**INSTALLING PROLOG ON YOUR COMPUTER**

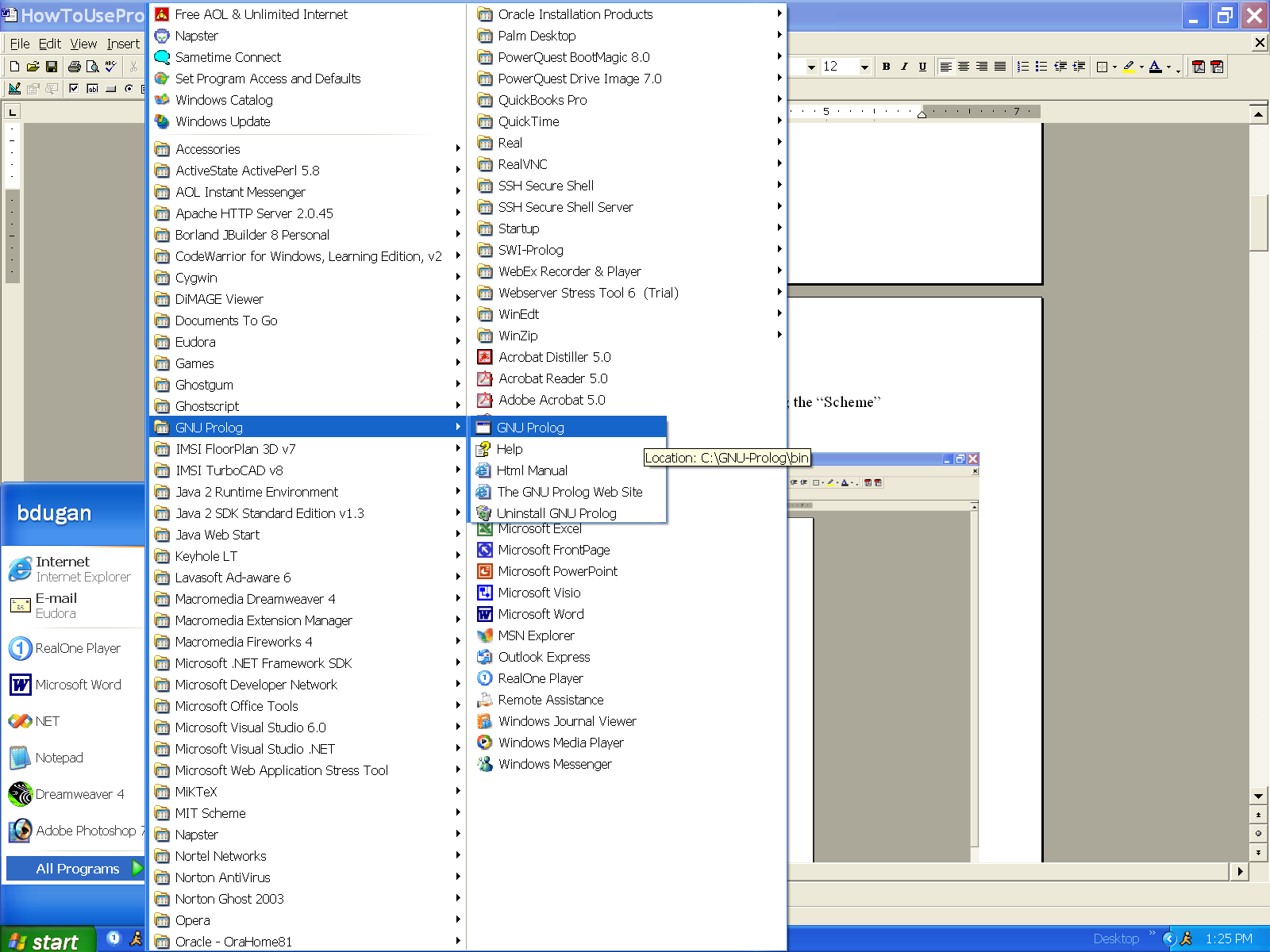
Locate the installation program here:

