



Staffordshire University

–

Artificial Intelligence

**Assessment level 3**

**Holiday Travel Guide**

Christiane Schmidt

November 27, 2003

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	About Prolog . . . . .	1
<b>2</b>	<b>The Holiday Travel Guide</b>	<b>2</b>
2.1	Purpose of the system . . . . .	2
<b>3</b>	<b>Knowledge representation</b>	<b>3</b>
<b>4</b>	<b>The implementation</b>	<b>4</b>
4.1	The user interface . . . . .	6
<b>5</b>	<b>Appendix</b>	<b>I</b>
5.1	Program Source Code . . . . .	II
5.2	Testruns . . . . .	XV

## List of Figures

1	semantic network . . . . .	3
2	main menu . . . . .	XV
3	help menu . . . . .	XVI
4	menu choice a . . . . .	XVII
5	menu choice b . . . . .	XVIII
6	menu choice c . . . . .	XIX
7	menu choice d . . . . .	XX
8	menu choice e . . . . .	XXI
9	end program . . . . .	XXII

# 1 Introduction

This coursework task was to choose and develop an intelligent system for a special domain. This coursework is about the development of an artificial intelligence agent that recommends counties after the input of country-specific properties.

This project is implemented in Prolog, a programming language for artificial intelligence.

## 1.1 About Prolog

The word Prolog is derived from '**P**rogramming in **L**ogic'.

Prolog is said to be a programming language of the fourth generation. Though developed at the beginning of the 1970ies, Prolog changed only few - the first version has already a high level of syntax complexity.

Like a database a Prolog programm consists of a description of a special domain - a reproduction of the real world in a 'mini-world'. The Prolog system provides the in the program saved knowledge for the user.

The difference to other programming language is that in Prolog is not specified how to solve a specific problem but to list of which sub-problems a problem consists and in which relations are between them.

In general Prolog programming is a declarative programming style and Prolog finds a solution without having programmed the way for solving it.

## 2 The Holiday Travel Guide

### 2.1 Purpose of the system

I really like and I am very interested traveling to different countries. So the idea for a traveling system was born.

The system is able to search for countries by specific criteria. Every country is, like in the real world, defined with country-specific properties. One country is defined by its name, the way of how to get there (by plane or by car), the most typical season when to travel to it (in this database Austria is, for example, defined as a typical winter holiday destination) and some general characteristics of the country (e.g. skiing or snow for winter countries). Additionally, every country is either in Europe, or far away from the UK (meaning not in Europe).

The target user of the system is a person that plans to go to holiday. He or she has the possibility to choose a list of all available countries that are saved in the database. With this list the user can choose to get all the properties that are saved in the database for one country. Additionally the user can check, if a country has specific properties, e.g. if Australia has a fascinating landscape or if not. Further on it is possible to choose a country the user likes and get all other countries that have similar properties, e.g. Spain has the same properties in the database like Greece.

The main part of the program is to type in the desired properties of the holiday destination and get out a list of all countries that fulfil those properties.

### 3 Knowledge representation

This system is a Prolog program to represent the 'Travel Guide' semantic network. It is a simple version of an semantic network consisting of different lands with properties and a season that every country refers to. The season (winter or summer) is defined by a number of months (in total 12, from January to December), 11 to 4 for winter months and 5 to 9 for summer months in the database.

Each country is defined by its name, a list of properties, the best months when to go there (derivated from the season winter or summer), the region (if it is far away or in Europe) and the way of traveling there. Countries, properties and the seasons represent the nodes of the semantic networks. The relations between those nodes are links, the second element that characterizes a semantic network.

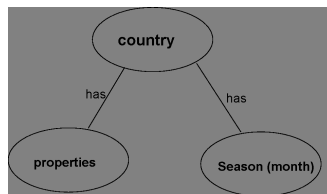


Figure 1: semantic network

By defining only few entries that refer to a bigger list of properties, the knowledge base did not get too big. I first tried other implementations of the Travel Guide, e.g. implementing all the different properties with rules, which led to a huge database although I had just a few countries in it for testing.

For the implementation with frames some more definitions would have been needed, e.g. cities or hotels that are in the countries. Adding more features would have made the program more authentic and more complex, but for the program's goals, to implement a Travel Guide that recommends different countries, it would have been too complex.

By implementing the program with semantic networks the program code will be easy to understand and new countries or properties could be added very easily, but is partially limited in adding new rules. They have to be not only added but implemented in the 5 main routines. The disadvantages with the implementation how it is at the moment is that there could not be drawn a real semantic network. It is more like a database that stores the entries 'land'(which stands for country) than representing a complex network structure. Though for the aims it is the best solution that allows a wide range of queries and rule (or 'method') definitions.

## 4 The implementation

I expected the implementation of the program to be easier than it really was. There were two main tasks, and three additional ones, the program should be able to do:

- First the program should print out a list of all available countries in the database. The implementation of this part was not really difficult. The problem was solved by using lists. The program looks with the built-in predicate `findall` for all countries that are defined in `land(Name, Properties, Season, Region, Goby)`. The predicate writes all elements that satisfy the query in a list. The bigger problem was to print out the list so that no duplicates are shown. The method `noRepetition(List1, List2)` avoid repetition in a list. It takes the first element (head) of a list (List1) and checks if this element is an element of the tail of the list. If it is tail of the list it would be ignored, otherwise the method writes this element in another list (List2). It works recursive, and does this check as long as there are no more elements to check in List1.

The method then print out a list of all available countries in the database. This function is used in some other methods to give the user an idea of out of which countries he may choose.

- The second choice of the menu is to choose a country with special properties. The user is asked questions about his preferences: First the region to which the user wants to travel, the way of traveling, the month in which he wants to travel (as a number between 1 and 12), and finally he should give a list of his preferred activities in holidays. For all those required values the user get proposals and has to choose one of the list.

The program reads in all the values. It searches the database for a list of countries (using the built-in predicate `findall`), later on calls `ListLand`. The program checks with this list which country definitions the desired properties contain. To do so the method `readlist(ListLand, Prop, LandList)` is called. The Prop-list is thereby the list of the desired properties. `LandList` is the list that would be returned. `readlist` does the following procedure: It takes the first element (the head) of `ListLand`, that means the first country. It searches for the list of properties that is defined in the knowledge base (`PropList`). This list contains a some properties which are not ordered in any way. The list of desired properties is also not ordered. Thats why two other methods are called, `permutation` and `sublist`. `permutation` changes the order of the `PropList` as often as there are possibilities. Every time a new ordered `PropList` is created, the method `sublist`

checks, if the Prop-list is a sublist of it. If this method succeeds, it will add the country to the LandList. If it fails, the procedures go on working until readlist is an empty list.

readlist returns then the LandList that contains all countries with the by the user entered properties. This list is then the final output of the second choice.

- The third choice uses the methods described before to print out a list of all available countries. It reads the input of the user who is asked to type in one country of which he wants to get the properties. The query is done like in the first part of the second choice and results all defined properties of the country.
- Another main task of the program is to find a country that has similar properties than another one, but that can have some other definitions. An example for that query is the following:

The user is asked for his preferred country. Assuming the user wants a country that is similar to Canada, but in the region europe. The program has to find then all countries in europe that have the properties skiing, snow and fun and the season winter. With this values the program can do the search like described above.

- The fifth task is a method that is already used above, but this time as a single query. The user enters a country and a list of properties and the program should check if the property list of the country contains the list of entered properties or if not. Like already detailed the query is executed with `permutation(List1,P)` and `sublist(List2,P)`.

