My Program:

1st program:

```
% facts
m([sam, chris, marshall, steven, jack, sean, james]).
f([jenny, myra, claire, mari, penelope, nicole, elizabeth]).

family([sam, jenny, [steven, claire]]).
family([marshall, myra, [jack, sean]]).
family([chris, claire, [penelope, nicole, elizabeth]]).
family([steven, elizabeth, [james]]).

% rules
male(X) :-
        m(Male),
        member(X, Male).
female(X) :-
        f(Female),
        member(X, Female).

father(F, X) :-
        family([F, _, Children]),
        member(X, Children).
mother(M, X) :-
        family([_, M, Children]),
        member(X, Children).
parent(P, X) :-
        father(P, X).
parent(P, X) :-
        mother(P, X).

siblings1(X, Y) :-
        family([_, _, Children]),
        member(X, Children),
        member(Y, Children),
        X \= Y.
siblings2(X, Y) :-
        siblings1(X, Y).
```

```
brother1(X, Y) :-
        male(X),
        siblings1(X, Y).
brother2(X, Y) :-
        brother1(X, Y).

sister1(X, Y) :-
        female(X),
        siblings1(X, Y).
sister2(X, Y) :-
        sister1(X, Y).

cousins(X, Y) :-
        parent(PX, X),
        parent(PY, Y),
        siblings1(PX, PY).

uncle(U, Z) :-
        male(U),
        parent(P, Z),
        siblings1(U, P).
aunt(A, Z) :-
        female(A),
        parent(P, Z),
        siblings1(A, P).

grandchild(GC, B) :-
        parent(B, P),
        parent(P, GC).
grandson(GS, B) :-
        male(GS),
        grandchild(GS, B).
granddaughter(GD, B) :-
        female(GD),
        grandchild(GD, B).

greatgrandparent(GGP, G) :-
        parent(GGP, GP),
        grandchild(G, GP).
```

```
ancestor(A, C) :-
        parent(A, C).
ancestor(A, C) :-
        parent(P, C),
        ancestor(A, P).
```

2nd program:

```
membership_test(Element, [Element|_]).
membership_test(Element, [_|List]) :-
  membership_test(Element, List).

first_element(Element, [Element|_]).

last_element(Last_Element, [Last_Element]).
last_element(List, [_|Last_Element]) :-
  last_element(List, Last_Element).

two_adjacent_elements(Element1, Element2, [Element1, Element2|_]).
two_adjacent_elements(Element1, Element2, [_|List]) :-
  two_adjacent_elements(Element1, Element2, List).

three_adjacent_elements(Element1, Element2, Element3, [Element1 , Element2, Element3|_]).
three_adjacent_elements(Element1, Element2, Element3, [_|List]) :-
  three_adjacent_elements(Element1, Element2, Element3, List).

append_list1_to_list2([], List, List).
append_list1_to_list2([List_Head|List_Tail], List2, [List_Head|Append]) :-
        append_list1_to_list2(List_Tail, List2, Append).

delete_element_from_list(Element, [Element|List_Tail], List_Tail).
delete_element_from_list(Element, [H|List_Tail], [H|Delete]) :-
  delete_element_from_list(Element, List_Tail, Delete).

%append_element_to_list([], List, List).

insert_to_list(Element, 1, List, [Element|List]).
insert_to_list(Element, Position, [List_Head|List_Tail], [List_Head|Insert]) :-
        Position > 1,
        New_Position is Position - 1,
            insert_to_list(Element, New_Position, List_Tail, Insert).

list_length([], 0).
list_length([_|List_Tail], Length) :-
  list_length(List_Tail, Updated_Length),
  Length is Updated_Length + 1.

reverse_list([], []).
reverse_list([List_Head|List_Tail], Reverse) :-
  reverse_list(List_Tail, Reversed_Tail),
  append_list1_to_list2(Reversed_Tail, [List_Head], Reverse).

palindrome_list_check([_]).
palindrome_list_check([List_Head|List_Tail]) :-
  palindrome_list_check(List_Head, List_Tail).


display_list([List_Head|List_Tail]) :-
  write(List_Head),
  write(' '),
  display_list(List_Tail).
```

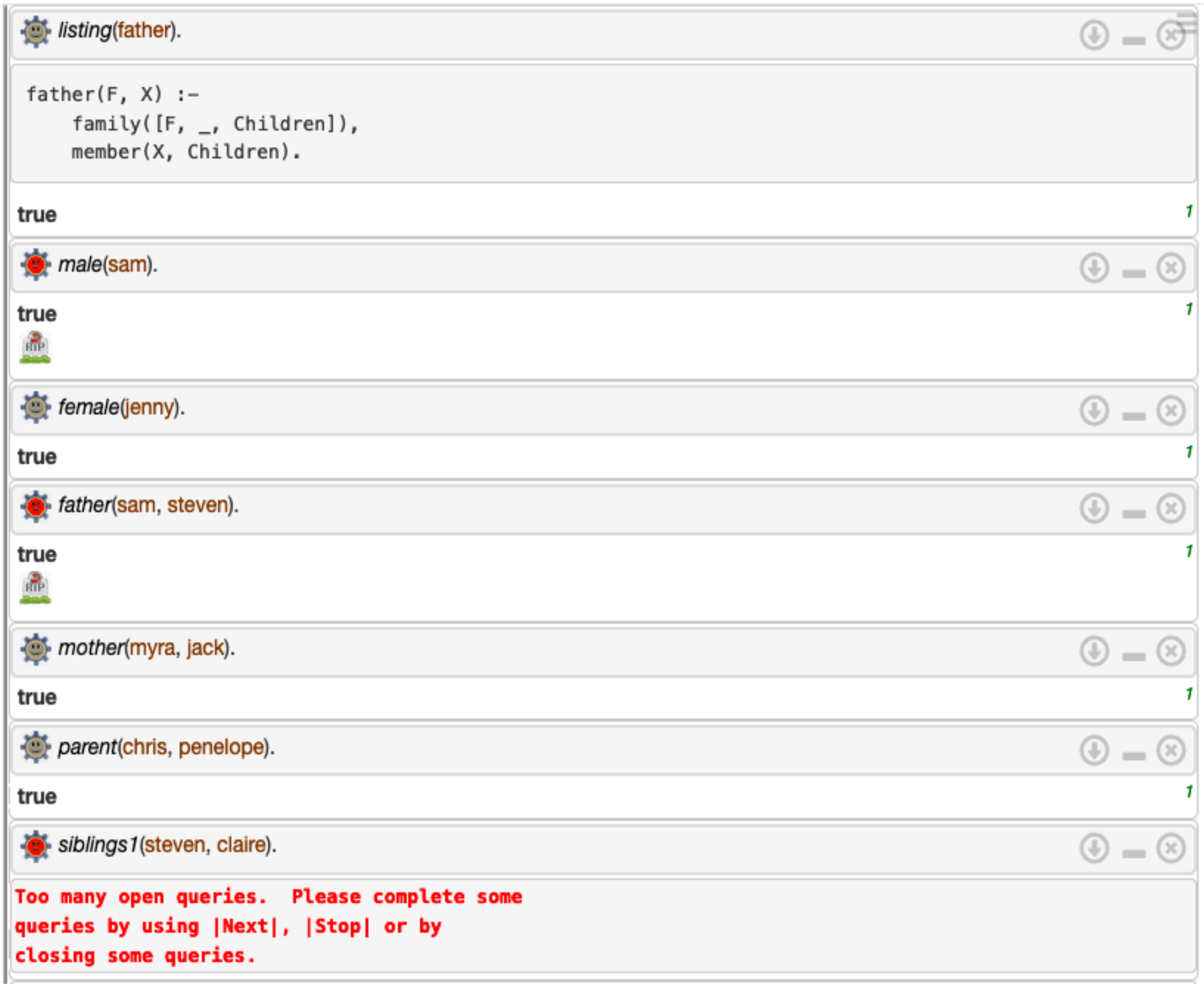
3rd program:

```
safe_movement ([]) .
safe_movement ([Queen |Not_Queen])   :-
  safe_movement (Not_Queen) ,
  do_not_move (Queen, Not_Queen,1) .

do_not_move (_,[],_) .
do_not_move (Current_Position,  [Possible_Position|Queen] , Previous_Position)   :-
  Possible_Position  -  Current_Position =\= Previous_Position,
  Current_Position  -  Possible_Position =\= Previous_Position,
  do_not_move (Current_Position,  Queen,  Previous_Position).

possible_movements (_,[],_) .
bad_movements ([], [] ) .
make_columns ( [], _) .
make_rows([], _) .
```

