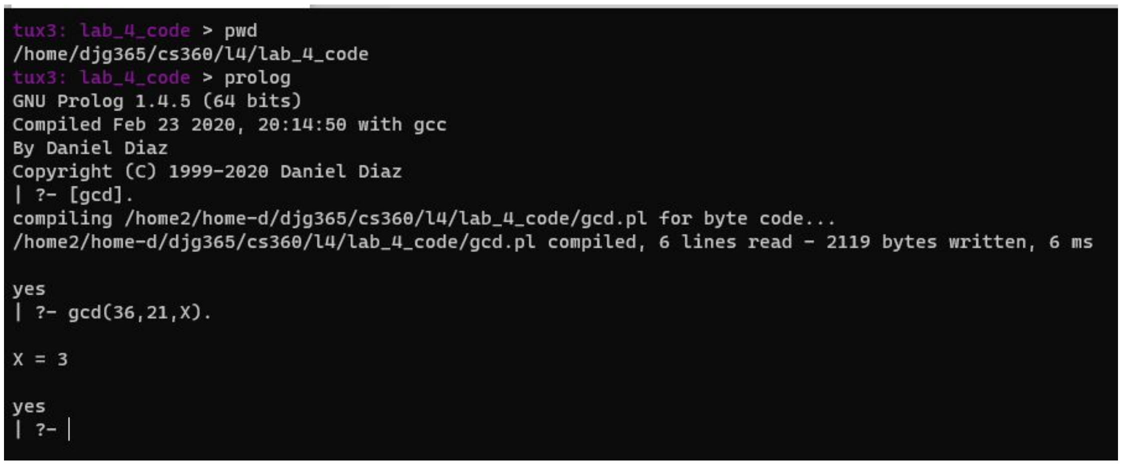
Dennis George, Stephen Hansen, Shivanshi Nagar
Group 4
CS360 Lab 4 Writeup
August 16, 2020

1. (i) In Prolog, load the file gcd.pl. It contains a prolog predicate for finding the greatest common divisor of two numbers. Find the greatest common divisor of 36 and 21.



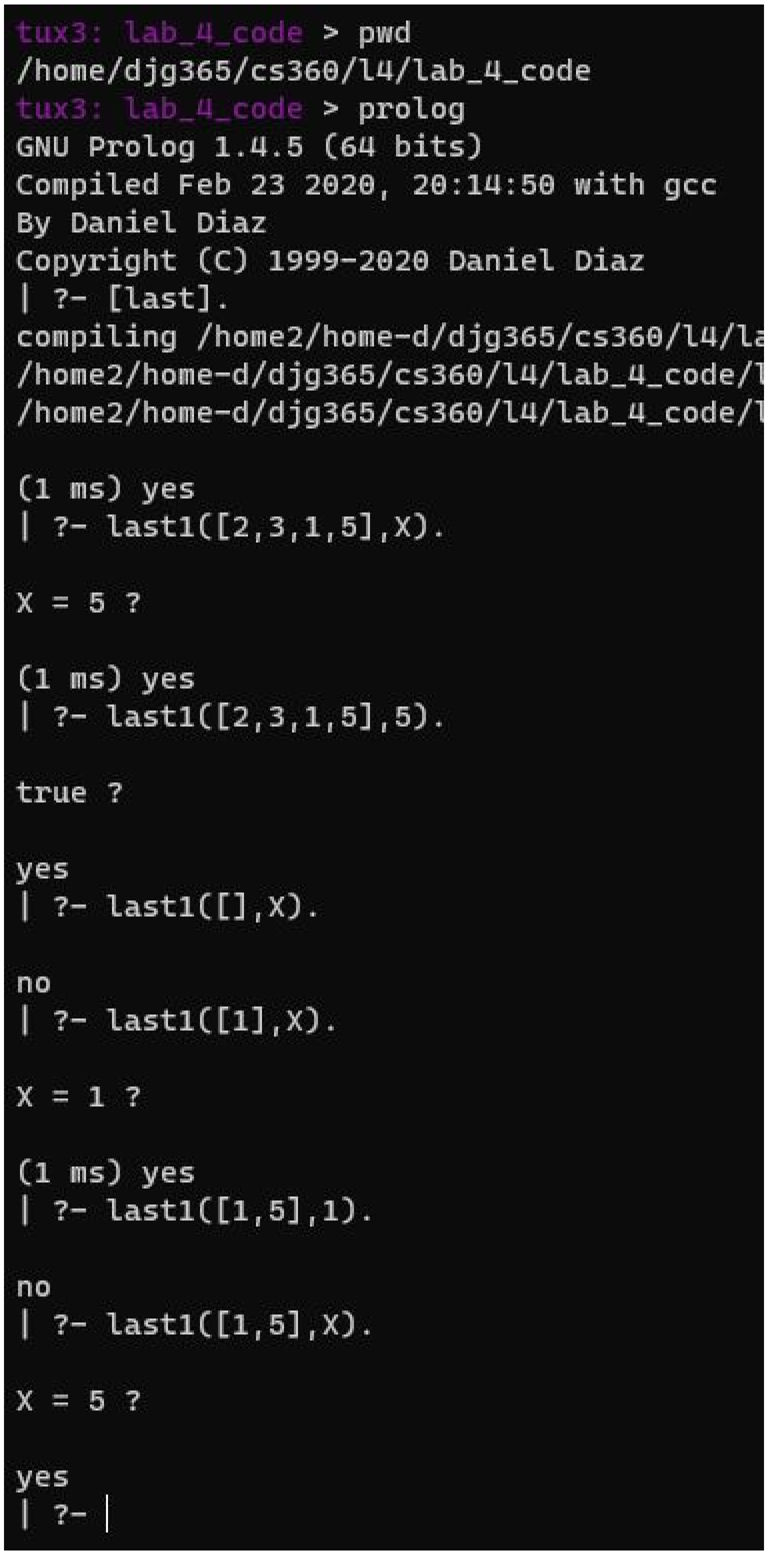
```
tux3: lab_4_code > pwd
/home/djg365/cs360/l4/lab_4_code
tux3: lab_4_code > prolog
GNU Prolog 1.4.5 (64 bits)
Compiled Feb 23 2020, 20:14:50 with gcc
By Daniel Diaz
Copyright (C) 1999-2020 Daniel Diaz
| ?- [gcd].
compiling /home2/home-d/djg365/cs360/l4/lab_4_code/gcd.pl for byte code...
/home2/home-d/djg365/cs360/l4/lab_4_code/gcd.pl compiled, 6 lines read - 2119 bytes written, 6 ms

yes
| ?- gcd(36,21,X).

X = 3

yes
| ?- |
```

The greatest common divisor of 36 and 21 is **3**.

Describe the computational complexity of the provided Prolog gcd code and compare it with the computational complexity of the Euclid's algorithm.

Euclidean -> O(log N)
Prolog gcd code -> O(N)
The computational complexity of prolog gcd code is linear. Gcd(x,y) calls min() which takes in x and y and gives the minimum number out of the two as the output, m. greatestdivisor() takes x, y and m, and checks if m divides both inputs completely, it it does then m is the gcd of x and y.
If it doesn't then greatestdivisor() is called again with x, y and m-1. This goes on until we find a number 'p' that divides x and y completely. Thus the computational complexity is linear, as we keep subtracting 1.

