,

Оглавление

1 39 [4] Дописать заданную константу в конец списка N раз

1.1 Постановка задачи

Дописать заданную константу в конец списка N раз

1.2 Исходный код программы

```
:- initialization(main).

%% Добавиьт элемент в список
append_n_times([], _, 0, []).
append_n_times([], Elem, Repeats, [Elem|TailOut]) :-
    NewRepeats is Repeats - 1,
    append_n_times([], Elem, NewRepeats, TailOut).

append_n_times([Head|Tail], Elem, Repeats, [Head|TailOut]) :-
    append_n_times(Tail, Elem, Repeats, TailOut).

append_n_times_print(List, Elem, Repeats) :-
    append_n_times(List, Elem, Repeats, OutList),
    writeln(OutList).

test :-
    append_n_times_print([], 1, 2),
    append_n_times_print([], -1, 9),
    append_n_times_print([12,3], 1, 2)
.


run_main :-
    writeln("Введите список"), readlist(List),
    writeln("Введите элемент для повториения"), readint(Elem),
    writeln("Введите количество повторений"), readint(Repeats),
    append_n_times_print(List, Elem, Repeats), interactive.


main :-
    ensure_loaded("../common.splog"),
    writeln("39 [4] Дописать заданную константу в конец списка N раз"),
    startup_notice,
    ( non_interactive_test ; interactive ).
```

1.3 Примеры работы программы

```
39 [4] Дописать заданную константу в конец списка N раз
Все введенные строки должны оканичиваться точкой
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите список
|: [1,2,3].
Введите элемент для повториения
|: 0.
Введите количество повторений
|: 11.
[1,2,3,0,0,0,0,0,0,0,0,0,0,0]
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
```