Main problems in the implementation were the recursion of some methods, the returning of variables and especially reading out lists and the comparison of two different lists with the check of a sublist.

Unfortunately the user could not input single elements when a list of input is required. I wanted to implement this with a getList method, but this method did not work well in my program. I tested it several times in other programs and it worked, but together with my program it produced a list of errors, so I decided to leave it out and ask the user to enter a list.

## 4.1 The user interface

The user interface is designed to help the user, give him hints and guide him through the program. To start the program the user has to type initialise. . This will print a banner and then the programs' menu. The user has the choice between menu items a to e. These items will call one of the methods described above in *Implementation*. For example, menu item a is to get a list of all available countries. The program is so structured, that all data input queries are well described so that the user will know at every time what he is expected to do and what the program is going to do.

The following example will prove this:

The user enters the menu choice d. The user will now be asked by the program to choose a country he likes. Which country do you like? Choose one from the list! The program shows a list of all available countries. Assuming the user enters canada.. He gets now some details about canada: in which region it is and how to travel to canada, and he is asked to choose the region he wants to travel to.

This country is in region\_faraway and you can travel there by plane.

Please choose if you want to travel to a country with the same region or with another:

```
region_europe,
region_faraway?
```

Further on assuming the user wants to get a country similar to canada, but in europe. He enters then region\_europe and get the following output to retrace what the program is doing next:

```
The Travel Guide will search for countries with the properties of canada
and the region region_europe
and does not care about the way of traveling..
```

The final output is emphasised with a line of stars.

```
*****
```

The following countries will fulfil those requirements:

```
::- austria
::- norway
::- switzerland
```

```
*****
```

At the end of every query the program asks the user to type any word to proceed, which will clear the screen and go back to the main menu.



Type any word to proceed.

The menu contains another choice: The help menu. Typing help. will show a new menu: The help menu. Here the user can try to input data correctly or get information about the program. Choice 1 for example will ask the user to enter a word and give back as a result the entered word. The fourth choice in the help menu will jump back to the main menu.

To finish the whole program the user has to be in the main menu and type f, which will end the complete program.

To avoid wrong user inputs, the program has a loop implemented which will check if the entry is correct and, e.g. in the menu advise the user that he should only input one of the given possibilities and loop back to the menu choices.

```
*** MENUE ***
```

Please remember to end every entry with an '.'! This is really important! Thank you.

```
::help:: Learn how to use this program correctly type in 'help' (don't forget the '.'!)
```

```
::a:: To get a list of all available countries type in 'a'
```

```
::b:: To choose a country with special properties type in 'b'
```

```
::c:: To get the properties of a special country type in 'c'
```

```
::d:: To find a land with similar properties than your preferred land type in 'd'
```

```
::e:: Try to find out if your preferred land has special properties type in 'e'
```

```
::f:: To end program type in 'f'
```

```
—: aa.
```

The entry was not correct, please try again

Please remember to end every entry with an '.'! This is really important! Thank you.

```
::help:: Learn how to use this program correctly type in 'help' (don't forget the '.'!)
```

```
::a:: To get a list of all available countries type in 'a'
```

```
::b:: To choose a country with special properties type in 'b'
```

```
::c:: To get the properties of a special country type in 'c'
```

```
::d:: To find a land with similar properties than your preferred land type in 'd'
```

```
::e:: Try to find out if your preferred land has special properties type in 'e'
```

```
::f:: To end program type in 'f'
```

```
—:
```

## 5 Appendix

### References

- [1] Russel, S., Norvig, P., 1995 *Artificial Intelligence - a modern approach*, New Jersey, Prentice Hall International Editions
- [2] Callear, D., 1994 *Prolog Programming for Students*, London, DP Publications Ltd
- [3] Bratko, I., 2001 *Prolog Programming for Artificial Intelligence*, England, Pearson Education Limited
- [4] Homepage SWI Prolog <http://www.swi-prolog.org>
- [5] Brna, P. *Prolog Programming, A First Course*, <http://www.cbl.leeds.ac.uk/paul/prologbook/>
- [6] *Expert Systems in Prolog*, <http://www.amzi.com/ExpertSystemsInProlog/>

## 5.1 Program Source Code

This is the complete listing of the program code.

```

/*Travel Guide*/

initialise:-cls,
show_banner,
menue.
show_banner:-
write('*****'),nl,
write('*                               *'),nl,
write('*           Welcome to Travel Guide           *'),nl,
write('*                               *'),nl,
write('*           programed by Christiane Schmidt           *'),nl,
write('*                               *'),nl,
write('*                               *'),nl,
write('*****'),nl,nl,
write('Welcome to Travel Guide. '), nl,
write('This program will recommend you the best holiday country. '),nl,
write('Enjoy it!'),nl,nl,nl.

/**MENUE CHOICE***/
menue:-write('*** MENUE ***'), nl, nl,
write('Plase remember to end every entry with an ''.''! This is really important!
Thank you. '),nl,nl,
write('::help:: Learn how to use this program correctly type in ''help''
(don''t forget the ''.''!)'),nl,
write('::a:: To get a list of all available countries type in ''a'' '), nl,
write('::b:: To choose a country with special properties type in ''b'' '), nl,
write('::c:: To get the properties of a special country type in ''c'' '), nl,
write('::d:: To find a country with similar properties than your
prefered country type in ''d'' '),nl,
write('::e:: Try to find out if your prefered country has special
properties type in ''e'' '),nl,

```

```

write('::f:: To end program type in ''f'' '), nl,
read(Choice), choice(Choice).

choice(a):-write('These are all countries saved in the database: '),nl,
write
('-----'),nl,
findall(Country, land(Country, _,_,_), CountryList),
noRepetition(CountryList, CountryList_new),
write_elt(CountryList_new),nl,fail.
choice(a):-next.

choice(b):-write('To which region do you want to tavel?:
region_europe,
region_faraway? '),
read(Reg),nl,
write('How do you want to travel: by plane or by car? type ''plane'' or ''car'' '),
read(Goby),nl,
write('In which month do you want to travel?:'),
read(Month),nl,
season(Season, Month),
write('          *The season in which you want to travel is: '),
write(Season),write('*'),nl,nl,
write('What activities do you want to do in your holidays? Write as list!
Choose from:
skiing, snow, fun, sightseeing, sun, diving, surfing, sport,
tennis, riding,
visiting the USA - type ''usa'',
lying on the beach - type''beach'',
seeing the fascinating_landscape - type ''fascinating_landscape'' '),
read(Prop),nl,
write
('*****'),nl,
write('''Travel Guide'' searches a country with the following properties:'),nl,nl,

```

```

write('Poperties: '),nl, write_elt(Prop), nl,
write('Region: '), write(Reg),nl,
write('Travel by: '), write(Goby),nl,
write('Season: '), write(Season),nl,
findall(Country, land(Country,_,Season,Reg,Goby), ListLand),
readlist(ListLand,Prop, LandList),
write
('-----'),nl,
write('The Team of ''Travel Guide'' recommends you the following countries: '), nl,nl,
write_elt(LandList),nl,nl,
write
('*****'),nl,next;
write
('*****'),nl,
write('The Team of ''Travel Guide'' could not find any country'), nl,nl,
write
('*****'),nl,next.

choice(c):-write('Here you get a list of all available countries: '),
findall(Country, land(Country, _,_,_,_), CountryList),
noRepetition(CountryList, CountryList_new),
write_elt(CountryList_new),nl,
write('The properties of which country do you like? '),
read(Country),
findall(Season, land(Country, Prop, Season, Reg, Goby), SeasonList),
noRepetition(SeasonList, SeasonList_norep),
findall(Goby, land(Country, Prop, Season, Reg, Goby), GobyList),
noRepetition(GobyList, GobyList_norep),
land(Country, Prop, _, Reg, _),nl,nl,nl,
write
('-----'),nl,nl,
write('The properties of '), write(Country), write(' are: '), nl,
write_elt(Prop), nl,