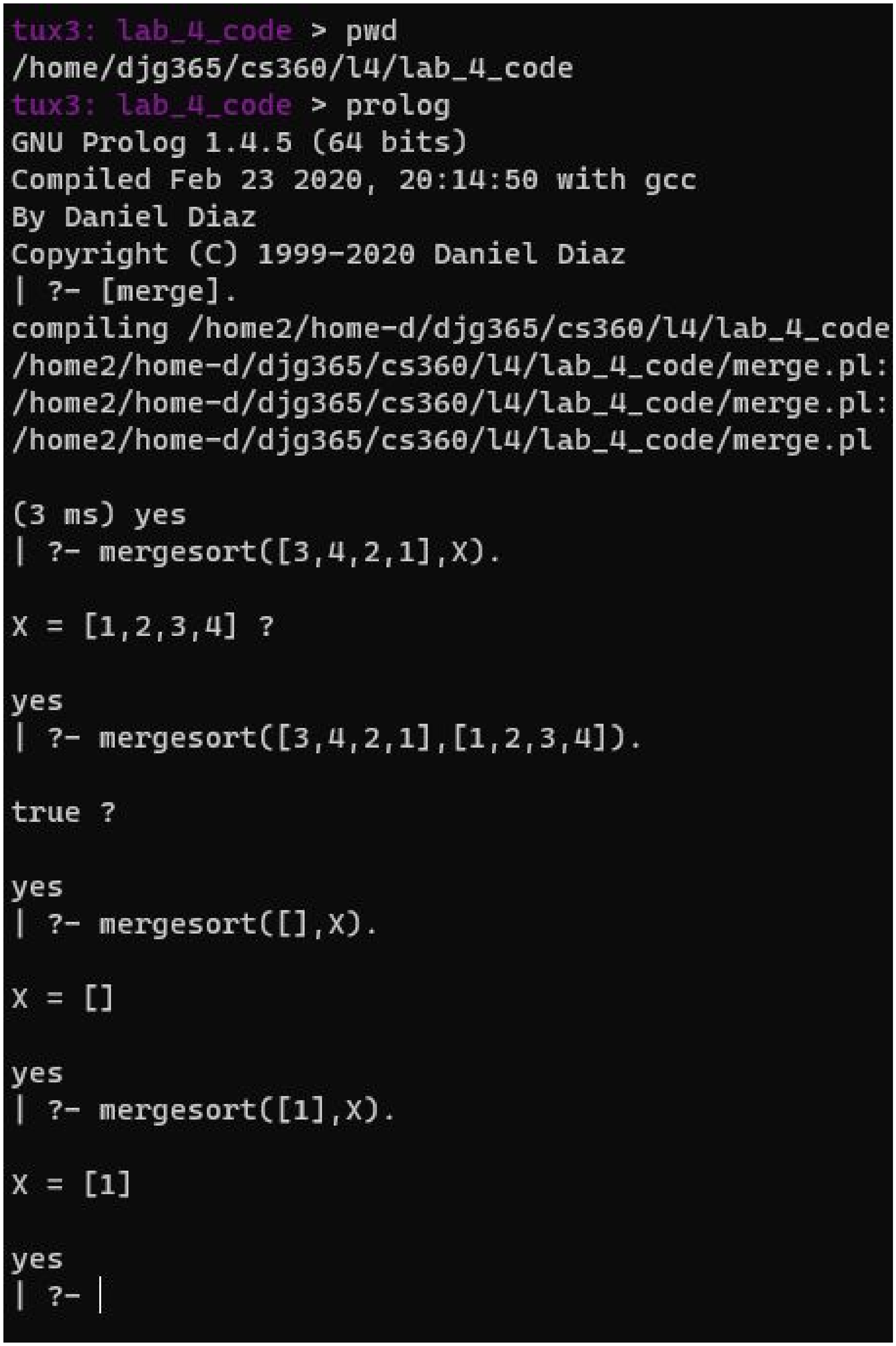
(ii) In Prolog, load the file last.pl. Demonstrate that the last1 predicate works correctly. What Prolog data structures and language features are used in the provided code? What is the target of its computation?



```
tux3: lab_4_code > pwd
/home/djg365/cs360/l4/lab_4_code
tux3: lab_4_code > prolog
GNU Prolog 1.4.5 (64 bits)
Compiled Feb 23 2020, 20:14:50 with gcc
By Daniel Diaz
Copyright (C) 1999-2020 Daniel Diaz
| ?- [last].
compiling /home2/home-d/djg365/cs360/l4/la
/home2/home-d/djg365/cs360/l4/lab_4_code/l
/home2/home-d/djg365/cs360/l4/lab_4_code/l

(1 ms) yes
| ?- last1([2,3,1,5],X).

