**Задача №1 (Lab7-1-1.pl):**

:- use\_module(library(pce)).

% Запуск окна меню menu().

menu():-

new(Main, dialog('Меню')),

new(Shell, button('Сортировка Шелла', message(@prolog, shell\_win))),

new(Qsort, button('Сортировка Хоара', message(@prolog, qsort\_win))),

new(Number, button('Числа по периметру', message(@prolog, number\_win))),

new(DeleteEl, button('Удалить элементы', message(@prolog, delete\_win))),

new(Exit, button('Выход', message(Main, destroy))),

send(Main, append, Shell),

send(Main, append, Qsort, below),

send(Main, append, Number, below),

send(Main, append, DeleteEl, below),

send(Main, append, Exit, below),

send(Main, open).

% Запуск окна сортировки Шелла

shell\_win() :-

new(Shell, dialog('Сортировка Шелла')),

new(List, text\_item('Список')),

new(ButtonSort,button('Сортировать', message(@prolog, shell\_sort\_win, List))),

send(Shell, append, List),

send(Shell, append, ButtonSort),

send(Shell, open).

%Сортировка введенного списка

shell\_sort\_win(L) :-

get(L, value, L1),

split\_string(L1, " ", "", LTemp),

writeln(LTemp),

convert(LTemp, ListToSort),

writeln(ListToSort),

shell\_sort(ListToSort, Result),

shell\_sort\_result(Result).

%Запуск окна-результата для сортировки Шелла

shell\_sort\_result(Result) :-

writeln(Result),

new(Dialog, dialog("Отсортированный список")),

new(F, text("Отсортированный список")),

new(B, button("X")),

send(Dialog, append, F),

send(Dialog, append, B),

send(Dialog, open),

p(Result, ResultStr),

send(B, name, ResultStr).

% Запуск окна сортировки Хоара

qsort\_win() :-

new(Qsort, dialog('Сортировка Хоара')),

new(List1, text\_item('Список')),

new(ButtonSort1,button('Сортировать', message(@prolog, qsort\_s\_win, List1))),

send(Qsort, append, List1),

send(Qsort, append, ButtonSort1),

send(Qsort, open).

%Сортировка введенного списка

qsort\_s\_win(L) :-

get(L, value, L1),

split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToSort),

qsort(ListToSort, Result),

qsort\_result(Result).

%Запуск окна-результата сортировки Хоара

qsort\_result(Result) :-

writeln(Result),

new(Dialog1, dialog("Отсортированный список")),

new(F1, text("Отсортированный список")),

new(B1, button("X")),

send(Dialog1, append, F1),

send(Dialog1, append, B1),

send(Dialog1, open),

p(Result, ResultStr),

send(B1, name, ResultStr).

% Запуск окна-результа для задания 3

number\_win() :-

solve(Result),

writeln(Result),

new(Dialog2, dialog("Решение головоломки")),

new(F2, text("Последовательность чисел")),

new(B2, button("X")),

send(Dialog2, append, F2),

send(Dialog2, append, B2),

send(Dialog2, open),

p(Result, ResultStr),

send(B2, name, ResultStr).

% Запуск окна задания 4.

delete\_win() :-

new(DeleteI, dialog('Удалить элементы')),

new(List2, text\_item('Список')),

new(Dig, text\_item('I')),

new(ButtonSort2,button('Изменить список', message(@prolog, delete\_s\_win, List2, Dig))),

send(DeleteI, append, List2),

send(DeleteI, append, Dig),

send(DeleteI, append, ButtonSort2),

send(DeleteI, open).

%Удаление из списка элементов на i\*n позициях n = 1, 2, 3 ...

delete\_s\_win(L, N) :-

get(L, value, L1),

split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToWork),

get(N, value, N1),

atom\_number(N1, NRes),

delete\_n\_i\_elem(ListToWork, NRes, Result),

delete\_result(Result).

%Запуск окна-результата задания 4.

delete\_result(Result) :-

writeln(Result),

new(Dialog2, dialog("Изменненый список")),

new(F2, text("Новый список")),

new(B2, button("X")),

send(Dialog2, append, F2),

send(Dialog2, append, B2),

send(Dialog2, open),

p(Result, ResultStr),

send(B2, name, ResultStr).

p([]," ").

p([H|T],S):-

p(T,SS),

concat(" ",SS,SSS),

concat(H,SSS,S).

convert([], []).

convert([H|T], [Elm|L]) :-

atom\_number(H, Elm),

convert(T,L).

/\*Проверка на то, что суммы чисел в концах любого отрезка не делились ни на 3, ни на 5, ни на 7.\*/

check\_solve([A], B):-

C is A + B,

X is C mod 3, X > 0,

Y is C mod 5, Y > 0,

Z is C mod 7, Z > 0.

check\_solve([A, B|Tail], К):-

C is A + B,

X is C mod 3, X > 0,

Y is C mod 5, Y > 0,

Z is C mod 7, Z > 0,

check\_solve([B|Tail], К).

solve([Head|Tail]) :-

%Генерации размещений без повторений и перестановок

permutation([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [Head|Tail]),

check\_solve([Head|Tail], Head).

%Если на входе пустой список, то удалять ничего не нужно

delete\_n\_i\_elem([], \_, []).

%Если на вход подается 1 удаляется весь список

delete\_n\_i\_elem(\_, 1, []).

%Если длина списка меньше, чем указанный I, то удалять ничего не нужно

delete\_n\_i\_elem(List, I, List) :-

length(List, Length),

Length < I.

/\*Удаление из списка элементов, находящихся на (i \* n) – х местах, где i задается в качестве аргумента, а в n = 1, 2, 3, …\*/

delete\_n\_i\_elem(List, I, Result) :-

length(List, Length),

get\_number\_delete(Length, I, 1, ListDelete),

reverse(ListDelete, ReverseListDelete),

delete\_list\_elem(List, ReverseListDelete, Result).

%Получение списка индексов для удаления

get\_number\_delete(Length, I, N, ListDelete) :-

Length >= (I \* N), !,

Value is (I \* N - 1),

NewN is (N + 1),

get\_number\_delete(Length, I, NewN, ListDeleteTail),

ListDelete = [Value|ListDeleteTail];

ListDelete = [].

%Удалить элементы по списку индексов

delete\_list\_elem(Result, [], Result).

delete\_list\_elem(List, [H|T], Result) :-

delete\_elem\_list(List, H, TempList),

delete\_list\_elem(TempList, T, Result).

%На вход подается пустой список.

qsort([], []).

%На вход подается список из одного элемента.

qsort([Elem], [Elem]).

%Сортировка Хоара

qsort([Head|Tail], ListSort):-

divide(Tail, Head, GrList, SmList),

qsort(GrList, GrListSort),

qsort(SmList, SmListSort), !,

append(SmListSort, [Head|GrListSort], ListSort).

%Разделение списка на 2 относительно опорного элемента

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

divide([Head|Tail], Elem, [Head|GrList], SmList):-

Head >= Elem, !,

divide(Tail, Elem, GrList, SmList).

divide([Head|Tail], Elem, GrList, [Head|SmList]):-

divide(Tail, Elem, GrList, SmList).

%На вход подается пустой список

shell\_sort([], []).

%На вход подается список из 1 элемента

shell\_sort(List, List) :-

length(List, Length),

Length = 1.

%На вход подается список из 2 элементов, первый элемент меньше

shell\_sort([ELEM1, ELEM2|TAIL], [ELEM1, ELEM2]) :-

length([ELEM1, ELEM2|TAIL], Length),

Length = 2,

ELEM1 < ELEM2.

%На вход подается список из 2 элементов, второй элемент меньше

shell\_sort([ELEM1, ELEM2|TAIL], [ELEM2, ELEM1]) :-

length([ELEM1, ELEM2|TAIL], Length),

Length = 2,

ELEM1 > ELEM2.

/\*Сортировка Шелла. Вычисление последовательности шагов сортировки производится методом, предложенным Дональдом Кнутом. \*/

shell\_sort(List, SortList) :-

length(List, Length),

list\_steps(Length, Steps),

shell\_sort\_steps(List, Steps, SortList).