```

```

write('You can travel to this country in: '),write_hlt(SeasonList_norep), nl,
write('The country is in the following region: '),write(Reg),nl,
write('You can go there by: '),write_hlt(GobyList_norep), nl,nl,
write
('*****'),
nl,nl, nl, next.
choice(c):-write
('*****'),nl,
write('Sorry, this country is not in the database!
Please try again or choose another option. '),
write
('*****'),
nl,nl,next.

choice(d):-write
('_____'),nl,nl,
write('Which country do you like? Choose one from the list! '),nl,
findall(Country, land(Country, _,-,-), CountryList),
noRepetition(CountryList, CountryList_new),
write_elt(CountryList_new),nl,
read(Country),
findall(Goby, land(Country, Prop, Season, Reg, Goby), GobyList),
noRepetition(GobyList, GobyList_norep),
land(Country, Prop, Season, Reg, _),
write
('_____'),nl,
write('This country is in '), write(Reg),
write(' and you can travel there by '), write_hlt(GobyList_norep),nl,nl,
write('Plase choose if you want to travel to a country
with the same region or with another: '),nl,
write('
region_europe,
region_faraway? '),

```

```

read(Reg_new),nl,
write
('-----'),nl,nl,
write('The Travel Guide will search for countries with the properties of '),
write(Country),nl,
write(' and the region '),write(Reg_new),
nl,write(' and does not care about the way of traveling. '),nl,nl,
findall(X, land(X,Prop,Season,Reg_new,_),List_1),
noRepetition(List_1, List_1_new),
write
('*****'),nl,
write('The following countries will fullfil those requirements: '),nl,
write_elt_2(List_1_new),nl,
write
('*****'),
nl,nl,next;
write('Sorry, there are no entries available in the database for this
country. '),nl,nl,next.

choice(e):-write('Please type in the country: '),nl,
read(Land),
land(Land,List1,_,_,_),
write('Insert List of properties the country should have: (write the list in []) '),
read(List2),
permutation(List1,P),
sublist(List2,P),
nl,
write
('*****'),nl,
write('The country ''' ), write(Land), write('''' has these properties: '),nl,
write_hlt(List1),nl,
write
('*****'),

```

```
next;
nl,
write
('*****'),nl,
write('The ''Travel Guide'' could not find those properties for this country. '),nl,
write
('*****'),
next.

choice(f):-write('Good Bye!').

choice(help):-test.

choice(_):-write('The entry was not correct, please try again!') ,nl,  menue.

goon(_):-cls,
menue.

/* **HELP MENUE** */
test:-cls,
write('*****'),nl,
write('Welcome to the Help-Menue!'),nl,nl,
write('Here you can test how to enter the required data correctly. '),nl,nl,
write('First, choose what you want to do!'),nl,
write('::1:: Enter one word'),nl,
write('::2:: Enter a list'),nl,
write('::3:: Further information'),nl,
write('::4:: Go back to main menue '),nl,
read(Choice), help(Choice).

help(1):-write('To enter a word, type the word and end with a ''.'' '),nl,
write(' Do not enter more than one word! '), nl,
read(Item), nl,
```



```
write('-----'),nl,
write('Congratulations! You passed the first test!'),nl,
write('This is the word you have entered: '),
write(Item),nl,nl,nl,
next_1.

help(2):-write('To enter a list, type in the list in [] and end with ''.''''),nl,
write('Separate every list entry with a comma!'),nl,
read(List),nl,
write('-----'),nl,
write('Congratulations! You passed the second test!'),
write('The list you entered: '),nl,
write(List),nl,nl,
next_1.

help(3):-write
('-----'),nl,
write('No information available at the moment'),nl,nl,
next_1.

help(4):-write
('-----'),nl,
write('You leave the help menu now. '), nl,nl,
cls, menue.

help(_):-write
('-----'),nl,
write('The entry was not correct, please try again!') ,nl, nl,
test.

/****RULES****/
next:-nl,nl,nl,nl,nl,
```

```

write('-----'),nl,
write('Type any word to proceed. '),
read(Goon), goon(Goon).

noRepetition([], []).
noRepetition([H|T1],T2):-member(H, T1),
noRepetition(T1,T2).
noRepetition([H|T1],[H|T2]):-not(member(H,T1)),
noRepetition(T1,T2).

/*clears the screen*/
cls :- put(27), put("[", put("2"), put("J").

next_1:-nl,nl,nl,nl,nl,
write('-----'),nl,
write('Type any word to proceed. '),
read(Goon), goon_1(Goon).

/*clears screen and go back to menu*/
goon_1(_):-cls, test.

/*write user entries in a list*/
getlist([H|T]):-read(X),not(X=end), X=H, getlist(T).
getlist([]).

/*write all entries of a list*/
readlist([],_,_).
readlist([H|T], Prop, LandList):-H=Country,
land(Country, PropList, _, _, _),
permutation(PropList, PropList_perm),
sublist(Prop, PropList_perm),
add(Country, LandList, LandList2),
readlist(T, Prop, LandList2).

