University of Chittagong

Department of Computer Science & Engineering

Assignment on Turbo PROLOG

Artificial Intelligence

CSE 713

Submitted To

Dr. Mohammad Shahadat Hossain

Professor

Computer Science & Engineering

University of Chittagong

Submitted By

Toasean Elmah Tasean

Department of Computer Science and Engineering

University of Chittagong

ID : 20701051

August 25, 2025

Chapter 1: About Prolog

Chapter 1 presents Turbo Prolog as a fifth-generation declarative programming language. Unlike procedural languages like Pascal or BASIC, Turbo Prolog uses deductive reasoning, enabling programmers to define problems and rules while the system finds the solutions. This makes it well-suited for building expert systems, knowledge bases, and intelligent tools.

Exercise: No exercise in this chapter.

Chapter 2: A Short Introduction to the Turbo Prolog System

Exercise

Write a program that reads a user's name and outputs a message.

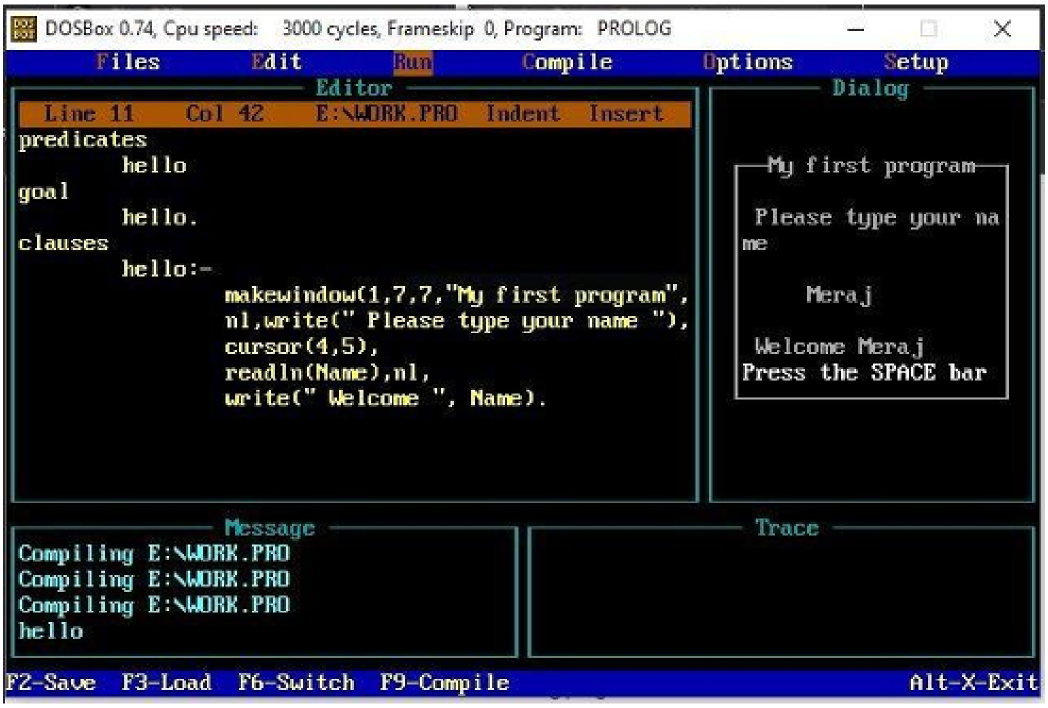

Figure 1: Result of running the program

```
predicates hello
goal hello.
```

```
clauses hello :-
    makewindow(1,7,7,"My first program",4,56,10,22),
    nl, write(" Please type your name "),
    cursor(4,5), readln(Name),
    nl, write(" Welcome ",Name).
```