%Шаги сортировки

shell\_sort\_steps(SortList, [], SortList).

shell\_sort\_steps(List, [H|T], SortList) :-

shell\_sort\_step(List, H, TempList),

shell\_sort\_steps(TempList, T, SortList).

shell\_sort\_step(List, ValueStep, Result) :-

shell\_sort\_step\_value(List, ValueStep, 0, ValueStep, Result).

shell\_sort\_step\_value(Result, ValueStep, N1, \_, Result) :-

length(Result, Length),

Max is (N1 + ValueStep),

Max >= Length.

shell\_sort\_step\_value(List, ValueStep, N1, N2, Result) :-

length(List, Length),

N2 >= Length,

N3 is (N1 + 1),

N4 is (N3 + 1),

shell\_sort\_step\_value(List, ValueStep, N3, N4, Result).

shell\_sort\_step\_value(List, ValueStep, N1, N2, Result) :-

get\_elem(List, N1, Elem1),

get\_elem(List, N2, Elem2),

Elem1 > Elem2,

swap\_elem(List, N1, N2, NewList),

N3 is (N2 + ValueStep),

shell\_sort\_step\_value(NewList, ValueStep, N1, N3, Result).

shell\_sort\_step\_value(List, ValueStep, N1, N2, Result) :-

get\_elem(List, N1, Elem1),

get\_elem(List, N2, Elem2),

Elem1 =< Elem2,

N3 is (N2 + ValueStep),

shell\_sort\_step\_value(List, ValueStep, N1, N3, Result).

%Перестановка элементов

swap\_elem(List, N1, N2, NewList) :-

get\_elem(List, N1, Elem1),

get\_elem(List, N2, Elem2),

delete\_elem\_list(List, N1, List1),

insert\_list(List1, Elem2, N1, List2),

delete\_elem\_list(List2, N2, List3),

insert\_list(List3, Elem1, N2, NewList).

%Вставить элемент в список

insert\_list([], Elem, 0, [Elem]).

insert\_list([H|T], Elem, 0, [Elem|[H|T]]).

insert\_list([H|T], Elem, N, [H|T1]) :-

N1 is (N - 1),

insert\_list(T, Elem, N1, T1).

%Удалить элемент из списка

delete\_elem\_list([\_|T], 0, T).

delete\_elem\_list([H|T], N, [H|T1]) :-

N1 is (N - 1),

delete\_elem\_list(T, N1, T1).

%Получить значение элемента по индексу

get\_elem([H|\_], 0, H).

get\_elem([\_|[H1|T]], N, Elem) :-

N1 is (N - 1),

get\_elem([H1|T], N1, Elem).

%Получение последовательности шагов

list\_steps(Length, Steps) :-

count\_steps(Length, 2, CountSteps),

list\_steps\_temp(CountSteps, Steps).

list\_steps\_temp(CountSteps, Steps) :-

CountSteps > -1, !,

value\_step(CountSteps, Value),

NewCountSteps is (CountSteps - 1),

list\_steps\_temp(NewCountSteps, StepsTail),

Steps = [Value|StepsTail];

Steps = [].

%Количество шагов сортировки

count\_steps(Length, Start, Steps) :-

Steps is Start - 2,

Length < 3 ^ Start;

NextStart is Start + 1,

count\_steps(Length, NextStart, Steps).

%Значение шагов сортировки

value\_step(0, 1).

value\_step(Step, Value) :-

Step1 is Step - 1,

value\_step(Step1, Value1),

Value is 3 \* Value1 + 1.

**Задача №2 (Lab7-2.pl):**

:- use\_module(library(pce)).

% Запуск окна меню

menu():-

new(Main, dialog('Меню')),

new(Shell, button('Сортировка Шелла', message(@prolog, shell\_win))),

new(Bubble, button('Сортировка прямым выбором', message(@prolog, selection\_win))),

new(Number, button('Задача "Числа по периметру"', message(@prolog, number\_result))),

new(Inser, button('Вставка подсписка в список', message(@prolog, insert\_win))),

new(Exit, button('Выход', message(Main, destroy))),

send(Main, append, Shell),

send(Main, append, Bubble, below),

send(Main, append, Number, below),

send(Main, append, Inser, below),

send(Main, append, Exit, below),

send(Main, open).

% Запуск окна сортировки Шелла

shell\_win() :-

new(Shell, dialog('Сортировка Шелла')),

new(List, text\_item('Список')),

new(ButtonSort,button('Сортировать', message(@prolog, shell\_result, List))),

send(Shell, append, List),

send(Shell, append, ButtonSort),

send(Shell, open).

%Сортировка введенного списка и вывод результата в новое окно.

shell\_result(L) :-

get(L, value, L1),

new(Dialog, dialog("Отсортированный список")),

new(F, text("Отсортированный список")),

new(B, button(" ")),

send(Dialog, append, F),

send(Dialog, append, B),

send(Dialog, open),

(L1 \= '',

(split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToSort),

shell\_sort(ListToSort, Result),

p(Result, ResultStr),

send(B, name, ResultStr));

(send(B, name, " "))).

% Запуск окна сортировки пузырьком

selection\_win() :-

new(Bubble, dialog('Сортировка прямым выбором')),

new(List1, text\_item('Список')),

new(ButtonSort1,button('Сортировать', message(@prolog, selection\_result, List1))),

send(Bubble, append, List1),

send(Bubble, append, ButtonSort1),

send(Bubble, open).

%Сортировка введенного списка и вывод результата в новое окно.

selection\_result(L) :-

get(L, value, L1),

new(Dialog1, dialog("Отсортированный список")),

new(F1, text("Отсортированный список")),

new(B1, button(" ")),

send(Dialog1, append, F1),

send(Dialog1, append, B1),

send(Dialog1, open),

(L1 \= '',

(split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToSort),

selection\_sort(ListToSort, Result),

p(Result, ResultStr),

send(B1, name, ResultStr));

(send(B1, name, " "))).

% Запуск окна-результа для решения задачи №3

number\_result() :-

get\_answer(Result),

new(Dialog2, dialog("Решение задачи")),

new(F2, text("Решение задачи")),

new(B2, button(" ")),

send(Dialog2, append, F2),

send(Dialog2, append, B2),

send(Dialog2, open),

p(Result, ResultStr),

send(B2, name, ResultStr).

% Запуск окна "Вставка подсписка в список".

insert\_win() :-

new(Rever, dialog('Вставка подсписка в список')),

new(List2, text\_item('Список')),

new(SubList, text\_item('Подсписок')),

new(Dig2, text\_item('I')),

new(ButtonAnswer,button('Изменить список', message(@prolog, insert\_result, List2, SubList, Dig2))),

send(Rever, append, List2),

send(Rever, append, SubList),

send(Rever, append, Dig2),

send(Rever, append, ButtonAnswer),

send(Rever, open).

%Запуск окна-результата задания 4.

insert\_result(L, S, I) :-

get(L, value, L1),

new(Dialog3, dialog("Изменённый список")),

new(F3, text("Изменённый список")),

new(B3, button(" ")),

send(Dialog3, append, F3),

send(Dialog3, append, B3),

send(Dialog3, open),

get(I, value, I1),

atom\_number(I1, IRes),

(L1 \= '',

(split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToWork),

get(S, value, S1),

(S1 \= '',

(split\_string(S1, " ", "", STemp),

convert(STemp, SubListToWork),

insert\_sublist(ListToWork, SubListToWork, IRes, Result),

p(Result, ResultStr),

send(B3, name, ResultStr));

(insert\_sublist(ListToWork, [], IRes, Result),

p(Result, ResultStr),

send(B3, name, ResultStr))));

(get(S, value, S1),

(S1 \= '',

(split\_string(S1, " ", "", STemp),

convert(STemp, SubListToWork),

insert\_sublist([], SubListToWork, IRes, Result),

p(Result, ResultStr),

send(B3, name, ResultStr));

(send(B3, name, " "))))).