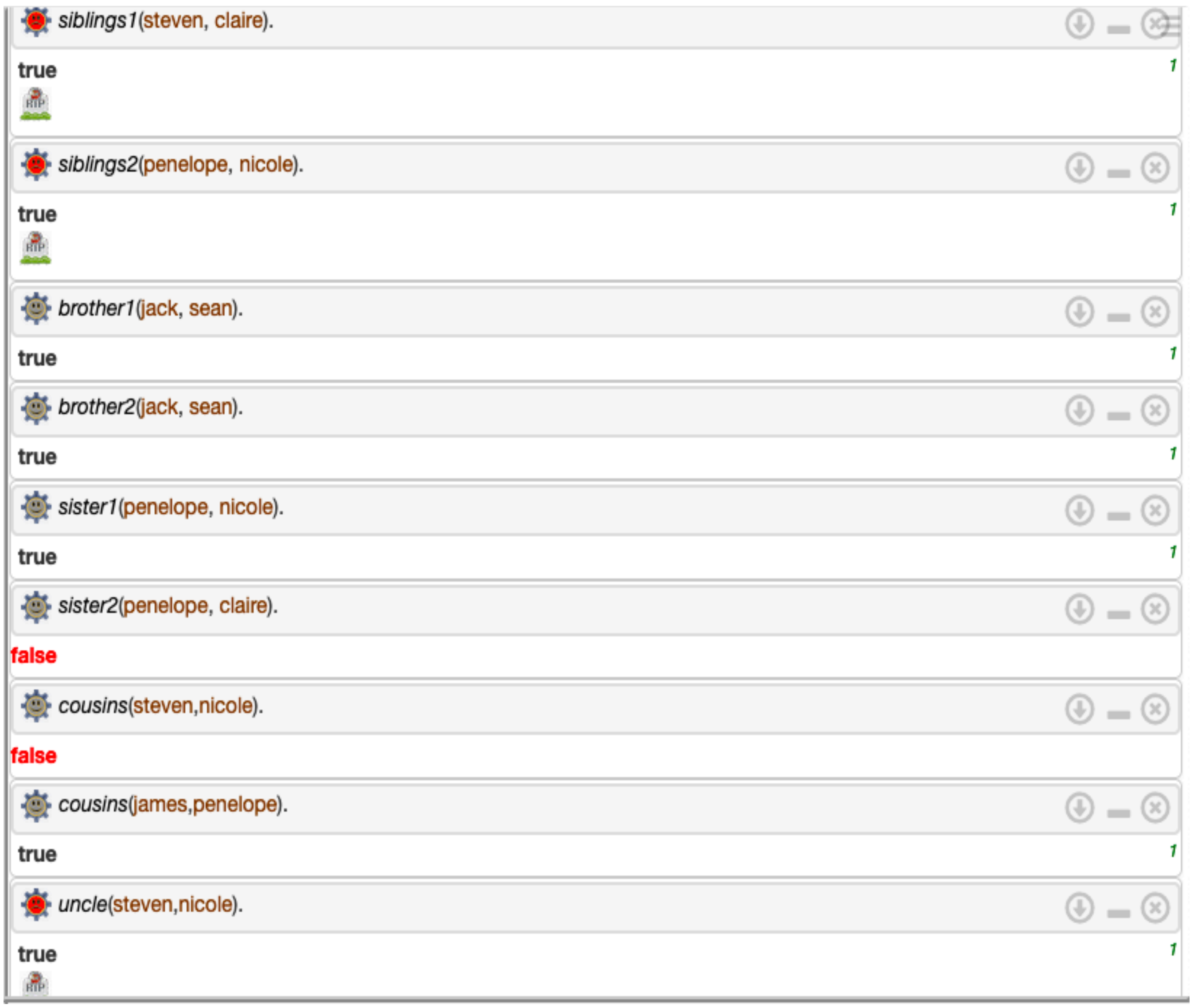
```
placing_queens (Eight_Queens)   :-
  possible_movements ([1,2,3,4,5,6,7,8] ,  Eight_Queens) ,
  safe_movement (Eight_Queens) .

make_moves  :-
     make_columns (A, 1) ,
     make_row (1, A) ,
     write (“-----------------------” ) ,
     placing_queens (Queen) .
```

Output:

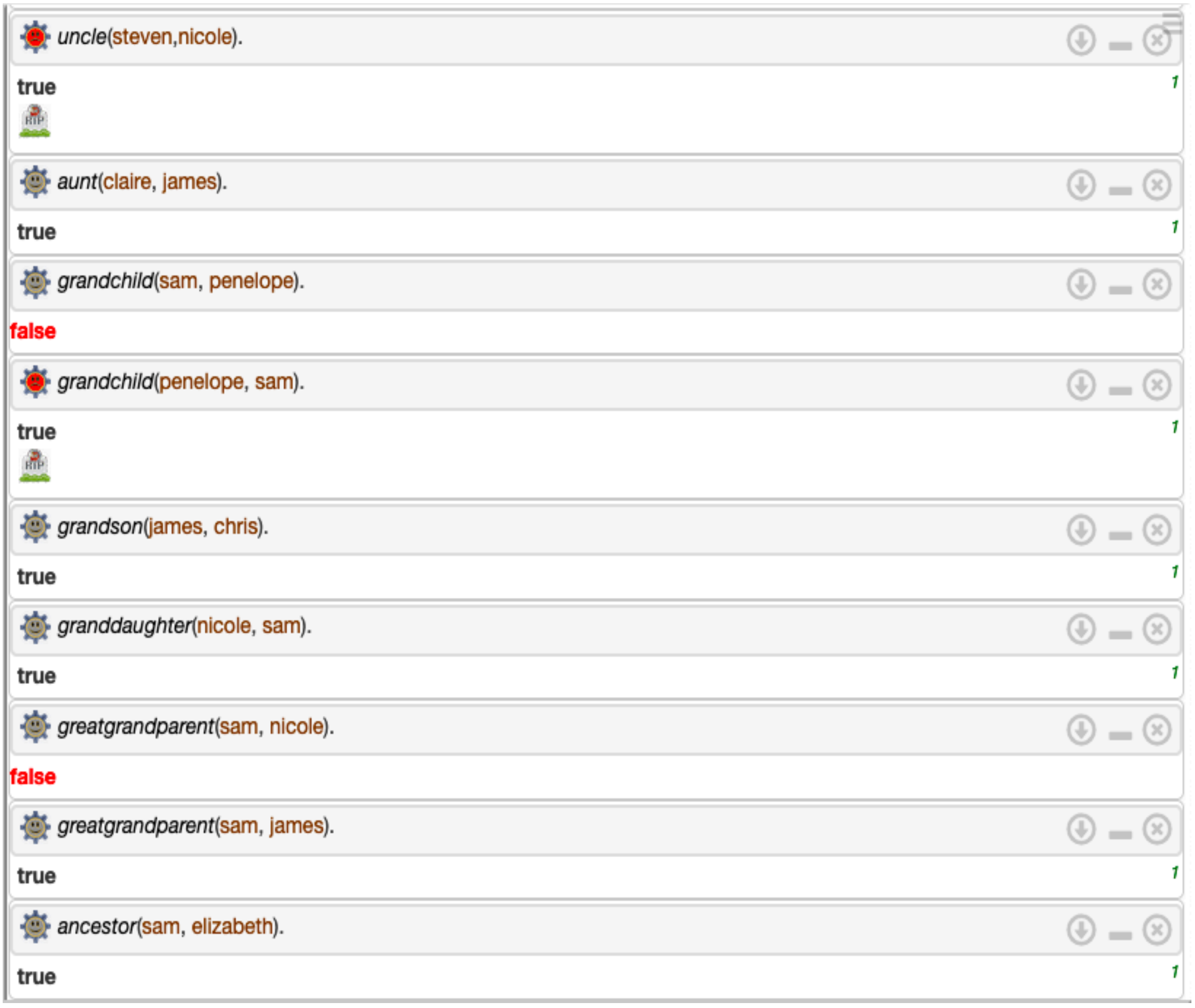
1st Program’s Output:



```
listing(father).

father(F, X) :-
    family([F, _, Children]),
    member(X, Children).

true

male(sam).
true

female(jenny).
true

father(sam, steven).
true

mother(myra, jack).
true

parent(chris, penelope).
true

siblings1(steven, claire).
Too many open queries.  Please complete some
queries by using |Next|, |Stop| or by
closing some queries.
```

