

Name/

Najd Nader Alsubaie

ID/

431001012

1. Ask Prolog if Albert is the parent of Peter by entering the query.

Answer:

The screenshot shows the SWISH Prolog environment. On the left, a code editor displays a Prolog program named 'family-program'. The program defines a 'parent' predicate with 17 facts and a 'female' predicate with 6 facts. On the right, a query window shows the query 'parent(albert, peter).' being executed, with the result 'true' displayed. The bottom right corner of the interface includes tabs for 'Examples', 'History', and 'Solutions', along with a 'table results' checkbox and a 'Run!' button.

```
1 % parent(Parent, Child)
2 parent(albert, jim).
3 parent(albert, peter).
4 parent(jim, brian).
5 parent(john, darren).
6 parent(peter, lee).
7 parent(peter, sandra).
8 parent(peter, james).
9 parent(peter, kate).
10 parent(peter, kyle).
11 parent(brian, jenny).
12 parent(irene, jim).
13 parent(irene, peter).
14 parent(pat, brian).
15 parent(pat, darren).
16 parent(amanda, jenny).
17
18
19 % female(Person)
20 female(irene).
21 female(pat).
22 female(lee).
23 female(sandra).
24 female(jenny).
25 female(amanda).
26 female(kate).
```

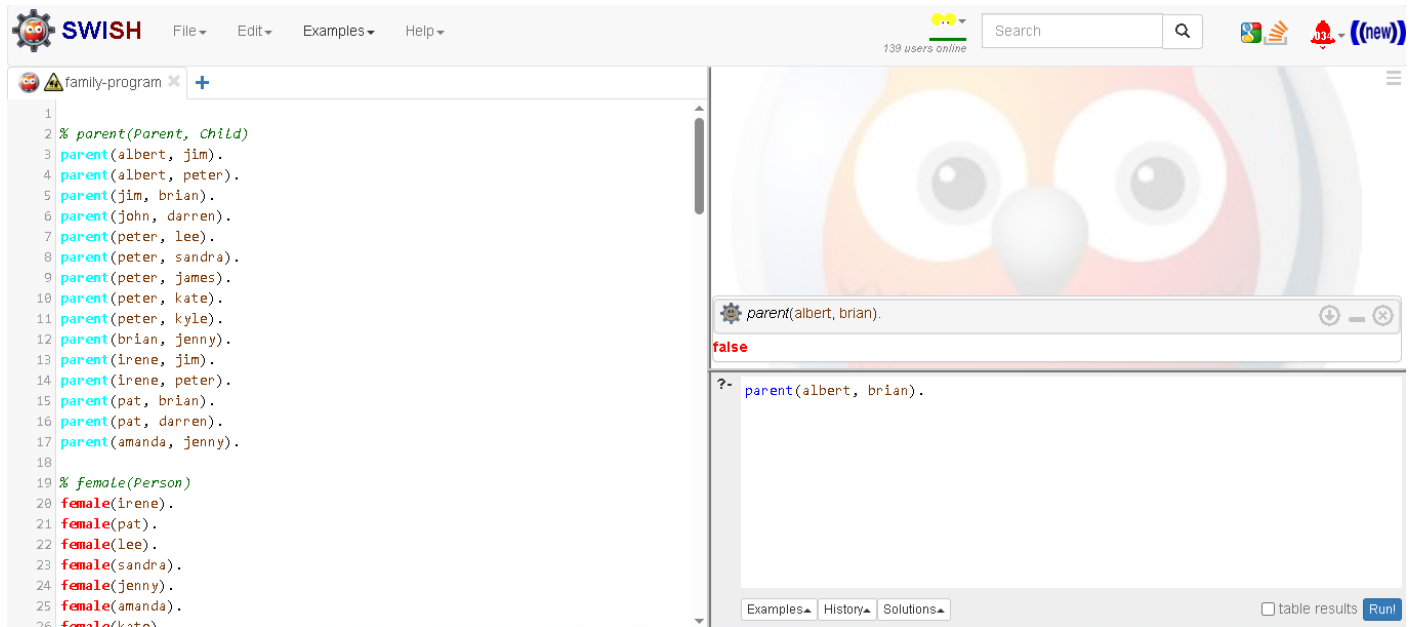
parent(albert, peter).
true

?- parent(albert, peter).