2 78 [8] Квадрат 0-1 матрицы

2.1 Постановка задачи

Квадрат 0-1 матрицы

2.2 Исходный код программы

```
:- initialization(main).

%% TODO

%% Вывести одну строку матрицы
print_row([]).
print_row([Head|Tail]) :-
    write(Head), write(" "),
    print_row(Tail).

%% Вывести матрицу
print_matrix([]).
print_matrix([FirstRow|Others]) :-
    write("[ "), print_row(FirstRow), writeln("]"),
    print_matrix(Others).

%% Получить первую колонку матрицы
first_col([], [], []).
first_col(
    [
[ Cell
  | RowTail %% Хвост результирующей строки
] | OtherRows %% Хвост матрицы будет использован для получения
      %% остальных частей результирующей строки.
    ], [
Cell %% Первый столбец первой строки равен первому элементу
     %% первой колонки и первому элементу результирующей
     %% строки
| ResRowTail %% Хвост результирующей строки равен первой
     %% колонке хвоста входной матрицы.
    ] , [
RowTail
| XXXVarName
    ] ) :-
    first_col(
OtherRows,
ResRowTail,
XXXVarName
    ).
```

```
%% Транспонировать матрицу
transpose([[]|_], []).
transpose([],[]).
transpose(InMatrix, [FirstRow|TailRows]) :-
    first_col( %% Находим первую колонку в матрице.
InMatrix,
FirstRow, %% Так как ‘FirstRow‘ было использовано как голова
%% результирующей матрицы то первая колонка входной
%% матрицы будет приравнена к первой строке выходной
%% матрицы

RestMatrix %% Оставшиеся колонки будут рекурсивно добавлены в
%% хвост результирующей матрицы.
    ),
    transpose(RestMatrix, TailRows).


%% Произвести логическое сложение
logic_mult(1, 1, 1).
logic_mult(_, _, 0).
%% Произвести логическое умножение
logic_add(0, 0, 0).
logic_add(_, _, 1).

%% Вычислить логическое произведение векторов (скалярное произведение
%% в покоординатной форме используя логические сложение и умножение)
cross_mult_list([Head1], [Head2], Out) :-
    logic_mult(Head1, Head2, Out).

cross_mult_list([Head1|Tail1], [Head2|Tail2], Out) :-
    cross_mult_list(Tail1, Tail2, TailProd),
    logic_mult(Head1, Head2, HeadProd),
    logic_add(TailProd, HeadProd, Out)
.

%% Умножить вектор на матрицу
cross_mult_vec_matrixT(Vector, [LastRow], [Out]) :-
    cross_mult_list(Vector, LastRow, Out).

cross_mult_vec_matrixT(Vector, [Row|OtherRows], [HeadOut|TailOut]) :-
    cross_mult_list(Vector, Row, HeadOut),
    cross_mult_vec_matrixT(Vector, OtherRows, TailOut)
.

%% Умножить матрицу на транспонированную матрицу
```

```
mult_matrix_matrixT(
    [LastTail],
    MatrTransp,
    [OutTail]) :-
    cross_mult_vec_matrixT(LastTail, MatrTransp, OutTail).

mult_matrix_matrixT(
    [MHead  | MTail  ],
    MatrTransp,
    [RHead  | RTail  ]
) :-
    mult_matrix_matrixT(MTail, MatrTransp, RTail),
    cross_mult_vec_matrixT(MHead, MatrTransp, RHead),

    true.


%% Вычислиьт квадрат матрицы
matrix_square(Matrix, Res) :-
    transpose(Matrix, Transposed),
    mult_matrix_matrixT(Matrix, Transposed, Res)
.

test_mult_matrix_matrixT(M1, M2) :-
    writeln("matrix-matrix test"),
    writeln("in:"),
    print_matrix(M1),
    writeln("---"),
    print_matrix(M2),
    writeln("out:"),
    %% trace(cross_mult_vec_matrixT),
    %% trace(mult_matrix_matrixT),
    mult_matrix_matrixT(M1, M2, Out),
    %% writeln(Out),
    print_matrix(Out),
true.

test_vec_mult_matrix(Vec, Matrix) :-
    writeln("vector-matrix test"),
    write("in: "), writeln(Vec),
    print_matrix(Matrix),
    cross_mult_vec_matrixT(Vec, Matrix, Out),
    writeln("out:"),
    writeln(Out),
true.
```

```
test_matrix_transpose(M) :-
    writeln("transpose test"),
    writeln("in:"),
    print_matrix(M),
    transpose(M, T),
    writeln("out:"),
    print_matrix(T),
true.

test_cross_mult_list(List1, List2) :-
    writeln("list-list test"),
    write(List1),
    write("  "),
    write(List2),
    write(" = "),
    cross_mult_list(List1, List2, Out),
    writeln(Out)
.

test_matrix_square(Matr) :-
    writeln("matrix square test"),
    writeln("in:"),
    print_matrix(Matr),
    %% trace(matrix_square),
    %% trace(transpose),
    matrix_square(Matr, Out), !,
    writeln("out:"),
    print_matrix(Out),
true.

%% Проверка на равенство строк по длинне
equal_rows([], _).
equal_rows([Row], PrevLen) :- length(Row, PrevLen).
equal_rows([Row|Tail], PrevLen) :-
    ( var(PrevLen) ; length(Row, PrevLen) ),
    length(Row, PrevLen),
    equal_rows(Tail, PrevLen).

%% Проверка на квадратность (количество строк должно быть равно
%% количеству столбцов)
is_square([Row|Tail], Len) :-
    length([Row|Tail], Len),
    length(Row, Len).

to_boollist([], []).
to_boollist([Head|Tail], [HeadOut|TailOut]) :-
```

```
    ( ( Head = 0, HeadOut = 0 ) ; HeadOut = 1 ),
    to_boollist(Tail, TailOut).


to_boolean([], []).
to_boolean([Row|Tail], [RowOut|TailOut]) :-
to_boollist(Row, RowOut),
to_boolean(Tail, TailOut).

%% Считать квадратную матрицу, провести проверки и вывести сообщения
%% об ошибках (если требуется)
read_square_matrix(Out) :-
    read(InList),
    ( equal_rows(InList, _) ;
      writeln_err("Все строки должны иметь одинаковую длинну") , fail ), !,
    ( is_square(InList, _) ;
      writeln_err("Требуется ввести квадратную! матрицу"), fail ),
    to_boolean(InList, Out),
true.

%% Запуск интерактивной части
run_main :-
    writeln(
"Введите квадратную булеву матрицу как вложенный список
(например: [[1,1], [1,1]]) Элементами матрицы должны быть
числа, любое ненулевое число будет интерпретрованно как 1"),
    read_square_matrix(InMatr),
    writeln("Вы ввели:"),
    print_matrix(InMatr),
    matrix_square(InMatr, Square),
    writeln("Результат:"),
    print_matrix(Square),
    interactive.

test :-
    test_matrix_transpose([[1]]), !,
    test_matrix_transpose([[0,0], [1,1]]), !,
    %% test_mult_matrix_matrixT([[0,0], [0,0]], [[0,0], [0,0]]),
    %% test_mult_matrix_matrixT([[1,1], [1,1]], [[1,1], [1,1]]),
    test_matrix_square([[1,1], [0,0]]), !,
    test_matrix_square([[1,1,1], [0,0,0], [1,1,1]]), !,
    test_matrix_square([[1,1,0,0], [1,1,0,0], [1,0,1,1], [0,0,0,1]]), !,
halt,
true.
```

```
main :-
    ensure_loaded("../common.splog"),
    writeln_inf("78 [8] булев квадрат матрицы"),
    startup_notice,
    ( non_interactive_test ; interactive ).
```