```

```
readlist([H|T], Prop, LandList):-H=Country,
land(Country, PropList, _, _, _),
permutation(PropList, PropList_perm),
not(sublist(Prop, PropList_perm)),
readlist(T, Prop, LandList).

write_elt([]).
write_elt([H|T]):-write(':- '), write(H), nl, write_elt(T).

write_elt_2([]):-nl,write('No entries available.').
write_elt_2([H|T]):-write(':- '), write(H), nl, write_elt(T).

write_hlt([]).
write_hlt([H|T]):-write(H), write(' , '), write_hlt(T).

/*check if a list is sublist of another*/
sublist(S,L):-conc(_,L2,L),
conc(S,_,L2).

/*append two lists*/
conc([],L,L).
conc([X|L1],L2,[X|L3]):-conc(L1,L2,L3).

/*permute list entries*/
permutation([],[]).
permutation([H|T],P):-permutation(T,T1),
insert(H,T1,P).

/*insert item in list*/
insert(X, List, BiggerList):-del(X, BiggerList, List).

/*delete item from list*/
```

```
del(X, [X|Tail], Tail).
del(X, [Y|Tail], [Y|Tail1]):-del(X, Tail, Tail1).

/*add element to list*/
testAdd(X,L):-member(X,L),
hans=L;
hans(X, [X|L]).

add(Element, List, List):- member(Element,List),!.
add(Element,List, [Element|List]).

add2end(X, [_|T], [_|NewT]):-add2end(X,T,NewT).
add2end(X, [], [X]).

add_front(L,E,NL) :- NL = node(E,L).

add_1(X, []):-add_1(X, [X|_]).

/*add_1(X, [X|_]).*/
add_2(X, [X|_]).
add_2(X, [_|_]):-add_2(X).

/**KNOWLEDGE BASE**/
/**Countries**/
land(austria, [skiing, snow, fun] , winter, region_europe, car).

land(australia, [beach, surfing, fascinating_landscape],
summer, region_faraway, plane).
land(australia, [beach, surfing, fascinating_landscape],
winter, region_faraway, plane).

land(california, [usa, tennis, beach], summer, region_faraway, plane).
land(california, [usa, tennis, beach], winter, region_faraway, plane).
```

```
land(canada, [skiing, snow, fun], winter, region_faraway, plane).
```

```
land(costaRica, [beach, sport, sun], summer, region_faraway, plane).
```

```
land(costaRica, [beach, sport, sun], winter, region_faraway, plane).
```

```
land(fijiIsland, [beach, surfing, fascinating_landscape],  
summer, region_faraway, plane).
```

```
land(france, [sightseeing, beach], summer, region_europe, car).
```

```
land(france, [sightseeing, beach], summer, region_europe, plane).
```

```
land(germany, [sightseeing], summer, region_europe, car).
```

```
land(germany, [sightseeing], winter, region_europe, car).
```

```
land(greece, [sightseeing, beach, sun], summer, region_europe, plane).
```

```
land(hawaii, [beach, sport, sun], summer, region_faraway, plane).
```

```
land(hawaii, [beach, sport, sun], winter, region_faraway, plane).
```

```
land(italy, [sightseeing, beach], summer, region_europe, plane).
```

```
land(maledivs, [diving, beach, sun], summer, region_faraway, plane).
```

```
land(maledivs, [diving, beach, sun], winter, region_faraway, plane).
```

```
land(netherlands, [sightseeing], summer, region_europe, car).
```

```
land(netherlands, [sightseeing], winter, region_europe, car).
```

```
land(newZealand, [diving, beach, sun], summer, region_faraway, plane).
```

```
land(newZealand, [diving, beach, sun], winter, region_faraway, plane).
```

```
land(norway, [skiing, snow, fun], winter, region_europe, plane).
```

```
land(philipins, [diving, beach, sun], summer, region_faraway, plane).
```

```
land(philipins, [diving, beach, sun], winter, region_faraway, plane).
```

```
land(portugal,[riding, beach, fun], summer, region_europe, plane).
```

```
land(santaBarbara, [usa, tennis, beach], summer, region_faraway, plane).
```

```
land(santaBarbara, [usa, tennis, beach], winter, region_faraway, plane).
```

```
land(scotland, [sightseeing], summer, region_europe, car).
```

```
land(southAfrica, [sightseeing, beach, fascinating_landscape],  
summer, region_faraway, plane).
```

```
land(southAfrica, [sightseeing, beach, fascinating_landscape],  
winter, region_faraway, plane).
```

```
land(spain, [sightseeing, beach, sun], summer, region_europe, plane).
```

```
land(switzerland, [skiing, snow, fun], winter, region_europe, car).
```

```
land(switzerland, [skiing, snow, fun], winter, region_europe, plane).
```

```
land(texas, [usa, riding, fascinating_landscape],  
summer, region_faraway, plane).
```

```
land(texas, [usa, riding, fascinating_landscape],  
winter, region_faraway, plane).
```

```
land(thailand, [sightseeing, beach, fascinating_landscape],  
summer, region_faraway, plane).
```

```
land(thailand, [sightseeing, beach, fascinating_landscape],  
winter, region_faraway, plane).
```

```
land(turkey, [beach, sport, sun], summer, region_europe, plane).
```

```
land(turkey, [beach, sport, sun], winter, region_europe, plane).
```

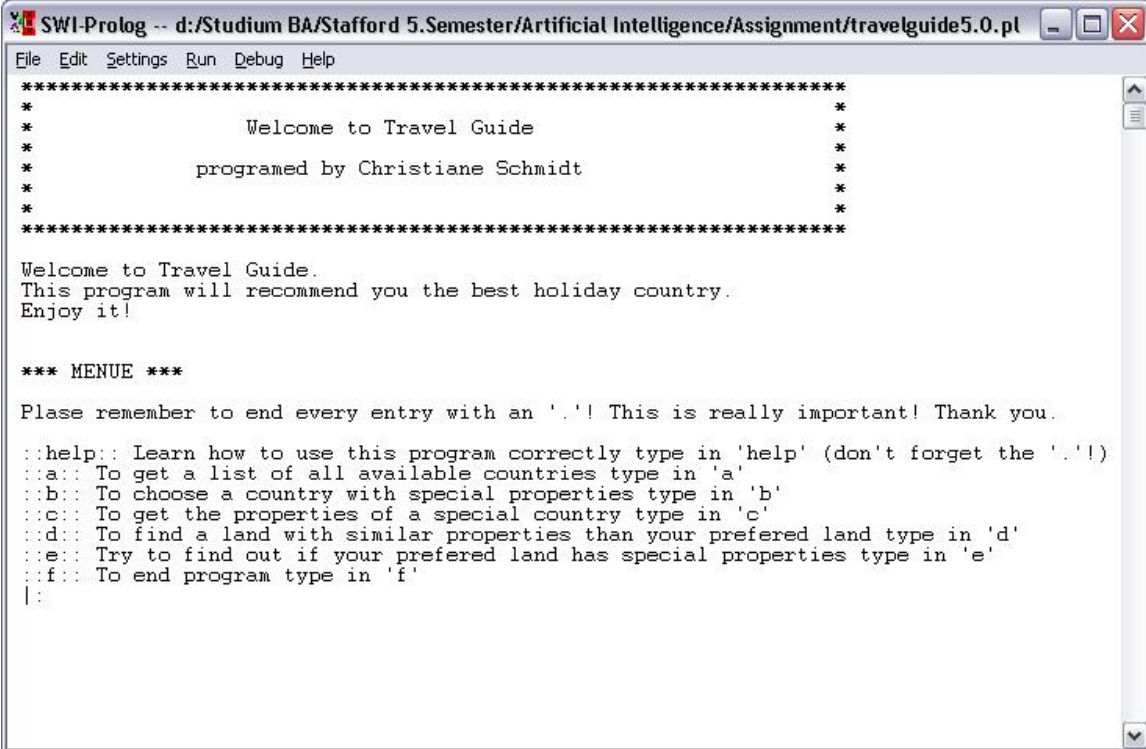
```
land(UnitedArabEmirates, [sightseeing, fascinating_landscape],
summer, region_faraway, plane).
land(UnitedArabEmirates, [sightseeing, fascinating_landscape],
winter, region_faraway, plane).

land(washington, [usa, sightseeing], summer, region_faraway, plane).

/**Seasons**/
season(winter, 1).
season(winter, 2).
season(winter, 3).
season(summer, 4).
season(summer, 5).
season(summer, 6).
season(summer, 7).
season(summer, 8).
season(summer, 9).
season(winter, 10).
season(winter, 11).
season(winter, 12).
```