Chapter 3: Tutorial I – Five Simple Programs

Exercise 3.1

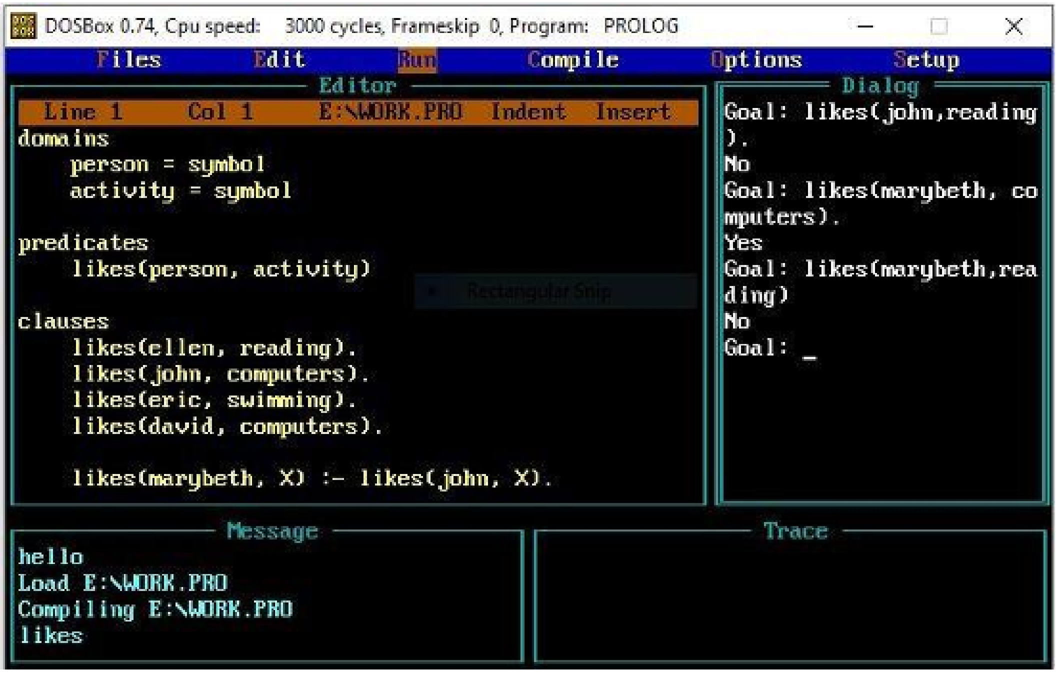

Figure 2: Result of running the program

Facts and rules:

```
domains
    person = symbol
    activity = symbol

predicates
    likes(person, activity)

clauses
    likes(ellen, reading).
```

```
likes(john, computers).
likes(eric, swimming).
likes(david, computers).
likes(marybeth, X) :- likes(john, X).
```

Exercise 3.2

Facts about pupils:

```
domains
    child = symbol
    age = integer

predicates
    pupil(child, age)

clauses
    pupil(peter,9).
    pupil(paul,10).
    pupil(chris,9).
    pupil(susan,9).
```

Goal:

```
pupil(Person1,9), pupil(Person2,10).
```

Exercise 3.3

Thesaurus program:

```
similar_meaning(big,enormous).
similar_meaning(big,tall).
similar_meaning(big,huge).
similar_meaning(happy,cheerful).
similar_meaning(happy,gay).
similar_meaning(happy,contented).
```