2.3 Примеры работы программы

```
Inf: 78 [8] булев квадрат матрицы
Все введенные строки должны оканичиваться точкой
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите квадратную булеву матрицу как вложенный список
(например: [[1,1], [1,1]]) Элементами матрицы должны быть
числа, любое ненулевое число будет интерпретированно как 1
|: [[1,0], [0,1]].
Вы ввели:
[ 1 0 ]
[ 0 1 ]
Результат:
[ 1 0 ]
[ 0 1 ]
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите квадратную булеву матрицу как вложенный список
(например: [[1,1], [1,1]]) Элементами матрицы должны быть
числа, любое ненулевое число будет интерпретированно как 1
|: [[]].
Err: Требуется ввести квадратную! матрицу
```

3 75.2 [6] циклический сдвиг списка вправо

3.1 Постановка задачи

75.2 [6] циклический сдвиг списка вправо

3.2 Исходный код программы

```
:- initialization(main).

append([], OtherList, OtherList).

append([OutHead|InTail], OtherList, [OutHead|OutTail]) :-
    append(InTail, OtherList, OutTail).

split_rhead([Head], Head, []).

split_rhead([Head|Tail], RHeadOut, RTailOut) :-
    split_rhead(Tail, RHeadOut, RLTail),
    append([Head], RLTail, RTailOut).

cyclic_shift([], []).
cyclic_shift([Head], [Head]).
cyclic_shift(List, OutList) :-
    split_rhead(List, RHead, RTail),
    append([RHead], RTail, OutList).

shift_print(List) :-
    write(List), write(" --> "),
    cyclic_shift(List, Out), writeln(Out).

run_main :-
    writeln("Введите список"), readlist(List),
    cyclic_shift(List, Out),
    write(" --> "), writeln(Out),
    interactive.

test :-
    %% trace(split_rhead),
    %% trace(append),
    shift_print([1,2,3]),
    shift_print([1,2]),
    shift_print([1]),
    shift_print([]),
    shift_print([1,2,3,4,5,0])
.
```

```
main :-
    ensure_loaded("../common.splog"),
    writeln_inf("75.2 [6] циклический сдвиг списка вправо"),
    startup_notice,
    ( non_interactive_test ; interactive ).
```

3.3 Примеры работы программы

```
Inf: 75.2 [6] циклический сдвиг списка вправо
Все введенные строки должны оканичиваться точкой
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите список
|: [1,2,3].
 --> [3,1,2]
```