Examples History Solutions table results Run!

2. Ask Prolog if Albert is the parent of Brian by entering the query:

Answer:



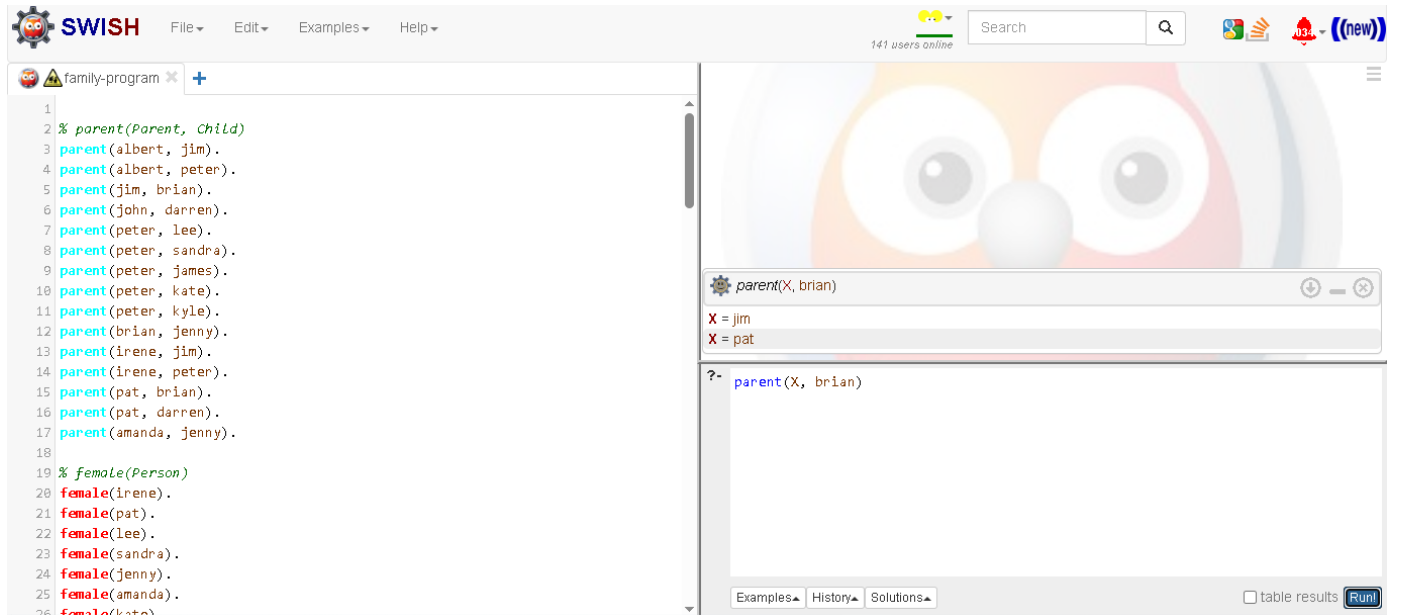
The screenshot shows the SWISH Prolog environment. On the left, a file named 'family-program' is open, containing a Prolog database with the following rules and facts:

```
1 % parent(Parent, Child)
2 parent(albert, jim).
3 parent(albert, peter).
4 parent(jim, brian).
5 parent(john, darren).
6 parent(peter, lee).
7 parent(peter, sandra).
8 parent(peter, james).
9 parent(peter, kate).
10 parent(peter, kyle).
11 parent(brian, jenny).
12 parent(irene, jim).
13 parent(irene, peter).
14 parent(pat, brian).
15 parent(pat, darren).
16 parent(amanda, jenny).
17
18
19 % female(Person)
20 female(irene).
21 female(pat).
22 female(lee).
23 female(sandra).
24 female(jenny).
25 female(amanda).
26 female(kate).
```

On the right, a query window shows the query `parent(albert, brian).` with the result `false`. Below the query window, there is a text area with the query `?- parent(albert, brian).` and buttons for 'Examples', 'History', 'Solutions', 'table results', and 'Run!'.

