QUIZ 14 (TAKE-HOME)

PROLOG PROGRAM

Submitted By-

Nahida Sultana Chowdhury

Student ID: 2000189256

Date: 11/29/2017

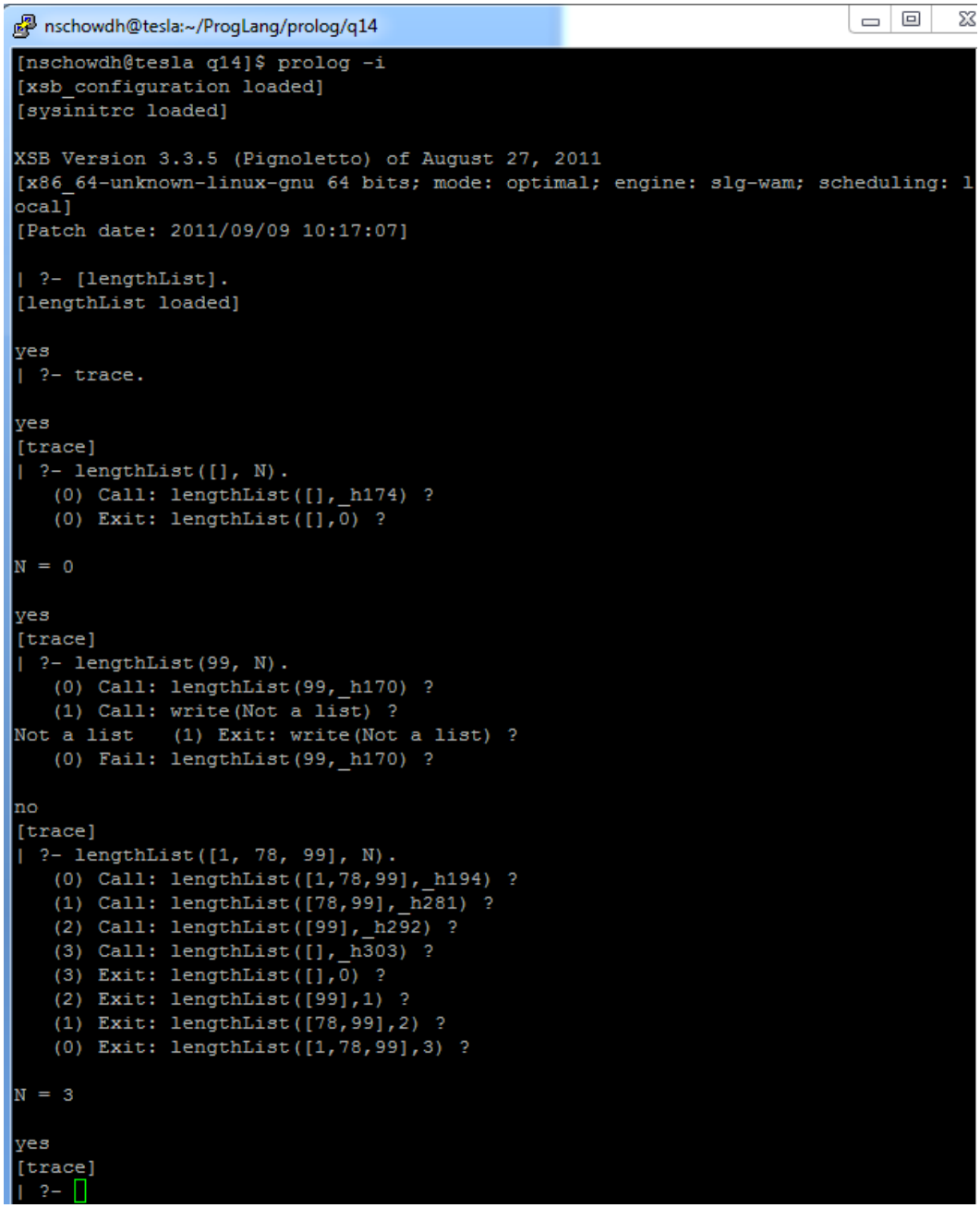
Problem 1: Compute the length of a given list. Program should handle lists of various sizes.

