|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА** – **Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных Технологий

Кафедра Вычислительной Техники (ВТ)

**Отчёт по практической работе**

по дисциплине

«Мат.логика и ТА»

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы | ИВБО-03-17 | Ковалев Д.Д. | |
| Ведущий преподаватель |  | Антик М.И. | |
| Зачтено | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2018 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  | *(подпись преподавателя)* |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2018 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  | *(подпись студента)* |

Москва 2018

**Задание с 1 занятия.**

**Условие задачи:**

Решение квадратичного уравнения.

**Код программы:**

predicates

nondeterm solve(real,real,real)*%функция подсчеты дискриминанта*

nondeterm reply(real,real,real)*%*

nondeterm run

nondeterm do(char)

clauses

solve(A,B,C):- *%если А не 0, то дискриминант считается по формуле*

A<>0,

D=B\*B-4\*A\*C,

reply(A,B,D).

solve(A,B,C):-*%если А*

A=0, B<>0,

P=-C/B,

write(" X = ", P).

solve(A,B,C):-*%если все значения 0, то Х – любое число,*

A=0, B=0, C=0,

write(" X - any ").

reply(A,B,**\_**):-*%вывод результата при А и В = 0 – то есть отсутствие решений*

A=0, B=0,

write ("no solution\n").

reply(A,B,C):-*%если А = 0, то X высчитывается по формуле*

A=0,

X = -C/B,

write("X = ", X).

reply(A,B,D):-*%если дискриминант меньше 0, то выводится в комплексной форме*

D<0,

Des = sqrt(-D)/(2\*A),

Resh = -B/(2\*A),

write("X1 = ", Resh, "-", Des, "i, X2 = ", Resh, "+", Des, "i").

reply(A,B,D):-*%если D=0, то X считается по такой формуле*

D=0,

X=-B/(2\*A), write("x=", X).

reply(A,B,D):-*%дискриминант считается по формуле*

X1=(-B+sqrt(D))/(2\*A),

X2=(-B-sqrt(D))/(2\*A),

write("x1=",X1,", x2=", X2), nl.

*%меню выбора*

**run**:-

write(" \*\*\*\* Enter \*\*\*\*\n"),

write("- 1 - to solve quadratic equation\n"),

write(" ax^2+bx+c=0\n"),

write("- 0 to EXIT\n"),

write("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"),

write(">>"), readchar(X), write(X), nl,

do(X),

run.

do('1'):-

write("a="), readreal(A), nl,

write("b="), readreal(B), nl,

write("c="), readreal(C), nl,

solve(A,B,C).

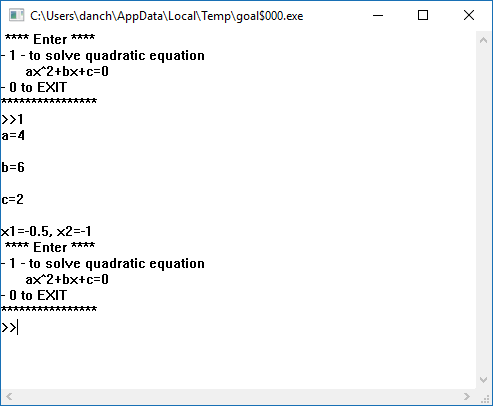
do('0'):-write(" - That's all."), exit.

do(**\_**):- write("no solution\n\n").

goal

run.

**Тесты:**



**Задание с 2 занятия.**

**Код программы:**

domains

i = integer *%целое число*

predicates

nondeterm counter(i,i,i,i) *%функция счетчика*

nondeterm run

nondeterm do(char)

clauses

counter(M,S,N,I):- *%функция вывода. Подается количество элементов (и максимальное число), модуль и начальная позиция. С начальной позиции идёт сложение, когда достигает максимальное число – происходит переход.*

write(N,". "),

NewN = (N+S) mod M,

NewI = I + 1, NewI < M,

counter(M,S,NewN,NewI);

nl.

**run**:-

nl,

write("\*\*\*Enter\*\*\*\n\n"),

write("- 1 - to start counter\n\n"),

write("- 0 - to EXIT\n\n"),

write("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"),

write("»"), readchar(X), write(X),nl,nl,

do(X),

run.

do('1'):-

write("MODUL» "), readint(M),

write("STEP» "), readint(S),

write("START» "), readint(N),

I=0,

counter(M,S,N,I).

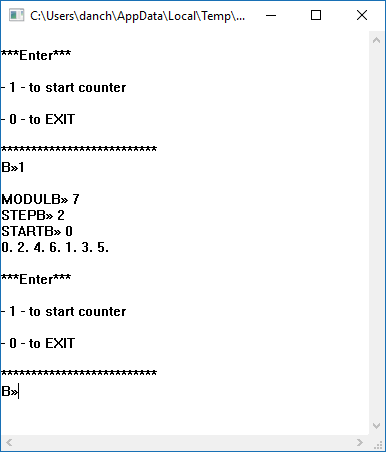
do('0'):- write(" - That`s all."), exit.

do(**\_**):- write("bad value\n\n").

goal

run.