%Сортировка Шелла

%Пустой список

shell\_sort([], []).

%Список из одного элемента

shell\_sort([H], [H]).

shell\_sort([H, H1|Tail], [H1, H]) :-

length([H, H1|Tail], Length),

Length = 2,

H1 < H.

%Сортировка Шелла

shell\_sort(List, Result) :-

get\_steps(List, Steps),

shell\_sort\_steps(List, Steps, Result).

%Получение списка шагов

get\_steps(List, Result) :-

length\_list(List, Length),

get\_steps\_temp(Length, Result, [], 1, 1).

get\_steps\_temp(Length, Result, Result, \_, Element) :-

(Element \* 3) > Length.

get\_steps\_temp(Length, Result, List, N, Element) :-

get\_step(N, Element1),

N1 is (N + 1),

insert\_element\_in\_list(List, Element, 0, List3),

get\_steps\_temp(Length, Result, List3, N1, Element1).

%Значение одного шага по методу Р. Седжвика

get\_step(N, Result) :-

get\_step\_temp(1, Result, 0, N).

get\_step\_temp(0, Result, Result, \_).

get\_step\_temp(Length, Result, 0, N) :-

Y is (N mod 2),

Y = 1,

pow\_element(2, N, Result1),

N3 is ((N+1) div 2),

pow\_element(2, N3, Result2),

H is (8 \* Result1 - 6 \* Result2 + 1),

Length1 is (Length - 1),

get\_step\_temp(Length1, Result, H, N).

get\_step\_temp(Length, Result, 0, N) :-

Y is (N mod 2),

Y = 0,

pow\_element(2, N, Result1),

N3 is (N div 2),

pow\_element(2, N3, Result2),

H is (9 \* Result1 - 9 \* Result2 + 1),

Length1 is (Length - 1),

get\_step\_temp(Length1, Result, H, N).

%Проход по списку шагов

shell\_sort\_steps(Result, [], Result).

shell\_sort\_steps(List, [H|T], Result) :-

shell\_sort\_step(List, H, List1),

shell\_sort\_steps(List1, T, Result).

%Шаг сортировки Шелла

shell\_sort\_step(List, Step, Result) :-

shell\_sort\_step\_temp(List, Step, 0, Step, Result).

shell\_sort\_step\_temp(Result, Step, N1, \_, Result) :-

length\_list(Result, Length),

Maximum is (N1 + Step),

Maximum >= Length.

shell\_sort\_step\_temp(List, Step, N1, N2, Result) :-

length\_list(List, Length),

N2 >= Length,

N3 is (N1 + 1),

N4 is (N3 + Step),

shell\_sort\_step\_temp(List, Step, N3, N4, Result).

shell\_sort\_step\_temp(List, Step, N1, N2, Result) :-

element\_n(List, N1, Element1),

element\_n(List, N2, Element2),

Element1 > Element2,

swap\_elements(List, N1, N2, List1),

N3 is (N2 + Step),

shell\_sort\_step\_temp(List1, Step, N1, N3, Result).

shell\_sort\_step\_temp(List, Step, N1, N2, Result) :-

element\_n(List, N1, Element1),

element\_n(List, N2, Element2),

Element1 =< Element2,

N3 is (N2 + Step),

shell\_sort\_step\_temp(List, Step, N1, N3, Result).

%Получить элемент по индексу

element\_n([H|\_], 0, H).

element\_n([\_|[H1|T]], N, Element) :-

N1 is (N - 1),

element\_n([H1|T], N1, Element).

%Вставка элемента в список

insert\_element\_in\_list([], Element, 0, [Element]).

insert\_element\_in\_list([H|T], Element, 0, [Element|[H|T]]).

insert\_element\_in\_list([H|T], Element, N, [H|T1]) :-

N1 is (N - 1),

insert\_element\_in\_list(T, Element, N1, T1) .

%Удалить элемент из списка по индексу

delete\_element([\_|T], 0, T).

delete\_element([H|T], N, [H|T1]) :-

N1 is (N - 1),

delete\_element(T, N1, T1).

%Перестановка элементов

swap\_elements(List, N1, N2, NewList) :-

element\_n(List, N1, Element1),

element\_n(List, N2, Element2),

delete\_element(List, N1, List1),

insert\_element\_in\_list(List1, Element2, N1, List2),

delete\_element(List2, N2, List3),

insert\_element\_in\_list(List3, Element1, N2, NewList).

%Возвести число в степень

pow\_element(Number, Degree, Result) :-

pow\_element\_temp(Number, Degree, Result, 0, 1).

pow\_element\_temp(\_, Degree, Result, Degree, Result).

pow\_element\_temp(Number, Degree, Result, Degree1, Result1) :-

Degree2 is (Degree1 + 1),

Result2 is (Result1 \* Number),

pow\_element\_temp(Number, Degree, Result, Degree2, Result2).

%Длина списка

length\_list(List, Length):-

length\_list\_temp(List, Length, 0).

length\_list\_temp([], Length, Length).

length\_list\_temp([\_|T], Length, Length1) :-

Length2 is (Length1 + 1),

length\_list\_temp(T, Length, Length2).

p([]," ").

p([H|T],S):-

p(T,SS),

concat(" ",SS,SSS),

concat(H,SSS,S).

convert([], []).

convert([H|T], [Elm|L]) :-

atom\_number(H, Elm),

convert(T,L).

%Сортировка пузырьком

%Пустой список

selection\_sort([], []).

%Список из одного элемента

selection\_sort([H], [H]).

%Сортировка методом прямого выбора

selection\_sort(List, SortList):-

selection\_sort(List, [], SortListTemp),

reverse(SortListTemp, SortList).

selection\_sort([], SortList, SortList) :- !.

selection\_sort(UnSortPart, SortPart, SortList):-

min\_list(UnSortPart, MinElement),

delete\_single\_element(UnSortPart, MinElement, UnSortTail),

selection\_sort(UnSortTail, [MinElement|SortPart], SortList).

%Удаления элемента с заданным значением

delete\_single\_element(List, Element, ListWithoutElement):-

select(Element, List, ListWithoutElement), !.

get\_answer([H|T]) :-

my\_permutation([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [H|T]),

check([H|T], H).

%Генерации размещений без повторений и перестановок

my\_permutation(List, Permutation):-

length(List, Length),

permutation\_temp(List, Length, Permutation).

permutation\_temp(\_List, 0, []):-!.

permutation\_temp(List, PermutationLength, [H|PermutationTail]):-

member(H, List),

delete(List, H, ListTail),

PermutationTailLength is (PermutationLength - 1),

permutation\_temp(ListTail, PermutationTailLength, PermutationTail).

/\*Проверка на то, что суммы соседних чисел не делились ни на 3, ни на 5, ни на 7.\*/

check([A], B):-

Sum is A + B,

X is Sum mod 3, X > 0,

Y is Sum mod 5, Y > 0,

Z is Sum mod 7, Z > 0.

check([A, B|T], К):-

Sum is A + B,

X is Sum mod 3, X > 0,

Y is Sum mod 5, Y > 0,

Z is Sum mod 7, Z > 0,

check([B|T], К).

%Если подсписок пустой, то вернуть исходный список

insert\_sublist(List1, [], \_, List1).

%Если исходный список пустой, то вернуть подсписок

insert\_sublist([], List2, \_, List2).

/\*Если элемент с которого необходимо вставлять список, больше чем количество элементов исходного списка, то подсписок вставить в конец исходного\*/

insert\_sublist(List1, List2, I, Result) :-

length(List1, Length),

Length < I,

I is Length,

insert\_sublist(List1, List2, I, Result).

%Вставка подсписка в начало списка.

insert\_sublist(List1, [A|B], 1, [A|Result]):- !,

insert\_sublist(List1, B, 1, Result).

%Вставка подсписка в список, начиная с i-го элемента

insert\_sublist([A|T], List2, I, [A|Result]):-

I1 is I - 1,

insert\_sublist(T, List2, I1, Result).

**Задача №3 (Lab7-3.pl):**

:- use\_module(library(pce)).