Goal:

```
similar_meaning(big, X).
```

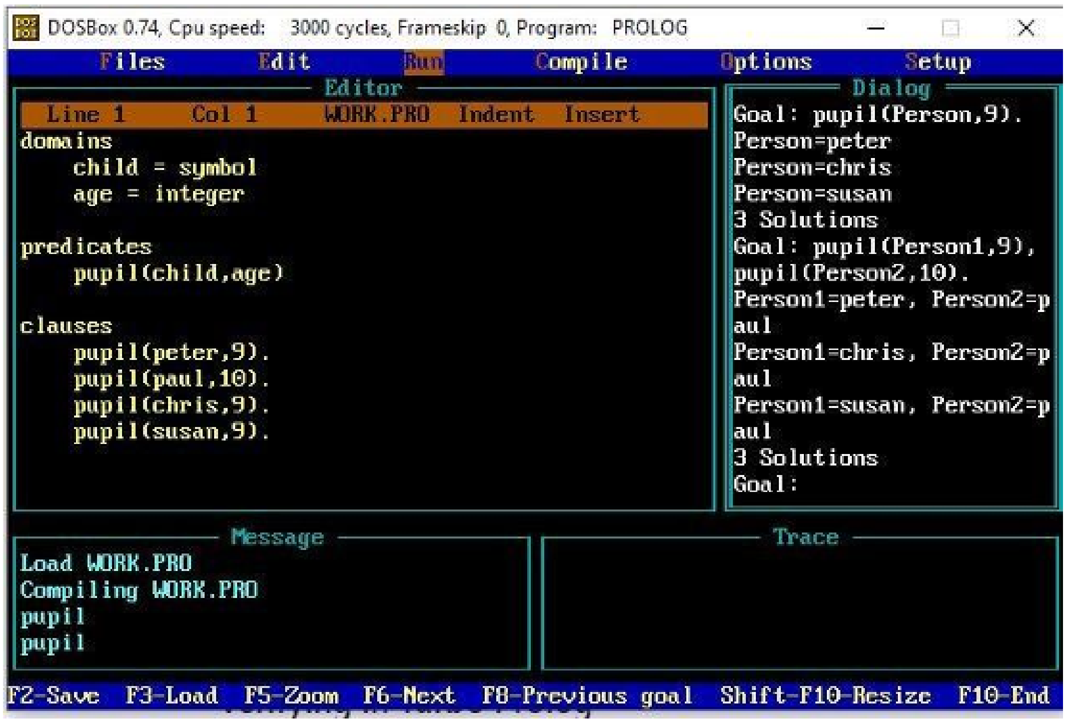

Figure 3: Result of running the program

Exercise 3.4

Murder mystery problem:

```
person(allan, 25, m, football_player).
person(allan, 25, m, butcher).
person(barbara, 22, f, hairdresser).
person(bert, 55, m, carpenter).
person(john, 25, m, pickpocket).

had_affair(barbara, john).
had_affair(barbara, bert).
had_affair(susan, john).

killed_with(susan, club).

motive(money).
motive(jealousy).

smeared_in(catherine, blood).
smeared_in(allan, mud).
```