**Тесты:**



**Задание с 3 занятия.**

**Код программы:**

Predicates

Nondeterm nok(real,real,real,real)

nondeterm run

nondeterm do(char)

Clauses

nok(OX,OY,X,Y):-*%если второе число меньше первого, то прибавляем к первому числу изначальное второе число*

OX>OY, NY = OY + Y, nok(OX,NY,X,Y).

nok(OX,OY,X,Y):-*%если первое число меньше второго, то к первому числу прибавляется первое изначальное число*

OX<OY, NX = OX + X, nok(NX,Y,X,Y).

nok(OX,OY,**\_**,**\_**):-*%когда они равны, то выводится число*

OX=OY, write("NOK = ",OX),nl.

**run**:- *%меню выбора*

nl,

write("\*\*\*Enter\*\*\*\n\n"),

write("- 1 - to start counter\n\n"),

write("- 0 - to EXIT\n\n"),

write("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"),

readchar(X),

do(X).

do('1'):-*%ввод чисел*

write("Write first number\n"),

write(">>"), readreal(X),nl,

write("Write second number\n"),

write(">>"), readreal(Y), nl,

nok(X,Y,X,Y),

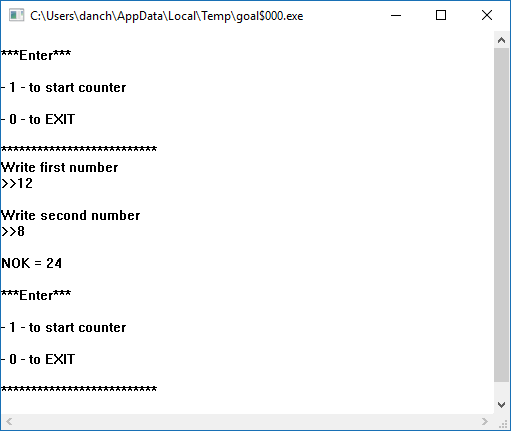
run.

do('0'):- write(" - That`s all."), exit.

do(**\_**):- write("bad value\n\n").

goal run.

**Тесты:**



**Задача №12.1**

**Условие задачи:**

Вычислить сумму элементов списка целых чисел, находящихся на чётных позициях.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% задание 12.2 - Вычислить сумму элементов списка, находящихся на нечётных позициях.*

nondeterm task\_12(list, i, i)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 12.1\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 12.1*

write("LIST: "), nl, readList(L),

write("LIST: ", L), nl,

task\_12(L, 1, Summ),

write("Summ of elements on even positions: ", Summ), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% задание 12.1 - Вычислить сумму элементов списка, находящихся на чётных позициях.*

*% На основе счётчика (N), который на каждом элементе увеличивается на 1, проверяем является*

*% ли текущая позиция нечётной (остаток от деления счётчика на 2 равен 0), и если да, то*

*% сумма будет равна текущему элементу + сумме элементов на нечётных позициях хвоста*

task\_12([], **\_**, 0).

task\_12([H|T], N, S) :-

N mod 2 = 0,

NewN = N + 1,

task\_12(T, NewN, TailS),

S = TailS + H.

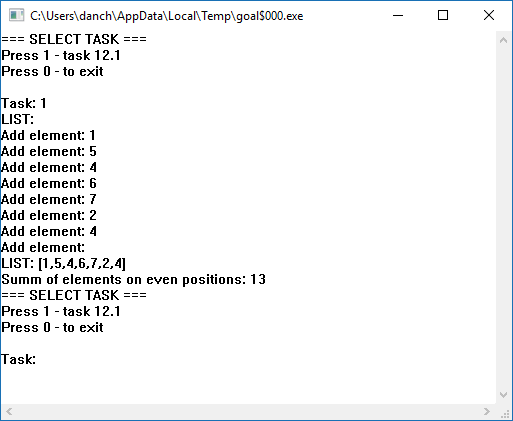
task\_12([**\_**|T], N, S) :-

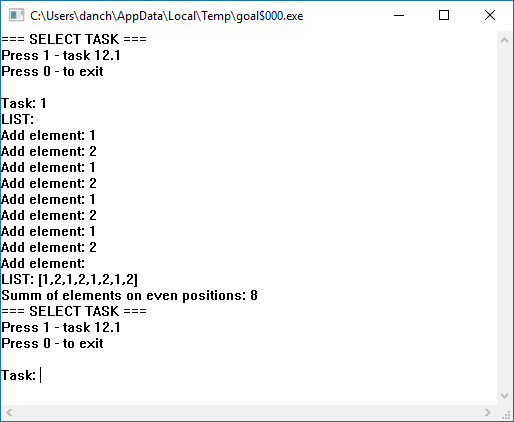
NewN = N + 1,

task\_12(T, NewN, S).

goal go.