4 23 [2] Получить последнее значение в списке

4.1 Постановка задачи

Получить последнее значение в списке

4.2 Исходный код программы

```
:- initialization(main).

get_last([Head], Head).
get_last([_|Tail], Elem) :- get_last(Tail, Elem).

get_last_print([]) :- writeln("-> ").
get_last_print(List) :- get_last(List, Out), write("-> "), writeln(Out).

test :-
    get_last_print([1,2,3]),
    get_last_print([1])
.

run_main :-
    writeln("Введите список"), readlist(List),
    get_last_print(List), interactive.

main :-
    ensure_loaded("../common.splog"),
    writeln("23 [2] Получить последнее значение в списке"),
    startup_notice,
    ( non_interactive_test ; interactive ).
```

5 61 [6] Указать характер упорядоченности списка

5.1 Постановка задачи

Указать характер упорядоченности списка

5.2 Исходный код программы

```
:- initialization(main).


strictly_increasing([Head, Last]) :- Head < Last.
strictly_increasing([Head1, Head2|Tail]) :-
    Head1 < Head2,
    strictly_increasing([Head2|Tail]).


strictly_decreasing([Head, Last]) :- Head > Last.
strictly_decreasing([Head1, Head2|Tail]) :-
    Head1 > Head2,
    strictly_decreasing([Head2|Tail]).

non_increasing([]).
non_increasing([_]).
non_increasing([Head, Last]) :- Head =< Last.
non_increasing([Head1, Head2|Tail]) :-
    Head1 =< Head2,
    non_increasing([Head2|Tail]).

non_decreasing([]).
non_decreasing([_]).
non_decreasing([Head, Last]) :- Head >= Last.
non_decreasing([Head1, Head2|Tail]) :-
    Head1 >= Head2,
    non_decreasing([Head2|Tail]).


check_list(List) :-
    write("---> "),
    (
( strictly_increasing(List), writeln("Строго возрастает")) ;
( strictly_decreasing(List), writeln("Строго убывает")) ;
(
    non_increasing(List),
    non_decreasing(List),
    writeln("Не возрастает и не убывает (постоянный)")
) ;
```

```
( non_increasing(List), writeln("Невозрастающий")) ;
( non_decreasing(List), writeln("Неубывающий")) ;
( writeln("Нельзя определить") )
    ), nl.

test :-
    check_list([1, 2, 3]),
    check_list([1, -1, -2]),
    check_list([0, 0, 0]),
    check_list([0, 0, 1]).


run_main :-
    writeln("Введите список"), readlist(List),
    check_list(List),
    interactive.

main :-
    ensure_loaded("../common.splog"),
    writeln("61 [6] Указать характер упорядоченности списка"),
    startup_notice,
    ( non_interactive_test ; interactive ).
```

5.3 Примеры работы программы

```
61 [6] Указать характер упорядоченности списка
Все введенные строки должны оканичиваться точкой
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите список
|: [1,2,3,3].
---> Невозрастающий

Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите список
|: [1,1,1,1].
---> Не возрастает и не убывает (постоянный)
```

6 54 [6] Удалить K элементов начиная с индекса N

6.1 Постановка задачи

Удалить K элементов начиная с индекса N

6.2 Исходный код программы

```
:- initialization(main).

remove_nk([], _, _, []).
remove_nk(List, 0, 0, List).

remove_nk([_|Tail], 0, K, ListOut) :-
    NewK is K - 1,
    remove_nk(Tail, 0, NewK, ListOut).

remove_nk(List, N, K, ListOut) :-
    NewN is N - 1,
    remove_nk(List, NewN, K, ListOut).

remove_nk_print(List, N, K) :-
    write(List), write(" N:"),
    write(N), write(" K:"),
    write(K),  write(" --> "),
    call_with_depth_limit(remove_nk(List, N, K, Out), 100, _),
    writeln(Out).

test :-
    %% trace(remove_nk),
    remove_nk_print([1], 2, 10),
    remove_nk_print([1,2,3], 10, 90),
    remove_nk_print([], 1, 2),
    remove_nk_print([1,2,3], 90, 100),
    remove_nk_print([1,2,3], 1, 2),
    remove_nk_print([], 1, 2),
    remove_nk_print([12], 0, 1)
.

run_main :-
    writeln("Введите список"), readlist(List),
    writeln("Введите индекс начального элемента"), readint(N),
    writeln("Введите количество элементов для удаления"), readint(K),
    remove_nk_print(List, N, K),
    interactive.
```

```
main :-
    ensure_loaded("../common.splog"),
    writeln("54 [6] Удалить K элементов начиная с индекса N"),
    startup_notice,
    ( non_interactive_test ; interactive ).
```