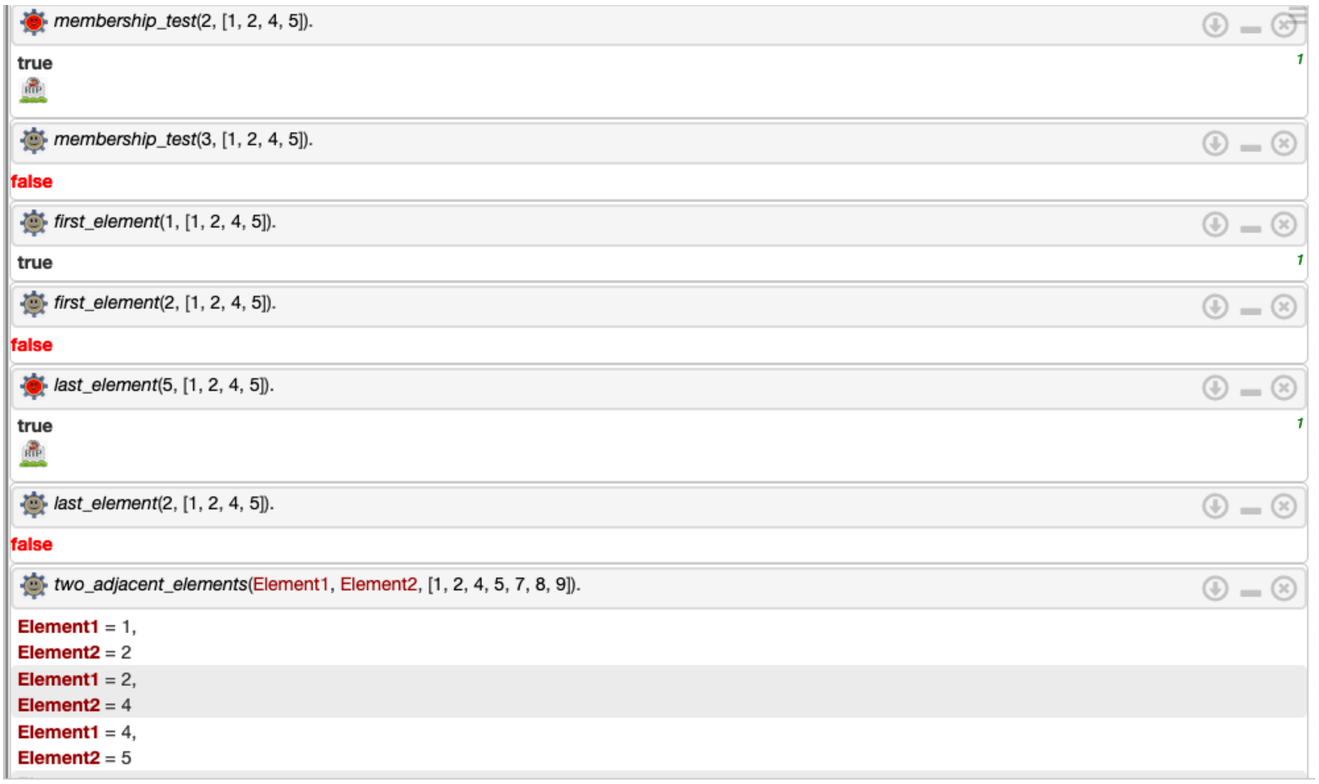


```
siblings1(steven, claire).
true
siblings2(penelope, nicole).
true
brother1(jack, sean).
true
brother2(jack, sean).
true
sister1(penelope, nicole).
true
sister2(penelope, claire).
false
cousins(steven,nicole).
false
cousins(james,penelope).
true
uncle(steven,nicole).
true
```