**Тесты:**





**Задача №18.**

**Условие задачи:**

Удалить из списка все отрицательные элементы.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% задание 18 - Удалить отрицательные элементы списка*

nondeterm task\_18(list, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 18\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 18*

write("LIST: "), nl, readList(LIST), nl,

write("LIST: ", LIST), nl,

task\_18(LIST, NewList),

write("RESULT LIST: ", NewList), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% задание 18 - Удалить отрицательные элементы списка*

*% Если элемент списка >= 0, заносим его в список-результат*

*% Если нет - пропускаем. И повторяем для хвостов списков.*

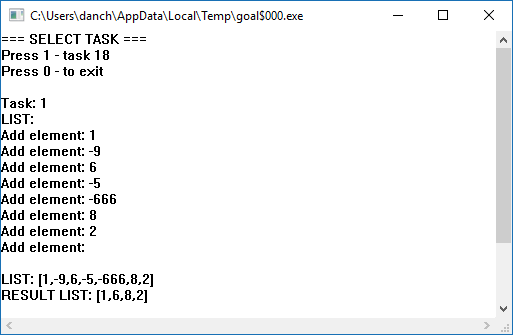
*% Для пустого списка результат - пуской список*

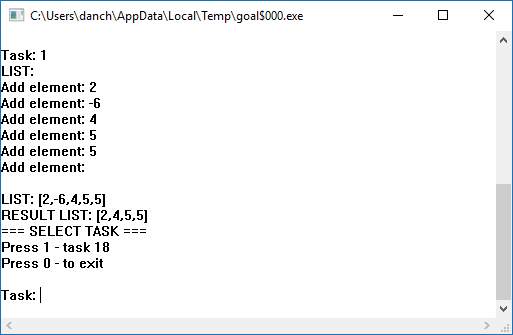
task\_18([H|T], [H|T2]) :- H >= 0, task\_18(T, T2).

task\_18([],[]).

goal go.

**Тесты:**





**Задача №19.**

**Условие задачи:**

Создать список из N начальных чисел ряда Фибоначчи.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% задание 19 - Создать список из N начальных чисел ряда Фибоначчи*

nondeterm task\_19(i, i, i, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 19\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 19*

write("N = "), readint(N),

task\_19(N, 0, 0, R),

write(R), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% задание 19 - Создать список из N начальных чисел ряда Фибоначчи*

*% если предыдущие числа равны 0 и 0, то мы находимся в начале ряда*

*% устанавливаем первое число в 1*

*% если предыдущие числа равны 0 и 1, то мы на втором элементе ряда*

*% а он тоже будет единицей*

*% для остальных случаев новое число равно сумме двух предыдущих*

*% при этом в рекурсию мы передаём уже второе предыдущее и текущее число*

*% N - счётчик, который уменьшается на 1 с каждым числом, и процесс остановится*

*% когда он будет равен 0 (вернётся пустой список)*

task\_19(0, **\_**, **\_**, []).

task\_19(N, 0, 0, [1|T]) :- N2 = N - 1, task\_19(N2, 1, 0, T).

task\_19(N, 1, 0, [1|T]) :- N2 = N - 1, task\_19(N2, 1, 1, T).

task\_19(N, A, B, [C|T]) :-

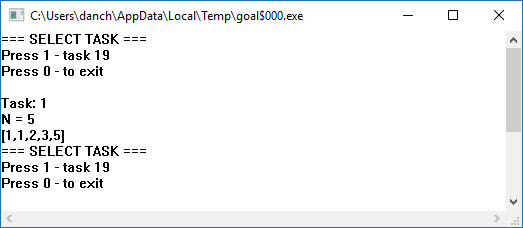
C = A + B,

N2 = N - 1,

task\_19(N2, B, C, T).

goal go.

Тесты:



Задача №26.

**Условие задачи:**

Заменить в исходном списке N-ый элемент заданной константой.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% задача 26 - Заменить в исходном списке N-ый элемент заданной константой.*

nondeterm task\_26(list, i, i, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 26\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 26*

write("List: "), nl, readList(L),

write("N = "), readint(N),

write("C = "), readint(C),

task\_26(L, N, C, R),

write("Result: ", R), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% задача 26 - Заменить в исходном списке N-ый элемент заданной константой.*

*% идём по списку и уменьшаем N на 1 на каждом элементе, и копируем элементы в конечный список*

*% и когда N = 0, значит, мы дошли до нужного элемента, помещаем в результат константу вместо исходного элемента*

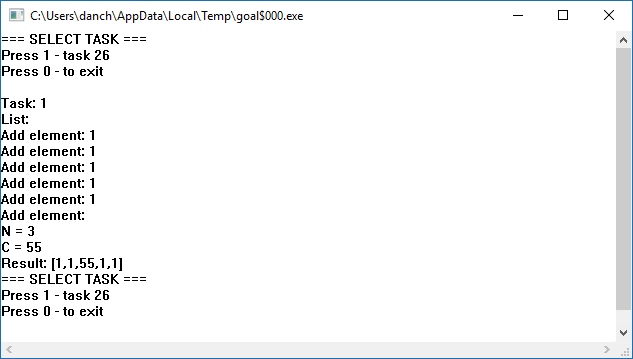
task\_26([], **\_**, **\_**, []).

task\_26([**\_**|T], 1, C, [C|RT]) :- task\_26(T, 0, C, RT).

task\_26([H|T], N, C, [H|R]) :- N2 = N - 1, task\_26(T, N2, C, R).

goal go.

