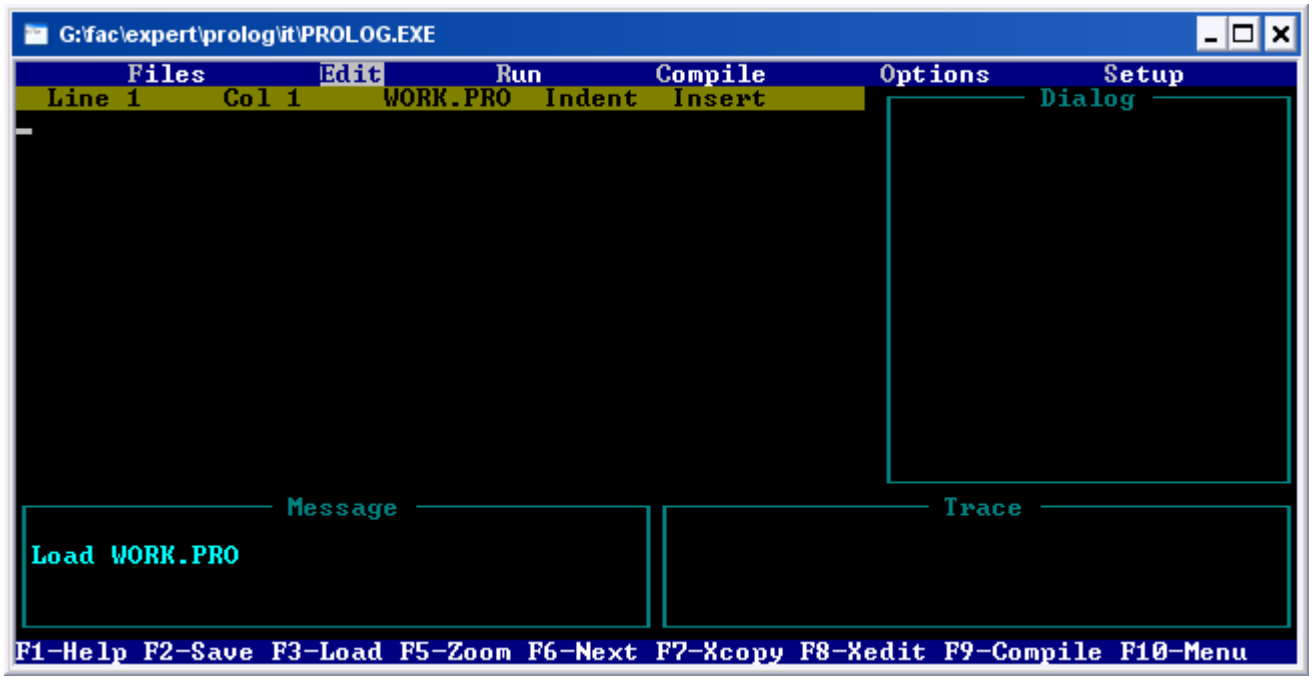
Thinking in Prolog imperatively !!

for c++/java/c# programmers

BonAppetite!©

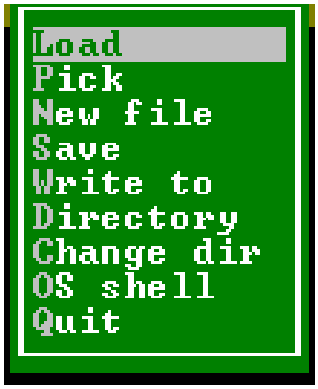
dr_dos_ok@yahoo.com

(0) user interface



No mouse !! Menus reached using Alt + first letter

File Menu



Load : open a certain file
Pick : choose from the last opened files
New file : clear the edit area.
Save : save the file, follows a "write to"
Write to : the first save, to write the file
Directory : open a certain directory
Change directory
Os shell : open the command prompt
Quit

Edit Menu
just send you to the edit area.

Run Menu
just send you to the run area. If there is an internal goal it is executed.

Compile Menu
generate .obj file then the .exe file.
there must be an internal goal.

Debugging
put the word **trace** at the first line of the program, then run (Alt + R), then enter a goal, trace by continue pressing F10. Keep your eye on *Edit, Message, Trace* areas.

(1) prolog program sections

1. Clauses section

clauses = facts and rules

Fact :

a relation name followed by objects enclosed in parentheses. The facts ends with a period (.)

relation	Ayman likes Saly	likes(ayman , saly)
property	Kermit is geen	green (kermit)

Rule :

Prolog rule has two parts Head and Body

```
Head : <subgoal1>,<subgoal2> , … , <subgoalN>
```

2 Predicate Section

If you define your own predicate in the clauses section of prolog program, you must declare it in a predicates section.

```
predicate_name ( arg_typel , arg_type2, … , arg_typeN)
```

You can have two predicates with the same name but different arity (number of arguments that a predicate takes).

3. Domains Section

☺ giving meaningful names to existing domains

☺ declare data structures. More about this later.

Built-in domains

char, real , string, symbol, integer

4. Goal Section

☺ goal is same as the **body of a Rule** but not followed by :-

☺ goal is similar to main() function in c++ programs

☺ Two types of goal are

External goal , which gives the ability to change goals

Internal goal , we can compile to a .exe file

5. Database Section

like predicates section but enables to update, remove, or add the facts at runtime.

6.Constants section

```
hundred  = (10*10)
pi       = 3.141592653
```

(2) variables

The first character of the name must be an upper case letter or an underscore after which any number of letters , digits or underscore can be used .

Anonymous Variables

☺ Anonymous variables used to ignore the values you don't need. The anonymous variable matches anything.