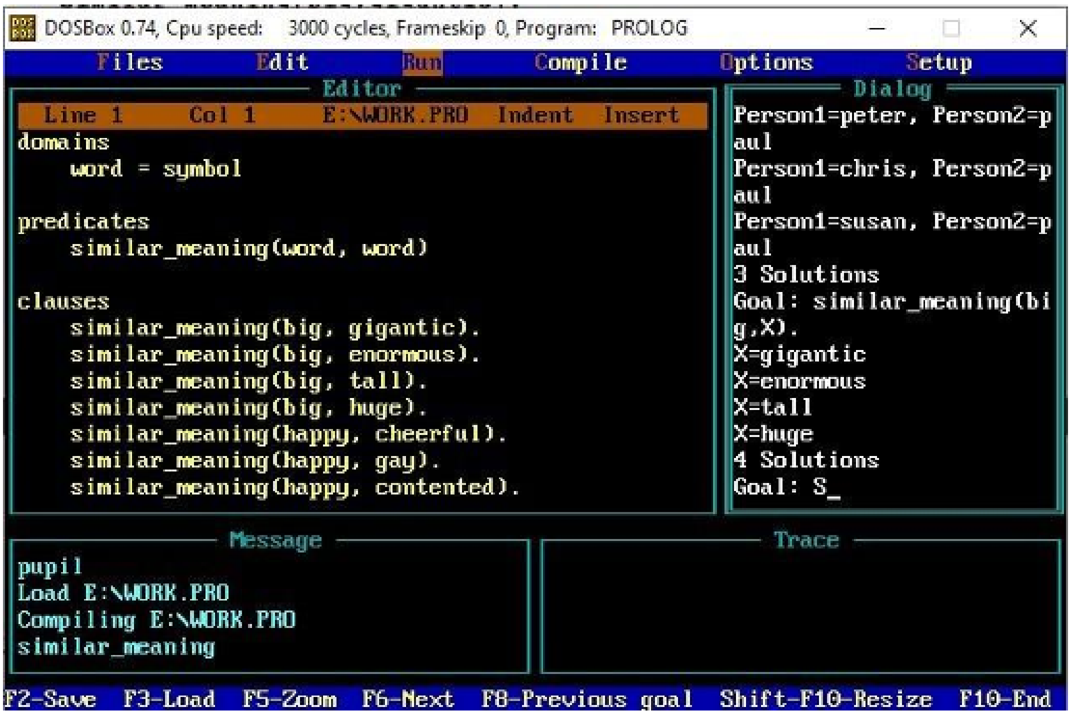

Figure 4: Result of running the program

```
owns(bert, wooden_leg).
owns(john, pistol).

/* Background knowledge */
operates_identically(wooden_leg, club).
operates_identically(bar, club).
operates_identically(pair_of_scissors, knife).
operates_identically(football_boot, club).

owns_probably(X, football_boot) :- person(X, _, _, football_player).
owns_probably(X, pair_of_scissors) :- person(X, _, _, hairdresser).
owns_probably(X, Object) :- owns(X, Object).

/* Suspect rules */
suspect(X) :- killed_with(susan, Weapon),
              operates_identically(Object, Weapon),
              owns_probably(X, Object).

suspect(X) :- motive(jealousy),
              person(X, _, m, _),
```

```
                had_affair(susan, X).

suspect(X) :- motive(jealousy),
              person(X, _, f, _),
              had_affair(X, Man),
              had_affair(susan, Man).

suspect(X) :- motive(money),
              person(X, _, _, pickpocket).
```

Solution in Turbo Prolog form:

```
domains
    name = symbol
    age = integer
    gender = symbol
    profession = symbol
    weapon = symbol
    motive_type = symbol

predicates
    person(name, age, gender, profession)
    had_affair(name, name)
    killed_with(name, weapon)
    motive(motive_type)
    smeared_in(name, symbol)
    owns(name, symbol)
    operates_identically(symbol, symbol)
    owns_probably(name, symbol)
    suspect(name)

clauses
    person(allan, 25, m, football_player).
    person(allan, 25, m, butcher).
    person(barbara, 22, f, hairdresser).
    person(bert, 55, m, carpenter).
    person(john, 25, m, pickpocket).

    had_affair(barbara, john).
    had_affair(barbara, bert).
```

```prolog
had_affair(susan, john).

killed_with(susan, club).

motive(money).
motive(jealousy).

smeared_in(catherine, blood).
smeared_in(allan, mud).

owns(bert, wooden_leg).
owns(john, pistol).

operates_identically(wooden_leg, club).
operates_identically(bar, club).
operates_identically(pair_of_scissors, knife).
operates_identically(football_boot, club).

owns_probably(X, football_boot) :- person(X, _, _, football_playe
owns_probably(X, pair_of_scissors) :- person(X, _, _, hairdresser
owns_probably(X, Object) :- owns(X, Object).

suspect(X) :- killed_with(susan, Weapon),
              operates_identically(Object, Weapon),
              owns_probably(X, Object).

suspect(X) :- motive(jealousy),
              person(X, _, m, _),
              had_affair(susan, X).

suspect(X) :- motive(jealousy),
              person(X, _, f, _),
              had_affair(X, Man),
              had_affair(susan, Man).

suspect(X) :- motive(money),
              person(X, _, _, pickpocket).
```

Figure 5: Result of running the murder mystery program

Chapter 4: Tutorial II – Domains, Objects and Lists

Exercise 4.1

Consider the following program:

```
predicates
    reference(name, phone_no)

goal
    write("Please type a name: "),
    readln(The_Name),
    reference(The_Name, Phone_No),
    write("The phone number is "),
    write(Phone_No), nl.

clauses
    reference("Albert", "01-1234561").
    reference("Betty", "01-569767").
    reference("Carol", "01-2671001").
    reference("Dorothy", "01-191051").
```

Try each of these queries in turn:

1. reference("Carol", Y).

2. reference(X, "01-191051").

3. reference("Mavis", Y).

4. reference(X, Y).

Solution: After running the following completed code:

Figure 6: Result of running the program

```
domains
    name = symbol
    phone_no = symbol

predicates
    reference(name, phone_no)

clauses
    reference("Albert", "01-1234561").
    reference("Betty", "01-569767").
    reference("Carol", "01-2671001").
    reference("Dorothy", "01-191051").
```

Exercise 4.2

Write a suitable `domains` declaration using compound objects that could be used in a Turbo Prolog catalog of musical shows. A typical entry in the catalog might be:

- Show: West Side Story
- Lyrics: Stephen Sondheim
- Music: Leonard Bernstein

Solution:

```
domains
    show = symbol
    name = symbol
    musical_show = catalog_entry(show, name, name)

predicates
    catalog(musical_show)

clauses
    catalog(catalog_entry("West_Side_Story",
                          "Stephen_Sondheim",
                          "Leonard_Bernstein")).
```

Exercise 4.3

Using compound objects wherever possible, write a Turbo Prolog program to keep a database of the current Top Ten hit records. Entries should include the name of the song, the name of the singer or group, its position in the Top Ten chart, and the number of weeks in the charts.

