали генерировать новые варианты.

**Комментированный текст программы.** Комментарий записывается после соответствующего утверждения программы.

**/\* Разрабатываемая процедура \*/**

proc(List,List1,Result):-

/\* искомый список Result как слияние списков List и List1 если \*/

concatх(List,List1,R\_prom),

/\* R\_prom – слияние общих элементов списков List,List1 и \*/

spi(R\_prom,Result).

/\* Result – список подуровней списка R\_prom \*/

**/\* Вспомогательные процедуры \*/**

concatx([],Z,[]). /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* Результат слияния пустого списка со списком Z – пустой спи-сок\*/

concatx([X1|Y1],X2,Z1):-

/\* первый список состоит из головы X1 и хвоста X2, а результирующий - Z1\* **если** \*/

nomemb(X1,X2),

/\* элемент X1 не встречается в списке X2 **и** \*/

concatx(Y1,X2,Z1);

/\* Z1- объединение списков Y1 и X2 **или** \*/

concatx(Y1,X2,Z),

/\* Z – объединение списков Y1 и X2 **и** \*/

add(X1,Z,Z1).

/\* Z1 состоит из головы X и хвоста Z\*/

add(X,T,[X|T]).

/\* добавление элемента X в голову списка T \*/

nomemb(N,X):-

/\* элемент N не встречается в списке X **если** \*/

not memb(N,X).

/\* нет принадлежности элемента N списку X \*/

memb(N,[N|X]). /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* элемент N совпадает с головой списка \*/

memb(N,[Y|T]):-

/\* элемент N не совпадает с головой Y следовательно\*/

memb(N,T).

/\* проверить принадлежность элемента N хвосту T \*/

spo([],[]). /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* для пустого исходного списка и список уровней тоже пустой \*/

spo([X1|Y1],[X1|Y3]):-

/\* голова исходного списка X1 становится головой результи-рующего **если** \*/

udal(X1,Y1,T),!,

/\* удалить повторы X1 в хвосте Y1 **и** прекратить перебор\*/

spo([X1|T],[X1|Y3]);

/\* перенести элемент X1 в результирующий список **или \*/**

spisok([X1|Y1],[\_|Y2]),

/\* сформировать подсписок уровня иp элемента **и** \*/

spo(Y2,Y3).

/\* сформировать список уровней \*/

spisok([],[]). /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* для пустого исходного и список уровней пустой \*/

spisok([X|Y],[[X]|Z]):-

/\* голова исходного списка становится подсписком [X] в резуль-тирующем списке если \*/

spisok(Y,Z).

/\* списки уровней хвост Y и Z \*/

udal(X,[X|T],T):-!. /\* НАЧАЛЬНЫЕ УСЛОВИЯ РЕКУРСИИ \*/

/\* если элемент совпадает с головой то результат удаления хвост и прекратить перебор \*/

udal(X,[Y|T],[Y|T1]):-udal(X,T,T1)./\* в противном случае удалить элемент из хвоста \*/

**Комментированный результат тестирования**

?- proc([1,2,3,4,2],[3,7,4,2],Res).

Res = [2,[3],[[4]]] ;

no

Искомое решение

?- concatx([1,2,3,4],[2,6,4],Res).

Res = [2,4] ;

no

Искомое решение

?- nomemb(3,[1,2,4,5]).

yes

?- memb(3,[1,2,4,5]).

no

?- spisok([[2]],Res).

Res = [[[2]]] ;

no

?- spo([1,2,3,4,2,5,4],Res).

Res = [1,[2],[[3]],[[[4]]],[[[[5]]]]] ;

no

Искомое решение

?- udal(2,[3,2,1,4,2,5],Res).

Res = [3,1,4,2,5] ;

no

Искомое решение