- **Code:**

```
lengthList(List,_) :- not(is_list(List)), write('Not a list'), !, fail.
lengthList([]    , 0 ).
lengthList([_|Tail] , Len ) :- lengthList(Tail, Counter) , Len is Counter + 1.
```

- **Screenshot (including program trace information):**

In the output screenshot three cases has been displayed:
1. When the list contains zero elements then it will return 0
2. When the input is an atom then it will display a message contains "Not a list", to check imposed *is_list* method.
3. When a list contains one or more items inside a list then it will return the total number of elements.



```
nschowdh@tesla:~/ProgLang/prolog/q14
[nschowdh@tesla q14]$ prolog -i
[xsb_configuration loaded]
[sysinitrc loaded]

XSB Version 3.3.5 (Pignoletto) of August 27, 2011
[x86_64-unknown-linux-gnu 64 bits; mode: optimal; engine: slg-wam; scheduling: l
ocal]
[Patch date: 2011/09/09 10:17:07]

| ?- [lengthList].
[lengthList loaded]

yes
| ?- trace.

yes
[trace]
| ?- lengthList([], N).
   (0) Call: lengthList([],_h174) ?
   (0) Exit: lengthList([],0) ?

N = 0

yes
[trace]
| ?- lengthList(99, N).
   (0) Call: lengthList(99,_h170) ?
   (1) Call: write(Not a list) ?
Not a list   (1) Exit: write(Not a list) ?
   (0) Fail: lengthList(99,_h170) ?

no
[trace]
| ?- lengthList([1, 78, 99], N).
   (0) Call: lengthList([1,78,99],_h194) ?
   (1) Call: lengthList([78,99],_h281) ?
   (2) Call: lengthList([99],_h292) ?
   (3) Call: lengthList([],_h303) ?
   (3) Exit: lengthList([],0) ?
   (2) Exit: lengthList([99],1) ?
   (1) Exit: lengthList([78,99],2) ?
   (0) Exit: lengthList([1,78,99],3) ?

N = 3

yes
[trace]
| ?-
```

Problem 2: Compute the second to last element of a given list. Your program should handle lists of various sizes.

- **Code:**

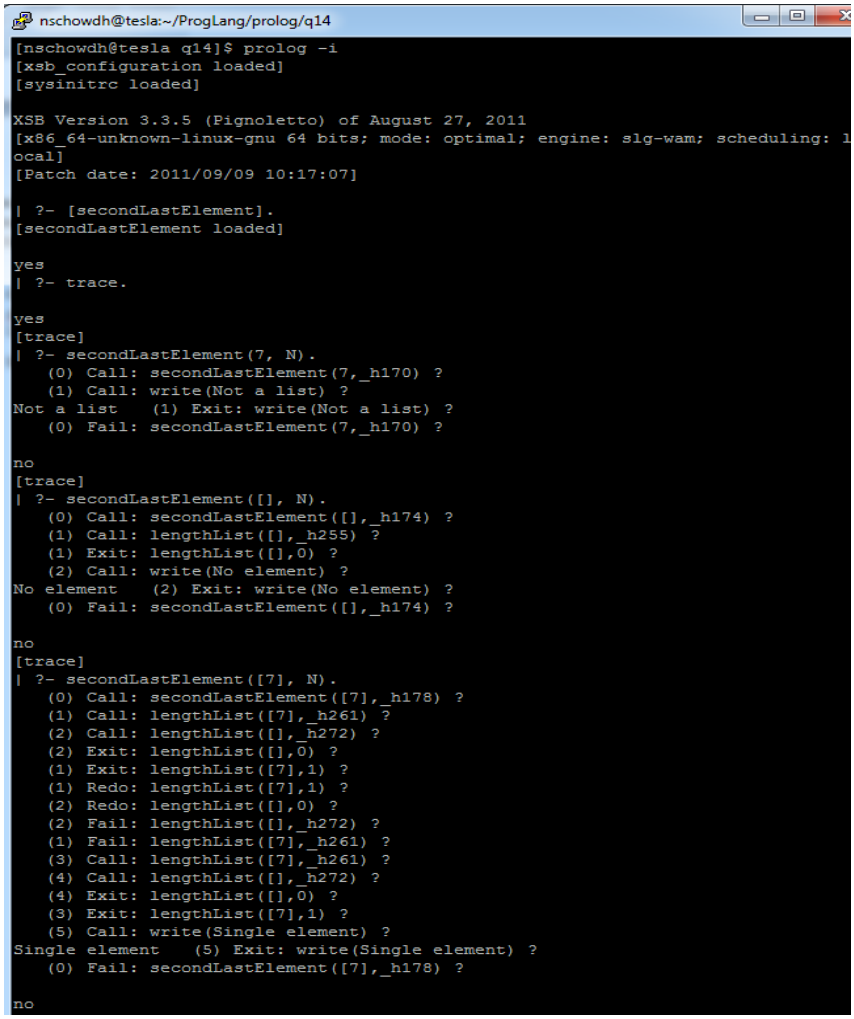
```
lengthList([]    , 0 ).
lengthList([_|Tail] , Len) :- lengthList(Tail, Counter) , Len is Counter + 1.

secondLastElement(List,_) :- not(is_list(List)), write('Not a list'), !, fail.
secondLastElement(List,_) :- lengthList(List, Len), Len == 0, write('No element'), !, fail.
secondLastElement(List,_) :- lengthList(List, Len), Len == 1, write('Single element'), !, fail.
secondLastElement([Head|Tail], X) :- lengthList([Head|Tail], Len), Len == 2, !, X is Head.
secondLastElement([_|Tail], X) :- secondLastElement(Tail, X).
```