3. Ask, "Who are the parents of Brian?":

Answer:

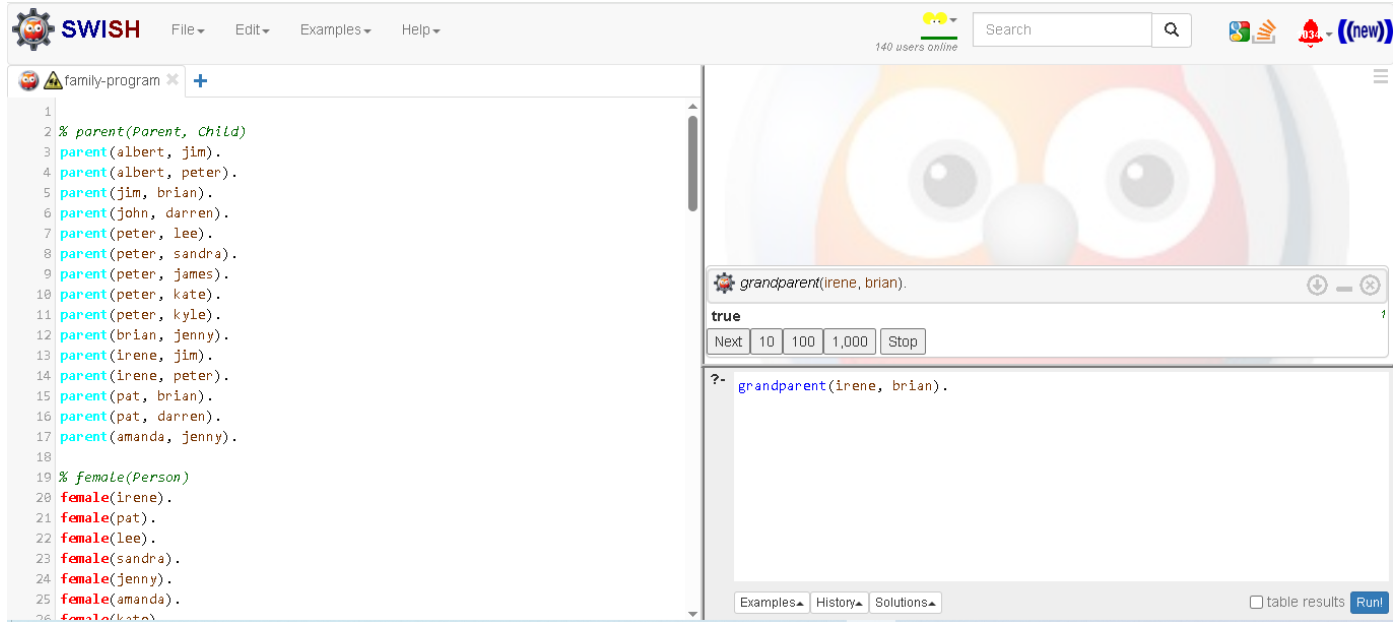


The screenshot shows the SWISH Prolog environment. On the left, the same 'family-program' file is open, containing the same Prolog database as in the previous screenshot.

On the right, a query window shows the query `parent(X, brian)` with the results `X = jim` and `X = pat`. Below the query window, there is a text area with the query `?- parent(X, brian)` and buttons for 'Examples', 'History', 'Solutions', 'table results', and 'Run!'.