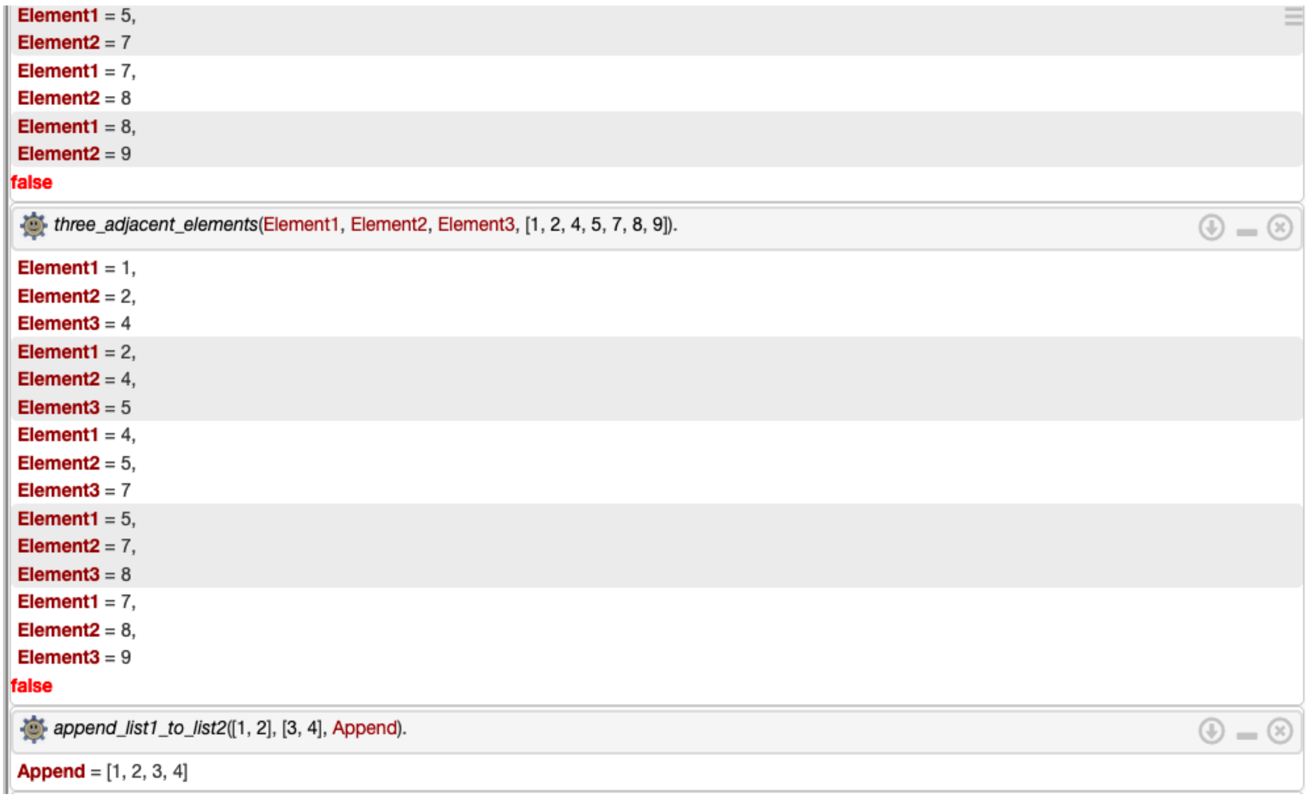


```
uncle(steven,nicole).
true
aunt(claire, james).
true
grandchild(sam, penelope).
false
grandchild(penelope, sam).
true
grandson(james, chris).
true
granddaughter(nicole, sam).
true
greatgrandparent(sam, nicole).
false
greatgrandparent(sam, james).
true
ancestor(sam, elizabeth).
true
```

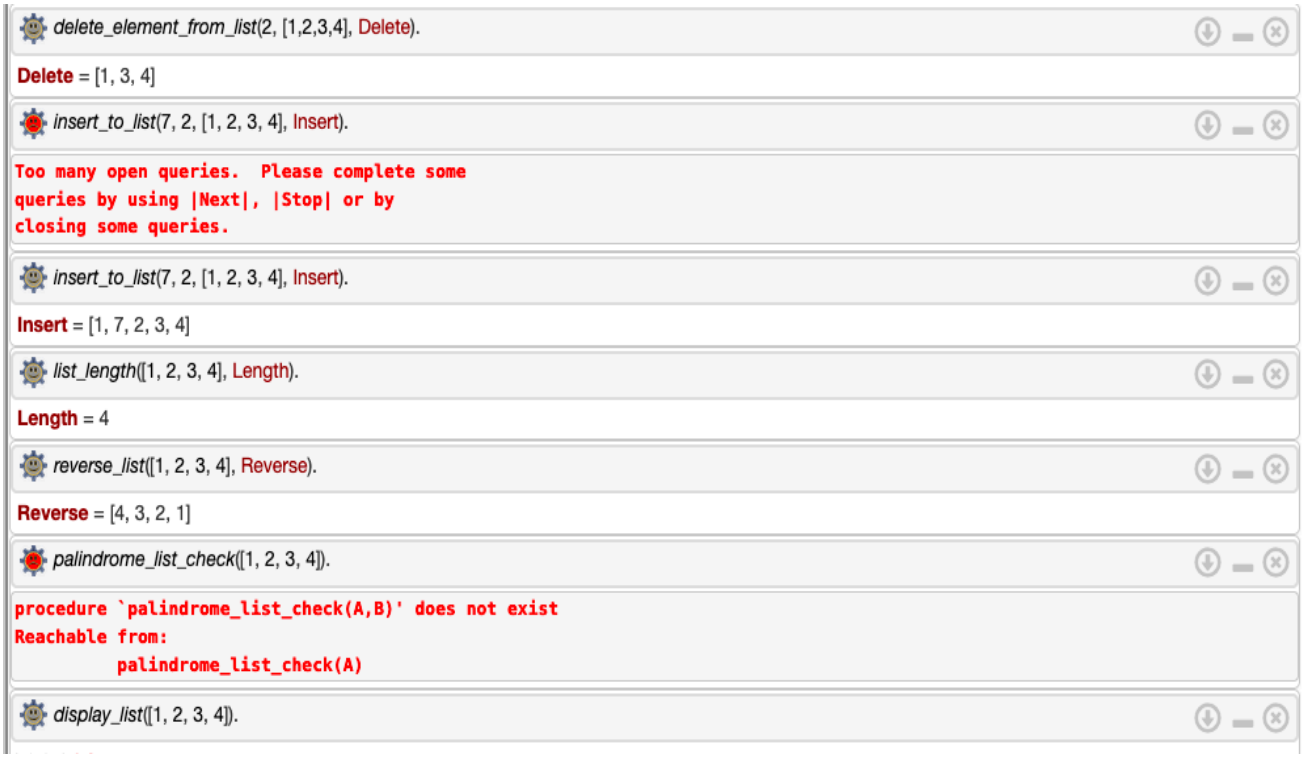
2nd Program's Output:



```
membership_test(2, [1, 2, 4, 5]).
true                                                                  1

membership_test(3, [1, 2, 4, 5]).
false

first_element(1, [1, 2, 4, 5]).
true                                                                  1

first_element(2, [1, 2, 4, 5]).
false

last_element(5, [1, 2, 4, 5]).
true                                                                  1

last_element(2, [1, 2, 4, 5]).
false

two_adjacent_elements(Element1, Element2, [1, 2, 4, 5, 7, 8, 9]).
Element1 = 1,
Element2 = 2
Element1 = 2,
Element2 = 4
Element1 = 4,
Element2 = 5
```