6.3 Примеры работы программы

```
54 [6] Удалить K элементов начиная с индекса N
Все введенные строки должны оканичиваться точкой
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите список
|: [1,2,3].
Введите индекс начального элемента
|: 1.
Введите количество элементов для удаления
|: 2.
[1,2,3] N:1 K:2 --> [3]
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: t.
[1] N:2 K:10 --> []
[1,2,3] N:10 K:90 --> []
[] N:1 K:2 --> []
[1,2,3] N:90 K:100 --> []
[1,2,3] N:1 K:2 --> [3]
[] N:1 K:2 --> []
[12] N:0 K:1 --> []
```

7 28 [2] Удалить элемент по индексу из списка

7.1 Постановка задачи

28 [2] Удалить элемент по индексу из списка

7.2 Исходный код программы

```
:- initialization(main).

remove_elem([_], Idx, []) :- Idx = 0.

remove_elem([_|Tail], Idx, ListOut) :-
    Idx = 0, ListOut = Tail.

remove_elem([Head|Tail], Idx, [HeadOut|TailOut]) :-
    Idx > 0,
    HeadOut = Head,
    NewIdx is Idx - 1,
    remove_elem(Tail, NewIdx, TailOut).

remove_element_print(InputList, Index) :-
    ( remove_elem(InputList, Index, OutList), writeln(OutList) ) ;
    (
write("Невозможно удалить элемент с индексом "),
write(Index), write(" . Список: "), writeln(InputList)
    ).

test :-
    remove_element_print([1,2,3,4], 3),
    remove_element_print([1,2,3,4], 5)
.

run_main :-
    writeln("Введите список"), readlist(List),
    writeln("Введите индекс"), readint(Int),
    remove_element_print(List, Int),
    interactive.


main :-
    ensure_loaded("../common.splog"),
    writeln_inf("28 [2] Удалить элемент по индексу из списка"),
    startup_notice,
    ( non_interactive_test ; interactive ).
```

7.3 Примеры работы программы

```
Inf: 28 [2] Удалить элемент по индексу из списка
Все введенные строки должны оканичиваться точкой
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите список
|: [1,2,3].
Введите индекс
|: 1.
[1,3]
```

8 33 [4] Вычислить число число элементов

списка после элемента с заданным значанием

8.1 Постановка задачи

Вычислить число число элементов списка после элемента с заданным значанием

8.2 Исходный код программы

```
:- initialization(main).

count_list([], 0).
count_list([_], 1).

count_list([_|Tail], Out) :-
    count_list(Tail, TailCount),
    Out is 1 + TailCount.

count_elems_after([], _, 0).

count_elems_after([Head|Tail], Value, OutSum) :-
    Head =\= Value,
    count_elems_after(Tail, Value, OutSum).

count_elems_after([Head|Tail], Head, OutSum) :-
    count_list(Tail, OutSum).


count_elems_after_print(InputList, Value) :-
    count_elems_after(InputList, Value, Out),
    write("-->"),
    writeln(Out).

test :-
    count_elems_after_print([1,2,3,4], 3),
    count_elems_after_print([1,2,3,4], 12),
    count_elems_after_print([1,2,3,4], 1).

run_main :-
    writeln("Введите список"), readlist(List),
    writeln("Введите элемент"), readint(Int),
    count_elems_after_print(List, Int),
    interactive.

main :-
    ensure_loaded("../common.splog"),
```

```
writeln("33 [4] Вычислить число число элементов списка после элемента с заданным знача
startup_notice,
( non_interactive_test ; interactive ).
```