<http://gprolog.org/#download>

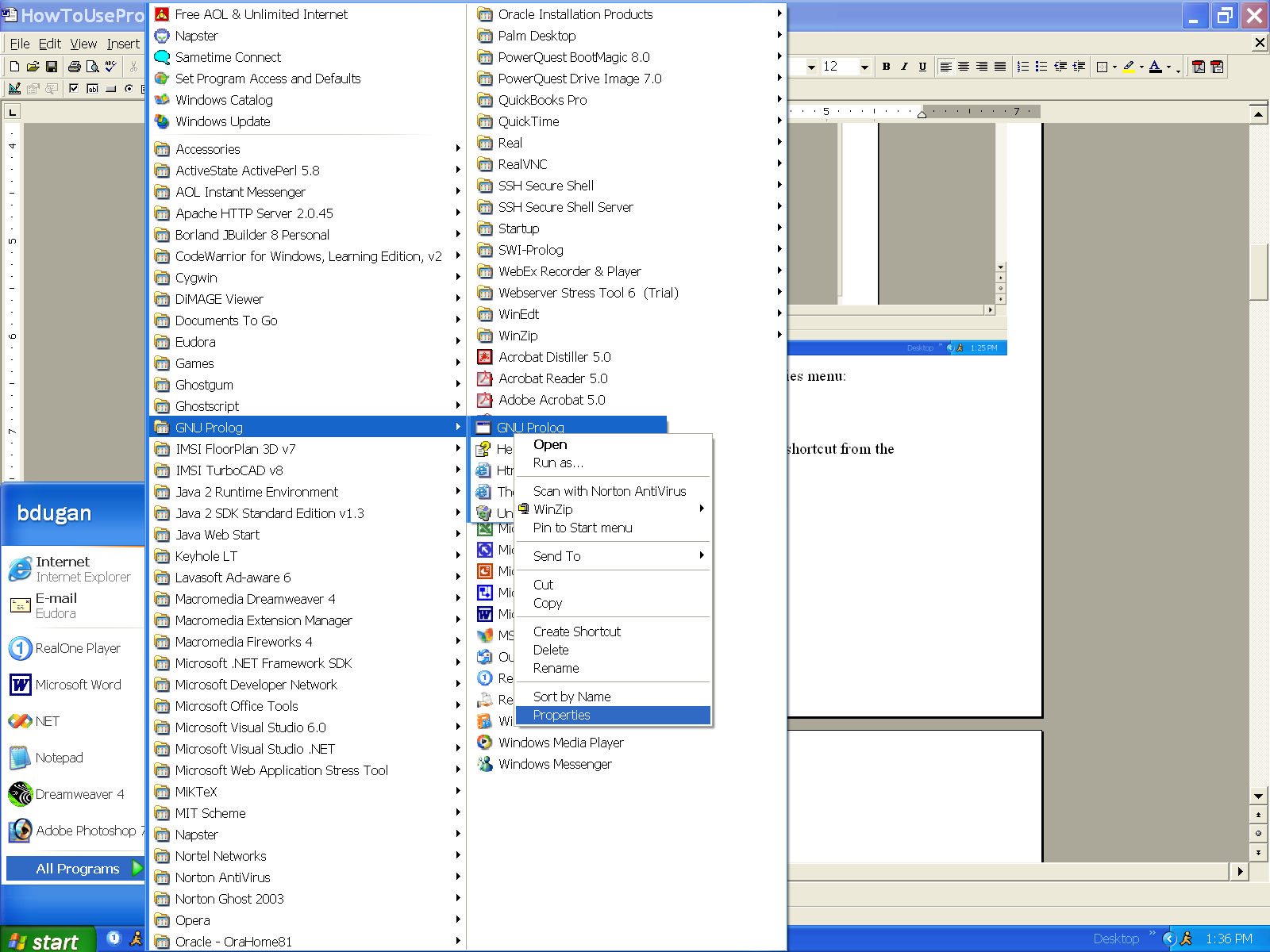
The Prolog documentation here:

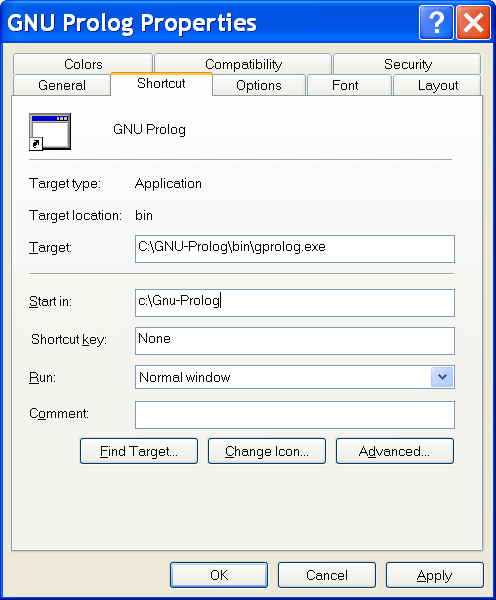
<http://gprolog.org/#manual>

Install the program, and go to the prolog startup shortcut in the startup menu.



