Name: Andrew Hayes
E-mail: a.hayes18@universityofgalway.ie
ID: 21321503

# CT331

2023–11–25

Assignment 3: Declarative Programming with Prolog

## 1 Question 1

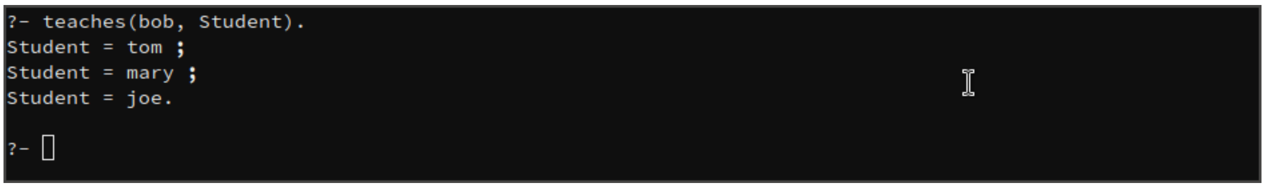
### 1.1 Rule that returns true if a given instructor teaches a given student

```
teaches(Instructor, Student) :- instructs(Instructor, Course), takes(Student, Course).
```

### 1.2 Query that uses the teaches rule to show all students instructed by bob

For this, I wasn't sure if the desired answer was a query that returned a student instructed by bob, followed by a couple semi-colons to get every student instructed by bob, or if the desired answer was a single query that returned a list of students taught by bob, so I did both.

```
?- teaches(bob, Student).
```

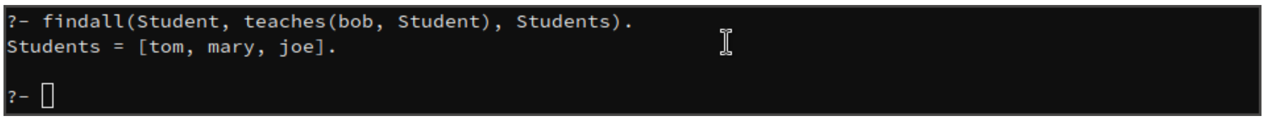


```
?- teaches(bob, Student).
Student = tom ;
Student = mary ;
Student = joe.

?-
```

Figure 1: Using the teaches rule to show all students instructed by bob

Alternatively, this could be done using the findall() predicate:

```
?- findall(Student, teaches(bob, Student), Students).
```

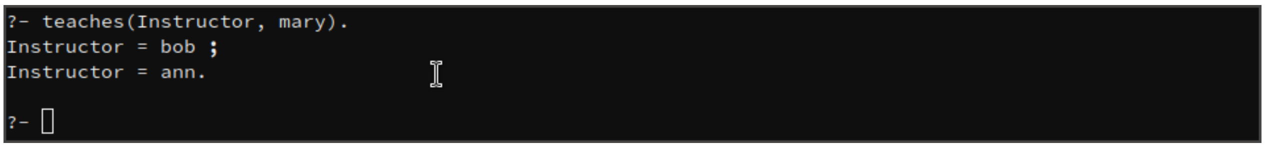


```
?- findall(Student, teaches(bob, Student), Students).
Students = [tom, mary, joe].

?-
```

Figure 2: Using the teaches rule & the findall predicate to show all students instructed by bob

### 1.3 Query that uses the teaches rule to show all instructors that instruct mary

```
?- teaches(Instructor, mary).
```

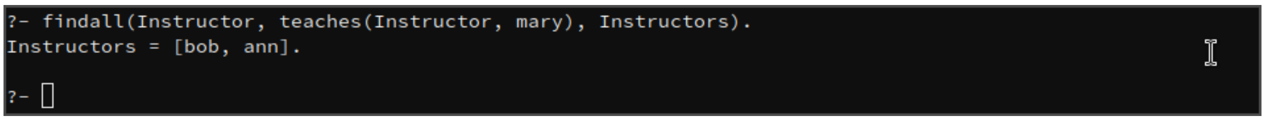


```
?- teaches(Instructor, mary).
Instructor = bob ;
Instructor = ann.

?-
```

Figure 3: Using the teaches rule to show all instructors that instruct mary

Alternatively, this could be done using the findall() predicate:

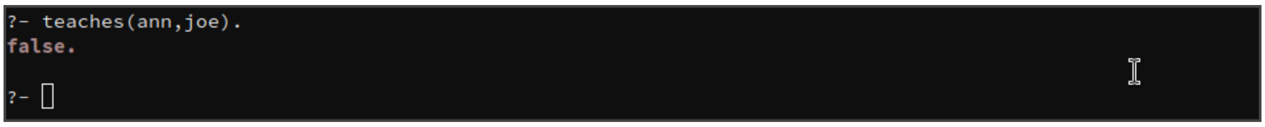
```
?- findall(Instructor, teaches(Instructor, mary), Instructors).
```



```
?- findall(Instructor, teaches(Instructor, mary), Instructors).
Instructors = [bob, ann].

?-
```

Figure 4: Using the teaches() rule & the findall() predicate to show all instructors that instruct mary

### 1.4 Result of query `teaches(ann,joe).`



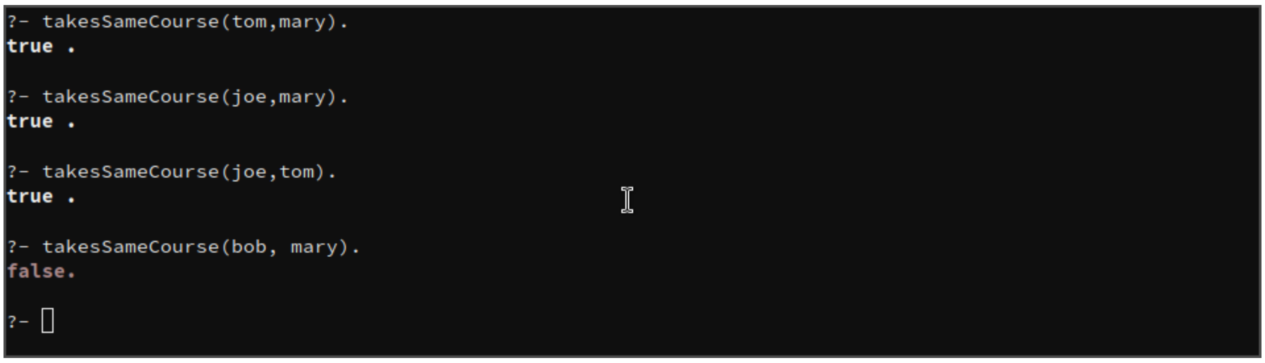
```
?- teaches(ann,joe).
false.

?- 
```

Figure 5: Result of query `teaches(ann,joe).`

The result of the query `teaches(ann,joe).` is `false.` because `ann` only instructs `ct345` and `joe` only takes `ct331`, and therefore `ann` does not teach `joe` because `ann` does not teach a course that `joe` takes.

### 1.5 Rule that returns `true` if two students take the same course

```
takesSameCourse(Student1, Student2) :- takes(Student1, Course), takes(Student2, Course).
```



```
?- takesSameCourse(tom,mary).
true .

?- takesSameCourse(joe,mary).
true .

?- takesSameCourse(joe,tom).
true .

?- takesSameCourse(bob, mary).
false.

?- 
```

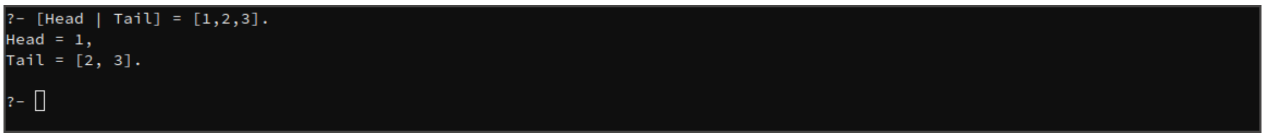
Figure 6: Queries to test `takesSameCourse()`

```
?- takesSameCourse(tom,mary).
?- takesSameCourse(joe,mary).
?- takesSameCourse(joe,tom).
?- takesSameCourse(bob, mary).
```

## 2 Question 2

### 2.1 Query that displays the head & tail of a list

```
?- [Head | Tail] = [1,2,3].
```

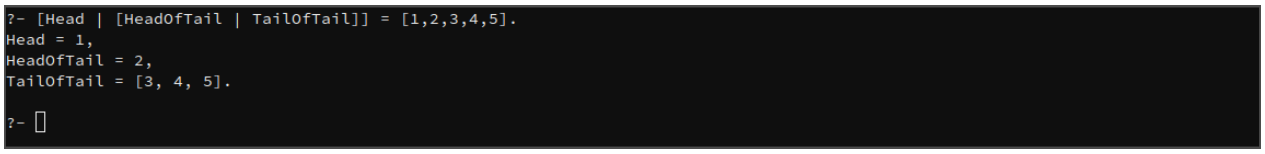


```
?- [Head | Tail] = [1,2,3].
Head = 1,
Tail = [2, 3].

?- 
```