% Запуск окна меню

menu():-

new(Main, dialog('Меню')),

new(Shell, button('Сортировка Шелла', message(@prolog, shell\_win))),

new(Bubble, button('Сортировка пузырьком', message(@prolog, bubble\_win))),

new(Number, button('Задача "Числа по периметру"', message(@prolog, number\_result))),

new(Rever, button('Реверсировать часть списка', message(@prolog, revers\_win))),

new(Exit, button('Выход', message(Main, destroy))),

send(Main, append, Shell),

send(Main, append, Bubble, below),

send(Main, append, Number, below),

send(Main, append, Rever, below),

send(Main, append, Exit, below),

send(Main, open).

% Запуск окна сортировки Шелла

shell\_win() :-

new(Shell, dialog('Сортировка Шелла')),

new(List, text\_item('Список')),

new(ButtonSort,button('Сортировать', message(@prolog, shell\_result, List))),

send(Shell, append, List),

send(Shell, append, ButtonSort),

send(Shell, open).

%Сортировка введенного списка и вывод результата в новое окно.

shell\_result(L) :-

get(L, value, L1),

new(Dialog, dialog("Отсортированный список")),

new(F, text("Отсортированный список")),

new(B, button(" ")),

send(Dialog, append, F),

send(Dialog, append, B),

send(Dialog, open),

(L1 \= '',

(split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToSort),

shell\_sort(ListToSort, Result),

p(Result, ResultStr),

send(B, name, ResultStr));

(send(B, name, " "))).

% Запуск окна сортировки пузырьком

bubble\_win() :-

new(Bubble, dialog('Сортировка пузырьком')),

new(List1, text\_item('Список')),

new(ButtonSort1,button('Сортировать', message(@prolog, bubble\_result, List1))),

send(Bubble, append, List1),

send(Bubble, append, ButtonSort1),

send(Bubble, open).

%Сортировка введенного списка и вывод результата в новое окно.

bubble\_result(L) :-

get(L, value, L1),

new(Dialog1, dialog("Отсортированный список")),

new(F1, text("Отсортированный список")),

new(B1, button(" ")),

send(Dialog1, append, F1),

send(Dialog1, append, B1),

send(Dialog1, open),

(L1 \= '',

(split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToSort),

bubble\_sort(ListToSort, Result),

p(Result, ResultStr),

send(B1, name, ResultStr));

(send(B1, name, " "))).

% Запуск окна-результа для решения задачи №3

number\_result() :-

answer(Result),

new(Dialog2, dialog("Решение задачи")),

new(F2, text("Полученная последовательность чисел")),

new(B2, button(" ")),

send(Dialog2, append, F2),

send(Dialog2, append, B2),

send(Dialog2, open),

p(Result, ResultStr),

send(B2, name, ResultStr).

% Запуск окна "Реверсировать часть списка"

revers\_win() :-

new(Rever, dialog('Реверсировать часть списка')),

new(List2, text\_item('Список')),

new(Dig, text\_item('I')),

new(Dig2, text\_item('N')),

new(ButtonAnswer,button('Изменить список', message(@prolog, revers\_result, List2, Dig, Dig2))),

send(Rever, append, List2),

send(Rever, append, Dig),

send(Rever, append, Dig2),

send(Rever, append, ButtonAnswer),

send(Rever, open).

%Запуск окна-результата задания 4.

revers\_result(L, I, N) :-

get(L, value, L1),

new(Dialog3, dialog("Изменённый список")),

new(F3, text("Изменённый список")),

new(B3, button(" ")),

send(Dialog3, append, F3),

send(Dialog3, append, B3),

send(Dialog3, open),

(L1 \= '',

(split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToWork),

get(I, value, I1),

atom\_number(I1, IRes),

get(N, value, N1),

atom\_number(N1, NRes),

revers(ListToWork, IRes, NRes, Result),

p(Result, ResultStr),

send(B3, name, ResultStr));

(send(B3, name, " "))).

%Сортировка Шелла

%Пустой список

shell\_sort([], []).

%Список из 1 элемента

shell\_sort([H], [H]).

%Список из 2 элементов, второй элемент меньше

shell\_sort([A, B|TAIL], [B, A]) :-

length\_list([A, B|TAIL], Length),

Length = 2,

A > B.

%Список из 2 элементов, первый элемент меньше

shell\_sort([A, B|T], [A, B]) :-

length\_list([A, B|T], Length),

Length = 2,

A < B.

%Сортировка Шелла.

shell\_sort(List, Result) :-

length\_list(List, Length),

list\_steps(Length, Steps),

shell\_sort\_steps(List, Steps, Result).

%Проход по шагам сортировки

shell\_sort\_steps(Result, [], Result).

shell\_sort\_steps(List, [H|T], Result) :-

shell\_sort\_step(List, H, TempList),

shell\_sort\_steps(TempList, T, Result).

%Шаг сортировки

shell\_sort\_step(List, Step, Result) :-

shell\_sort\_step\_value(List, Step, 0, Step, Result).

shell\_sort\_step\_value(Result, Step, N1, \_, Result) :-

length\_list(Result, Length),

Max is (N1 + Step),

Max >= Length.

shell\_sort\_step\_value(List, Step, N1, N2, Result) :-

length\_list(List, Length),

N2 >= Length,

N3 is (N1 + 1),

N4 is (N3 + 1),

shell\_sort\_step\_value(List, Step, N3, N4, Result).

shell\_sort\_step\_value(List, Step, N1, N2, Result) :-

get\_element\_list(List, N1, Element1),

get\_element\_list(List, N2, Element2),

Element1 > Element2,

swap\_elem(List, N1, N2, NewList),

N3 is (N2 + Step),

shell\_sort\_step\_value(NewList, Step, N1, N3, Result).

shell\_sort\_step\_value(List, Step, N1, N2, Result) :-

get\_element\_list(List, N1, Element1),

get\_element\_list(List, N2, Element2),

Element1 =< Element2,

N3 is (N2 + Step),

shell\_sort\_step\_value(List, Step, N1, N3, Result).

%Получение последовательности шагов

list\_steps(Length, Steps) :-

number\_steps(Length, 2, NumberSteps),

list\_steps\_temp(NumberSteps, Steps).

list\_steps\_temp(NumberSteps, Steps) :-

NumberSteps > -1, !,

step\_value(NumberSteps, Value),

NewNumberSteps is (NumberSteps - 1),

list\_steps\_temp(NewNumberSteps, StepsTail),

Steps = [Value|StepsTail];

Steps = [].

%Количество шагов сортировки

number\_steps(Length, Start, Steps) :-

Steps is Start - 2,

Length < 3 ^ Start;

NextStart is Start + 1,

number\_steps(Length, NextStart, Steps).

%Значение шагов сортировки

step\_value(0, 1).

step\_value(Step, Value) :-

Step1 is Step - 1,

step\_value(Step1, Value1),

Value is 3 \* Value1 + 1.

%Длина списка

length\_list(List, Length):-

length\_list\_temp(List, Length, 0).

length\_list\_temp([], Length, Length).

length\_list\_temp([\_|T], Length, Length1) :-

Length2 is (Length1 + 1),

length\_list\_temp(T, Length, Length2).

%Получить значение элемента по индексу

get\_element\_list([H|\_], 0, H).

get\_element\_list([\_|[H1|T]], N, Element) :-

N1 is (N - 1),

get\_element\_list([H1|T], N1, Element).

%Перестановка элементов

swap\_elem(List, N1, N2, NewList) :-

get\_element\_list(List, N1, Element1),

get\_element\_list(List, N2, Element2),

delete\_element\_list(List, N1, List1),

insert\_element\_list(List1, Element2, N1, List2),

delete\_element\_list(List2, N2, List3),

insert\_element\_list(List3, Element1, N2, NewList).

%Вставить элемент в список по индексу

insert\_element\_list([], Element, 0, [Element]).

insert\_element\_list([H|T], Element, 0, [Element|[H|T]]).

insert\_element\_list([H|T], Element, N, [H|T1]) :-

N1 is (N - 1),

insert\_element\_list(T, Element, N1, T1).