**Тесты:**



**Задача №31.**

**Условие задачи:**

По списку элементов сформировать новый список элементов согласно заданному списка порядковых номеров.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% задание 31 – По списку элементов сформировать новый список элементов согласно списка порядковых номеров*

nondeterm task\_31(list, list, list, i, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 31\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 31*

write("LIST OF ELEMENTS: "), nl, readList(LIST), nl,

write("LIST OF NUMBERS: "), nl, readList(NUMS), nl,

write("LIST OF ELEMENTS: ", LIST), nl,

write("LIST OF NUMBERS: ", NUMS), nl,

task\_31(LIST, LIST, NUMS, 1, NewList),

write("RESULT LIST: ", NewList), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% задание 31 – По списку элементов сформировать новый список элементов согласно списка порядковых номеров*

*% идём по списку, сравнивая текущий элемент списка позиций с текущей позицией (POS)*

*% Если они совпадают - то вставляем текущий элемент списка в результат, сбрасываем счётчик позиций,*

*% и повторяем процесс заново для всего списка. Если же элемент списка позиций с текущей не совпал,*

*% то двигаемся по списку дальше, пока не найдём нужную позицию и соответствующий ей элемент*

task\_31(**\_**, **\_**, [], **\_**, []).

task\_31(FullList, [H|**\_**], [Pos|NT], Pos, [H|RT]) :-

task\_31(FullList, FullList, NT, 1, RT).

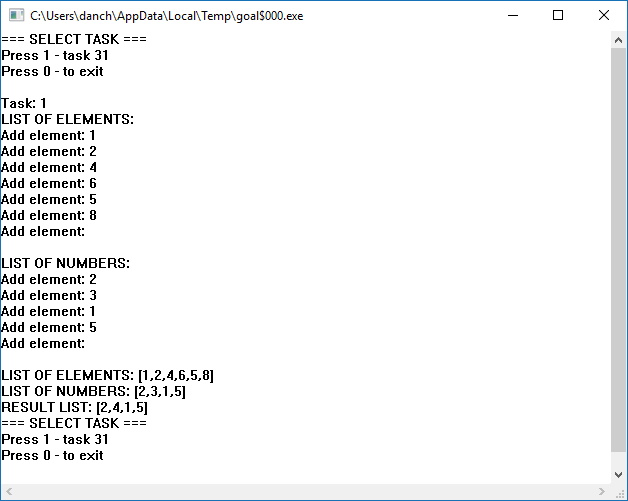
task\_31(FullList, [**\_**|T], PosList, Pos, Res) :-

NewPos = Pos + 1,

task\_31(FullList, T, PosList, NewPos, Res).

goal go.

**Тесты:**



**Задача №32.2.**

**Условие задачи:**

Объединить два списка одинаковой длины в третий так,

чтобы вначале списка были элементы с н/ч позиций первого

списка, а в конце с н/ч - из второго. н - нечётные, ч - чётные. 2(нч).

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

nondeterm add\_end(i, list, list) *% добавляет элемент в конец списка*

*% задание 32.2 - создаёт список из элементов с чётных (нечётных позиций) исходного списка*

nondeterm task\_32(list, i, list, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 32.2\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 32.2*

write("LIST 1: "), nl, readList(ListA),

write("LIST 2: "), nl, readList(ListB),

write("LIST 1: ", ListA), nl,

write("LIST 2: ", ListB), nl,

StartA = 0, *% для нечётных позиций*

task\_32(ListA, StartA, [], R1),

StartB = 1, *% для чётных позиций*

task\_32(ListB, StartB, R1, R2),

write("RESULT: ", R2), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% добавляет элемент в конец списка (из лекций)*

add\_end(X, [H|T], [H|Tf]) :- add\_end(X, T, Tf).

add\_end(X, [], [X]).

*% задание 32.2 - создаёт список из элементов с чётных позиций исходного списка*

*% создаёт список из элементов на чётных позициях, основываясь на переданном числе,*

*% которое считается началом отсчёта. Если передана 1 или любое нечётное – то будут элементы с чётных позиций,*

*% но если передать любое чётное число, то будут элементы с нечетных позиций*

*% Предикат добавляет элемент в конец существующего списка, а не создаёт новый*

*% специально для условий задания, что бы можно было добавить в конец уже частично сформированного списка*

task\_32([], **\_**, ListResult, ListResult).

task\_32([H|T], Counter, PrevList, ListResult) :-

Counter mod 2 = 0,

NextCounter = Counter + 1,

add\_end(H, PrevList, NewList),

task\_32(T, NextCounter, NewList, ListResult).