Figure 7: Query to display the head & tail of the list `[1,2,3]`

### 2.2 Display the head of a list, the head of the tail of the list, & the tail of the tail of the list

```
?- [Head | [HeadOfTail | TailOfTail]] = [1,2,3,4,5].
```

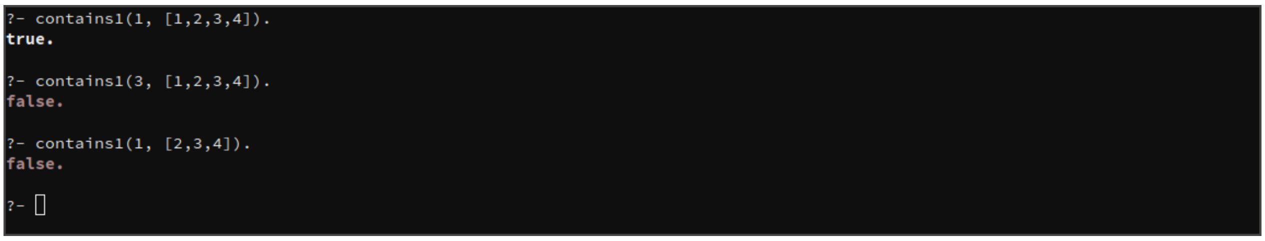


```
?- [Head | [HeadOfTail | TailOfTail]] = [1,2,3,4,5].
Head = 1,
HeadOfTail = 2,
TailOfTail = [3, 4, 5].

?- 
```

Figure 8: Query to display the head of the list, the head of the tail of the list, & the tail of the tail of the list `[1,2,3,4,5]`

## 2.3 Rule that returns `true` if a given element is the first element of a given list

```
contains1(Element, [Element | Tail]).

?- contains1(1, [1,2,3,4]).
?- contains1(3, [1,2,3,4]).
?- contains1(1, [2,3,4]).
```

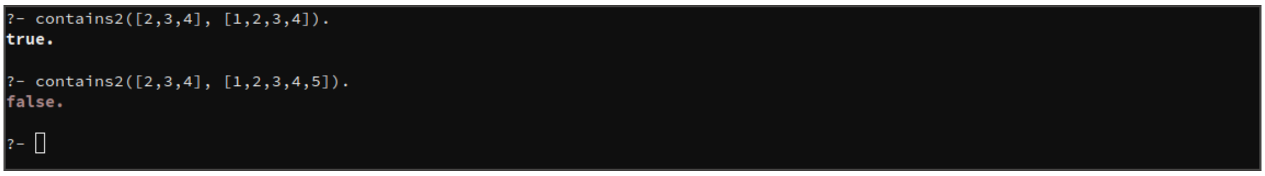


```
?- contains1(1, [1,2,3,4]).
true.

?- contains1(3, [1,2,3,4]).
false.

?- contains1(1, [2,3,4]).
false.

?- 
```

Figure 9: `contains1()` testing

## 2.4 Rule that returns `true` if a given list is the same as the tail of another given list

```
contains2(Sublist, [Head | Sublist]).

?- contains2([2,3,4], [1,2,3,4]).
?- contains2([2,3,4], [1,2,3,4,5]).
```

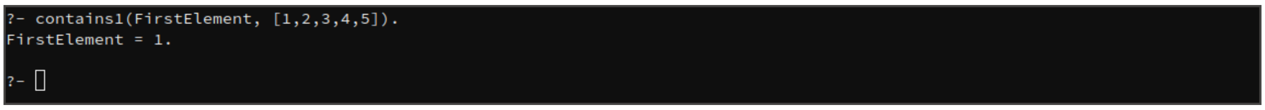


```
?- contains2([2,3,4], [1,2,3,4]).
true.

?- contains2([2,3,4], [1,2,3,4,5]).
false.

?- 
```

Figure 10: `contains2()` testing

## 2.5 Query to display the first element of a given list using `contains1()`

```
?- contains1(FirstElement, [1,2,3,4,5]).
```