X = 5 ?

(1 ms) yes
| ?- last1([2,3,1,5],5).

true ?

yes
| ?- last1([],X).

no
| ?- last1([1],X).

X = 1 ?

(1 ms) yes
| ?- last1([1,5],1).

no
| ?- last1([1,5],X).

X = 5 ?

yes
| ?-
```

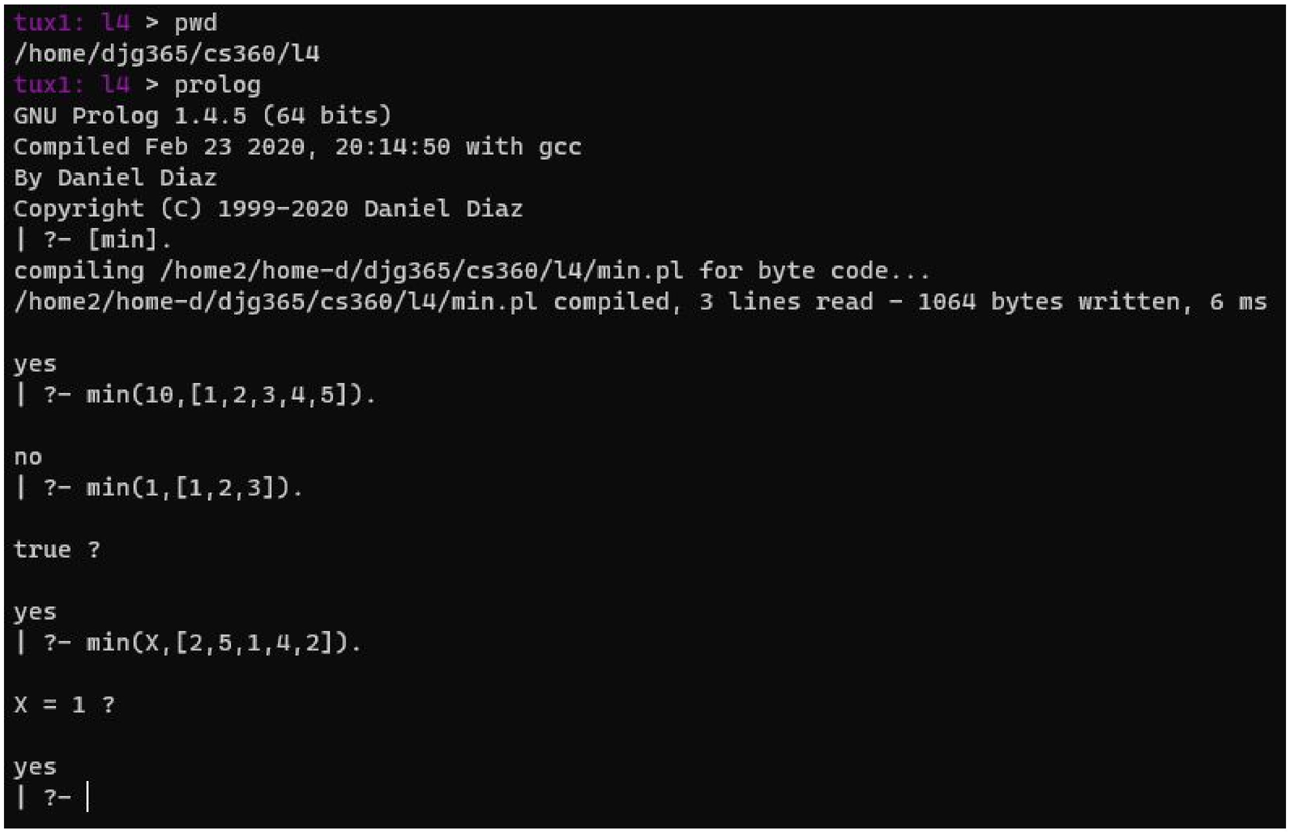
Language features - object predicate values, strongly typed, algebraic data types, pattern matching
Data structures - Terms -> Variables and atomic terms, Lists, comparison.