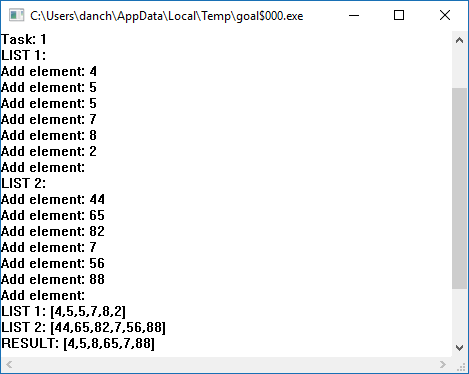
task\_32([**\_**|T], Counter, PrevList, ListResult) :-

NextCounter = Counter + 1,

task\_32(T, NextCounter, PrevList, ListResult).

goal go.

**Тесты:**



**Задача №37.**

**Условие задачи:**

Подсписок, в котором нет К первых элементов исходного списка.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% задание 37 - Подсписок, в котором нет К первых элементов исходного списка.*

nondeterm task\_37(list, i, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 37\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 37*

readList(LIST),

write("List: ", LIST), nl,

write("K: "), readint(K),

task\_37(LIST, K, RESULT),

write("Result: ", RESULT), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% задание 37 - Подсписок, в котором нет К первых элементов исходного списка.*

*% пока K > 0, пропускаем элементы и уменьшаем K на каждом элементе*

*% как только К дойдёт до 0, начинаем копировать элементы в конечный список*

task\_37([], **\_**, []).

task\_37([SH|ST], K, [RH|RT]) :-

K < 1,

NewK = K - 1,

RH = SH,

task\_37(ST, NewK, RT).

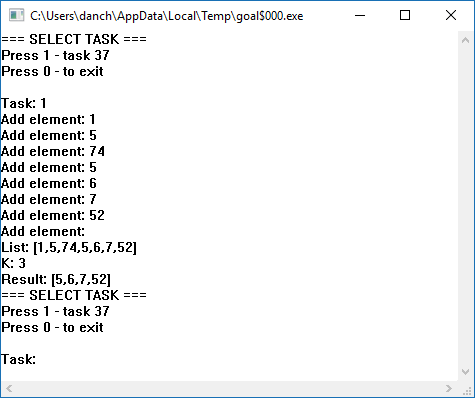
task\_37([**\_**|ST], K, Result) :-

NewK = K - 1,

task\_37(ST, NewK, Result).

goal go.

Тесты:



**Задача №42.**

**Условие задачи:**

Каждый из N последних элементов списка заменить заданной константой.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

nondeterm length(list, i) *% вычисляет длину списка*

*% задание 42 - заменяет N последних элементов списка на заданную константу.*

nondeterm task\_42(list, i, i, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 42\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 42*

readList(L), nl,

write("List: ", L), nl,

write("N: "), readint(N),

write("C: "), readint(C),

length(L, Length),

From = Length - N,

task\_42(L, From, C, R),

write("Result: ", R), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% вычисляет длину списка*

length([], 0). *% длина пустого списка - 0*

length([**\_**|T], L) :-

length(T, TL),

L = 1 + TL. *% длина списка на 1 больше длины его хвоста*

*% задание 42 - заменяет N последних элементов списка на заданную константу.*

*% в этот предикат в качестве N передаётся не количество элементов, а длина*

*% списка минус количество элементов, то есть, позиция, с которой нужно заменять элементы*

*% идём по списку, уменьшая N на 1 и копируя элементы в результатирующий список, когда N*

*% стала равна 1 (мы дошли до позиции N), то начинаем ставить в голову результата константу C*

task\_42([], **\_**, **\_**, []).

task\_42([H|T], N, C, [H|RT]) :-

N > 0,

NewN = N - 1,

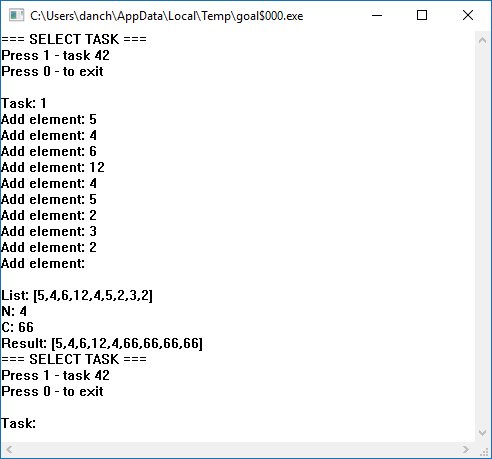
task\_42(T, NewN, C, RT).

task\_42([**\_**|T], N, C, [C|RT]) :-

task\_42(T, N, C, RT).

goal go.

**Тесты:**



**Задача №43.**

**Условие задачи:**

Вместо N последних элементов списка оставить одну заданную константу.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

nondeterm length(list, i) *% вычисляет длину списка*

*% задание 43 - заменяет N последних элементов списка на одну заданную константу.*

nondeterm task\_43(list, i, i, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 43\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 43*

write("LIST OF ELEMENTS: "), nl, readList(L), nl,

write("List: ", L), nl,

write("N: "), readint(N),

write("C: "), readint(C),

length(L, Length),

From = Length - N,

task\_43(L, From, C, R),

write("Result: ", R), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% вычисляет длину списка*

length([], 0). *% длина пустого списка - 0*

length([**\_**|T], L) :-

length(T, TL),

L = 1 + TL. *% длина списка на 1 больше длины его хвоста*

*% задание 43 - заменяет N последних элементов списка на одну заданную константу.*