4. Ask, "Is Irene a grandparent of Brian?" This can be answered by finding out if Irene is the parent of someone who is the parent of Brian.

Answer:



The screenshot shows the SWISH Prolog IDE interface. On the left, a code editor displays a Prolog program named 'family-program'. The program defines a 'parent' predicate with 17 facts and a 'female' predicate with 6 facts. On the right, a query window shows the query 'grandparent(irene, brian).' which has returned the result 'true'. Below the query window, there are buttons for 'Next', '10', '100', '1,000', and 'Stop'. At the bottom right, there are tabs for 'Examples', 'History', and 'Solutions', and a 'Run!' button.

```
1 % parent(Parent, Child)
2 parent(albert, jim).
3 parent(albert, peter).
4 parent(jim, brian).
5 parent(john, darren).
6 parent(peter, lee).
7 parent(peter, sandra).
8 parent(peter, james).
9 parent(peter, kate).
10 parent(peter, kyle).
11 parent(brian, jenny).
12 parent(irene, jim).
13 parent(irene, peter).
14 parent(pat, brian).
15 parent(pat, darren).
16 parent(amanda, jenny).
17
18
19 % female(Person)
20 female(irene).
21 female(pat).
22 female(lee).
23 female(sandra).
24 female(jenny).
25 female(amanda).
26 female(kate).
```

grandparent(irene, brian).

true

Next 10 100 1,000 Stop

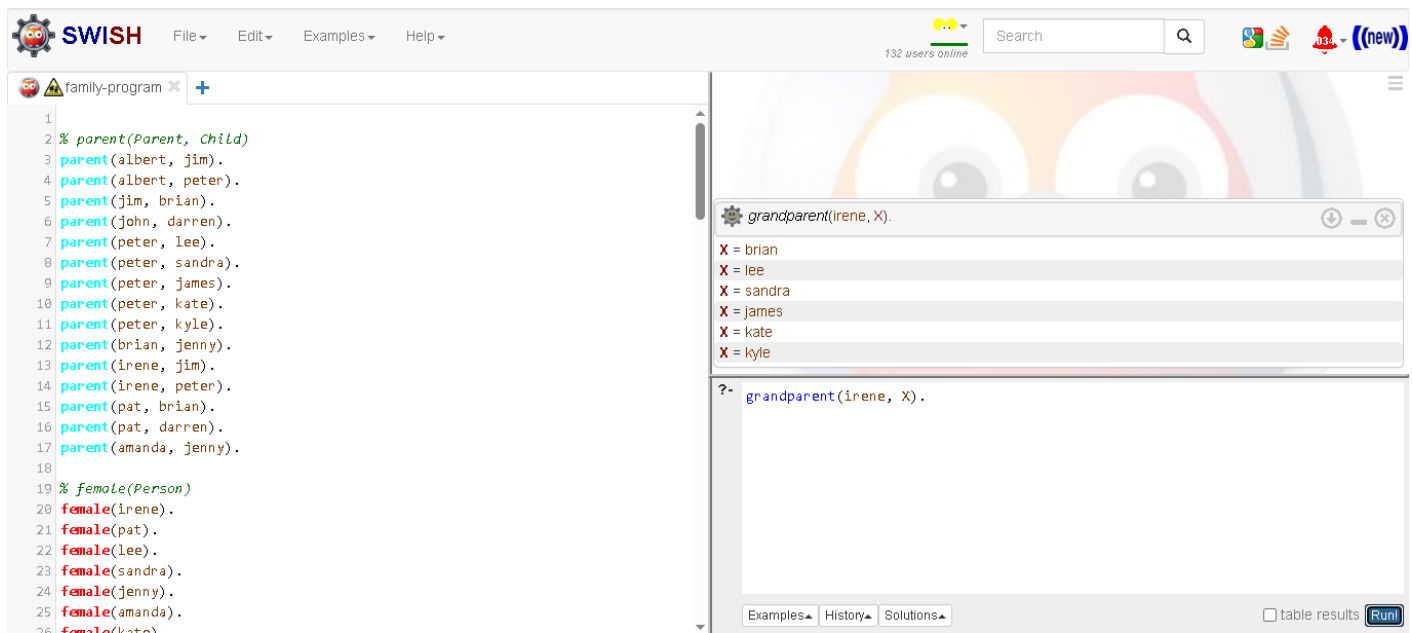
?- grandparent(irene, brian).