RIGHT-CLICK on the Gnu-Prolog shortcut to and select the properties menu:



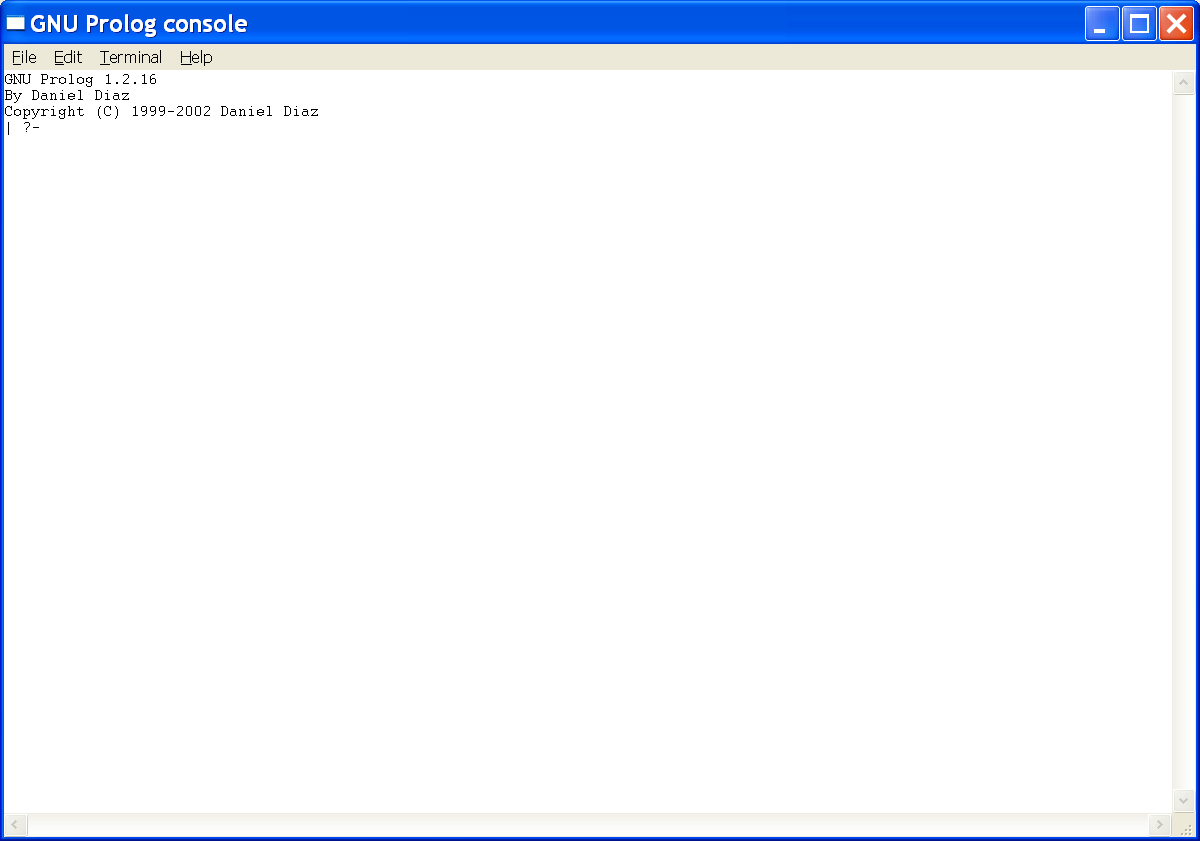


Change the “Start in:” property to “c:\gnu-prolog”. This will allow the prolog interpreter to load program files that you save in the directory “c:\gnu-prolog”.

Click the “APPLY” button, then click the “OK” button.

Use the start menu again to start the gnu-prolog interpreter.

Now you’ll see the Prolog interpreter waiting for your instructions:



To get anything to work in Prolog, you need to provide a basic set of rules, facts, or relations that you know to be true. Facts cannot be defined in the interpreter; they must be defined in a file that is LOADED into the interpreter.