%Удалить элемент из списка по индексу

delete\_element\_list([\_|T], 0, T).

delete\_element\_list([H|T], N, [H|T1]) :-

N1 is (N - 1),

delete\_element\_list(T, N1, T1).

p([]," ").

p([H|T],S):-

p(T,SS),

concat(" ",SS,SSS),

concat(H,SSS,S).

convert([], []).

convert([H|T], [Elm|L]) :-

atom\_number(H, Elm),

convert(T,L).

%Сортировка пузырьком

%Если список пуст

bubble\_sort([], []).

%Если список из 1-го элемента

bubble\_sort([H], [H]).

%Сортировка методом пузырька

bubble\_sort(SortList, SortList):-

move\_max\_to\_end(SortList, DoubleSortList),

SortList = DoubleSortList, !.

bubble\_sort(List, SortList):-

move\_max\_to\_end(List, SortPart),

bubble\_sort(SortPart, SortList).

move\_max\_to\_end([], []):-!.

move\_max\_to\_end([Head], [Head]):-!.

move\_max\_to\_end([First, Second|Tail], [Second|ListWithMaxEnd]):-

First > Second, !,

move\_max\_to\_end([First|Tail], ListWithMaxEnd).

move\_max\_to\_end([First, Second|Tail], [First|ListWithMaxEnd]):-

move\_max\_to\_end([Second|Tail], ListWithMaxEnd).

answer([HeadAnswer|TailAnswer]) :-

list\_number\_1\_10(L),

permutation(L, [HeadAnswer|TailAnswer]),

check\_answer([HeadAnswer|TailAnswer], HeadAnswer).

%Список чисел от 1 до 10

list\_number\_1\_10([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).

%Проверка на то, что суммы соседних чисел не делятся ни на 3, ни на 5, ни на 7.

check\_answer([A], B):-

SumAB is A + B,

X is SumAB mod 3, X > 0,

Y is SumAB mod 5, Y > 0,

Z is SumAB mod 7, Z > 0.

check\_answer([A, B|Tail], К):-

SumAB is A + B,

X is SumAB mod 3, X > 0,

Y is SumAB mod 5, Y > 0,

Z is SumAB mod 7, Z > 0,

check\_answer([B|Tail], К).

%Если список пустой

revers([], \_, \_, []).

%Если i больше, чем длина списка

revers(List, I, \_, List) :-

length(List, Length),

Length < I.

%Если N больше, чем длина списка

revers(List, I, N, Result) :-

length(List, Length),

Length < N,

NewN is (Length - I + 1),

revers(List, I, NewN, Result).

%Если i = 1

revers(List, 1, N, Result):-

rev(List, N, [], Result), !.

%В остальных случаях

revers([HList|TList], I, N, [HList|Result]):-

In is I - 1,

revers(TList, In, N, Result).

cnct([], List, List).

cnct([HeadA|TailA], B, [HeadA|Res]):-

cnct(TailA, B, Res).

rev(List, 0, R, Res):-

cnct(R, List, Res).

rev([HList|TList], N, R, Res):-

Nn is N-1,

rev(TList, Nn, [HList|R], Res).

**Задача №4 (Lab7-1-2.pl):**

game():-

initial\_state(State),

show(State),

next\_move(State, 0).

%Начальное состояние

initial\_state([0,0,0,0,0,0,0,0,0]).

%0 - Незанятое поле

from\_symbol\_to\_digit(0, ' ').

%1 - крестик

from\_symbol\_to\_digit(1, 'x').

%2 - нолик

from\_symbol\_to\_digit(2, 'o').

%Вывод поля игры

show(State) :-

get\_element(State, 0, SymbolA),

get\_element(State, 1, SymbolB),

get\_element(State, 2, SymbolC),

get\_element(State, 3, SymbolD),

get\_element(State, 4, SymbolE),

get\_element(State, 5, SymbolF),

get\_element(State, 6, SymbolG),

get\_element(State, 7, SymbolH),

get\_element(State, 8, SymbolI),

from\_symbol\_to\_digit(SymbolA, DigitA),

from\_symbol\_to\_digit(SymbolB, DigitB),

from\_symbol\_to\_digit(SymbolC, DigitC),

from\_symbol\_to\_digit(SymbolD, DigitD),

from\_symbol\_to\_digit(SymbolE, DigitE),

from\_symbol\_to\_digit(SymbolF, DigitF),

from\_symbol\_to\_digit(SymbolG, DigitG),

from\_symbol\_to\_digit(SymbolH, DigitH),

from\_symbol\_to\_digit(SymbolI, DigitI),

write("0|"),

write(DigitA),

write(DigitB),

writeln(DigitC),

write("1|"),

write(DigitD),

write(DigitE),

writeln(DigitF),

write("2|"),

write(DigitG),

write(DigitH),

writeln(DigitI),

writeln(" 012"),

writeln("-----------").

%Проверка того, что данное поле свободно

is\_free(X, Y, State) :-

N is (Y \* 3 + X),

get\_element(State, N, A),

A = 0.

%Выполнение хода

make\_move(X, Y, State, Symbol, State\_next) :-

is\_free(X, Y, State),

N is (Y \* 3 + X),

delete\_element\_list(State, N, State1),

insert\_element\_list(State1, Symbol, N, State\_next),

show(State\_next).

%Проверка того, что победили крестики

check(State) :-

state\_win\_x(StateX),

member(State, [StateX]),

writeln("Крестики победили"), !.

%Проверка того, что победили нолики

check(State) :-

state\_win\_o(StateO),

member(State, [StateO]),

writeln("Нолики победили"), !.

%Проверка того, что игра закончилачь ничьей

check(State) :-

not(member(0, State)),

writeln("Ничья"), !.

%Ход игрока

next\_move(State, N) :-

I is (N mod 2), I = 0,

Symbol is 1,

writeln("Введите номер столбца Вашего хода"),

read(X),

writeln("Введите номер строки Вашего хода"),

read(Y),

make\_move(X, Y, State, Symbol, NewState), !,

not(check(NewState)),

N1 is (N + 1),

next\_move(NewState, N1).

%Следующий ход.

next\_move(State, N) :-

I is (N mod 2), I = 1,

Symbol is 2,

possible\_move\_o(State, X, Y),

make\_move(X, Y, State, Symbol, NewState), !,

not(check(NewState)),

N1 is (N + 1),

next\_move(NewState, N1).

%Варианты ситуаций игры когда побеждают крестики

state\_win\_x([1,1,1,\_,\_,\_,\_,\_,\_]).

state\_win\_x([\_,\_,\_,1,1,1,\_,\_,\_]).

state\_win\_x([\_,\_,\_,\_,\_,\_,1,1,1]).

state\_win\_x([1,\_,\_,1,\_,\_,1,\_,\_]).

state\_win\_x([\_,1,\_,\_,1,\_,\_,1,\_]).

state\_win\_x([\_,\_,1,\_,\_,1,\_,\_,1]).

state\_win\_x([1,\_,\_,\_,1,\_,\_,\_,1]).

state\_win\_x([\_,\_,1,\_,1,\_,1,\_,\_]).

%Варианты ситуаций игры когда побеждают нолики

state\_win\_o([2,2,2,\_,\_,\_,\_,\_,\_]).

state\_win\_o([\_,\_,\_,2,2,2,\_,\_,\_]).

state\_win\_o([\_,\_,\_,\_,\_,\_,2,2,2]).

state\_win\_o([2,\_,\_,2,\_,\_,2,\_,\_]).

state\_win\_o([\_,2,\_,\_,2,\_,\_,2,\_]).

state\_win\_o([\_,\_,2,\_,\_,2,\_,\_,2]).

state\_win\_o([2,\_,\_,\_,2,\_,\_,\_,2]).

state\_win\_o([\_,\_,2,\_,2,\_,2,\_,\_]).