Examples History Solutions

table results Run!

5. Find all the grandchildren of Irene

Answer:



The screenshot shows the SWISH Prolog IDE interface. On the left, a code editor displays the same Prolog program as in the previous screenshot. On the right, a query window shows the query 'grandparent(irene, X).' which has returned a list of results: 'X = brian', 'X = lee', 'X = sandra', 'X = james', 'X = kate', and 'X = kyle'. Below the query window, there are buttons for 'Next', '10', '100', '1,000', and 'Stop'. At the bottom right, there are tabs for 'Examples', 'History', and 'Solutions', and a 'Run!' button.

```
1 % parent(Parent, Child)
2 parent(albert, jim).
3 parent(albert, peter).
4 parent(jim, brian).
5 parent(john, darren).
6 parent(peter, lee).
7 parent(peter, sandra).
8 parent(peter, james).
9 parent(peter, kate).
10 parent(peter, kyle).
11 parent(brian, jenny).
12 parent(irene, jim).
13 parent(irene, peter).
14 parent(pat, brian).
15 parent(pat, darren).
16 parent(amanda, jenny).
17
18
19 % female(Person)
20 female(irene).
21 female(pat).
22 female(lee).
23 female(sandra).
24 female(jenny).
25 female(amanda).
26 female(kate).
```

grandparent(irene, X).

X = brian
X = lee
X = sandra
X = james
X = kate
X = kyle

?- grandparent(irene, X).

Examples History Solutions

table results Run!

6. Add rule to define grandparent relation.

Answer:

```
59  
60 % grandparent(Grandparent, Grandchild)  
61 grandparent(Grandparent, Grandchild) :-  
62     parent(Grandparent, Parent),  
63     parent(Parent, Grandchild).  
64
```

7. Add rule to define who is the older.

Answer:

```
64  
65 % older(A, B)  
66 older(A, B) :-  
67     yearOfBirth(A, YearA),  
68     yearOfBirth(B, YearB),  
69     YearA < YearB.  
70
```

8. Using the rules that you are defined in 7, and 6 ,(siblings, older), define new rule: bigBrother. Hint the big brother should be male.

Answer:

```
90 % bigBrother(Brother, Sibling)
91 bigBrother(Brother, Sibling) :-
92     siblings(Brother, Sibling),
93     male(Brother),
94     older(Brother, Sibling).
95
```

Test:

The screenshot shows the SWISH Prolog IDE interface. The left pane displays a Prolog program named 'family-program' with the following rules:

```
78 sibling_list(Child, Siblings) :-
79     findall(Sibling, siblings(Child, Sibling), List),
80     remove_duplicates(List, Siblings).
81
82 % remove_duplicates(List, Result)
83 remove_duplicates([], []).
84 remove_duplicates([X|Rest], Result) :-
85     member(X, Rest), !,
86     remove_duplicates(Rest, Result).
87 remove_duplicates([X|Rest], [X|Result]) :-
88     remove_duplicates(Rest, Result).
89
90 % bigBrother(Brother, Sibling)
91 bigBrother(Brother, Sibling) :-
92     siblings(Brother, Sibling),
93     male(Brother),
94     older(Brother, Sibling).
95
96 % descendant(Person, Descendant)
97 descendant(Person, Descendant) :-
98     parent(Person, Descendant).
99 descendant(Person, Descendant) :-
100     parent(Person, Child),
101     descendant(Child, Descendant).
102
103 % ancestor(Person, Ancestor)
```

The right pane shows a query window with the query `bigBrother(james, kate).` and the result `true`. Below the query window, there is a text area with the query `?- bigBrother(james, kate).` and a 'Run!' button. The bottom of the interface includes tabs for 'Examples', 'History', and 'Solutions', and a checkbox for 'table results'.