*% в этот предикат в качестве N передаётся не количество элементов, а длина*

*% списка минус количество элементов, то есть, позиция, с которой нужно заменить элементы*

*% идём по списку, уменьшая N на 1 и копируя элементы в результатирующий список, когда N*

*% стала равна 0 (мы дошли до позиции N), то начинаем ставим в голову списка результата C*

task\_43([H|T], N, C, [H|RT]) :-

N > 0,

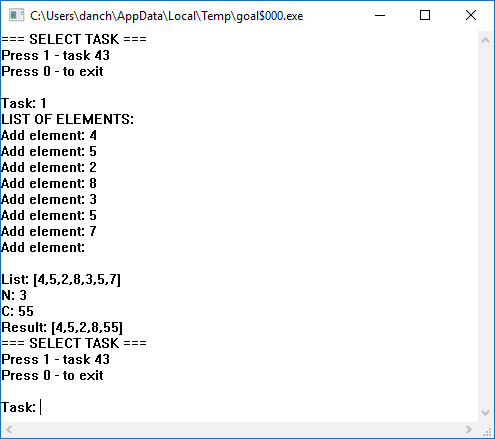
NewN = N - 1,

task\_43(T, NewN, C, RT).

task\_43(**\_**, **\_**, C, [C]).

task\_43([], **\_**, **\_**, []).

goal go.

Тесты :

**Задача №53.**

**Условие задачи:**

Вставить в первый список другой список, начиная с N-ой позиции

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% задание 53 – вставляет один список в другой, начиная с N-ной позиции*

nondeterm task\_53(list, i, list, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 53\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 53*

write("LIST 1: "), nl, readList(L1),

write("LIST 2: "), nl, readList(L2),

write("LIST 1: ", L1), nl,

write("LIST 2: ", L2), nl,

write("N = "), readint(N),

task\_53(L1, N, L2, R),

write("RESULT: ", R), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% задание 53 – вставляет один список в другой, начиная с N-ной позиции*

*% идём по списку, уменьшая N на 1 для каждого элемента и копируем каждый*

*% элемент из исходного списка в результат, и если N равно 1, то вставляем*

*% элементы из 2 списка, пока они не кончатся (пока хвост 2 списка не будет пустым),*

*% затем продолжаем вставлять элементы из 1*

task\_53([], **\_**, [], []).

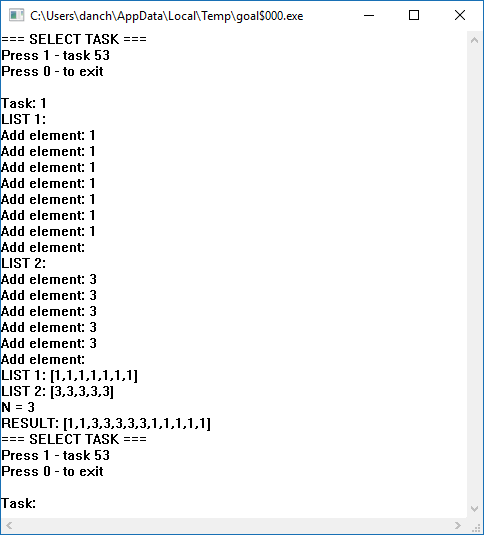
task\_53([H1|T1], N, L2, [H1|RT]) :- N > 1, NextN = N - 1, task\_53(T1, NextN, L2, RT).

task\_53([H1|T1], N, [], [H1|RT]) :- task\_53(T1, N, [], RT).

task\_53(L1, N, [H2|T2], [H2|RT]) :- task\_53(L1, N, T2, RT).

goal go.

Тесты:



**Задача №63.**

**Условие задачи:**

Вычислить значение логической формулы, используя обратную польскую запись.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

tokenType = number;operation *% тип токена в формуле (число или операция)*

tokenTypes = tokenType\* *% множество типов*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% первый этап чтения вводимой формулы - определение типа токена (число/операция) [n/o]*

nondeterm readFormula(list, tokenTypes)

*% второй этап чтения вводимой формулы - добавление токена в список токенов*

nondeterm saveToFormula(list, tokenTypes, char)

*% красиво выводит формулу*

nondeterm printFormula(list, tokenTypes)

*% задание 63 - вычисляет значение логической формулы по обратной польской записи*

nondeterm task\_63(list, tokenTypes, list, i)

*% предикаты логических операций*

nondeterm op\_or(i,i,i)

nondeterm op\_and(i,i,i)

nondeterm op\_xor(i,i,i)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 63\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 63*

write("Enter formula: "), nl,

readFormula(FORMULA, TYPES), nl,

write("Formula: "), printFormula(FORMULA, TYPES), nl,

write("Types: "), write(TYPES), nl,

task\_63(FORMULA, TYPES, [], R),

write("Result: ", R), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% первый этап чтения вводимой формулы - определение типа токена (число/операция) [n/o]*

*% читаем символ, если это Esc - прекращаем чтение формулы, если нет - запускаем saveToFormula*

readFormula(NUMS, TYPES) :-

write("Token type [n]umber/[o]peration/[Esc]-stop: "),

readchar(C),

C <> 27, *% escape key*

saveToFormula(NUMS, TYPES, C).

readFormula([], []).