- **Screenshot (including program trace information):**

In the output screenshot five cases has been displayed:

1. When the input is an atom then it will display a message contains "Not a list", to check imposed *is_list* method.
2. When the list contains zero elements then it will return "'No element".
3. When the list contains exactly one element then will return a message contains "Single element".
4. When the list contains exactly two elements then will return the head of the list.
5. When the list contains more than two items then will call *secondLastElement* recursively and when it will reach up to length of list 2 then will return the head element of that remain list.



```
nschowdh@tesla:~/ProgLang/prolog/q14
[nschowdh@tesla q14]$ prolog -i
[xsb_configuration loaded]
[sysinitrc loaded]

XSB Version 3.3.5 (Pignoletto) of August 27, 2011
[x86_64-unknown-linux-gnu 64 bits; mode: optimal; engine: slg-wam; scheduling: l
ocal]
[Patch date: 2011/09/09 10:17:07]

| ?- [secondLastElement].
[secondLastElement loaded]

yes
| ?- trace.

yes
[trace]
| ?- secondLastElement(7, N).
   (0) Call: secondLastElement(7,_h170) ?
   (1) Call: write(Not a list) ?
Not a list   (1) Exit: write(Not a list) ?
   (0) Fail: secondLastElement(7,_h170) ?

no
[trace]
| ?- secondLastElement([], N).
   (0) Call: secondLastElement([],_h174) ?
   (1) Call: lengthList([],_h255) ?
   (1) Exit: lengthList([],0) ?
   (2) Call: write(No element) ?
No element   (2) Exit: write(No element) ?
   (0) Fail: secondLastElement([],_h174) ?

no
[trace]
| ?- secondLastElement([7], N).
   (0) Call: secondLastElement([7],_h178) ?
   (1) Call: lengthList([7],_h261) ?
   (2) Call: lengthList([],_h272) ?
   (2) Exit: lengthList([],0) ?
   (1) Exit: lengthList([7],1) ?
   (1) Redo: lengthList([7],1) ?
   (2) Redo: lengthList([],0) ?
   (2) Fail: lengthList([],_h272) ?
   (1) Fail: lengthList([7],_h261) ?
   (3) Call: lengthList([7],_h261) ?
   (4) Call: lengthList([],_h272) ?
   (4) Exit: lengthList([],0) ?
   (3) Exit: lengthList([7],1) ?
   (5) Call: write(Single element) ?
Single element   (5) Exit: write(Single element) ?
   (0) Fail: secondLastElement([7],_h178) ?

no
```