The Latest Version Of The Program

```
% parent(Parent, Child)
parent(albert, jim).
parent(albert, peter).
parent(jim, brian).
parent(john, darren).
parent(peter, lee).
parent(peter, sandra).
parent(peter, james).
parent(peter, kate).
parent(peter, kyle).
parent(brian, jenny).
parent(irene, jim).
parent(irene, peter).
parent(pat, brian).
parent(pat, darren).
parent(amanda, jenny).
```

```
% female(Person)
female(irene).
female(pat).
female(lee).
female(sandra).
female(jenny).
female(amanda).
female(kate).
```

```
% male(Person)
male(albert).
male(jim).
male(peter).
male(brian).
male(john).
male(darren).
```

```
male(james).
male(kyle).
```

```
% yearOfBirth(Person, Year)
yearOfBirth(irene, 1923).
yearOfBirth(pat, 1954).
yearOfBirth(lee, 1970).
yearOfBirth(sandra, 1973).
yearOfBirth(jenny, 1996).
yearOfBirth(amanda, 1979).
yearOfBirth(albert, 1926).
yearOfBirth(jim, 1949).
yearOfBirth(peter, 1945).
yearOfBirth(brian, 1974).
yearOfBirth(john, 1955).
yearOfBirth(darren, 1976).
yearOfBirth(james, 1969).
yearOfBirth(kate, 1975).
yearOfBirth(kyle, 1976).
```

```
% -----
% RULES
% -----
```

```
% grandparent(Grandparent, Grandchild)
grandparent(Grandparent, Grandchild) :-
    parent(Grandparent, Parent),
    parent(Parent, Grandchild).
```

```
% older(A, B)
older(A, B) :-
    yearOfBirth(A, YearA),
    yearOfBirth(B, YearB),
    YearA < YearB.
```

```
% siblings(A, B)
siblings(A, B) :-
    parent(X, A),
    parent(X, B),
```

A \== B.

```
% sibling_list(Child, Siblings)
sibling_list(Child, Siblings) :-
    findall(Sibling, siblings(Child, Sibling), List),
    remove_duplicates(List, Siblings).
```

```
% remove_duplicates(List, Result)
remove_duplicates([], []).
remove_duplicates([X|Rest], Result) :-
    member(X, Rest), !,
    remove_duplicates(Rest, Result).
remove_duplicates([X|Rest], [X|Result]) :-
    remove_duplicates(Rest, Result).
```

```
% bigBrother(Brother, Sibling)
bigBrother(Brother, Sibling) :-
    siblings(Brother, Sibling),
    male(Brother),
    older(Brother, Sibling).
```

```
% descendant(Person, Descendant)
descendant(Person, Descendant) :-
    parent(Person, Descendant).
descendant(Person, Descendant) :-
    parent(Person, Child),
    descendant(Child, Descendant).
```

```
% ancestor(Person, Ancestor)
ancestor(Person, Ancestor) :-
    parent(Ancestor, Person).
ancestor(Person, Ancestor) :-
    parent(Parent, Person),
    ancestor(Parent, Ancestor).
```

```
% children(Parent, ChildList)
children(Parent, ChildList) :-
    findall(Child, parent(Parent, Child), ChildList).
```

```
% listCount(List, Count)
```



```
listCount([], 0).
listCount(_|Tail, Count) :-
    listCount(Tail, TailCount),
    Count is TailCount + 1.

% countDescendants(Person, Count)
countDescendants(Person, Count) :-
    findall(Desc, descendant(Person, Desc), List),
    listCount(List, Count).
```