Solution:

```
domains
    song = symbol
    artist = symbol
    position = integer
    weeks = integer
    hit_record = record(song, artist, position, weeks)

predicates
    top_ten(hit_record)
```

```
clauses
    top_ten(record("Blinding_Lights", "The_Weeknd", 1, 45)).
    top_ten(record("Watermelon_Sugar", "Harry_Styles", 2, 35)).
    top_ten(record("Rockstar", "DaBaby_ft._Roddy_Ricch", 3, 30)).
    top_ten(record("Savage_Love", "Jawsh_685_and_Jason_Derulo", 4, 20
    top_ten(record("Roses", "SAINt_JHN", 5, 25)).
    top_ten(record("Before_You_Go", "Lewis_Capaldi", 6, 40)).
    top_ten(record("Say_So", "Doja_Cat", 7, 50)).
    top_ten(record("Adore_You", "Harry_Styles", 8, 45)).
    top_ten(record("Dance_Monkey", "Tones_and_I", 9, 60)).
    top_ten(record("Don't_Start_Now", "Dua_Lipa", 10, 55)).

goal
    top_ten(Record), write(Record), nl, fail.

goal
    top_ten(record("Blinding_Lights", Artist, _, _)),
    write("Artist: "), write(Artist), nl.

goal
    top_ten(record("Savage_Love", _, Position, Weeks)),
    write("Position: "), write(Position), nl,
    write("Weeks in chart: "), write(Weeks), nl.

goal
    top_ten(record(Song, "Harry_Styles", _, _)),
    write("Song: "), write(Song), nl.
```

Exercise 4.4

Add `domains` and `predicates` declarations to the following facts and rules:

```
factorial(X,Y) if newfactorial(0,1,X,Y).
newfactorial(X,Y,X,Y).
newfactorial(U,V,X,Y) if
    U1=U+1, U1V=U1*V,
    newfactorial(U1,U1V,X,Y).
```

and try out the resulting program with the following goals: `factorial(3,Answer).` `factorial(4,Answer).` `factorial(5,Answer).`

Solution:

Figure 7: Result of running the factorial program

```
domains
   num = integer

predicates
    factorial(num, num)
    newfactorial(num, num, num, num)

clauses
    factorial(X, Y) :- newfactorial(0, 1, X, Y).

    newfactorial(X, Y, X, Y).

    newfactorial(U, V, X, Y) :-
         U < X,
         U1 = U + 1,
         V1 = U1 * V,
         newfactorial(U1, V1, X, Y).
```

Chapter 5: Turbo Prolog's Relentless Search for Solutions

Exercise 5.1

Type in Program 15 and evaluate the following goals:

- father(X, Y).
- everybody.

Why are the solutions to `everybody` terminated by `False`? For a clue, append `everybody` as a second clause to the definition of predicate `everybody` and reevaluate the goal.

Solution:

Given Program 14:

```
/* Program 14 */
```

Program 15 in Turbo Prolog form:

```
/* Program 15 */

domains
    name = symbol

predicates
    father(name, name)
    everybody

clauses
    father(leonard, katherine).
    father(carl, jason).
    father(carl, marilyn).

    everybody :-
        father(X, Y),
        write(X, " is ", Y, " s   father"),
        nl, fail.
```

Running the code above with the given goals, we get the following result:

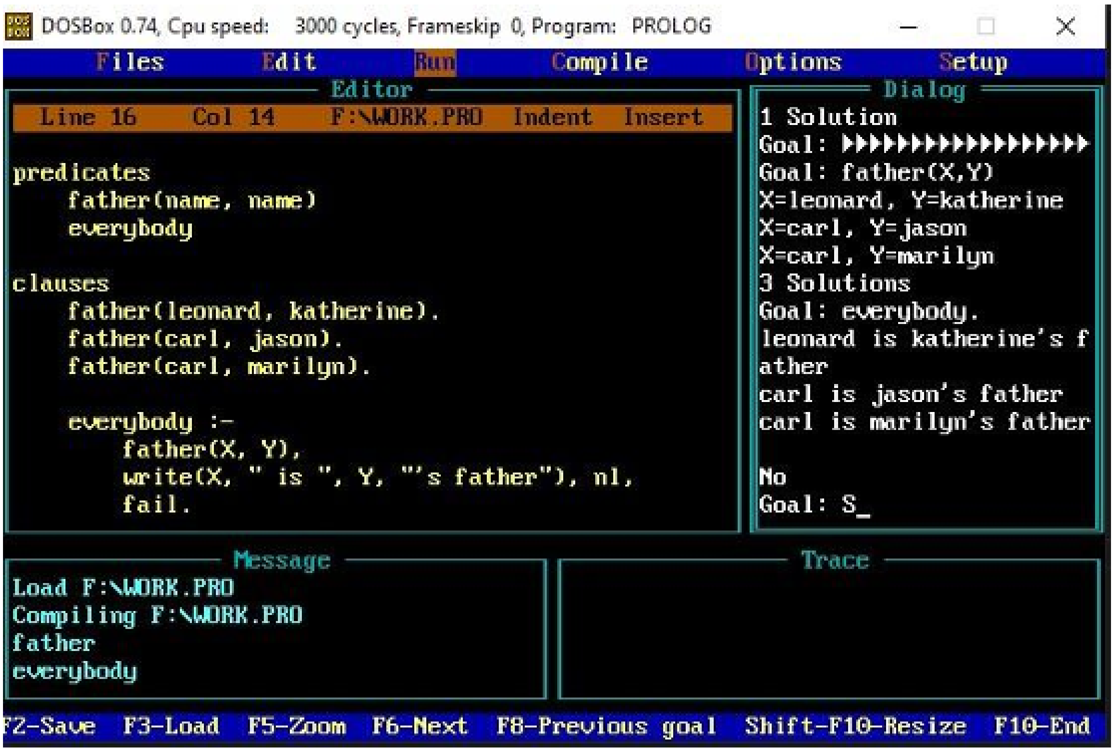

Figure 8: Result of running Program 15

Chapter 6: Arithmetic, Input/Output, and Debugging

Exercise 6.1

Consider the following program:

```
predicates
    solve(real, real, real)
    reply(real, real, real)
    mysqrt(real, real, real)
    equal(real, real)

clauses
    solve(A, B, C) :-
        D = B * B - 4 * A * C,
        reply(A, B, D), nl.

    reply(_, _, D) :- D < 0,
        write("No solution"), !.

    reply(A, B, D) :- D = 0,
        X = -B / (2 * A),
```

```
        write("x = "), write(X), nl, !.

    reply(A, B, D) :-
        mysqrt(D, D, SqrtD),
        X1 = (-B + SqrtD) / (2 * A),
        X2 = (-B - SqrtD) / (2 * A),
        write("x1 = "), write(X1),
        write(" and x2 = "), write(X2), nl.

    mysqrt(X, Guess, Root) :-
        NewGuess = Guess - (Guess * Guess - X) / (2 * Guess),
        not(equal(NewGuess, Guess)), !,
        mysqrt(X, NewGuess, Root).

    mysqrt(_, Guess, Guess).

    equal(X, Y) :-
        X / Y > 0.99999,
        X / Y < 1.00001.
```

Try the following goals:

- `solve(1,2,1).`
- `solve(1,1,1).`
- `solve(1,-3,2).`

Solution: After running the following code:

```
domains
    num = real

predicates
    solve(num, num, num)
    reply(num, num, num)
    mysqrt(num, num, num)
    equal(num, num)

clauses
    solve(A, B, C) :-
        D = B * B - 4 * A * C,
```

```
        reply(A, B, D), nl.

    reply(_, _, D) :- D < 0,
        write("No solution"), !.

    reply(A, B, D) :- D = 0,
        X = -B / (2 * A),
        write("x = "), write(X), nl, !.

    reply(A, B, D) :-
        mysqrt(D, D, SqrtD),
        X1 = (-B + SqrtD) / (2 * A),
        X2 = (-B - SqrtD) / (2 * A),
        write("x1 = "), write(X1),
        write(" and x2 = "), write(X2), nl.

    mysqrt(X, Guess, Root) :-
        NewGuess = Guess - (Guess * Guess - X) / (2 * Guess),
        not(equal(NewGuess, Guess)), !,
        mysqrt(X, NewGuess, Root).

    mysqrt(_, Guess, Guess).

    equal(X, Y) :-
        X / Y > 0.99999,
        X / Y < 1.00001.
```

Exercise 6.2

Turbo Prolog has a built-in square root function `sqrt`. Thus,

$$X = \sqrt{D}$$

will bind `X` to the square root of the value to which `D` is bound. Program 6.1 can be rewritten using `sqrt` and then compared with the original version.