## 5.2 Testruns

The main menu of the program (user interface after initialising):



```
SWI-Prolog -- d:/Studium BA/Stafford 5.Semester/Artificial Intelligence/Assignment/travelguide5.0.pl
File Edit Settings Run Debug Help
*****
*
*           Welcome to Travel Guide           *
*
*           programed by Christiane Schmidt   *
*
*
*
*****

Welcome to Travel Guide.
This program will recommend you the best holiday country.
Enjoy it!

*** MENUE ***

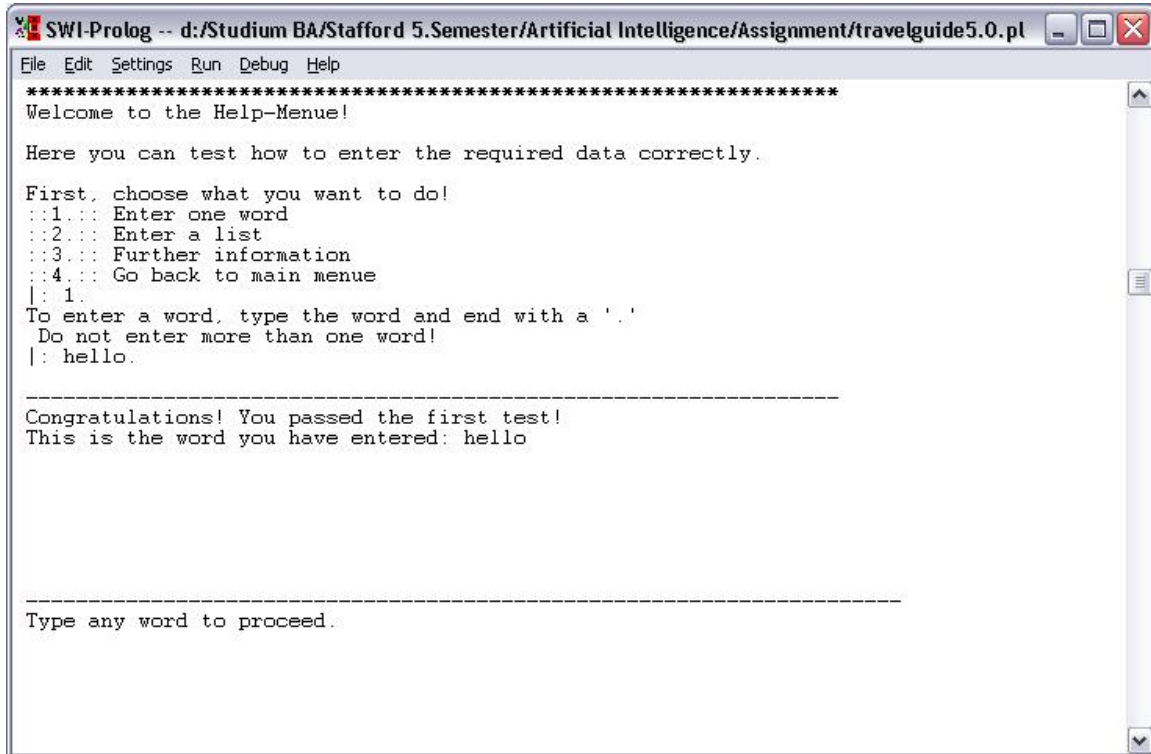
Plase remember to end every entry with an '.'! This is really important! Thank you.

::help:: Learn how to use this program correctly type in 'help' (don't forget the '.')
::a:: To get a list of all available countries type in 'a'
::b:: To choose a country with special properties type in 'b'
::c:: To get the properties of a special country type in 'c'
::d:: To find a land with similar properties than your prefered land type in 'd'
::e:: Try to find out if your prefered land has special properties type in 'e'
::f:: To end program type in 'f'
|:
```

Figure 2: main menue



The help menu:



```
SWI-Prolog -- d:/Studium BA/Stafford 5.Semester/Artificial Intelligence/Assignment/travelguide5.0.pl
File Edit Settings Run Debug Help
*****
Welcome to the Help-Menu!

Here you can test how to enter the required data correctly.

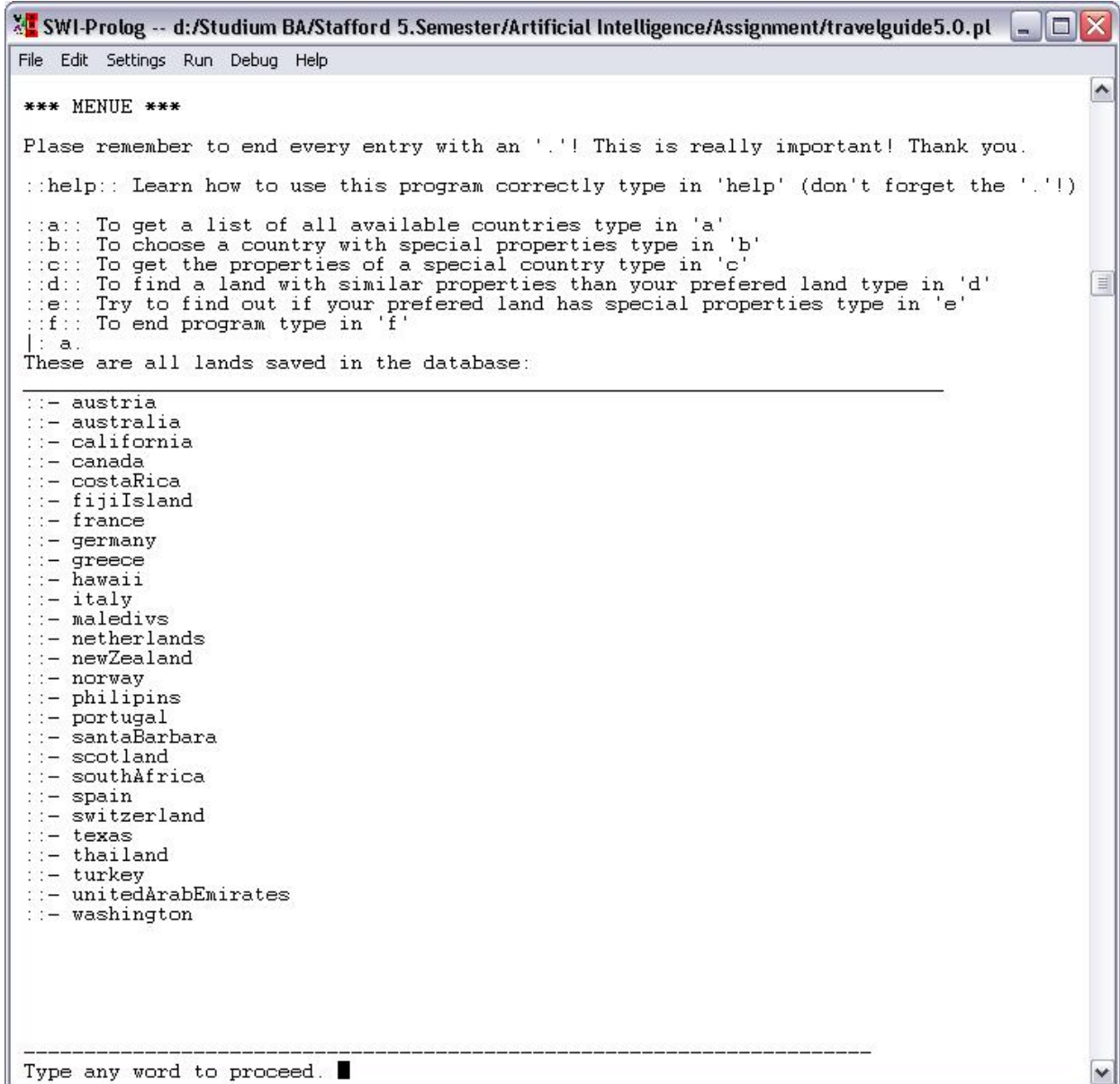
First, choose what you want to do!
::1:: Enter one word
::2:: Enter a list
::3:: Further information
::4:: Go back to main menu
|: 1.
To enter a word, type the word and end with a '.'
Do not enter more than one word!
|: hello.

-----
Congratulations! You passed the first test!
This is the word you have entered: hello

-----
Type any word to proceed.
```