*% второй этап чтения вводимой формулы -*

*% добавляем токен - в список токенов, а его тип - в список типов*

*% и читаем формулу дальше*

saveToFormula([NH|NT], [number|TT], 'n') :- write("N: "), readint(NH), readFormula(NT, TT).

saveToFormula([NH|NT], [operation|TT], 'o') :- write("O: "), readchar(NH), writef("%c", NH), nl, readFormula(NT, TT).

*% красиво выводит формулу:*

*% если очередной символ в ней - число, выводим как число*

*% если операция - выводим как симво (char) (%c)*

printFormula([], []).

printFormula([TOKEN|TOKENS], [number|TYPES]) :- write(TOKEN, ", "), printFormula(TOKENS, TYPES).

printFormula([TOKEN|TOKENS], [operation|TYPES]) :- writef("%c", TOKEN), write(", "), printFormula(TOKENS, TYPES).

*% задание 62 - вычисляет значение логической формулы по обратной польской записи*

*% Сначала читаем формулу, для числа вводим N, для операции - O*

*% доступные логические операции: & (И), | (ИЛИ), ^ (М2)*

*% и сохраняем в 2 списка: 1 - сами введенные данные, 2 - их тип (Ч/О)*

*% Далее идём по формуле, если нам попадается число - помещаем его в стек*

*% если попадается операция - берём последние 2 числа с верхушки стека,*

*% применяем к ним операцию, кладём результат обратно на стек*

*% если входные данные закончились - результат будет равен числу в стеке*

task\_63([NH|NT], [number|TT], STACK, R) :- task\_63(NT, TT, [NH|STACK], R).

task\_63(['&'|NT], [operation|TT], [O2,O1|ST\_T], R) :- op\_and(O1, O2, OpRes), task\_63(NT, TT, [OpRes|ST\_T], R).

task\_63(['|'|NT], [operation|TT], [O2,O1|ST\_T], R) :- op\_or(O1, O2, OpRes), task\_63(NT, TT, [OpRes|ST\_T], R).

task\_63(['^'|NT], [operation|TT], [O2,O1|ST\_T], R) :- op\_xor(O1, O2, OpRes), task\_63(NT, TT, [OpRes|ST\_T], R).

task\_63([], [], [R], R).

*% таблицы истинности логических операций*

op\_and(0, 0, 0).

op\_and(\_, 0, 0).

op\_and(0, \_, 0).

op\_and(\_, \_, 1).

op\_or(0, 0, 0).

op\_or(\_, \_, 1).

op\_xor(0, 0, 0).

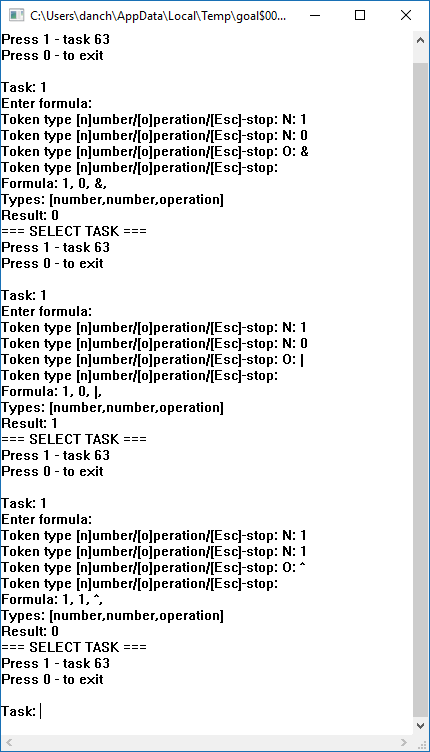
op\_xor(\_, 0, 1).

op\_xor(0, \_, 1).

op\_xor(\_, \_, 0).

goal go.

Тесты:



**Задача №71.**

**Условие задачи:**

По исходному списку целых чисел создать список пар, в которых первая компонента –элемент исходного списка, вторая – число его подряд идущих повторений.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

multiList = list\*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% задание 71 – создаёт список пар в которых первая компонента – элемент исходного списка, вторая – число его подряд идущих повторений.*

nondeterm task\_71(list, i, multiList)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 71\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 71*

write("LIST: "), nl, readList(L), nl,

write("LIST: ", L), nl,

task\_71(L, 0, R),

write("REPEATS: ", R), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% задание 71 – создаёт список пар в которых первая компонента – элемент исходного списка, вторая – число его подряд идущих повторений.*

*% список из одного элемента первращается в список [Этот элемент, N] N - число его повторений на предыдущих итерациях*

*% для списка, в голове которого идут 2 одинаковых элемента - увеличиваем N на 1*

*% для списка, в голове которого разные элементы - сохраняем первый элемент в список-результат вместе с его N*

*% а для второго элемента и остального списка - сбрасываем N в 0 идём по списку дальше*