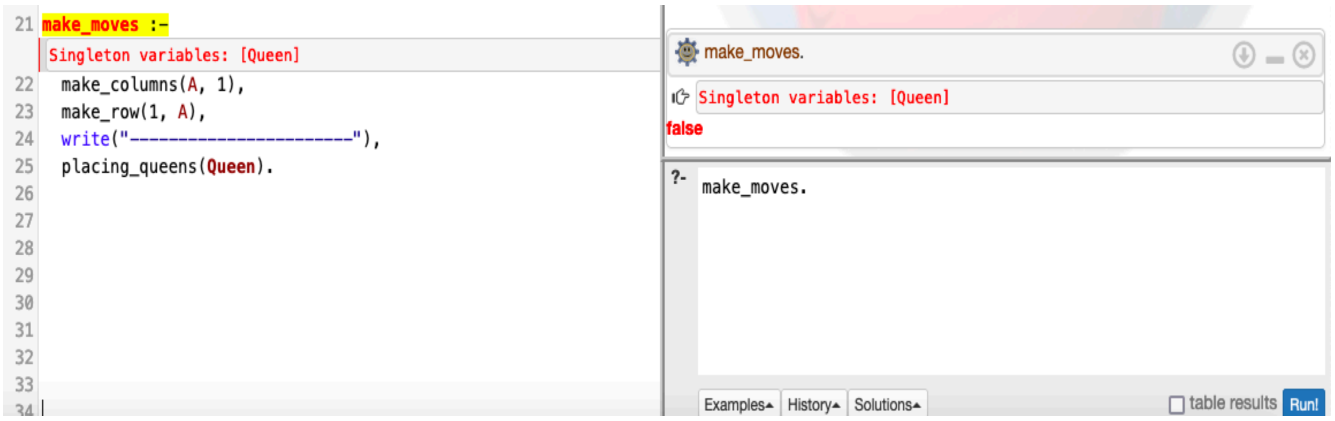


```
Element1 = 5,
Element2 = 7
Element1 = 7,
Element2 = 8
Element1 = 8,
Element2 = 9
false

three_adjacent_elements(Element1, Element2, Element3, [1, 2, 4, 5, 7, 8, 9]).
Element1 = 1,
Element2 = 2,
Element3 = 4
Element1 = 2,
Element2 = 4,
Element3 = 5
Element1 = 4,
Element2 = 5,
Element3 = 7
Element1 = 5,
Element2 = 7,
Element3 = 8
Element1 = 7,
Element2 = 8,
Element3 = 9
false

append_list1_to_list2([1, 2], [3, 4], Append).
Append = [1, 2, 3, 4]
```



```
delete_element_from_list(2, [1,2,3,4], Delete).
Delete = [1, 3, 4]

insert_to_list(7, 2, [1, 2, 3, 4], Insert).
Too many open queries.  Please complete some
queries by using |Next|, |Stop| or by
closing some queries.

insert_to_list(7, 2, [1, 2, 3, 4], Insert).
Insert = [1, 7, 2, 3, 4]

list_length([1, 2, 3, 4], Length).
Length = 4

reverse_list([1, 2, 3, 4], Reverse).
Reverse = [4, 3, 2, 1]

palindrome_list_check([1, 2, 3, 4]).
procedure `palindrome_list_check(A,B)' does not exist
Reachable from:
          palindrome_list_check(A)

display_list([1, 2, 3, 4]).
```

3rd Program's Output:



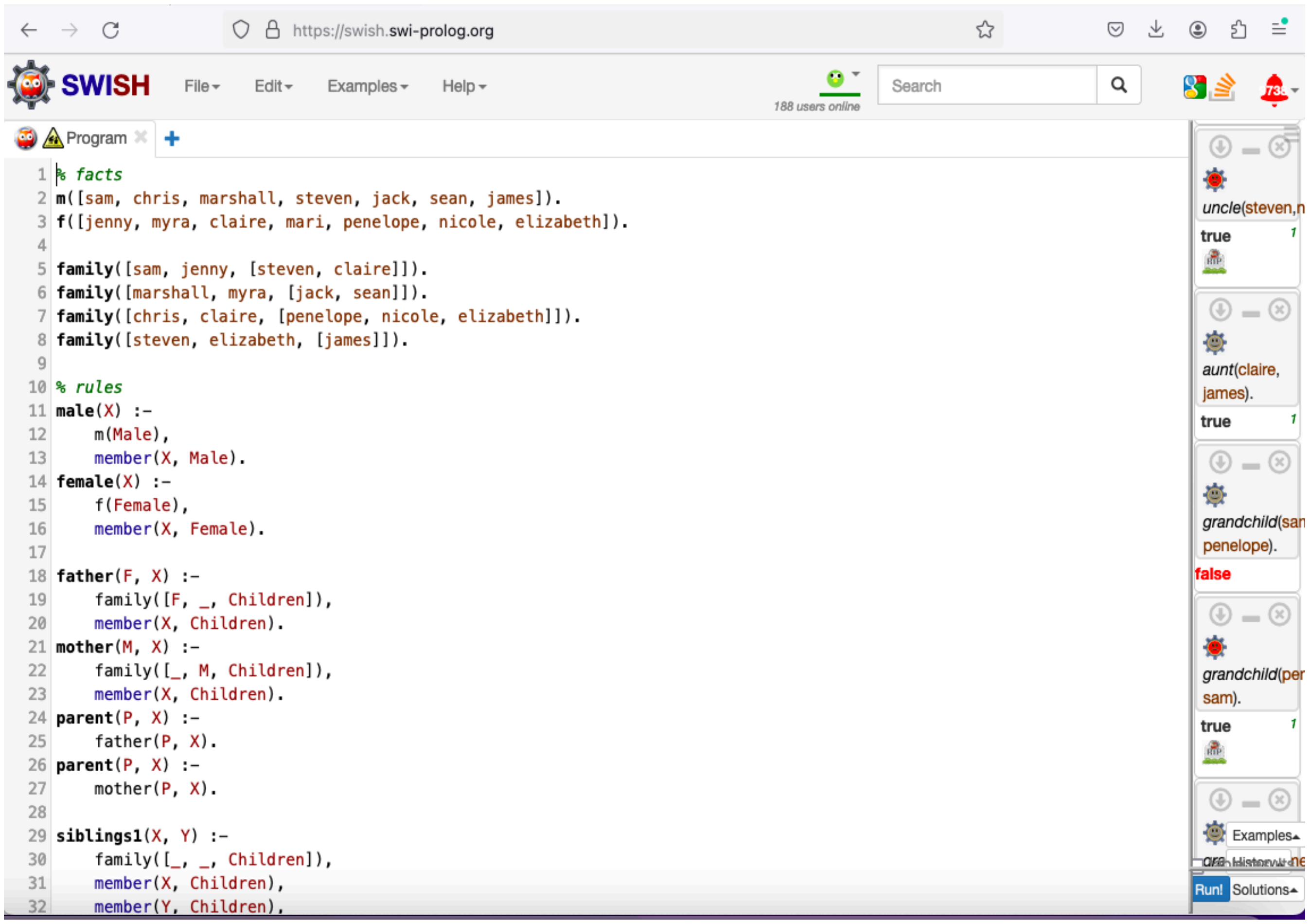
```
make_moves :-
Singleton variables: [Queen]
  make_columns(A, 1),
  make_row(1, A),
  write("----------------------"),
  placing_queens(Queen).
```

```
make_moves.
Singleton variables: [Queen]
false
```

```
?- make_moves.
```