nschowdh@tesla:~/ProgLang/prolog/q14

```
[trace]
| ?- secondLastElement([7, 9], N).
   (0) Call: secondLastElement([7,9],_h186) ?
   (1) Call: lengthList([7,9],_h271) ?
   (2) Call: lengthList([9],_h282) ?
   (3) Call: lengthList([],_h293) ?
   (3) Exit: lengthList([],0) ?
   (2) Exit: lengthList([9],1) ?
   (1) Exit: lengthList([7,9],2) ?
   (1) Redo: lengthList([7,9],2) ?
   (2) Redo: lengthList([9],1) ?
   (3) Redo: lengthList([],0) ?
   (3) Fail: lengthList([],_h293) ?
   (2) Fail: lengthList([9],_h282) ?
   (1) Fail: lengthList([7,9],_h271) ?
   (4) Call: lengthList([7,9],_h271) ?
   (5) Call: lengthList([9],_h282) ?
   (6) Call: lengthList([],_h293) ?
   (6) Exit: lengthList([],0) ?
   (5) Exit: lengthList([9],1) ?
   (4) Exit: lengthList([7,9],2) ?
   (4) Redo: lengthList([7,9],2) ?
   (5) Redo: lengthList([9],1) ?
   (6) Redo: lengthList([],0) ?
   (6) Fail: lengthList([],_h293) ?
   (5) Fail: lengthList([9],_h282) ?
   (4) Fail: lengthList([7,9],_h271) ?
   (7) Call: lengthList_#301(7,[9],_h272) ?
   (8) Call: lengthList([9],_h283) ?
   (9) Call: lengthList([],_h294) ?
   (9) Exit: lengthList([],0) ?
   (8) Exit: lengthList([9],1) ?
   (7) Exit: lengthList_#301(7,[9],2) ?
   (0) Exit: secondLastElement([7,9],7) ?

N = 7

yes
```

```
| ?-  secondLastElement([7, 9, 11], N).
   (0) Call: secondLastElement([7,9,11],_h194) ?
   (1) Call: lengthList([7,9,11],_h283) ?
   (2) Call: lengthList([9,11],_h294) ?
   (3) Call: lengthList([11],_h305) ?
   (4) Call: lengthList([],_h316) ?
   (4) Exit: lengthList([],0) ?
   (3) Exit: lengthList([11],1) ?
   (2) Exit: lengthList([9,11],2) ?
   (1) Exit: lengthList([7,9,11],3) ?
   (1) Redo: lengthList([7,9,11],3) ?
   (2) Redo: lengthList([9,11],2) ?
   (3) Redo: lengthList([11],1) ?
   (4) Redo: lengthList([],0) ?
   (4) Fail: lengthList([],_h316) ?
   (3) Fail: lengthList([11],_h305) ?
   (2) Fail: lengthList([9,11],_h294) ?
   (1) Fail: lengthList([7,9,11],_h283) ?
   (5) Call: lengthList([7,9,11],_h283) ?
   (6) Call: lengthList([9,11],_h294) ?
   (7) Call: lengthList([11],_h305) ?
   (8) Call: lengthList([],_h316) ?
   (8) Exit: lengthList([],0) ?
   (7) Exit: lengthList([11],1) ?
   (6) Exit: lengthList([9,11],2) ?
   (5) Exit: lengthList([7,9,11],3) ?
   (5) Redo: lengthList([7,9,11],3) ?
   (6) Redo: lengthList([9,11],2) ?
   (7) Redo: lengthList([11],1) ?
   (8) Redo: lengthList([],0) ?
   (8) Fail: lengthList([],_h316) ?
   (7) Fail: lengthList([11],_h305) ?
   (6) Fail: lengthList([9,11],_h294) ?
   (5) Fail: lengthList([7,9,11],_h283) ?
   (9) Call: lengthList_#301(7,[9,11],_h284) ?
   (10) Call: lengthList([9,11],_h295) ?
   (11) Call: lengthList([11],_h306) ?
   (12) Call: lengthList([],_h317) ?
   (12) Exit: lengthList([],0) ?
   (11) Exit: lengthList([11],1) ?
   (10) Exit: lengthList([9,11],2) ?
   (9) Exit: lengthList_#301(7,[9,11],3) ?
   (9) Redo: lengthList_#301(7,[9,11],3) ?
   (10) Redo: lengthList([9,11],2) ?
   (11) Redo: lengthList([11],1) ?
   (12) Redo: lengthList([],0) ?
   (12) Fail: lengthList([],_h317) ?
   (11) Fail: lengthList([11],_h306) ?
   (10) Fail: lengthList([9,11],_h295) ?
   (9) Fail: lengthList_#301(7,[9,11],_h284) ?
   (13) Call: secondLastElement([9,11],_h194) ?
```

```
   (14) Call: lengthList([9,11],_h294) ?
   (15) Call: lengthList([11],_h305) ?
   (16) Call: lengthList([],_h316) ?
   (16) Exit: lengthList([],0) ?
   (15) Exit: lengthList([11],1) ?
   (14) Exit: lengthList([9,11],2) ?
   (14) Redo: lengthList([9,11],2) ?
   (15) Redo: lengthList([11],1) ?
   (16) Redo: lengthList([],0) ?
   (16) Fail: lengthList([],_h316) ?
   (15) Fail: lengthList([11],_h305) ?
   (14) Fail: lengthList([9,11],_h294) ?
   (17) Call: lengthList([9,11],_h294) ?
   (18) Call: lengthList([11],_h305) ?
   (19) Call: lengthList([],_h316) ?
   (19) Exit: lengthList([],0) ?
   (18) Exit: lengthList([11],1) ?
   (17) Exit: lengthList([9,11],2) ?
   (17) Redo: lengthList([9,11],2) ?
   (18) Redo: lengthList([11],1) ?
   (19) Redo: lengthList([],0) ?
   (19) Fail: lengthList([],_h316) ?
   (18) Fail: lengthList([11],_h305) ?
   (17) Fail: lengthList([9,11],_h294) ?
   (20) Call: lengthList_#301(9,[11],_h295) ?
   (21) Call: lengthList([11],_h306) ?
   (22) Call: lengthList([],_h317) ?
   (22) Exit: lengthList([],0) ?
   (21) Exit: lengthList([11],1) ?
   (20) Exit: lengthList_#301(9,[11],2) ?
   (13) Exit: secondLastElement([9,11],9) ?
   (0) Exit: secondLastElement([7,9,11],9) ?
```