8.3 Примеры работы программы

```
33 [4] Вычислить число число элементов списка после элемента с заданным значанием
Все введенные строки должны оканичиваться точкой
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите список
|: [1,2,3,4].
Введите элемент
|: 1.
-->3
```

9 49 [6] Определить номер позиции с которой

начинается заданный подсписок

9.1 Постановка задачи

Определить номер позиции с которой начинается заданный подсписок

9.2 Исходный код программы

```
:- initialization(main).

algo([H|_], [H], Idx, Idx).

algo([_|T], [H], Idx,  Out) :-
    NewIdx is Idx + 1,
    algo(T, [H], NewIdx, Out).

algo(
    [Head1,Head2],
    [Head1,Head2],
    Idx,
    Idx).

algo(
    [Head1,Head2|Tail1],
    [Head1,Head2|Tail2],
    Idx,
    Out
) :-
    algo(
[Head2|Tail1],
[Head2|Tail2],
Idx,
Out
    ).

algo([_|Tail1], Sublist, Idx, Out) :-
    NewIdx is Idx + 1,
    algo(Tail1, Sublist, NewIdx, Out).

find_sublist_print(List, Sublist) :-
    write(List), write(" - "),
    write(Sublist), write(" -> "),
    %% (
    %%     List = Sublist,
    %%     List = [],
```

```
    %%     writeln("0")
    %% ) ;
    (
%% trace(algo),
algo(List, Sublist, 1, Start),
writeln(Start)
    ) ;
    (
writeln("Исходный список не содержит подсписка")
    )
.

test :-
    find_sublist_print( [1,2     ], [1   ] ), !,
    find_sublist_print( [1,2     ], [2   ] ), !,
    find_sublist_print( [1,2     ], [1,2 ] ), !,
    find_sublist_print( [1,2,3,4 ], [3,4 ] ), !,
    find_sublist_print( [9,109   ], [1   ] ), !,
    find_sublist_print( [1,1,1   ], [1   ] ), !,
    find_sublist_print( [        ], [1   ] ), !,
    find_sublist_print( [12,2    ], [    ] ), !,
    find_sublist_print( [1,2,3   ], [1,3 ] ), !,
    find_sublist_print( [1,2,3,4 ], [1,4 ] ), !,
    %%trace(algo),
    %% gtrace,
    find_sublist_print( [1,2,3   ], [1,2 ] ), !,
    find_sublist_print( [1,2,3,1,2   ], [1,2 ] ), !,

halt.


run_main :-
    writeln("Введите список"), readlist(List),
    writeln("Подсписок"), readlist(Sublist),
    find_sublist_print(List, Sublist),
    interactive
.


main :-
    ensure_loaded("../common.splog"),
    writeln("49 [6] Определить номер позиции с которой начинается заданный подсписок"),
    startup_notice,
    ( non_interactive_test ; interactive ).
```

9.3 Примеры работы программы

```
49 [6] Определить номер позиции с которой начинается заданный подсписок
Все введенные строки должны оканичиваться точкой
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите список
|: [1,2,3].
Подсписок
|: [2].
[1,2,3] - [2] -> 1
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: t.
[1,2] - [1] -> 0
[1,2] - [2] -> 1
[1,2] - [1,2] -> 0
[1,2,3,4] - [3,4] -> 2
[9,109] - [1] -> Исходный список не содержит подсписка
[1,1,1] - [1] -> 0
[] - [1] -> Исходный список не содержит подсписка
[12,2] - [] -> Исходный список не содержит подсписка
```