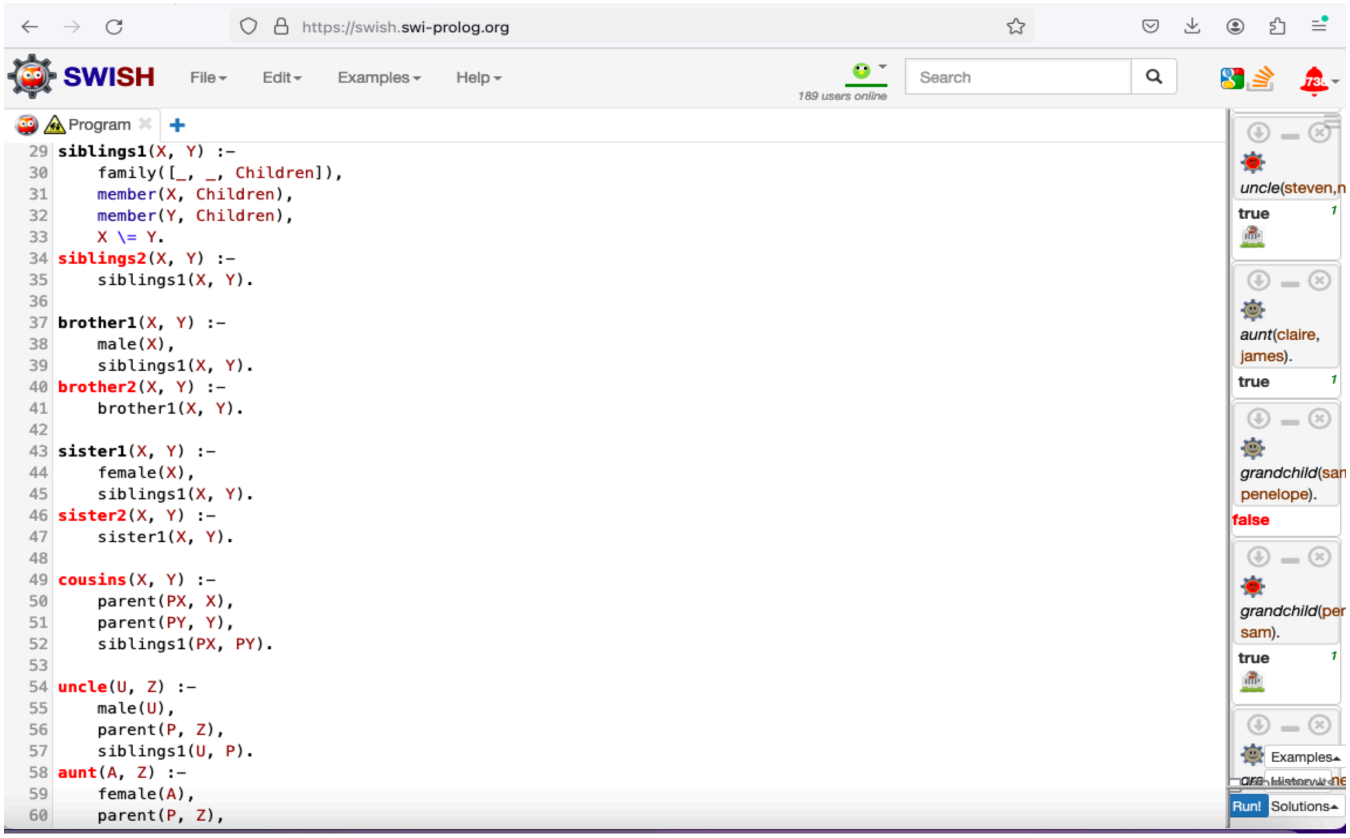
Screenshots of my code:

1st Program:



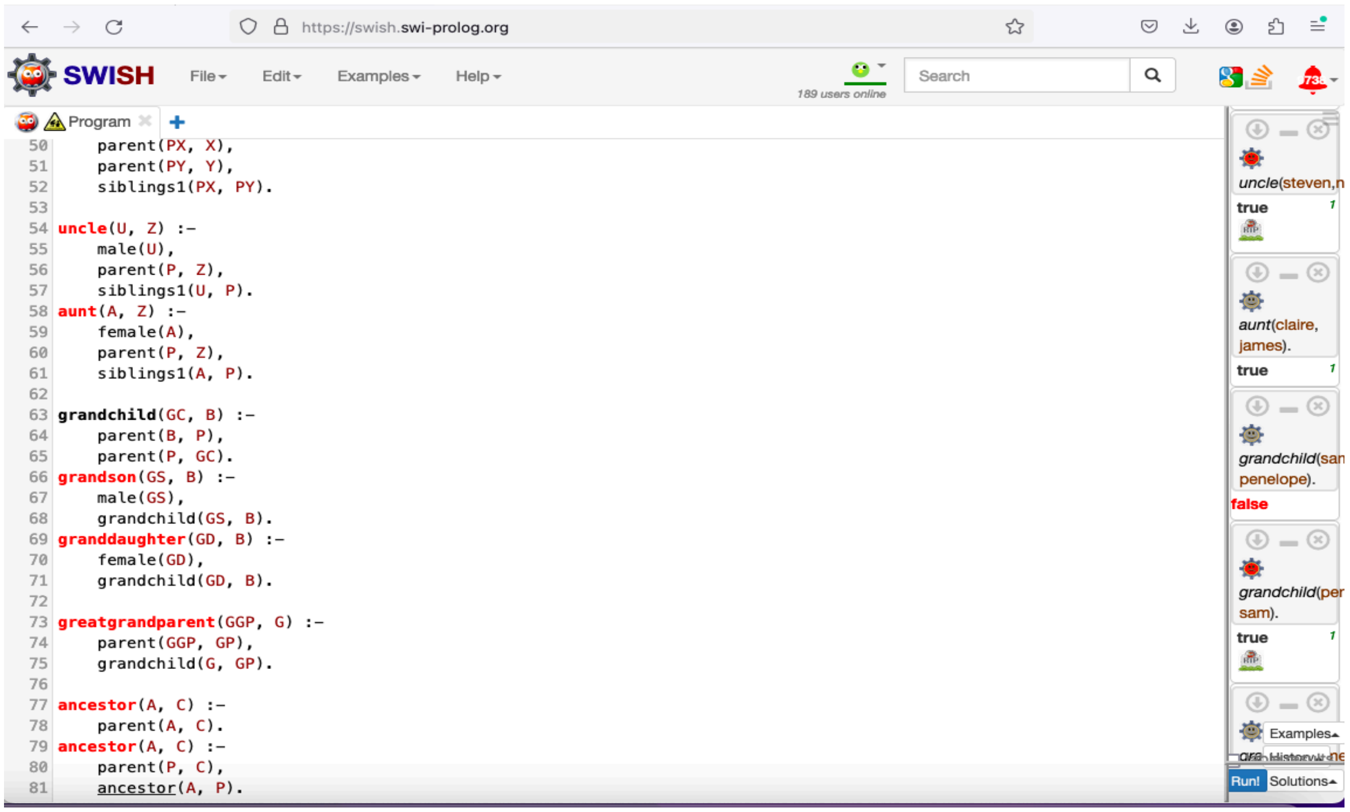
```
% facts
m([sam, chris, marshall, steven, jack, sean, james]).
f([jenny, myra, claire, mari, penelope, nicole, elizabeth]).

family([sam, jenny, [steven, claire]]).
family([marshall, myra, [jack, sean]]).
family([chris, claire, [penelope, nicole, elizabeth]]).
family([steven, elizabeth, [james]]).

% rules
male(X) :-
    m(Male),
    member(X, Male).
female(X) :-
    f(Female),
    member(X, Female).

father(F, X) :-
    family([F, _, Children]),
    member(X, Children).
mother(M, X) :-
    family([_, M, Children]),
    member(X, Children).
parent(P, X) :-
    father(P, X).
parent(P, X) :-
    mother(P, X).

siblings1(X, Y) :-
    family([_, _, Children]),
    member(X, Children),
    member(Y, Children),
```

```
uncle(steven,n
true

aunt(claire,
james).
true

grandchild(san
penelope).
false

grandchild(per
sam).
true
```



```
siblings1(X, Y) :-
    family([_, _, Children]),
    member(X, Children),
    member(Y, Children),
    X \= Y.
siblings2(X, Y) :-
    siblings1(X, Y).

brother1(X, Y) :-
    male(X),
    siblings1(X, Y).
brother2(X, Y) :-
    brother1(X, Y).

sister1(X, Y) :-
    female(X),
    siblings1(X, Y).
sister2(X, Y) :-
    sister1(X, Y).

cousins(X, Y) :-
    parent(PX, X),
    parent(PY, Y),
    siblings1(PX, PY).

uncle(U, Z) :-
    male(U),
    parent(P, Z),
    siblings1(U, P).
aunt(A, Z) :-
    female(A),
    parent(P, Z),
```

```
uncle(steven,n
true
aunt(claire,
james).
true
grandchild(san
penelope).
false
grandchild(per
sam).
true
```