N = 9

```
yes
[trace]
| ?-
```

Problem 3: Take two lists, L1 and L2, of same lengths and will return a merged list, L3, such that L3 is obtained by carrying out the perfect shuffle operation on L1 and L2. Your program should handle lists of various sizes. (Note: A perfect shuffle on two lists, [1, 2] and [3, 4], will result in [1, 3, 2, 4].)

- **Code:**

```
lengthList(List,_) :- not(is_list(List)), write('Not a list'), !, fail.
lengthList([]    , 0 ).
lengthList([_|Tail] , Len ) :- lengthList(Tail, Counter) , Len is Counter + 1.

shuffle(List1, List2, _) :- lengthList(List1, Len1), lengthList(List2, Len2), not(Len1 = Len2),
write('lists are not equal'), !, fail.
shuffle([], [], []).
shuffle([Head1|Tail1], [Head2|Tail2], [Head1, Head2|Tail3]) :- shuffle(Tail, Tail2, Tail3).
```

- **Screenshot (including program trace information):**

In the output screenshot three cases has been displayed:
1. When the List1 contains 2 elements and List2 contains 3 elements then it will return terminate with a message that “lists are not equal”
2. When the inputs are atom then it will display a message contains “Not a list”, to check imposed *is_list* method.
3. When both List1 and List2 contain equal number of elements then the shuffle take place which conduct the *shuffle* operation between List1 and List2 and return the resulted list.

```
nschowdh@tesla:~/ProgLang/prolog/q14
[nschowdh@tesla q14]$ prolog -i
[xsb_configuration loaded]
[sysinitrc loaded]

XSB Version 3.3.5 (Pignoletto) of August 27, 2011
[x86_64-unknown-linux-gnu 64 bits; mode: optimal; engine: slg-wam; sche
duling: local]
[Patch date: 2011/09/09 10:17:07]

| ?- [shuffle]
.
[Compiling ./shuffle]
[shuffle compiled, cpu time used: 0.0120 seconds]
[shuffle loaded]

yes
| ?- trace.

yes
[trace]
| ?- shuffle([1,3], [2, 4, 6], N).
   (0) Call: shuffle([1,3],[2,4,6],_h218) ?
   (1) Call: lengthList([1,3],_h316) ?
   (2) Call: lengthList([3],_h327) ?
   (3) Call: lengthList([],_h338) ?
   (3) Exit: lengthList([],0) ?
   (2) Exit: lengthList([3],1) ?
   (1) Exit: lengthList([1,3],2) ?
   (4) Call: lengthList([2,4,6],_h370) ?
   (5) Call: lengthList([4,6],_h381) ?
   (6) Call: lengthList([6],_h392) ?
   (7) Call: lengthList([],_h403) ?
   (7) Exit: lengthList([],0) ?
   (6) Exit: lengthList([6],1) ?
   (5) Exit: lengthList([4,6],2) ?
   (4) Exit: lengthList([2,4,6],3) ?
   (8) Call: write(lists are not equal) ?
lists are not equal   (8) Exit: write(lists are not equal) ?
   (0) Fail: shuffle([1,3],[2,4,6],_h218) ?

no
[trace]
| ?-

shuffle(4, 5, N).
   (0) Call: shuffle(4,5,_h178) ?
   (1) Call: lengthList(4,_h266) ?
   (2) Call: write(Not a list) ?
Not a list   (2) Exit: write(Not a list) ?
   (1) Fail: lengthList(4,_h266) ?
   (0) Fail: shuffle(4,5,_h178) ?

no
[trace]
| ?-
```

```
| ?- [shuffle].
[Compiling ./shuffle]
[shuffle compiled, cpu time used: 0.0020 seconds]
[shuffle loaded]
yes
| ?- trace.
yes
[trace]
| ?- shuffle([1, 3, 5], [2, 4, 6], N).
   (0) Call: shuffle([1,3,5],[2,4,6],_h226) ?
   (1) Call: lengthList([1,3,5],_h326) ?
   (2) Call: lengthList([3,5],_h337) ?