To get the examples we’re going to talk about in class to work, you are going to need to load the facts located in this file:

<http://www.stonehill.edu/compsci/CS323/prolog/family.pl>

Save this file to the directory: “c:\gnu-prolog”.

Now inside of the prolog interpreter type the following:

[family].

You should see the following:

| ?- [family].

compiling c:\Gnu-Prolog\family.pl for byte code...

c:\Gnu-Prolog\family.pl:27 warning: singleton variables [Tail] for head\_basket/1

c:\Gnu-Prolog\family.pl:33 warning: singleton variables [Tail] for mem/2

c:\Gnu-Prolog\family.pl:34 warning: singleton variables [Head] for mem/2

c:\Gnu-Prolog\family.pl:36 warning: singleton variables [List2] for union/3

c:\Gnu-Prolog\family.pl:37 warning: singleton variables [List1] for union/3

c:\Gnu-Prolog\family.pl compiled, 44 lines read - 5897 bytes written, 20 ms

yes

Let’s try a hello world program. The family.pl file you just loaded consists of a list of FACTS that the prolog programs we discuss in class will use. Facts are stated like this:

<relationname>(value1, value2, value3, ... valueN).

For example:

hello(world).

If we want to know whether a fact is true or not, we use the prolog interpreter to create a query. For example, if we want to know if the fact hello(world) is true, type the following query:

| ?- hello(world).

The interpreter responds with:

yes

Indicating that this fact is true. Notice that in the family.pl file, hello(world). is considered to be a FACT, but when you type the same thing in the prolog interpreter, it is considered to be a QUERY.

If we type:

| ?- hello(stonehill).

The interpreter responds with:

no

Because this fact has not been defined.

We can get a little fancier, by allowing prolog to perform something called a unification query. This type of query asks prolog to find all values that cause a relation to evaluate to true.

For example:

hello(X).

states, “find all values of X such that hello(X). evaluates to true”. If you look at the family.pl file you’ll see the following facts about the hello relation:

hello(world).

hello(earth).

hello(mars).

When you type this query:

|?- hello(X)

You get these results:

Type “;” after each X = ...? to continue the unification process. If you press ENTER instead, the unification process will terminate.

X = world ? ;

X = earth ? ;