%Получение элемента по индексу

get\_element([Head|\_], 0, Head).

get\_element([\_|[Head1|Tail]], N, Element) :-

N1 is (N - 1),

get\_element([Head1|Tail], N1, Element).

%Вставка элемента по индексу

insert\_element\_list([], Element, 0, [Element]).

insert\_element\_list([Head|Tail], Element, 0, [Element|[Head|Tail]]).

insert\_element\_list([Head|Tail], Element, N, [Head|Tail1]) :-

N1 is (N-1),

insert\_element\_list(Tail, Element, N1, Tail1) .

%Удаление элемента по индексу

delete\_element\_list([\_|Tail], 0, Tail).

delete\_element\_list([Head|Tail], N, [Head|Tail1]) :-

N1 is (N - 1),

delete\_element\_list(Tail, N1, Tail1).

%Если центр не занят, то занимаем его

possible\_move\_o([\_, \_,\_, \_, 0, \_, \_, \_, \_], 1, 1) :- !.

%Ситуации, при которых, данный ход является выигрышным, занимаем победную позицию.

possible\_move\_o([0,2,2,\_,\_,\_,\_,\_,\_], 0, 0) :- !.

possible\_move\_o([2,0,2,\_,\_,\_,\_,\_,\_], 1, 0) :- !.

possible\_move\_o([2,2,0,\_,\_,\_,\_,\_,\_], 2, 0) :- !.

possible\_move\_o([\_,\_,\_,0,2,2,\_,\_,\_], 0, 1) :- !.

possible\_move\_o([\_,\_,\_,2,0,2,\_,\_,\_], 1, 1) :- !.

possible\_move\_o([\_,\_,\_,2,2,0,\_,\_,\_], 2, 1) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,0,2,2], 0, 2) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,2,0,2], 1, 2) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,2,2,0], 2, 2) :- !.

possible\_move\_o([0,\_,\_,2,\_,\_,2,\_,\_], 0, 0) :- !.

possible\_move\_o([2,\_,\_,0,\_,\_,2,\_,\_], 0, 1) :- !.

possible\_move\_o([2,\_,\_,2,\_,\_,0,\_,\_], 0, 2) :- !.

possible\_move\_o([\_,0,\_,\_,2,\_,\_,2,\_], 1, 0) :- !.

possible\_move\_o([\_,2,\_,\_,0,\_,\_,2,\_], 1, 1) :- !.

possible\_move\_o([\_,2,\_,\_,2,\_,\_,0,\_], 1, 2) :- !.

possible\_move\_o([\_,\_,0,\_,\_,2,\_,\_,2], 2, 0) :- !.

possible\_move\_o([\_,\_,2,\_,\_,0,\_,\_,2], 2, 1) :- !.

possible\_move\_o([\_,\_,2,\_,\_,2,\_,\_,0], 2, 2) :- !.

possible\_move\_o([0,\_,\_,\_,2,\_,\_,\_,2], 0, 0) :- !.

possible\_move\_o([2,\_,\_,\_,0,\_,\_,\_,2], 1, 1) :- !.

possible\_move\_o([2,\_,\_,\_,2,\_,\_,\_,0], 2, 2) :- !.

possible\_move\_o([\_,\_,0,\_,2,\_,2,\_,\_], 2, 0) :- !.

possible\_move\_o([\_,\_,2,\_,0,\_,2,\_,\_], 1, 1) :- !.

possible\_move\_o([\_,\_,2,\_,2,\_,0,\_,\_], 0, 2) :- !.

%Ситуации, в которой нужно предотвратить выигрыш противника

possible\_move\_o([0,1,1,\_,\_,\_,\_,\_,\_], 0, 0) :- !.

possible\_move\_o([1,0,1,\_,\_,\_,\_,\_,\_], 1, 0) :- !.

possible\_move\_o([1,1,0,\_,\_,\_,\_,\_,\_], 2, 0) :- !.

possible\_move\_o([\_,\_,\_,0,1,1,\_,\_,\_], 0, 1) :- !.

possible\_move\_o([\_,\_,\_,1,0,1,\_,\_,\_], 1, 1) :- !.

possible\_move\_o([\_,\_,\_,1,1,0,\_,\_,\_], 2, 1) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,0,1,1], 0, 2) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,1,0,1], 1, 2) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,1,1,0], 2, 2) :- !.

possible\_move\_o([0,\_,\_,1,\_,\_,1,\_,\_], 0, 0) :- !.

possible\_move\_o([1,\_,\_,0,\_,\_,1,\_,\_], 0, 1) :- !.

possible\_move\_o([1,\_,\_,1,\_,\_,0,\_,\_], 0, 2) :- !.

possible\_move\_o([\_,0,\_,\_,1,\_,\_,1,\_], 1, 0) :- !.

possible\_move\_o([\_,1,\_,\_,0,\_,\_,1,\_], 1, 1) :- !.

possible\_move\_o([\_,1,\_,\_,1,\_,\_,0,\_], 1, 2) :- !.

possible\_move\_o([\_,\_,0,\_,\_,1,\_,\_,1], 2, 0) :- !.

possible\_move\_o([\_,\_,1,\_,\_,0,\_,\_,1], 2, 1) :- !.

possible\_move\_o([\_,\_,1,\_,\_,1,\_,\_,0], 2, 2) :- !.

possible\_move\_o([0,\_,\_,\_,1,\_,\_,\_,1], 0, 0) :- !.

possible\_move\_o([1,\_,\_,\_,0,\_,\_,\_,1], 1, 1) :- !.

possible\_move\_o([1,\_,\_,\_,1,\_,\_,\_,0], 2, 2) :- !.

possible\_move\_o([\_,\_,0,\_,1,\_,1,\_,\_], 2, 0) :- !.

possible\_move\_o([\_,\_,1,\_,0,\_,1,\_,\_], 1, 1) :- !.

possible\_move\_o([\_,\_,1,\_,1,\_,0,\_,\_], 0, 2) :- !.

%Если центр занят, то занимает левый правый угол.

possible\_move\_o([0, \_,\_, \_, 1, \_, \_, \_, \_], 0, 0):- !.

%Если центр заняли мы (o), а игрок занял одну из угловых позиций

possible\_move\_o([0, \_, \_, \_, 2, \_, \_, \_, 1], 0, 0):- !.

possible\_move\_o([\_, \_, 1, \_, 2, \_, 0, \_, \_], 0, 2):- !.

possible\_move\_o([\_, \_, 0, \_, 2, \_, 1, \_, \_], 2, 0):- !.

possible\_move\_o([1, \_, \_, \_, 2, \_, \_, \_, 0], 2, 2):- !.

%В остальных случаях возможны следующие ходы

possible\_move\_o(\_, 0, 0).

possible\_move\_o(\_, 0, 2).

possible\_move\_o(\_, 2, 0).

possible\_move\_o(\_, 2, 2).

possible\_move\_o(\_, 0, 1).

possible\_move\_o(\_, 1, 0).

possible\_move\_o(\_, 1, 1).

possible\_move\_o(\_, 1, 2).

possible\_move\_o(\_, 2, 1).

%Если центр заняли мы (o), а игрок занял одну из угловых позиций

possible\_move\_o([0, \_, \_, \_, 2, \_, \_, \_, 1], 0, 0):- !.

possible\_move\_o([\_, \_, 1, \_, 2, \_, 0, \_, \_], 0, 2):- !.

possible\_move\_o([\_, \_, 0, \_, 2, \_, 1, \_, \_], 2, 0):- !.

possible\_move\_o([1, \_, \_, \_, 2, \_, \_, \_, 0], 2, 2):- !.

possible\_move\_o([\_, 0,\_, \_, 2, \_, \_, \_, 1], 1, 0) :- !.

possible\_move\_o([\_, 0,1, \_, 2, \_, \_, \_, \_], 1, 0) :- !.

possible\_move\_o([\_, 0,\_, \_, 2, \_, 1, \_, \_], 1, 0) :- !.

possible\_move\_o([1, 0,\_, \_, 2, \_, \_, \_, \_], 1, 0) :- !.