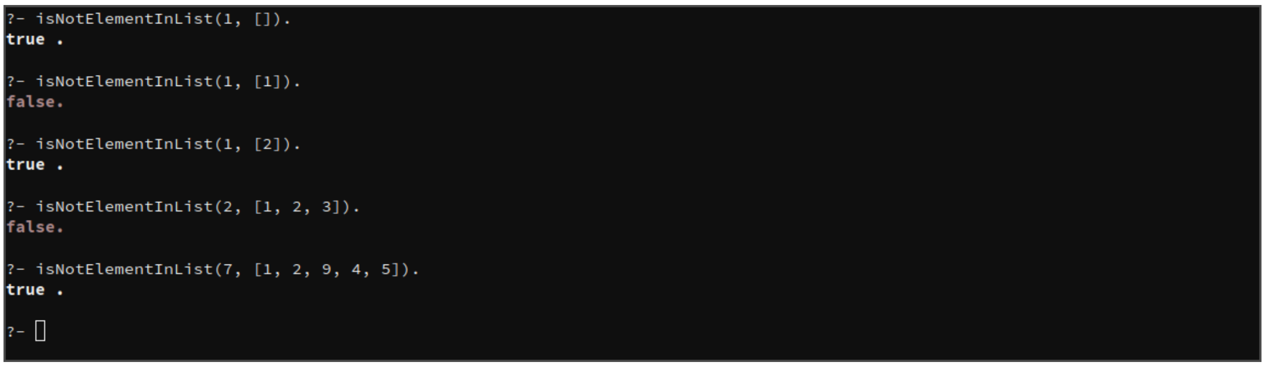


```
?- contains1(FirstElement, [1,2,3,4,5]).
FirstElement = 1.

?- 
```

Figure 11: Query to display the first element of a given list using `contains1()`

## 3 Determine if a given element is not in a given list

```
% base case: any element is not in an empty list
isNotElementInList(_, []).

% return true if Element is not the Head of the list and it's not found recursively searching the rest of
↪ the list
isNotElementInList(Element, [Head | Tail]) :- Element \= Head, isNotElementInList(Element, Tail).

% testing
isNotElementInList(1, []).
isNotElementInList(1, [1]).
isNotElementInList(1, [2]).
isNotElementInList(2, [1, 2, 3]).
isNotElementInList(7, [1, 2, 9, 4, 5]).
```

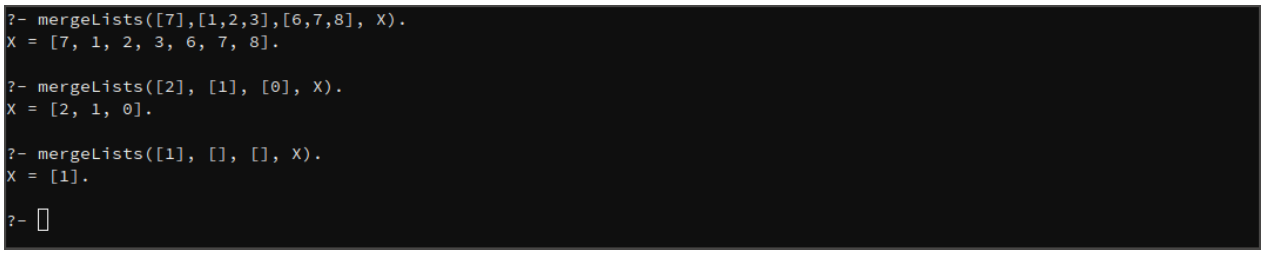


```
?- isNotElementInList(1, []).
true .

?- isNotElementInList(1, [1]).
false.

?- isNotElementInList(1, [2]).
true .

?- isNotElementInList(2, [1, 2, 3]).
false.

?- isNotElementInList(7, [1, 2, 9, 4, 5]).
true .

?-
```

Figure 12: Testing `isNotElementInList()`

## 4 Facts & rules to merge three lists

```
% predicate to merge two lists
% base case: if the first list is empty, just return the second
mergeTwoLists([], List, List).

% recursive predicate to merge two lists
% split the first list into head and tail, and recurse with its tail and the second list until the first
↪  list is empty (base case)
% then merge the original head of the first list with the resulting tail
mergeTwoLists([Head | Tail], List2, [Head | ResultTail]) :- mergeTwoLists(Tail, List2, ResultTail).

% predicate to merge 3 lists
% base case: merging an empty list and two others is the same as merging two lists
mergeLists([], List2, List3, Merged) :- mergeTwoLists(List2, List3, Merged).

% split the first list into head and tail, and recurse with its tail and the other two lists until the
↪  first list is empty (base case)
mergeLists([Head1 | Tail1], List2, List3, [Head1 | MergedTail]) :- mergeLists(Tail1, List2, List3,
↪  MergedTail).

?- mergeLists([7],[1,2,3],[6,7,8], X).
?- mergeLists([2], [1], [0], X).
?- mergeLists([1], [], [], X).
```