Target - to get the last element of a given list

(iii) In Prolog, load the file merge.pl. Demonstrate that the mergesort predicate runs correctly.

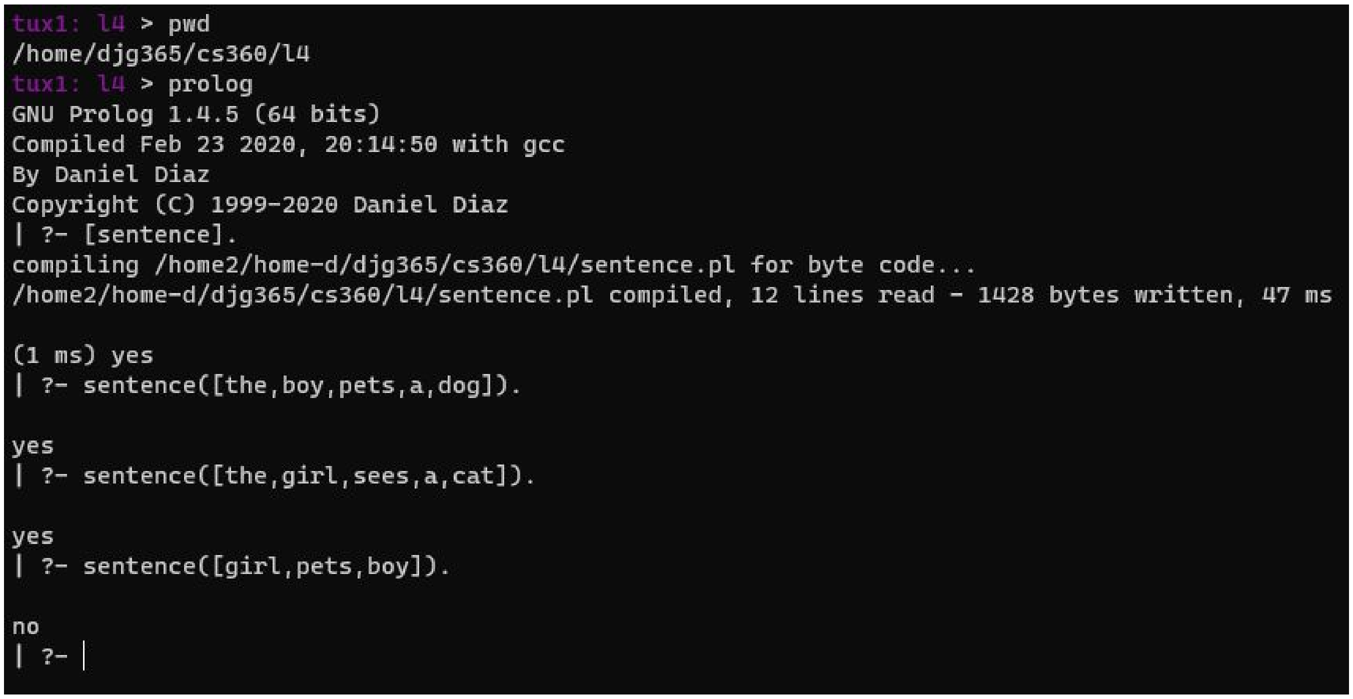


```
tux3: lab_4_code > pwd
/home/djg365/cs360/l4/lab_4_code
tux3: lab_4_code > prolog
GNU Prolog 1.4.5 (64 bits)
Compiled Feb 23 2020, 20:14:50 with gcc
By Daniel Diaz
Copyright (C) 1999-2020 Daniel Diaz
| ?- [merge].
compiling /home2/home-d/djg365/cs360/l4/lab_4_code
/home2/home-d/djg365/cs360/l4/lab_4_code/merge.pl:
/home2/home-d/djg365/cs360/l4/lab_4_code/merge.pl:
/home2/home-d/djg365/cs360/l4/lab_4_code/merge.pl

(3 ms) yes
| ?- mergesort([3,4,2,1],X).