%В остальных случаях возможны следующие ходы

possible\_move\_o(\_, 0, 0).

possible\_move\_o(\_, 1, 0).

possible\_move\_o(\_, 2, 0).

possible\_move\_o(\_, 1, 0).

possible\_move\_o(\_, 1, 1).

possible\_move\_o(\_, 1, 2).

possible\_move\_o(\_, 2, 0).

possible\_move\_o(\_, 2, 1).

possible\_move\_o(\_, 2, 2).

**Задача №5 (Lab7-1-3.pl):**

:- use\_module(library(pce)).

%Окно старта игры

menu():-

new(Main, dialog('Крестики нолики')),

new(Start,button('Начать игру', message(@prolog, init))),

new(Exit,button('Выход', message(Main, destroy))),

send(Main, append, Start),

send(Main, append, Exit, below),

send(Main, open).

%Окно игры

init() :-

new(Game, dialog('Игра')),

new(GameField1, button('X')),

new(GameField2, button('X')),

new(GameField3, button('X')),

new(GameField4, button('X')),

new(GameField5, button('X')),

new(GameField6, button('X')),

new(GameField7, button('X')),

new(GameField8, button('X')),

new(GameField9, button('X')),

new(ClickField1,button('1', message(@prolog, change, GameField1, GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9))),

new(ClickField2,button('2', message(@prolog, change, GameField2, GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9))),

new(ClickField3,button('3', message(@prolog, change, GameField3, GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9))),

new(ClickField4,button('4', message(@prolog, change, GameField4, GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9))),

new(ClickField5,button('5', message(@prolog, change, GameField5, GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9))),

new(ClickField6,button('6', message(@prolog, change, GameField6, GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9))),

new(ClickField7,button('7', message(@prolog, change, GameField7, GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9))),

new(ClickField8,button('8', message(@prolog, change, GameField8, GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9))),

new(ClickField9,button('9', message(@prolog, change, GameField9, GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9))),

send(Game, append, ClickField1),

send(Game, append, ClickField2, right),

send(Game, append, ClickField3, right),

send(Game, append, GameField1, right),

send(Game, append, GameField2, right),

send(Game, append, GameField3, right),

send(Game, append, ClickField4, below),

send(Game, append, ClickField5, right),

send(Game, append, ClickField6, right),

send(Game, append, GameField4, right),

send(Game, append, GameField5, right),

send(Game, append, GameField6, right),

send(Game, append, ClickField7, below),

send(Game, append, ClickField8, right),

send(Game, append, ClickField9, right),

send(Game, append, GameField7, right),

send(Game, append, GameField8, right),

send(Game, append, GameField9, right),

send(Game, open),

game1([GameField1, GameField2, GameField3, GameField4, GameField5, GameField6, GameField7, GameField8, GameField9]).

%Начально состояние

game1(ListField) :-

change\_field(ListField, [0,0,0,0,0,0,0,0,0]).

%0 - Незанятое поле

from\_symbol\_to\_digit(0, ' ').

%1 - крестик

from\_symbol\_to\_digit(1, 'X').

%2 - нолик

from\_symbol\_to\_digit(2, 'O').

%Изменение состояния полей GameFieldN

change\_field(ListField, State) :-

get\_element(State, 0, SymbolA),

get\_element(State, 1, SymbolB),

get\_element(State, 2, SymbolC),

get\_element(State, 3, SymbolD),

get\_element(State, 4, SymbolE),

get\_element(State, 5, SymbolF),

get\_element(State, 6, SymbolG),

get\_element(State, 7, SymbolH),

get\_element(State, 8, SymbolI),

from\_symbol\_to\_digit(SymbolA, DigitA),

from\_symbol\_to\_digit(SymbolB, DigitB),

from\_symbol\_to\_digit(SymbolC, DigitC),

from\_symbol\_to\_digit(SymbolD, DigitD),

from\_symbol\_to\_digit(SymbolE, DigitE),

from\_symbol\_to\_digit(SymbolF, DigitF),

from\_symbol\_to\_digit(SymbolG, DigitG),

from\_symbol\_to\_digit(SymbolH, DigitH),

from\_symbol\_to\_digit(SymbolI, DigitI),

get\_element(ListField, 0, A),

get\_element(ListField, 1, B),

get\_element(ListField, 2, C),

get\_element(ListField, 3, D),

get\_element(ListField, 4, E),

get\_element(ListField, 5, F),

get\_element(ListField, 6, G),

get\_element(ListField, 7, H),

get\_element(ListField, 8, I),

send(A, name, DigitA),

send(B, name, DigitB),

send(C, name, DigitC),

send(D, name, DigitD),

send(E, name, DigitE),

send(F, name, DigitF),

send(G, name, DigitG),

send(H, name, DigitH),

send(I, name, DigitI).

change(X, A, B, C, D, E, F, J, H, I) :-

get(X, name, X1),

from\_symbol\_to\_digit(N, X1),

N = 0,

send(X, name, 'X'),

get\_state([A, B, C, D, E, F, J, H, I], State), !,

not(check(State)),

Symbol is 2,

possible\_move\_o(State, X0, Y0),

make\_move(X0, Y0, State, Symbol, NewState), !,

change\_field([A, B, C, D, E, F, J, H, I], NewState),

not(check(NewState)).

%Проверка того, что поле свободно.

is\_free(X, Y, State) :-

N is (Y \* 3 + X),

get\_element(State, N, A),

A = 0.

%Выполнение хода

make\_move(X, Y, State, Symbol, State\_next) :-

is\_free(X, Y, State),

N is Y\*3+X,

delete\_element\_list(State, N, State1),

insert\_element\_list(State1, Symbol, N, State\_next).

%Проверка того, что победили крестики

check(State) :-

state\_win\_x(StateX),

member(State, [StateX]),

new(Dialog, dialog("Крестики победили")),

new(F, text("Крестики победили")),

send(Dialog, append, F),

send(Dialog, open), !.

%Проверка того, что победили нолики

check(State) :-

state\_win\_o(StateO),

member(State, [StateO]),

new(Dialog, dialog("Нолики победили")),

new(F, text("Нолики победили")),

send(Dialog, append, F),

send(Dialog, open), !.

%Проверка того, что игра закончилась ничьей

check(State) :-

not(member(0, State)),

new(Dialog, dialog("Ничья")),

new(F, text("Ничья")),

send(Dialog, append, F),

send(Dialog, open),!.

%Варианты ситуаций игры, когда побеждают крестики

state\_win\_x([1,1,1,\_,\_,\_,\_,\_,\_]).

state\_win\_x([\_,\_,\_,1,1,1,\_,\_,\_]).

state\_win\_x([\_,\_,\_,\_,\_,\_,1,1,1]).

state\_win\_x([1,\_,\_,1,\_,\_,1,\_,\_]).

state\_win\_x([\_,1,\_,\_,1,\_,\_,1,\_]).

state\_win\_x([\_,\_,1,\_,\_,1,\_,\_,1]).

state\_win\_x([1,\_,\_,\_,1,\_,\_,\_,1]).

state\_win\_x([\_,\_,1,\_,1,\_,1,\_,\_]).

%Варианты ситуаций игры, когда побеждают нолики

state\_win\_o([2,2,2,\_,\_,\_,\_,\_,\_]).

state\_win\_o([\_,\_,\_,2,2,2,\_,\_,\_]).

state\_win\_o([\_,\_,\_,\_,\_,\_,2,2,2]).

state\_win\_o([2,\_,\_,2,\_,\_,2,\_,\_]).

state\_win\_o([\_,2,\_,\_,2,\_,\_,2,\_]).

state\_win\_o([\_,\_,2,\_,\_,2,\_,\_,2]).

state\_win\_o([2,\_,\_,\_,2,\_,\_,\_,2]).

state\_win\_o([\_,\_,2,\_,2,\_,2,\_,\_]).