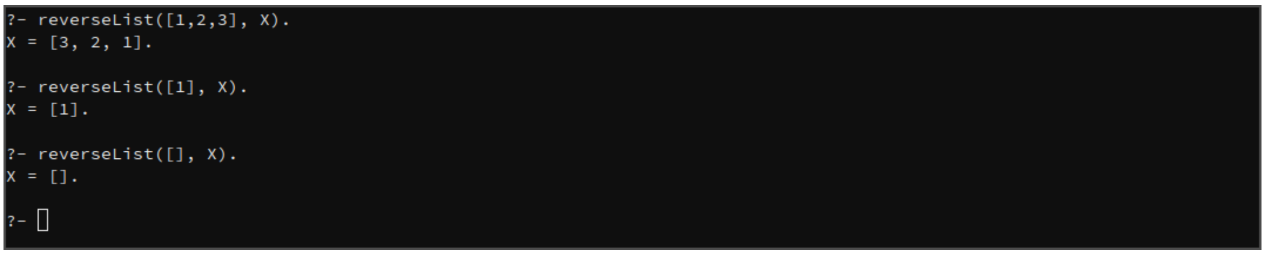
```
?- mergeLists([7],[1,2,3],[6,7,8], X).
X = [7, 1, 2, 3, 6, 7, 8].

?- mergeLists([2], [1], [0], X).
X = [2, 1, 0].

?- mergeLists([1], [], [], X).
X = [1].

?-
```

Figure 13: Testing `mergeLists()`

## 5 Facts & rules to reverse a given list

```
% call the helper predicate with the list to be reversed and an empty Accumulator to build up
reverseList(List, Reversed) :- reverseListHelper(List, [], Reversed).

```

```prolog
% base case fact: when the list to reverse is empty, the accumulator is the reversed list
reverseListHelper([], Accumulator, Accumulator).

% recurse with the tail after prepending the head to the accumulator
reverseListHelper([Head | Tail], Accumulator, Reversed) :- reverseListHelper(Tail, [Head | Accumulator],
↪  Reversed).

?- reverseList([1,2,3], X).
?- reverseList([1], X).
?- reverseList([], X).
```

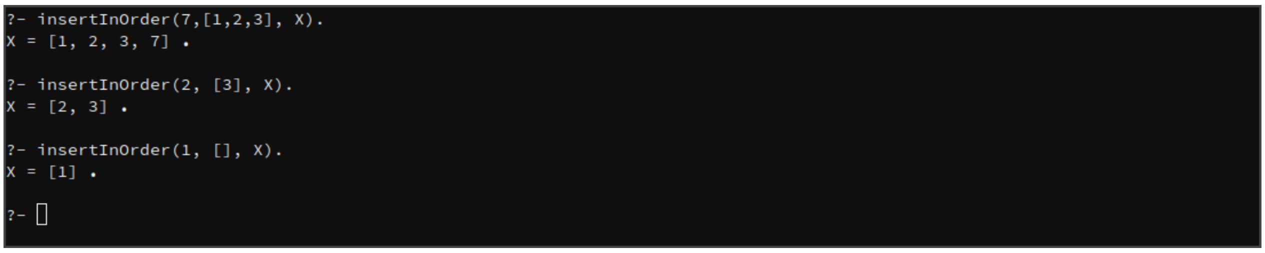


```
?- reverseList([1,2,3], X).
X = [3, 2, 1].

?- reverseList([1], X).
X = [1].

?- reverseList([], X).
X = [].

?- 
```

Figure 14: Testing `reverseList()`

## 6 Facts & rules to insert an element into its correct position in a given list

```prolog
% base fact: if the list is empty, the list to be returned is just the element
insertInOrder(Element, [], [Element]).

% if the element to be inserted is <= the head of the list, insert it at the head of the list
insertInOrder(Element, [Head | Tail], [Element, Head | Tail]) :- Element =< Head.

% if the element to be inserted is greater than the head of the list, recurse with the tail of the list
↪  until
insertInOrder(Element, [Head | Tail], [Head | NewTail]) :- Element > Head, insertInOrder(Element, Tail,
↪  NewTail).
```



```
?- insertInOrder(7,[1,2,3], X).
X = [1, 2, 3, 7] .

?- insertInOrder(2, [3], X).
X = [2, 3] .

?- insertInOrder(1, [], X).
X = [1] .

?- 
```

Figure 15: Testing `insertInOrder()`