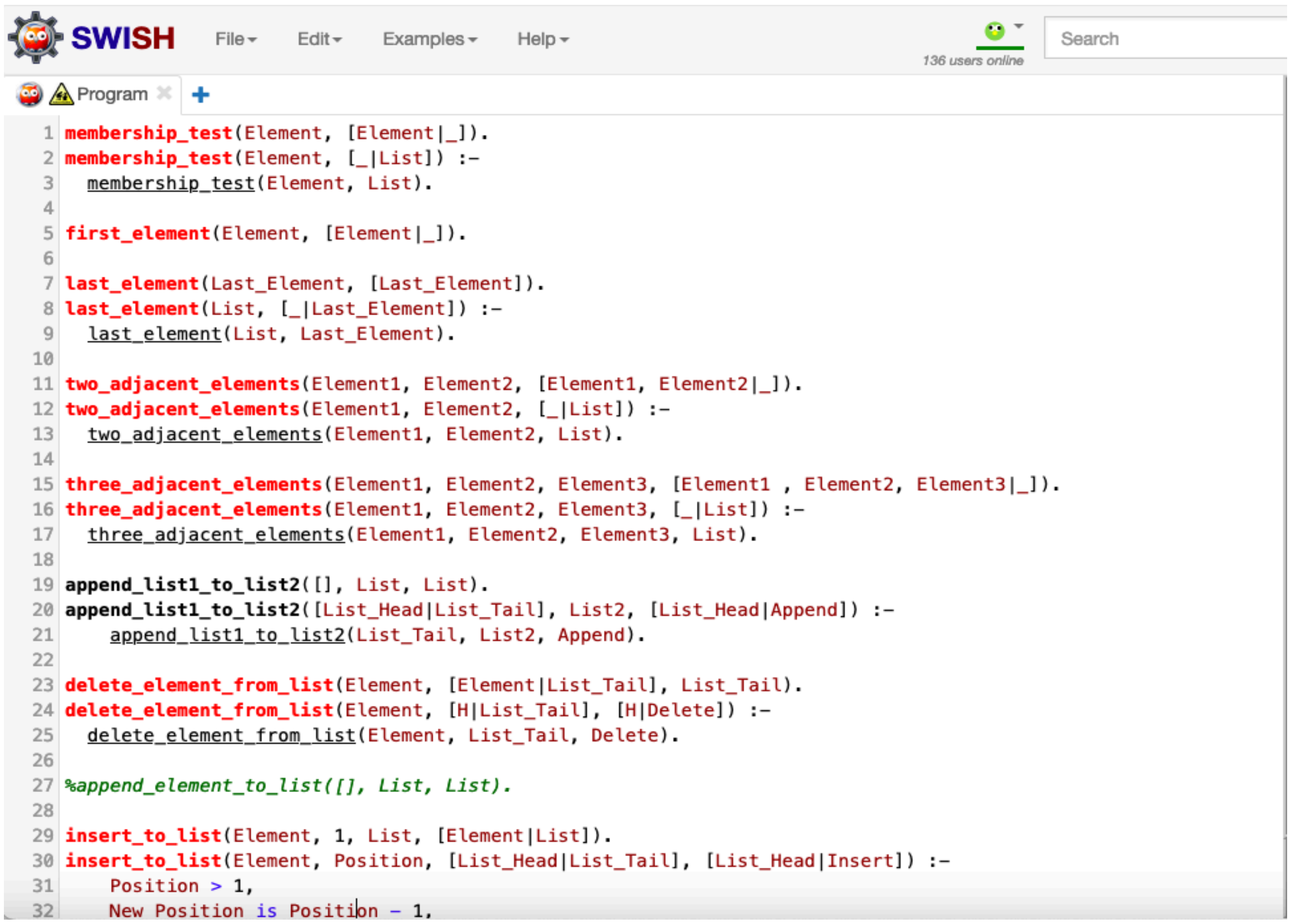


```
    parent(PX, X),
    parent(PY, Y),
    siblings1(PX, PY).

uncle(U, Z) :-
    male(U),
    parent(P, Z),
    siblings1(U, P).
aunt(A, Z) :-
    female(A),
    parent(P, Z),
```

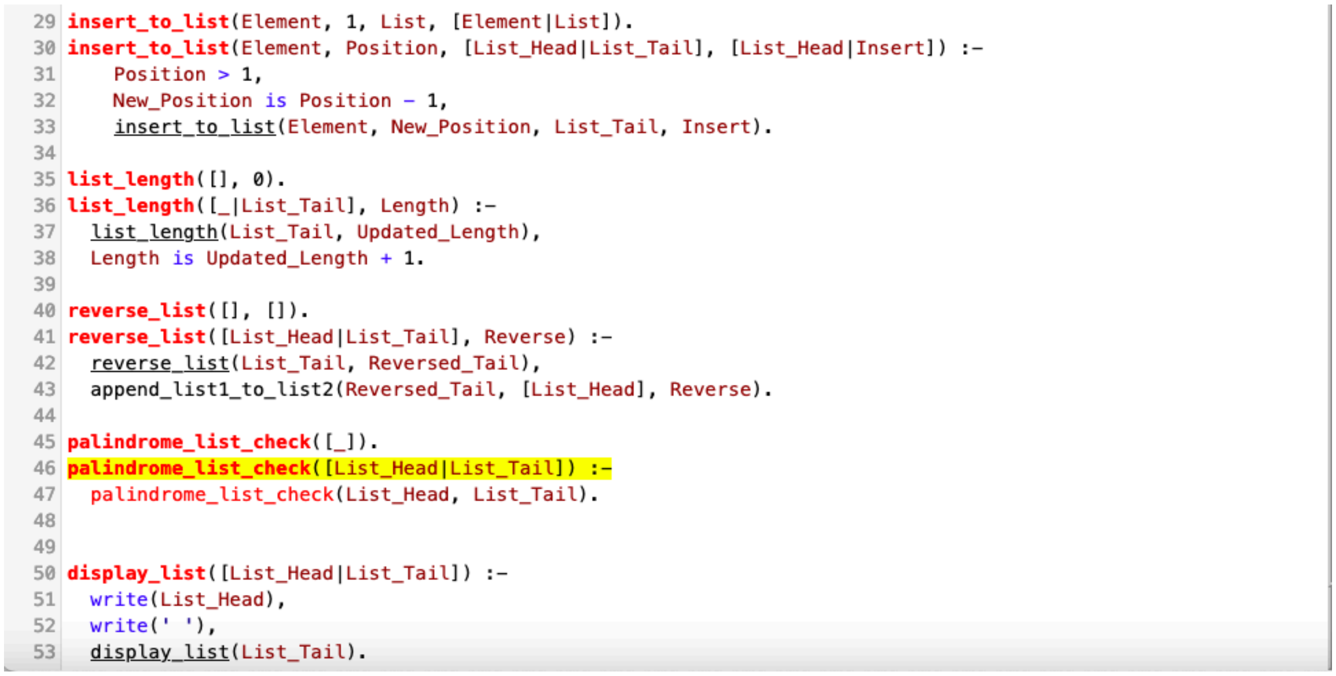
```
    siblings1(A, P).

grandchild(GC, B) :-
    parent(B, P),
    parent(P, GC).
grandson(GS, B) :-
    male(GS),
    grandchild(GS, B).
granddaughter(GD, B) :-
    female(GD),
    grandchild(GD, B).

greatgrandparent(GGP, G) :-
    parent(GGP, GP),
    grandchild(G, GP).

ancestor(A, C) :-
    parent(A, C).
ancestor(A, C) :-
    parent(P, C),
    ancestor(A, P).
```

```
uncle(steven,n
true
aunt(claire,
james).
true
grandchild(san
penelope).
false
grandchild(per
sam).
true
```