   (3) Call: lengthList([5],_h348) ?
   (4) Call: lengthList([],_h359) ?
   (4) Exit: lengthList([],0) ?
   (3) Exit: lengthList([5],1) ?
   (2) Exit: lengthList([3,5],2) ?
   (1) Exit: lengthList([1,3,5],3) ?
   (5) Call: lengthList([2,4,6],_h398) ?
   (6) Call: lengthList([4,6],_h409) ?
   (7) Call: lengthList([6],_h420) ?
   (8) Call: lengthList([],_h431) ?
   (8) Exit: lengthList([],0) ?
   (7) Exit: lengthList([6],1) ?
   (6) Exit: lengthList([4,6],2) ?
   (5) Exit: lengthList([2,4,6],3) ?
   (5) Redo: lengthList([2,4,6],3) ?
   (6) Redo: lengthList([4,6],2) ?
   (7) Redo: lengthList([6],1) ?
   (8) Redo: lengthList([],0) ?
   (8) Fail: lengthList([],_h431) ?
   (7) Fail: lengthList([6],_h420) ?
   (6) Fail: lengthList([4,6],_h409) ?
   (5) Fail: lengthList([2,4,6],_h398) ?
   (1) Redo: lengthList([1,3,5],3) ?
   (2) Redo: lengthList([3,5],2) ?
   (3) Redo: lengthList([5],1) ?
   (4) Redo: lengthList([],0) ?
   (4) Fail: lengthList([],_h359) ?
   (3) Fail: lengthList([5],_h348) ?
   (2) Fail: lengthList([3,5],_h337) ?
   (1) Fail: lengthList([1,3,5],_h326) ?
   (9) Call: shuffle([3,5],[4,6],_h327) ?
   (10) Call: lengthList([3,5],_h342) ?
   (11) Call: lengthList([5],_h353) ?
   (12) Call: lengthList([],_h364) ?
   (12) Exit: lengthList([],0) ?
   (11) Exit: lengthList([5],1) ?
   (10) Exit: lengthList([3,5],2) ?
   (13) Call: lengthList([4,6],_h396) ?
   (14) Call: lengthList([6],_h407) ?
```

```
   (15) Call: lengthList([],_h418) ?
   (15) Exit: lengthList([],0) ?
   (14) Exit: lengthList([6],1) ?
   (13) Exit: lengthList([4,6],2) ?
   (13) Redo: lengthList([4,6],2) ?
   (14) Redo: lengthList([6],1) ?
   (15) Redo: lengthList([],0) ?
   (15) Fail: lengthList([],_h418) ?
   (14) Fail: lengthList([6],_h407) ?
   (13) Fail: lengthList([4,6],_h396) ?
   (10) Redo: lengthList([3,5],2) ?
   (11) Redo: lengthList([5],1) ?
   (12) Redo: lengthList([],0) ?
   (12) Fail: lengthList([],_h364) ?
   (11) Fail: lengthList([5],_h353) ?
   (10) Fail: lengthList([3,5],_h342) ?
   (16) Call: shuffle([5],[6],_h343) ?
   (17) Call: lengthList([5],_h358) ?
   (18) Call: lengthList([],_h369) ?
   (18) Exit: lengthList([],0) ?
   (17) Exit: lengthList([5],1) ?
   (19) Call: lengthList([6],_h394) ?
   (20) Call: lengthList([],_h405) ?
   (20) Exit: lengthList([],0) ?
   (19) Exit: lengthList([6],1) ?
   (19) Redo: lengthList([6],1) ?
   (20) Redo: lengthList([],0) ?
   (20) Fail: lengthList([],_h405) ?
   (19) Fail: lengthList([6],_h394) ?
   (17) Redo: lengthList([5],1) ?
   (18) Redo: lengthList([],0) ?
   (18) Fail: lengthList([],_h369) ?
   (17) Fail: lengthList([5],_h358) ?
   (21) Call: shuffle([],[],_h359) ?
   (22) Call: lengthList([],_h374) ?
   (22) Exit: lengthList([],0) ?
   (23) Call: lengthList([],_h392) ?
   (23) Exit: lengthList([],0) ?
   (23) Redo: lengthList([],0) ?
   (23) Fail: lengthList([],_h392) ?
   (22) Redo: lengthList([],0) ?
   (22) Fail: lengthList([],_h374) ?
   (21) Exit: shuffle([],[],[]) ?
   (16) Exit: shuffle([5],[6],[5,6]) ?
   (9) Exit: shuffle([3,5],[4,6],[3,4,5,6]) ?
   (0) Exit: shuffle([1,3,5],[2,4,6],[1,2,3,4,5,6]) ?
```