X = [1,2,3,4] ?

yes
| ?- mergesort([3,4,2,1],[1,2,3,4]).

true ?

yes
| ?- mergesort([],X).

X = []

yes
| ?- mergesort([1],X).

X = [1]

yes
| ?- 
```

2. In Prolog, load the functions min and sentence you prepared for the lab. Demonstrate that your functions operate properly.

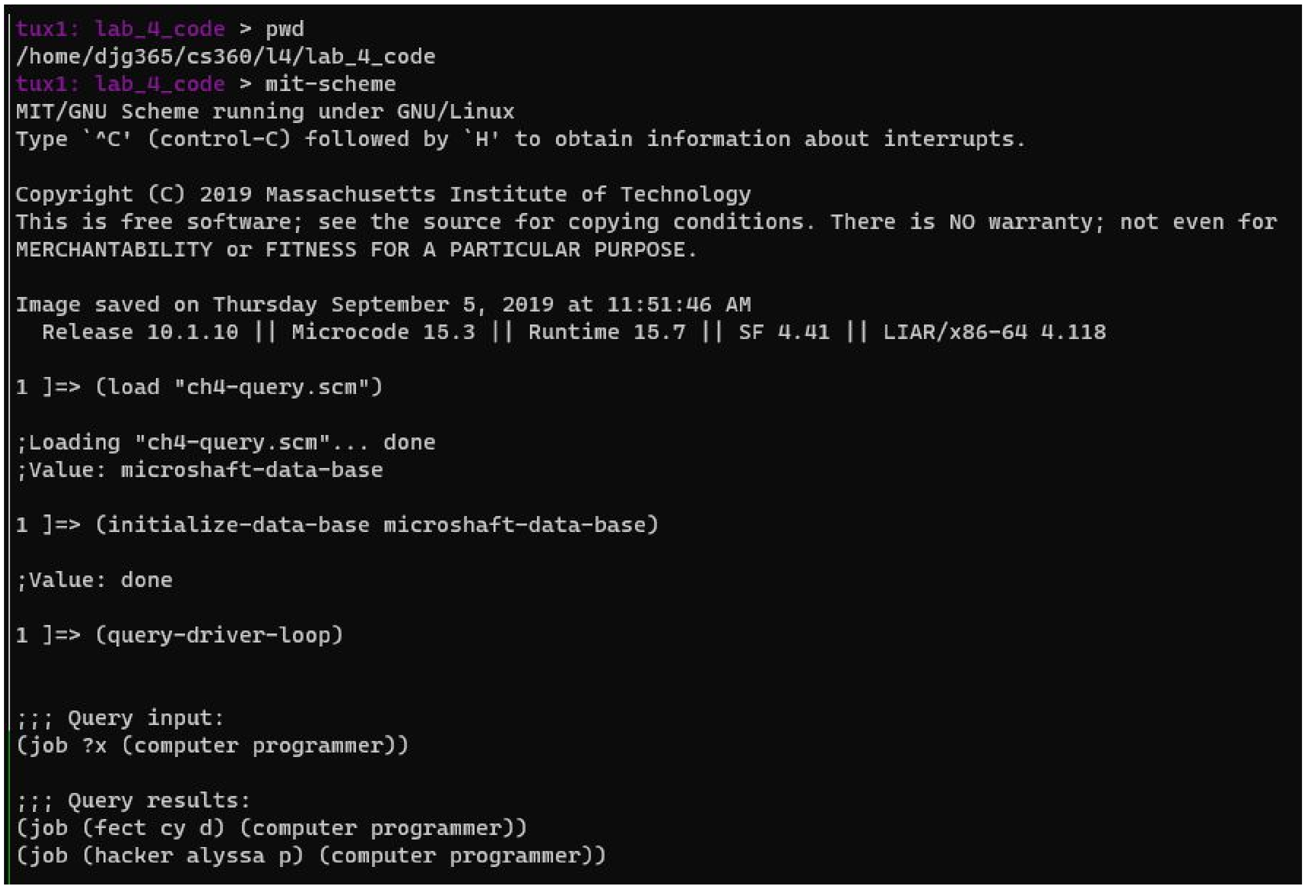


```
tux1: l4 > pwd
/home/djg365/cs360/l4
tux1: l4 > prolog
GNU Prolog 1.4.5 (64 bits)
Compiled Feb 23 2020, 20:14:50 with gcc
By Daniel Diaz
Copyright (C) 1999-2020 Daniel Diaz
| ?- [min].