2nd Program:



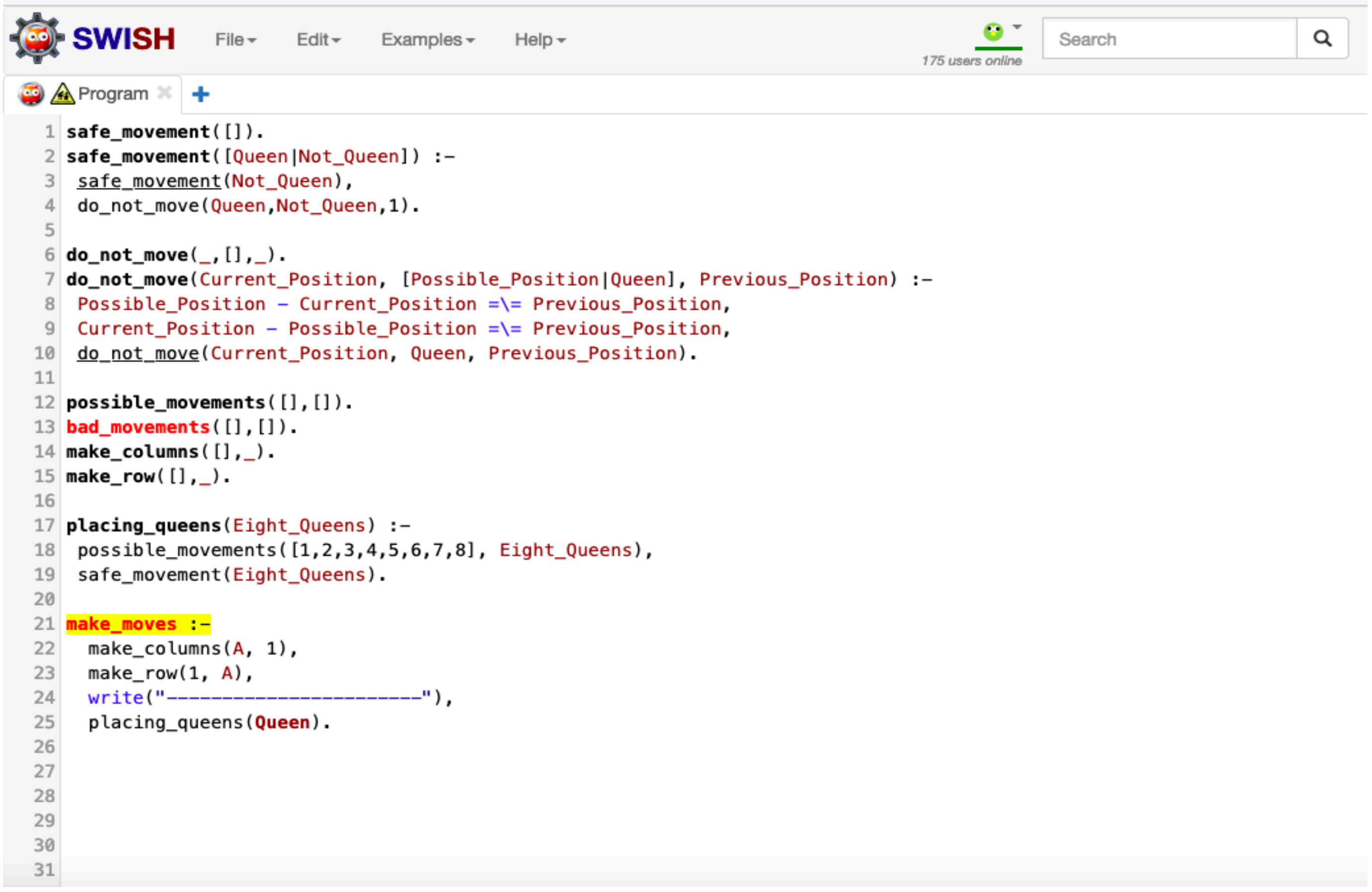
```
membership_test(Element, [Element|_]).
membership_test(Element, [_|List]) :-
  membership_test(Element, List).

first_element(Element, [Element|_]).

last_element(Last_Element, [Last_Element]).
last_element(List, [_|Last_Element]) :-
  last_element(List, Last_Element).

two_adjacent_elements(Element1, Element2, [Element1, Element2|_]).
two_adjacent_elements(Element1, Element2, [_|List]) :-
  two_adjacent_elements(Element1, Element2, List).

three_adjacent_elements(Element1, Element2, Element3, [Element1 , Element2, Element3|_]).
three_adjacent_elements(Element1, Element2, Element3, [_|List]) :-
  three_adjacent_elements(Element1, Element2, Element3, List).

append_list1_to_list2([], List, List).
append_list1_to_list2([List_Head|List_Tail], List2, [List_Head|Append]) :-
    append_list1_to_list2(List_Tail, List2, Append).

delete_element_from_list(Element, [Element|List_Tail], List_Tail).
delete_element_from_list(Element, [H|List_Tail], [H|Delete]) :-
  delete_element_from_list(Element, List_Tail, Delete).

%append_element_to_list([], List, List).

insert_to_list(Element, 1, List, [Element|List]).
insert_to_list(Element, Position, [List_Head|List_Tail], [List_Head|Insert]) :-
    Position > 1,
    New Position is Position - 1,
```



```
insert_to_list(Element, 1, List, [Element|List]).
insert_to_list(Element, Position, [List_Head|List_Tail], [List_Head|Insert]) :-
    Position > 1,
    New_Position is Position - 1,
```

```
    insert_to_list(Element, New_Position, List_Tail, Insert).

list_length([], 0).
list_length([_|List_Tail], Length) :-
  list_length(List_Tail, Updated_Length),
  Length is Updated_Length + 1.

reverse_list([], []).
reverse_list([List_Head|List_Tail], Reverse) :-
  reverse_list(List_Tail, Reversed_Tail),
  append_list1_to_list2(Reversed_Tail, [List_Head], Reverse).

palindrome_list_check([_]).
palindrome_list_check([List_Head|List_Tail]) :-
  palindrome_list_check(List_Head, List_Tail).


display_list([List_Head|List_Tail]) :-
  write(List_Head),
  write(' '),
  display_list(List_Tail).
```

3rd Program:



```
safe_movement([]).
safe_movement([Queen|Not_Queen]) :-
 safe_movement(Not_Queen),
 do_not_move(Queen,Not_Queen,1).

do_not_move(_,[],_).
do_not_move(Current_Position, [Possible_Position|Queen], Previous_Position) :-
 Possible_Position - Current_Position =\= Previous_Position,
 Current_Position - Possible_Position =\= Previous_Position,
 do_not_move(Current_Position, Queen, Previous_Position).

possible_movements([],[]).
bad_movements([],[]).
make_columns([],_).
make_row([],_).

placing_queens(Eight_Queens) :-
 possible_movements([1,2,3,4,5,6,7,8], Eight_Queens),
 safe_movement(Eight_Queens).

make_moves :-
  make_columns(A, 1),
  make_row(1, A),
  write("----------------------"),
  placing_queens(Queen).
```