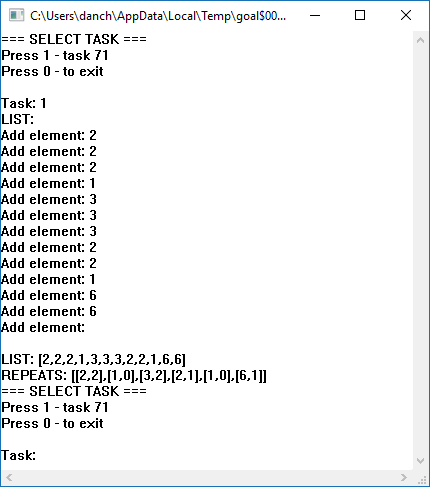
task\_71([H], N, [[H,N]]).

task\_71([H,H|T], N, R) :- NewN = N + 1, task\_71([H|T], NewN, R).

task\_71([H1,H2|T], N, [[H1, N]|RT]) :- task\_71([H2|T], 0, RT).

goal go.

**Тесты:**



**Задача №84.**

**Условие задачи:**

Напечатать путь от корня к указанной вершине в бинарном справочнике.

**Код программы:**

domains

i = integer *% целое число*

list = i\* *% список целых чисел*

bi\_t = tree\_b(bi\_t, i, bi\_t); nil *% функтор бинарного дерева (из лекций)*

predicates

nondeterm go *% главное меню программы*

nondeterm do(char) *% запускает выполнение полученного на вход задания*

nondeterm readList(list) *% вводит список от пользователя*

*% предикаты для бинарных деревьев*

*% выводит бинарное дерево (от левого края)*

nondeterm map(bi\_t, i)

*% делает отступ слева (выводит на консоль табуляцию D раз)*

nondeterm tab(i)

*% создаёт бинарное дерево как бинарный справочник (вводит с консоли)*

nondeterm create\_tree(bi\_t, bi\_t)

*% вставляет элемент в бинарный справочник*

nondeterm insert(i, bi\_t, bi\_t)

*% задание 84 - Напечатать путь от корня к указанной вершине в бинарном справочнике*

nondeterm task\_84(bi\_t, i, list)

clauses

go :-

write("=== SELECT TASK ===\n"),

write("Press 1 - task 84\n"),

write("Press 0 - to exit\n\n"),

write("Task: "), readchar(A),

write(A), nl, do(A),

go.

do('1') :- *% задание 84*

write("Enter tree: "), nl,

create\_tree(nil, Tree),

write("Enter vertex: "), readint(V),

task\_84(Tree, V, Path),

write("Path: ", Path), nl.

do('0') :-

write("Good bye!"), nl,

exit.

*% ввод списка от пользователя*

readList([H|T]) :-

write("Add element: "),

readint(H), *% читаем число и помещаем его в голову*

readList(T). *% запускаем себя для хвоста*

*% если введено не число – прекращаем рекурсию, возвращаем пустой список*

readList([]).

*% бинарное дерево*

*% выводим дерево на консоль (предикат из лекций)*

map(nil, **\_**).

map(tree\_b(Left, E, Right), D) :-

NewD = D + 1,

map(Right, NewD),

tab(D), write(E), nl,

map(Left, NewD).

*% делаем D отступов (из лекций)*

tab(0).

tab(D) :-

write("\t"),

NewD = D - 1,

tab(NewD).

*% созадём дерево (из лекций)*

create\_tree(Tree, NewTree) :-

readint(C),!,

insert(C, Tree, TempTree),

nl, map(TempTree, 0),

write("----------------------------------------------------------------------"), nl,

create\_tree(TempTree, NewTree).

create\_tree(Tree, Tree).

*% вставляем элемент в дерево (из лекций)*

insert(New, nil, tree\_b(nil, New, nil)).

insert(E, tree\_b(Left, E, Right), tree\_b(Left, E, Right)).

insert(New, tree\_b(Left, E, Right), tree\_b(NewLeft, E, Right)) :-

New < E,

insert(New, Left, NewLeft).

insert(New, tree\_b(Left, E, Right), tree\_b(Left, E, NewRight)) :-

E < New,

insert(New, Right, NewRight).

*% задание 84 - Напечатать путь от корня к указанной вершине в бинарном справочнике*

*% Если деерво закончилось - делаем откат, искомая вершина не найдена*

*% Если текущая вершина равна искомой - то это конец пути*

*% Если текущая вершина не равна искомой, то заносим её в список и повторяем процесс*

*% для левого поддерева. Если поиск в левом поддереве не удался, то ищем её в правом*

task\_84(nil, **\_**, **\_**) :- fail.

task\_84(tree\_b(**\_**, V, **\_**), V, [V]).

task\_84(tree\_b(nil, **\_**, nil), **\_**, **\_**) :- fail.

task\_84(tree\_b(L, E, **\_**), V, [E|T]) :- task\_84(L, V, T).

task\_84(tree\_b(**\_**, E, R), V, [E|T]) :- task\_84(R, V, T).

goal go.

**Тесты:**