%Получение состояния

get\_state(ListField, [A, B, C, D, E, F, G, H, I]) :-

get\_element(ListField, 0, A2),

get\_element(ListField, 1, B2),

get\_element(ListField, 2, C2),

get\_element(ListField, 3, D2),

get\_element(ListField, 4, E2),

get\_element(ListField, 5, F2),

get\_element(ListField, 6, G2),

get\_element(ListField, 7, H2),

get\_element(ListField, 8, I2),

get(A2, name, A1),

get(B2, name, B1),

get(C2, name, C1),

get(D2, name, D1),

get(E2, name, E1),

get(F2, name, F1),

get(G2, name, G1),

get(H2, name, H1),

get(I2, name, I1),

from\_symbol\_to\_digit(A, A1),

from\_symbol\_to\_digit(B, B1),

from\_symbol\_to\_digit(C, C1),

from\_symbol\_to\_digit(D, D1),

from\_symbol\_to\_digit(E, E1),

from\_symbol\_to\_digit(F, F1),

from\_symbol\_to\_digit(G, G1),

from\_symbol\_to\_digit(H, H1),

from\_symbol\_to\_digit(I, I1).

%Получение элемента по индексу

get\_element([Head|\_], 0, Head).

get\_element([\_|[Head1|Tail]], N, Element) :-

N1 is (N - 1),

get\_element([Head1|Tail], N1, Element).

%Вставка элемента по индексу

insert\_element\_list([], Element, 0, [Element]).

insert\_element\_list([Head|Tail], Element, 0, [Element|[Head|Tail]]).

insert\_element\_list([Head|Tail], Element, N, [Head|Tail1]) :-

(N1 is N - 1),

insert\_element\_list(Tail, Element, N1, Tail1).

%Удаление элемента по индексу

delete\_element\_list([\_|Tail], 0, Tail).

delete\_element\_list([Head|Tail], N, [Head|Tail1]) :-

N1 is (N - 1),

delete\_element\_list(Tail, N1, Tail1).

%Если центр не занят, то занимаем его

possible\_move\_o([\_, \_,\_, \_, 0, \_, \_, \_, \_], 1, 1) :- !.

%Ситуация при которой, данный ход является выигрышным

possible\_move\_o([0,2,2,\_,\_,\_,\_,\_,\_], 0, 0) :- !.

possible\_move\_o([2,0,2,\_,\_,\_,\_,\_,\_], 1, 0) :- !.

possible\_move\_o([2,2,0,\_,\_,\_,\_,\_,\_], 2, 0) :- !.

possible\_move\_o([\_,\_,\_,0,2,2,\_,\_,\_], 0, 1) :- !.

possible\_move\_o([\_,\_,\_,2,0,2,\_,\_,\_], 1, 1) :- !.

possible\_move\_o([\_,\_,\_,2,2,0,\_,\_,\_], 2, 1) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,0,2,2], 0, 2) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,2,0,2], 1, 2) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,2,2,0], 2, 2) :- !.

possible\_move\_o([0,\_,\_,2,\_,\_,2,\_,\_], 0, 0) :- !.

possible\_move\_o([2,\_,\_,0,\_,\_,2,\_,\_], 0, 1) :- !.

possible\_move\_o([2,\_,\_,2,\_,\_,0,\_,\_], 0, 2) :- !.

possible\_move\_o([\_,0,\_,\_,2,\_,\_,2,\_], 1, 0) :- !.

possible\_move\_o([\_,2,\_,\_,0,\_,\_,2,\_], 1, 1) :- !.

possible\_move\_o([\_,2,\_,\_,2,\_,\_,0,\_], 1, 2) :- !.

possible\_move\_o([\_,\_,0,\_,\_,2,\_,\_,2], 2, 0) :- !.

possible\_move\_o([\_,\_,2,\_,\_,0,\_,\_,2], 2, 1) :- !.

possible\_move\_o([\_,\_,2,\_,\_,2,\_,\_,0], 2, 2) :- !.

possible\_move\_o([0,\_,\_,\_,2,\_,\_,\_,2], 0, 0) :- !.

possible\_move\_o([2,\_,\_,\_,0,\_,\_,\_,2], 1, 1) :- !.

possible\_move\_o([2,\_,\_,\_,2,\_,\_,\_,0], 2, 2) :- !.

possible\_move\_o([\_,\_,0,\_,2,\_,2,\_,\_], 2, 0) :- !.

possible\_move\_o([\_,\_,2,\_,0,\_,2,\_,\_], 1, 1) :- !.

possible\_move\_o([\_,\_,2,\_,2,\_,0,\_,\_], 0, 2) :- !.

%Ситуации, в которой нужно предотвратить выигрыш противника

possible\_move\_o([0,1,1,\_,\_,\_,\_,\_,\_], 0, 0) :- !.

possible\_move\_o([1,0,1,\_,\_,\_,\_,\_,\_], 1, 0) :- !.

possible\_move\_o([1,1,0,\_,\_,\_,\_,\_,\_], 2, 0) :- !.

possible\_move\_o([\_,\_,\_,0,1,1,\_,\_,\_], 0, 1) :- !.

possible\_move\_o([\_,\_,\_,1,0,1,\_,\_,\_], 1, 1) :- !.

possible\_move\_o([\_,\_,\_,1,1,0,\_,\_,\_], 2, 1) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,0,1,1], 0, 2) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,1,0,1], 1, 2) :- !.

possible\_move\_o([\_,\_,\_,\_,\_,\_,1,1,0], 2, 2) :- !.

possible\_move\_o([0,\_,\_,1,\_,\_,1,\_,\_], 0, 0) :- !.

possible\_move\_o([1,\_,\_,0,\_,\_,1,\_,\_], 0, 1) :- !.

possible\_move\_o([1,\_,\_,1,\_,\_,0,\_,\_], 0, 2) :- !.

possible\_move\_o([\_,0,\_,\_,1,\_,\_,1,\_], 1, 0) :- !.

possible\_move\_o([\_,1,\_,\_,0,\_,\_,1,\_], 1, 1) :- !.

possible\_move\_o([\_,1,\_,\_,1,\_,\_,0,\_], 1, 2) :- !.

possible\_move\_o([\_,\_,0,\_,\_,1,\_,\_,1], 2, 0) :- !.

possible\_move\_o([\_,\_,1,\_,\_,0,\_,\_,1], 2, 1) :- !.

possible\_move\_o([\_,\_,1,\_,\_,1,\_,\_,0], 2, 2) :- !.

possible\_move\_o([0,\_,\_,\_,1,\_,\_,\_,1], 0, 0) :- !.

possible\_move\_o([1,\_,\_,\_,0,\_,\_,\_,1], 1, 1) :- !.

possible\_move\_o([1,\_,\_,\_,1,\_,\_,\_,0], 2, 2) :- !.

possible\_move\_o([\_,\_,0,\_,1,\_,1,\_,\_], 2, 0) :- !.

possible\_move\_o([\_,\_,1,\_,0,\_,1,\_,\_], 1, 1) :- !.

possible\_move\_o([\_,\_,1,\_,1,\_,0,\_,\_], 0, 2) :- !.

%Если центр занят, то занимает левый правый угл.

possible\_move\_o([0, \_,\_, \_, 1, \_, \_, \_, \_], 0, 0):- !.

%Если центр заняли мы (o), а игрок занял одну из угловых позиций

possible\_move\_o([0, \_, \_, \_, 2, \_, \_, \_, 1], 0, 0):- !.

possible\_move\_o([\_, \_, 1, \_, 2, \_, 0, \_, \_], 0, 2):- !.

possible\_move\_o([\_, \_, 0, \_, 2, \_, 1, \_, \_], 2, 0):- !.

possible\_move\_o([1, \_, \_, \_, 2, \_, \_, \_, 0], 2, 2):- !.

%В остальных случаях возможны следующие ходы

possible\_move\_o(\_, 0, 0).

possible\_move\_o(\_, 0, 2).

possible\_move\_o(\_, 2, 0).

possible\_move\_o(\_, 2, 2).

possible\_move\_o(\_, 0, 1).

possible\_move\_o(\_, 1, 0).

possible\_move\_o(\_, 1, 1).

possible\_move\_o(\_, 1, 2).

possible\_move\_o(\_, 2, 1).