compiling /home2/home-d/djg365/cs360/l4/min.pl for byte code...
/home2/home-d/djg365/cs360/l4/min.pl compiled, 3 lines read - 1064 bytes written, 6 ms

yes
| ?- min(10,[1,2,3,4,5]).

no
| ?- min(1,[1,2,3]).

true ?

yes
| ?- min(X,[2,5,1,4,2]).

X = 1 ?

yes
| ?- 
```

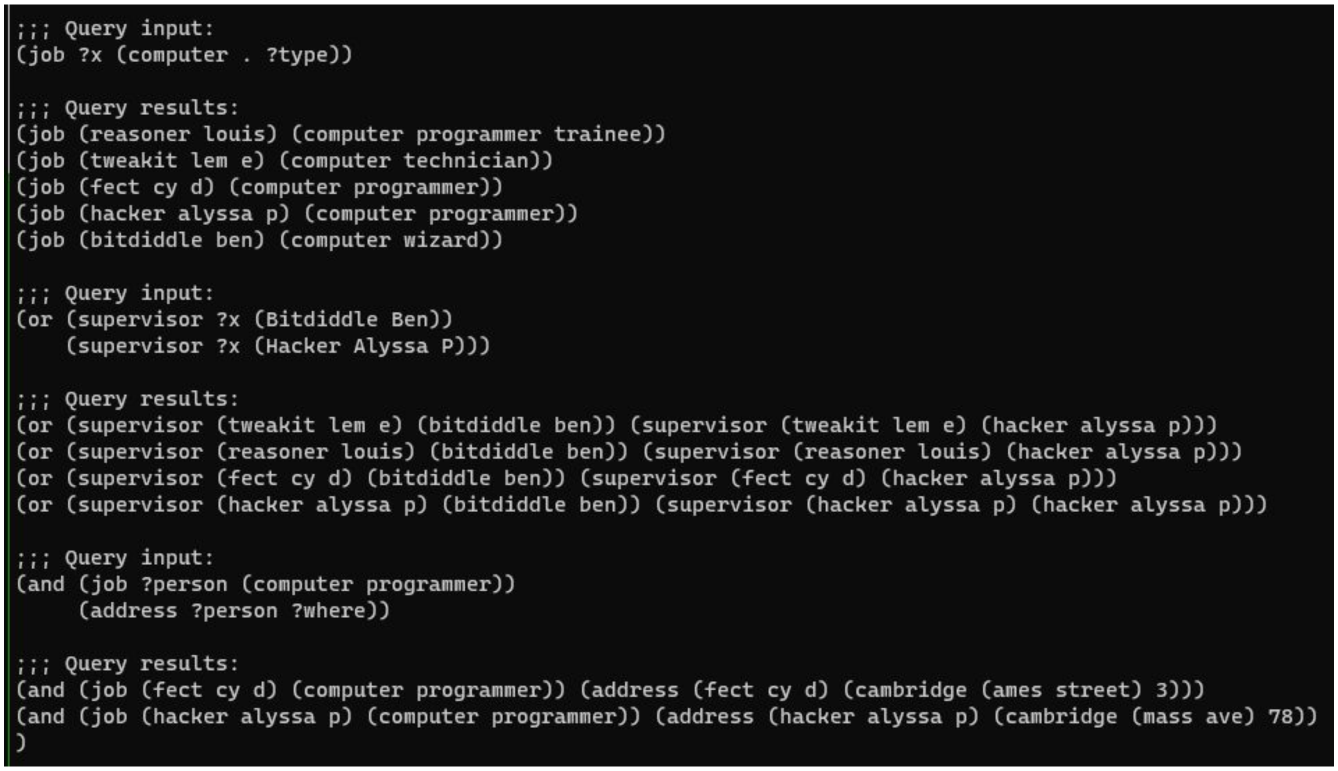


```
tux1: l4 > pwd
/home/djg365/cs360/l4
tux1: l4 > prolog
GNU Prolog 1.4.5 (64 bits)
Compiled Feb 23 2020, 20:14:50 with gcc
By Daniel Diaz
Copyright (C) 1999-2020 Daniel Diaz
| ?- [sentence].