Figure 3: help menu

menu choice a, a list of all countries that are saved in the database:



```
SWI-Prolog -- d:/Studium BA/Stafford 5.Semester/Artificial Intelligence/Assignment/travelguide5.0.pl
File Edit Settings Run Debug Help

*** MENUE ***

Please remember to end every entry with an '!' This is really important! Thank you.

::help:: Learn how to use this program correctly type in 'help' (don't forget the '!')

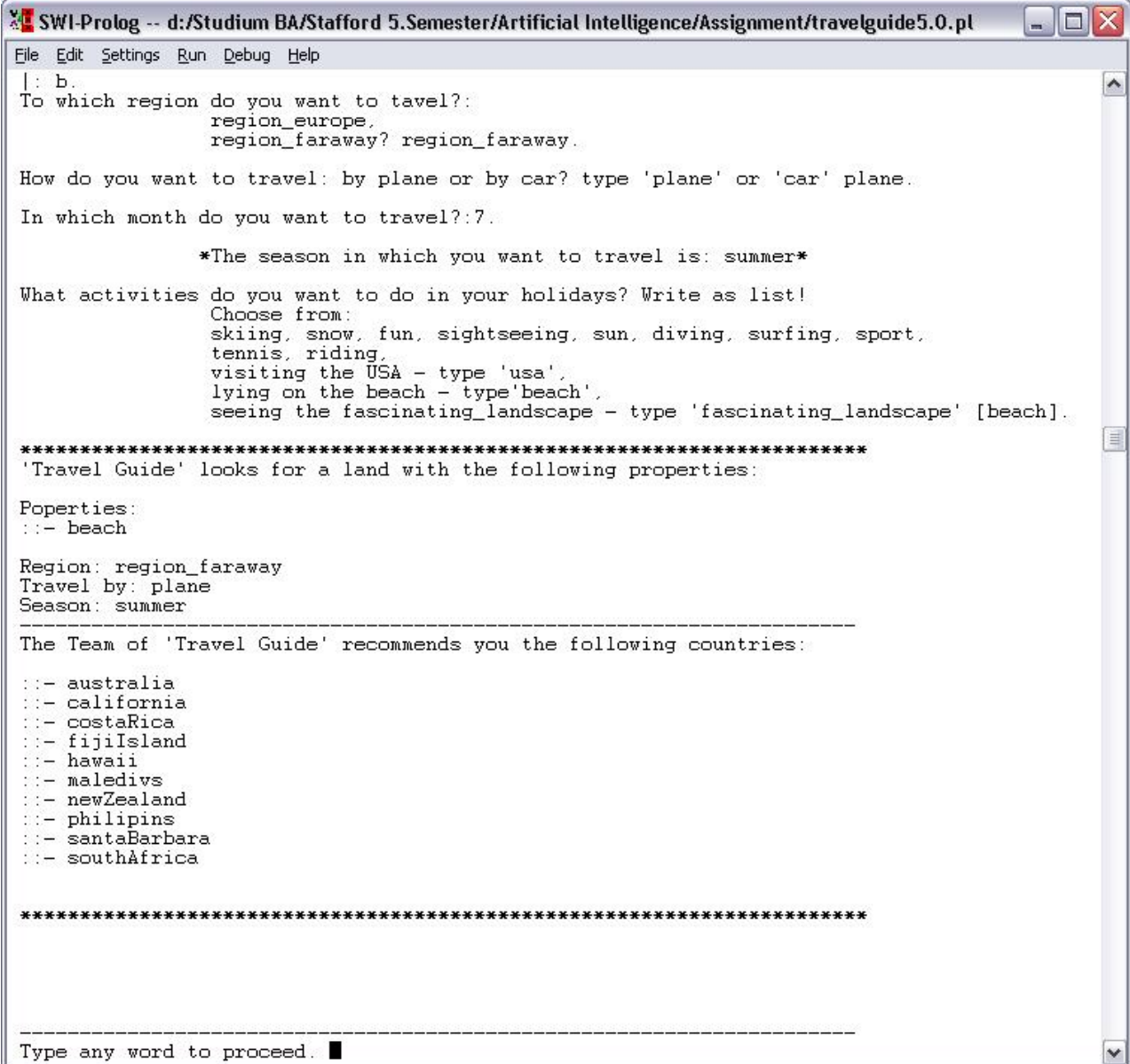
::a:: To get a list of all available countries type in 'a'
::b:: To choose a country with special properties type in 'b'
::c:: To get the properties of a special country type in 'c'
::d:: To find a land with similar properties than your preferred land type in 'd'
::e:: Try to find out if your preferred land has special properties type in 'e'
::f:: To end program type in 'f'
|: a.
These are all lands saved in the database:

-----
::- austria
::- australia
::- california
::- canada
::- costaRica
::- fijiIsland
::- france
::- germany
::- greece
::- hawaii
::- italy
::- maledivs
::- netherlands
::- newZealand
::- norway
::- philipins
::- portugal
::- santaBarbara
::- scotland
::- southAfrica
::- spain
::- switzerland
::- texas
::- thailand
::- turkey
::- unitedArabEmirates
::- washington

-----
Type any word to proceed. █
```

Figure 4: menu choice a

menu choice b, all countries with the properties "region\_faraway, month=july, travel by plane and property beach":



```

SWI-Prolog -- d:/Studium BA/Stafford 5.Semester/Artificial Intelligence/Assignment/travelguide5.0.pl
File Edit Settings Run Debug Help
|: b.
To which region do you want to tavel?:
    region_europe,
    region_faraway? region_faraway.

How do you want to travel: by plane or by car? type 'plane' or 'car' plane.

In which month do you want to travel?:7.

    *The season in which you want to travel is: summer*

What activities do you want to do in your holidays? Write as list!
Choose from:
skiing, snow, fun, sightseeing, sun, diving, surfing, sport,
tennis, riding,
visiting the USA - type 'usa',
lying on the beach - type 'beach',
seeing the fascinating_landscape - type 'fascinating_landscape' [beach].

*****
'Travel Guide' looks for a land with the following properties:

Properties:
::- beach

Region: region_faraway
Travel by: plane
Season: summer

-----
The Team of 'Travel Guide' recommends you the following countries:

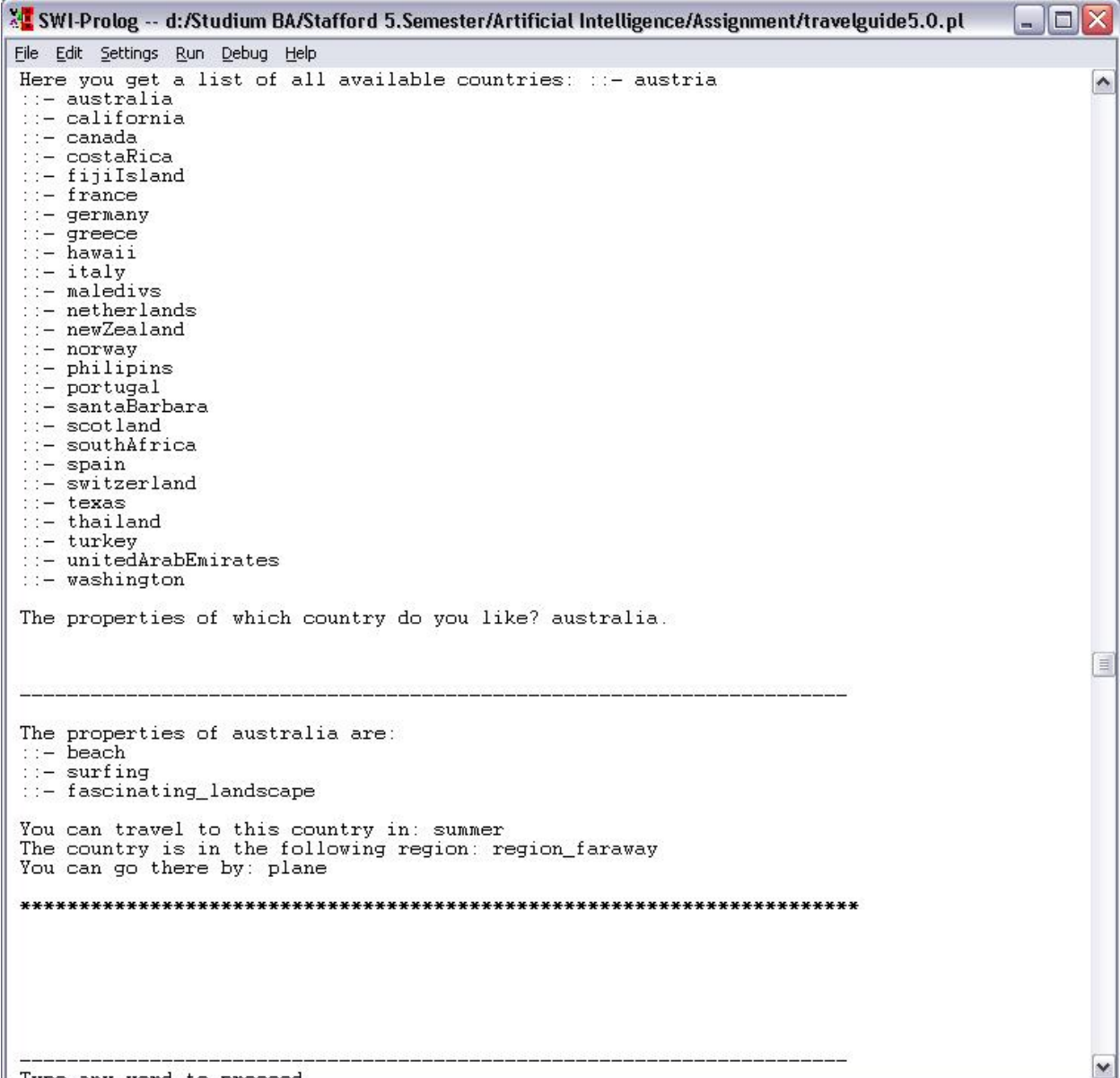
::- australia
::- california
::- costaRica
::- fijiIsland
::- hawaii
::- maledivs
::- newZealand
::- philipins
::- santaBarbara
::- southAfrica

*****

-----
Type any word to proceed. █
  