Modified Program

```
domains
    num = real
```

Figure 9: Result of running the quadratic solver program (custom `mysqrt`)

```
predicates
    solve(num, num, num)
    reply(num, num, num)
    equal(num, num)

clauses
    solve(A, B, C) :-
        D = B * B - 4 * A * C,
        reply(A, B, D), nl.

    reply(_, _, D) :-
        D < 0,
        write("No solution"), !.

    reply(A, B, D) :-
        D = 0,
        X = -B / (2 * A),
        write("x = "), write(X), nl, !.

    reply(A, B, D) :-
```

```
        X1 = (-B + sqrt(D)) / (2 * A),
        X2 = (-B - sqrt(D)) / (2 * A),
        write("x1␣=␣"), write(X1),
        write("␣and␣x2␣=␣"), write(X2), nl.

    equal(X, Y) :-
        X / Y > 0.99999,
        X / Y < 1.00001.
```

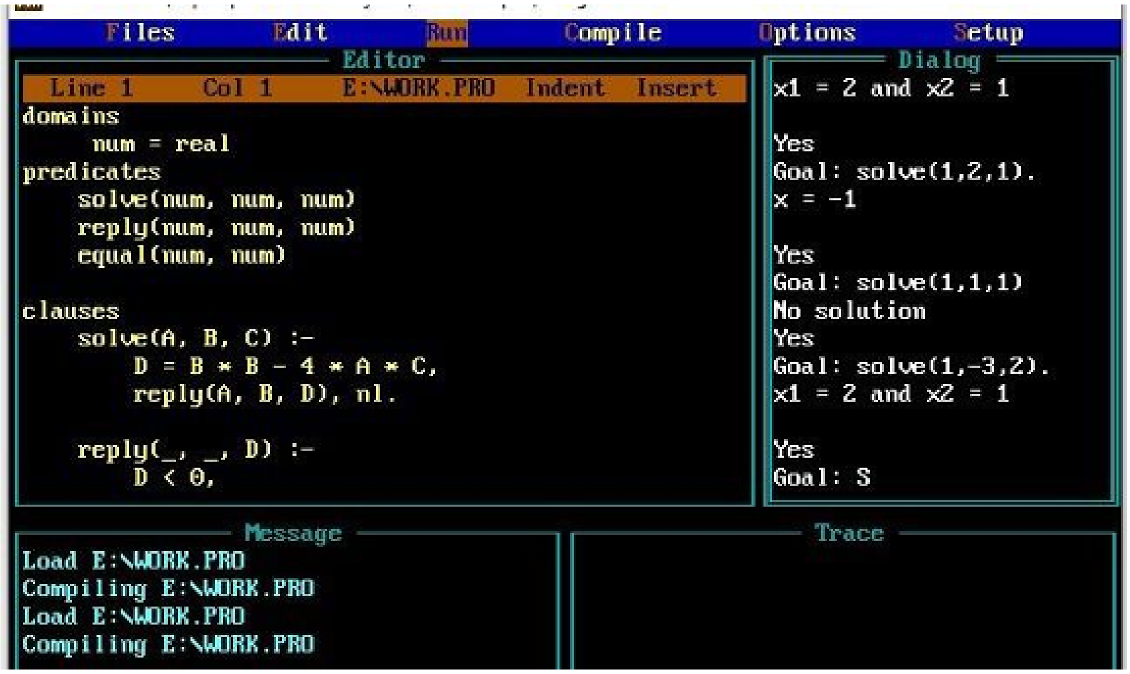

Figure 10: Result of running the program

Explanation of Modifications

- **Removed `mysqrt` Predicate:** Since Turbo Prolog provides the `sqrt` function, the custom `mysqrt` predicate is no longer needed.
- **Using `sqrt(D)`:** The expressions for the quadratic roots now directly calculate the square root of `D`.

Comparison

By using the built-in `sqrt` function, this version is both simpler and faster since it eliminates the need for iterative calculations of the square root. Both versions yield the same results for quadratic equations, but the modified version is more efficient and concise.