10 85 [4] Найти максимальный и минимальные элемент в бинарном справочнике

10.1 Постановка задачи

85 [4] Найти максимальный и минимальные элемент в бинарном справочнике

10.2 Исходный код программы

```
:- initialization(main).

%% Добавить элемент в словарь
insert_item(
    tree(Root, nil, nil),
    Item,
    tree(Root, Item, nil)
) :-
    Item < Root
.

insert_item(
    tree(Root, nil, nil),
    Item,
    tree(Root, nil, Item)
) :-
    Item > Root
.

insert_item(
    tree(Root, nil, nil),
    Root,
    tree(Root, nil, nil)
).

insert_item(
    tree(Root, Left, Right),
    Item,
    tree(Root, NewLeft, Right)
) :-
    Item < Root,
    insert_item(Left, Item, NewLeft)
.

insert_item(
    tree(Root, Left, Right),
    Item,
```

```
    tree(Root, Left, NewRight)
) :-
    Item > Root,
    insert_item(Right, Item, NewRight)
.

insert_item(
    tree(Root, Left, Right),
    Root,
    tree(Root, Left, Right)
).

insert_item(nil, nil, nil).
insert_item(Root, nil, Root).
insert_item(nil,  Item, tree(Item, nil, nil)).
insert_item(Root, Item, tree(Root, Item, nil)) :- Item < Root.
insert_item(Root, Item, tree(Root, nil, Item)) :- Item > Root.
insert_item(Root, Root, Root).

%% Добавить все элементы из списка в словарь
insertall(Tree, [], Tree).
insertall(Tree, [Head|Tail], OutTree) :-
    insert_item(Tree, Head, NewTree),
    insertall(NewTree, Tail, OutTree).

%% Найти максимальный элемент в списке
find_max(tree(Root, _, nil), Root).
find_max(tree(_, _, Right), Out) :- find_max(Right, Out).
find_max(Item, Item).

%% Найти минимальный элемент в списке
find_min(tree(Root, nil, _), Root).
find_min(tree(_, Left, _), Out) :- find_min(Left, Out).
find_min(Item, Item).

%% Вывести дерево растущив внизу
%%print_tree_down(Tree) :-
%%    height(Tree, Height),


%% Вывести дерево
print_tree(tree(Root, nil, nil), Level) :-
    print_ident(Root, Level).

print_tree(tree(Root, Left, nil), Level) :-
    NewLevel is Level + 1,
```

```
    print_tree(Root, Level),
    print_tree(Left, NewLevel)
.

print_tree(tree(Root, nil, Right), Level) :-
    NewLevel is Level + 1,
    print_tree(Right, NewLevel),
    print_tree(Root, Level)
.

print_tree(tree(Root, Left, Right), Level) :-
    NewLevel is Level + 1,
    print_tree(Right, NewLevel),
    print_ident(Root, Level),
    print_tree(Left, NewLevel).


print_tree(Item, Level) :-
    print_ident(Item, Level).

print_ident(Item, 0) :- writeln(Item).
print_ident(Item, 1) :- write("|=== "), writeln(Item).
print_ident(Item, Level) :-
    write("     "),
    NewLevel is Level - 1,
    print_ident(Item, NewLevel).

%% Тестирование добавление элемента в словарь
test_insert_item(Tree, Item) :-
    insert_item(Tree, Item, Out),
    writeln(Out).

%% Тестирование построения
test_insertall(Tree, Items) :-
    insertall(Tree, Items, Out),
    print_tree(Out, 0),
    test_find_max(Out),
    test_find_min(Out)
.

%% Построить бинарное словарь из списка
make_tree([Head|Tail], OutTree) :-
    Tree = tree(Head, nil, nil),
    insertall(Tree, Tail, OutTree).

%% Найти максимаальный элемент и напечатать результат
```

```
test_find_max([Head|Tail]) :-
    Tree = tree(Head, nil, nil),
    insertall(Tree, Tail, NewTree),
    test_find_max(NewTree).

test_find_max(Tree) :-
    find_max(Tree, Out),
    writeln(Out).

%% Найти минимальный элемент и напечатать результат
test_find_min([Head|Tail]) :-
    Tree = tree(Head, nil, nil),
    insertall(Tree, Tail, NewTree),
    test_find_min(NewTree).

test_find_min(Tree) :-
    find_min(Tree, Out),
    writeln(Out).


test :-
    (
test_insert_item(9, 9),
test_insert_item(nil, nil),
test_insert_item(tree(1,2,3), 9),
test_insert_item(tree(1,nil, 1), 2),
test_insertall(4, [1,2,34, 3, 4,5,6,7,8, 999]),
test_find_max([1,2,3,4]),
test_find_min([5,6,7,9, -1]),
test_find_max(tree(9,0, 99)),
true
    ) ;

( true, halt ).


run_main :-
    writeln("Введите список элементов из которых будет построен бинарный справочник"),
    writeln("Введите список"), readlist(List),
    make_tree(List, Tree),
    find_max(Tree, Max),
    find_min(Tree, Min),
    writeln("Построенный бинарный справочник:"),
    print_tree(Tree, 1),
    write("Максимальный элемент: "), writeln(Max),
    write("Минимальный элемент: "), writeln(Min),
```

```
    interactive.


main :-
    ensure_loaded("../common.splog"),
    writeln("85 [4] Найти максимальный и минимальные элемент в бинарном справочнике"),
    startup_notice,
    ( non_interactive_test ; interactive ).
```