N = [1,2,3,4,5,6]

yes
[trace]

Problem 4: Compare and Analyze the CLisp and Prolog version

- CLsip and Prolog both are declarative programming language.None of them have variable side effects.
- CLisp follows functional programming model whereas Prolog follow Logic programming model.
- CLisp is parenthesis based where parenthesis must be provided for each command. Prolog is not parenthesis based and to indicate the end of a statement its require to use period.
- In CLisp its use CAR and CDR that returns head and tail of the list accordingly where as Prolog use vertical bar to split a list into head and tail.
- In terms of execution steps Prolog takes more steps than CLisp as they follow backtracking mechanism which is visible to the trace information.
- To perform the execution time for both applied *time* method for CLisp and Prolog. For CLisp the *time* methods return the execution time properly. However for prolog every time it returns 0 so can't rely on the execution time to make a comparison. Again the total number of trace point if we focus for both cases then it's quite visible that Prolog should take more execution time than CLisp.

Table I: Execution time (CLisp vs Prolog)

Program name	**CLISP**	**Prolog**
Length of List	**> (time (lengthOfList '(1 2 3 4 5 6)))** Real time: 2.3E-5 sec. Run time: 2.2E-5 sec. Space: 0 Bytes 6	**?- time(lengthList([1,2,3,4,5,6], N)).** % 0 CPU in 0 seconds (Inf% CPU) N = 6 yes
secondLast Element	**> (time (secondLastElement '(1 2 3 4 5 6)))** Real time: 1.13E-4 sec. Run time: 1.08E-4 sec. Space: 0 Bytes 5	**?-time(secondLastElement([1, 2,3,4,5,6], N)).** % 0 CPU in 0 seconds (Inf% CPU) N = 5 yes
Shuffle	**> (time (shuffle '(1 3 5) '(2 4 6)))** Real time: 4.9E-5 sec. Run time: 4.4E-5 sec. Space: 96 Bytes (1 2 3 4 5 6)	**?- time (shuffle ([1, 3, 5], [2, 4, 6], N)).** (0) Call: time(shuffle([1,3,5],[2,4,6],_h234)) ? % 0 CPU in 0 seconds (Inf% CPU) (0) Exit: time(shuffle([1,3,5],[2,4,6],[1,2,3,4,5,6])) ? N = [1,2,3,4,5,6] yes [trace]