compiling /home2/home-d/djg365/cs360/l4/sentence.pl for byte code...
/home2/home-d/djg365/cs360/l4/sentence.pl compiled, 12 lines read - 1428 bytes written, 47 ms

(1 ms) yes
| ?- sentence([the,boy,pets,a,dog]).

yes
| ?- sentence([the,girl,sees,a,cat]).

yes
| ?- sentence([girl,pets,boy]).

no
| ?- 
```

3. In Scheme, load the file ch4-query.scm. Demonstrate that you can run the provided code and answer simple queries with its help.



```
tux1: lab_4_code > pwd
/home/djg365/cs360/l4/lab_4_code
tux1: lab_4_code > mit-scheme
MIT/GNU Scheme running under GNU/Linux
Type `^C' (control-C) followed by `H' to obtain information about interrupts.

Copyright (C) 2019 Massachusetts Institute of Technology
This is free software; see the source for copying conditions. There is NO warranty; not even for
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Image saved on Thursday September 5, 2019 at 11:51:46 AM
  Release 10.1.10 || Microcode 15.3 || Runtime 15.7 || SF 4.41 || LIAR/x86-64 4.118

1 ]=> (load "ch4-query.scm")

;Loading "ch4-query.scm"... done
;Value: microshaft-data-base

1 ]=> (initialize-data-base microshaft-data-base)

;Value: done

1 ]=> (query-driver-loop)


;;; Query input:
(job ?x (computer programmer))

;;; Query results:
(job (fect cy d) (computer programmer))
(job (hacker alyssa p) (computer programmer))
```



```
;;; Query input:
(job ?x (computer . ?type))

;;; Query results:
(job (reasoner louis) (computer programmer trainee))
(job (tweakit lem e) (computer technician))
(job (fect cy d) (computer programmer))
(job (hacker alyssa p) (computer programmer))
(job (bitdiddle ben) (computer wizard))

;;; Query input:
(or (supervisor ?x (Bitdiddle Ben))
    (supervisor ?x (Hacker Alyssa P)))

;;; Query results:
(or (supervisor (tweakit lem e) (bitdiddle ben)) (supervisor (tweakit lem e) (hacker alyssa p)))
(or (supervisor (reasoner louis) (bitdiddle ben)) (supervisor (reasoner louis) (hacker alyssa p)))
(or (supervisor (fect cy d) (bitdiddle ben)) (supervisor (fect cy d) (hacker alyssa p)))
(or (supervisor (hacker alyssa p) (bitdiddle ben)) (supervisor (hacker alyssa p) (hacker alyssa p)))

;;; Query input:
(and (job ?person (computer programmer))
     (address ?person ?where))

;;; Query results:
(and (job (fect cy d) (computer programmer)) (address (fect cy d) (cambridge (ames street) 3)))
(and (job (hacker alyssa p) (computer programmer)) (address (hacker alyssa p) (cambridge (mass ave) 78))
)
```