10.3 Примеры работы программы

```
85 [4] Найти максимальный и минимальные элемент в бинарном справочнике
Все введенные строки должны оканичиваться точкой
Inf: Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов
|: s.
Введите список элементов из которых будет построен бинарный справочник
Введите список
|: [6,7,2,1,45,34,1,4,5].
Построенный бинарный справочник:
    |---1
        |---1
|---2
    |---4
        |---5
6
|---7
        |---34
    |---45
Максимальный элемент: 45
Минимальный элемент: 1
```

11 Вспомогательный функции

11.1 Постановка задачи

Вспомогательные предикаты использоваанные в программах

11.2 Исходный код программы

```
%% Вывести сообщение с информацией для пользователя
writeln_inf(Message) :-
    ansi_format([bold, fg(green)], "Inf: ", []),
    writeln(Message).

%% Вывесте сообщение об ошибке
writeln_err(Message) :-
    ansi_format([bold, fg(red)], "Err: ", []),
    writeln(Message).

%% Вывести сообщение
writeln_log(Message) :-
    ansi_format([bold, fg(blue)], "Log: ", []),
    writeln(Message).

%% Запуск автоматических тестов и выход из программы
non_interactive_test() :-
    exists_file("no-interactive") ,
    writeln_log("running in non-interactive mode") ,
    test,
    writeln_log("done"),
    halt.

%% Запуск программы в интерактивном режиме
run_interactively() :-
    writeln_log("Running in interactive mode"),
    writeln_inf("Каждая введенная строка должна оканчиваться точкой."),
    interactive.

%% Обработка пользовательского ввода и интерактивный выбор действий
interactive :-
    writeln_inf(
"Введите q для прекращения работы программы,
s для начала/продолжения работы или
t для выполнения автоматических тестов"),
    readterm(Input),
    (( Input = q ; Input = s ; Input = t ) ,
     (
 ( Input = q, end_program ) ;
```

```
 ( Input = s, run_main ) ;
 ( Input = t, test ) ;
 interactive
     )
    ) ;
    interactive_fail.

startup_notice() :-
    ansi_format(
[bold, fg(red)],
"Все введенные строки должны оканичиваться точкой~n",
[]).

end_program :-
    ( exists_file("do-halt"), halt ) ;
    writeln_inf("done").

%% Перезапуск интерактивного режима в случае неправильного ввода
interactive_fail :-
    writeln_err("Ожидалось s или q"),
    interactive.


%% Считать список
readlist(List) :-
    read(List), (
is_list(List) ;
( writeln_err("Ввод не может быть распознан как лист"), fail )
    ).

% Считать целое число
readint(Int) :-
    ( read(Int), integer(Int) ) ;
    ( writeln_err("Ввод не может быть распознан как целое число"), fail ).

readterm(Term) :- read(Term).
```