```

Figure 5: menu choice b

menu choice c, a list of all properties of australia:



```
SWI-Prolog -- d:/Studium BA/Stafford 5.Semester/Artificial Intelligence/Assignment/travelguide5.0.pl
File Edit Settings Run Debug Help
Here you get a list of all available countries: :- australia
::- australia
::- california
::- canada
::- costaRica
::- fijiIsland
::- france
::- germany
::- greece
::- hawaii
::- italy
::- maledivs
::- netherlands
::- newZealand
::- norway
::- philipins
::- portugal
::- santaBarbara
::- scotland
::- southAfrica
::- spain
::- switzerland
::- texas
::- thailand
::- turkey
::- unitedArabEmirates
::- washington

The properties of which country do you like? australia.

-----

The properties of australia are:
::- beach
::- surfing
::- fascinating_landscape

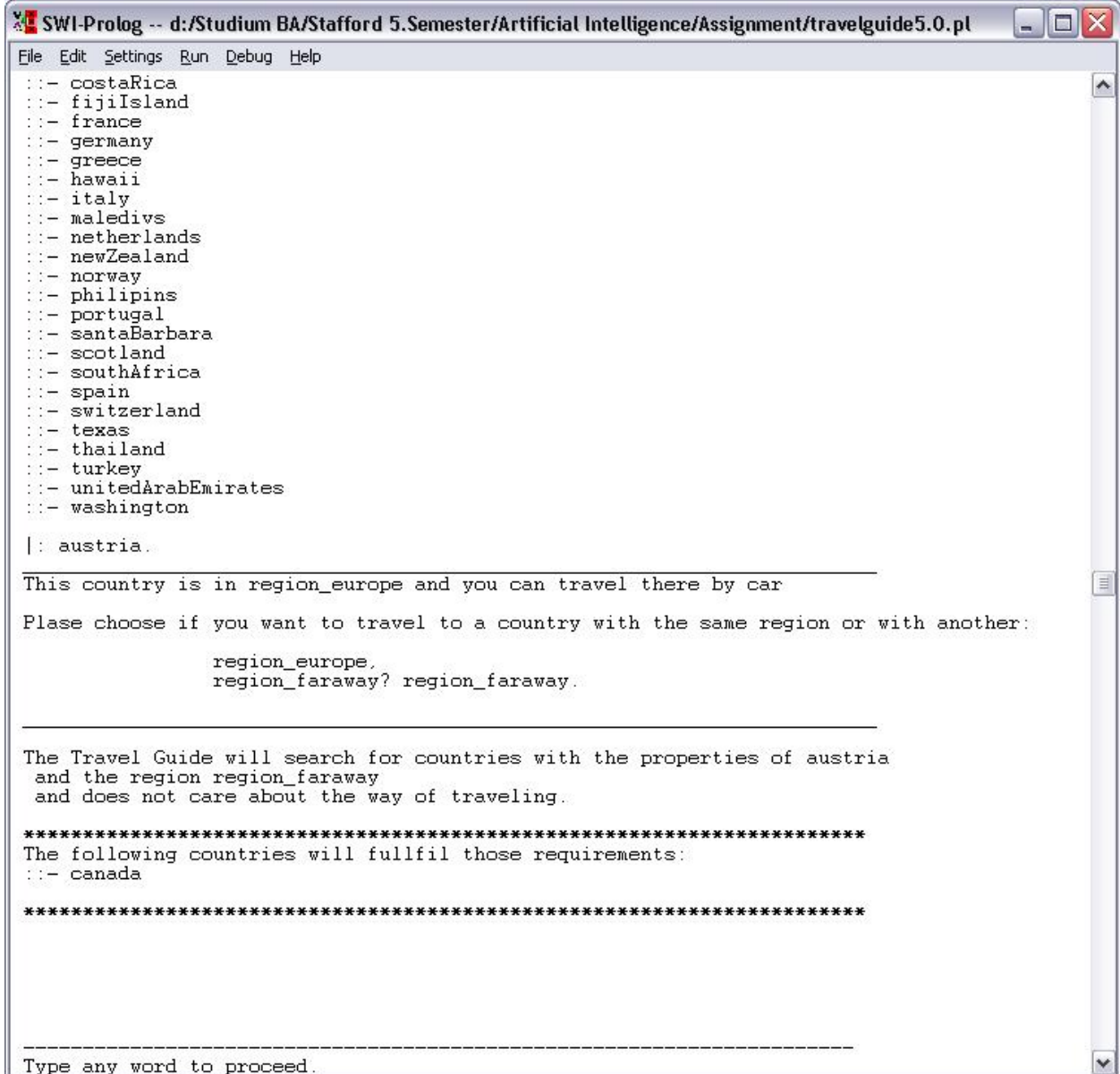
You can travel to this country in: summer
The country is in the following region: region_faraway
You can go there by: plane

*****

-----
Type any word to proceed.
```