X = mars

yes

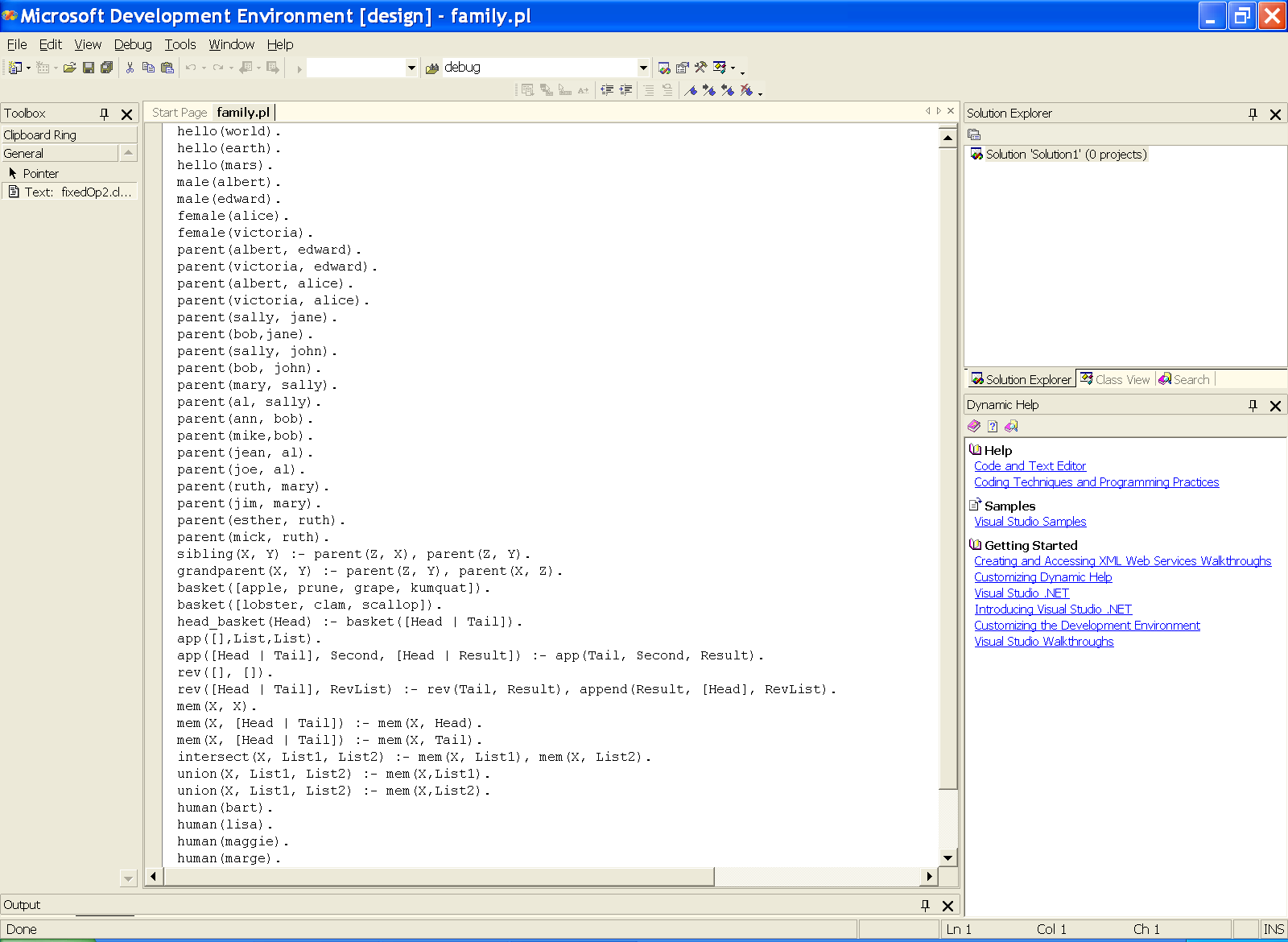
**USING AN EDITOR**

You use an editor to create prolog FACTS and GOALS.

You use the prolog interpreter to execute queries that use the facts and goals.

You can use any old editor. In my case, I just used Microsoft’s Visual Studio to open the file:

“c:\gnu-prolog\family.pl”



Notice that the beginning of the file contains the hello() relation facts we’ve been using to understand some prolog basics.

During the load process, the prolog interpreter is going to execute the code in your .pl file. Most of this code is just fact and goal definitions, so you’re not actually doing any work... just getting ready to do some work by typing a query in the prolog interpreter.

While executing these definitions, the interpreter may encounter syntax errors. For example, when I removed the last ‘)’ from the definition of the hello(mars fact I get this when I load the family.pl file:

| ?- [family].

compiling c:\Gnu-Prolog\family.pl for byte code...

c:\Gnu-Prolog\family.pl:3 error: syntax error: , or ) expected (char:11)

c:\Gnu-Prolog\family.pl:30 warning: singleton variables [Tail] for head\_basket/1

c:\Gnu-Prolog\family.pl:36 warning: singleton variables [Tail] for mem/2

c:\Gnu-Prolog\family.pl:37 warning: singleton variables [Head] for mem/2

c:\Gnu-Prolog\family.pl:39 warning: singleton variables [List2] for union/3

c:\Gnu-Prolog\family.pl:40 warning: singleton variables [List1] for union/3

1 error(s)

compilation failed

When this happens, you need to:

* look at the code in the XXX.pl file
* correct the problem
* save the XXX.pl file again
* RELOAD the .pl file in the prolog interpreter using the [XXX].

# OTHER USEFUL FUNCTIONS

Use:

trace.

To observe how prolog implements the resolution of a query. As you watch the traces of several queries you will develop an understanding how prolog uses a left to right order of evaluation of subgoals, depth first search, and backtracking for resolution. You’ll also see that the order you specify facts in your “.pl” file influences the resolution process.

For example, here’s a trace of the hello(X) query:

| ?- trace.

The debugger will first creep -- showing everything (trace)

(10 ms) yes

{trace}

| ?- hello(X).

1 1 Call: hello(\_16) ?

1 1 Exit: hello(world) ?

X = world ? ;

1 1 Redo: hello(world) ?

1 1 Exit: hello(earth) ?

X = earth ? ;

1 1 Redo: hello(earth) ?

1 1 Exit: hello(mars) ?

X = mars

(10 ms) yes

{trace}