Figure 6: menu choice c

menu choice d, a list of all countries that are similar to austria but not in europe:



```

SWI-Prolog -- d:/Studium BA/Stafford 5.Semester/Artificial Intelligence/Assignment/travelguide5.0.pl
File Edit Settings Run Debug Help
::- costaRica
::- fijiIsland
::- france
::- germany
::- greece
::- hawaii
::- italy
::- maledivs
::- netherlands
::- newZealand
::- norway
::- philipins
::- portugal
::- santaBarbara
::- scotland
::- southAfrica
::- spain
::- switzerland
::- texas
::- thailand
::- turkey
::- unitedArabEmirates
::- washington

|: austria.

-----
This country is in region_europe and you can travel there by car
Plase choose if you want to travel to a country with the same region or with another:

    region_europe,
    region_faraway? region_faraway.

-----

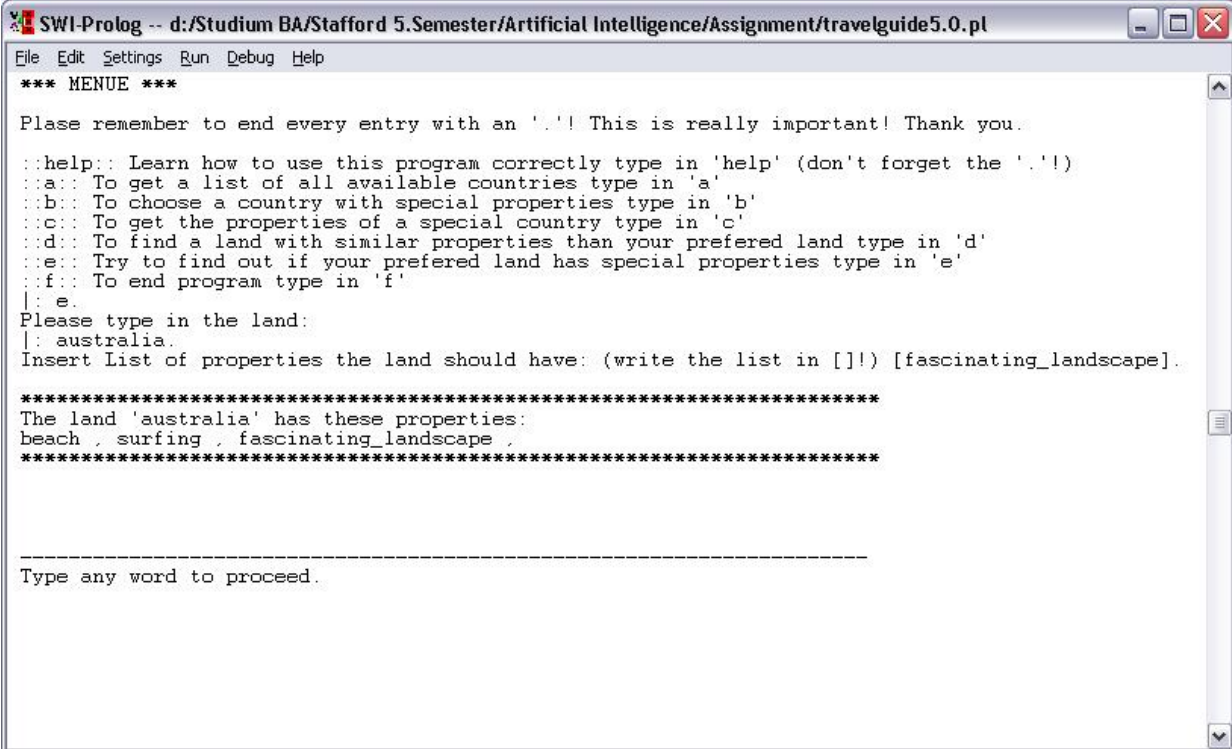
The Travel Guide will search for countries with the properties of austria
and the region region_faraway
and does not care about the way of traveling.

*****
The following countries will fullfil those requirements:
::- canada
*****

-----
Type any word to proceed.
  
```

Figure 7: menu choice d

menu choice e, checking if australia has the property fascinating\_landscape:



```
SWI-Prolog -- d:/Studium BA/Stafford 5.Semester/Artificial Intelligence/Assignment/travelguide5.0.pl
File Edit Settings Run Debug Help
*** MENUE ***

Please remember to end every entry with an '.'! This is really important! Thank you.

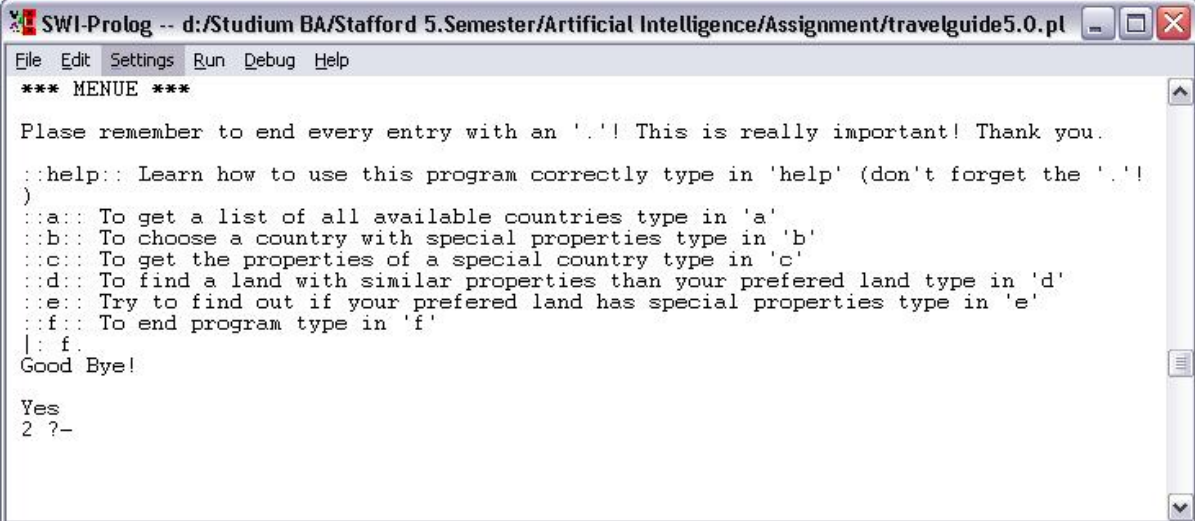
::help:: Learn how to use this program correctly type in 'help' (don't forget the '.')
::a:: To get a list of all available countries type in 'a'
::b:: To choose a country with special properties type in 'b'
::c:: To get the properties of a special country type in 'c'
::d:: To find a land with similar properties than your preferred land type in 'd'
::e:: Try to find out if your preferred land has special properties type in 'e'
::f:: To end program type in 'f'
|: e.
Please type in the land:
|: australia.
Insert List of properties the land should have: (write the list in []) [fascinating_landscape].

*****
The land 'australia' has these properties:
beach , surfing , fascinating_landscape ,
*****

-----
Type any word to proceed.
```

Figure 8: menu choice e

menu choice f, ending the program:



```
SWI-Prolog -- d:/Studium BA/Stafford 5.Semester/Artificial Intelligence/Assignment/travelguide5.0.pl
File Edit Settings Run Debug Help
*** MENUE ***

Plase remember to end every entry with an '.'! This is really important! Thank you.

::help:: Learn how to use this program correctly type in 'help' (don't forget the '.'!
)
::a:: To get a list of all available countries type in 'a'
::b:: To choose a country with special properties type in 'b'
::c:: To get the properties of a special country type in 'c'
::d:: To find a land with similar properties than your prefered land type in 'd'
::e:: Try to find out if your prefered land has special properties type in 'e'
::f:: To end program type in 'f'
|: f.
Good Bye!

Yes
2 ?-
```

Figure 9: end program