☺ It is represented by alone underscore _

☺ owns(_ , shoes) % Everyone owns shoes

(3) Compound data

```
class Date
{
    String month;
    int day;
    int year;

    Date(String m, int d, int y)
    {
        this.month = m;
        this.day = d;
        this.year = y;
    }
}
void Main ()
{
    Date x = new Date("oct", 3, 2007);
}
```

```
domains
    date_type = date(string, integer, integer)
goal
    X = date("oct", 3, 2007)
```

☺ date_type is the date type name

☺ date is called the **functor**, and its effect is similar to the constructor.

(4) comments

```
/*  Multiple
    line
    comment */

% single line comment
```

(5) Input/Output [from the system/user]

Writing

String line= "abc"; **int** i = 2; **float** r= 4.6; cout << line << 2 << r << endl;
Line = "abc", I = 2, R = 4.6, write (Line, I, R), nl

nl for new line, it generates a new line to display screen.

Reading

String line; cin >> line; **int** i; cin >> i; **float** r; cin >> r; **char** c; cin >> c;
readln (Line), readint(I), readreal (R), readchar (C), readterm(date_type, Term), file_str("C:\\1.TXT", Text)

(6) IF

if (*condition*) *if_body*; **else** *else_body*;

predicates
if(t1,t2, ..., tn)
clauses
if(V1,V2, ..., Vn) :-
condition, **!**,
if_body .
if(V1,V2, ..., Vn) :-
else_body .

Where

- ☺ V1,V2,..., Vn are variables appearing in *condition* .
- ☺ t1,t2, ..., tn are data types of V1,V2, ..., Vn
- ☺ Sentences of *if_body* and *else_body* will be separated by commas not semicolons.

Example : Minimum

input: two numbers *x1, x2*

output : the minimum of them

1	**#include** <iostream.h>	**predicates**
2	**void** Main()	if_minimum(integer, integer)
3	{	**clauses**
4	**int** x1, x2;	if_minimum(X1, X2) :-
5	cin >> x1;	X1 < X2, !,
6	cin >> x2;	write (X1).
7		if_minimum(_, X2) :-
8	**if** (x1 < x2)	write (X2).
9	cout << x1;	**goal**
10	**else**	readint(Y1),
11	cout << x2;	readint(Y2),
12	}	if_minimum(Y1, Y2)

Note

☺ In prolog code, line 7, we replace X1 with _ Generally, if a variable in the head *f(V1,..., Vn)* does not appear in the body , it is preferred to be replaced with _ otherwise a warning arises.

(7) SWITCH

```
switch (var)
{
    case v1 :   case_v1_body;   break;
    case v2 :   case_v2_body;   break;
    ⋮
    case vn :   case_vn_body;   break;
    default:    default_body;   break;
}
```

```
predicates
    case(t_var)
clauses
    case(v1) :-  case_v1_body,   !.
    case(v2) :-  case_v2_body,   !.
    ⋮
    case(vn) :-  case_vn_body,   !.
    case(_) :-   default_body,   !.
```

Where

☺ v1,v2,..., vn are possible values for *var* variable .

☺ t_var is data type of *var* variable.

☺ Sentences of *case_v1_body, case_v2_body,..., case_vn_body*, and *default_body* are separated by commas.

Example : Days

input: a number *n*

output : the day name

```
#include <iostream.h>
void Main()
{
    int n;
    cin >> n;

    switch (n)
    {
        case 1 :    cout << "Saturday";  break;
        case 2 :    cout << "Sunday";    break;
        case 3 :    cout << "Monday";    break;
        case 4 :    cout << "Tuesday";   break;
        case 5 :    cout << "Wednesday"; break;
        case 6 :    cout << "Thursday";  break;
        case 7 :    cout << "Friday";    break;
        default:    cout << "Bogus !!";  break;
    }
}
```

```
predicates
    case_days(integer)
clauses
    case_days(1) :-  write (saturday),    !.
    case_days(2) :-  write (sunday),      !.
    case_days(3) :-  write (monday),      !.
    case_days(4) :-  write (tuesday),     !.
    case_days(5) :-  write (wednesday),   !.
    case_days(6) :-  write (thursday),    !.
    case_days(7) :-  write (friday),      !.
    case_days(_) :-  write ("Bogus !!"),  !.
goal
    readint(N),
    case_days(N)
```

Note

- ☺ In prolog code, we output days without enclosing them in double quotes ,e.g. saturday rather than "Saturday". this is allowed since it is in the form of symbol data.
- ☺ ! is called the cut. Its effect is similar to **break** to prevent testing other cases when the current case is matched. However, it can be put anywhere not only the last sentence as the **break**. Usually it is put at the first , i.e. after :-

(8) FOR

```
for (i= initValue; condition, iteration)
    for_body;
```

```
predicates
   for(integer,integer)
clauses
    for(I, N) :-
        neg_condition,   !, .
    for(I, N) :-
        for_body ,
        iteration,
        for( NewI, N).
```

Where

- ☺ *neg_condition* is the negation of *condition* . while *condition* is a looping condition, *neg_condition* is stopping condition.
- ☺ *iteration* usually take the form i++ or i--. we cannot write I=I+1 in prolog , because this may be misunderstood as a condition test if I equals I+1. The correct form is NewI=I+1

Example : Counting

input: a number *n*
output : the numbers from 0 to n-1 , one per line

```
1  #include <iostream.h>
2  void Main()
3  {
4      int n;
5      cin >> n;
6
7      for(int i=0; i<n; i++)
8          cout << i << "\n";
9
10 }
11
12
```

```
predicates
   for_count(integer,integer)
clauses
    for_count(I, N) :-
        I = N, !.
    for_count(I, N) :-
        write (I, "\n"),
        NewI = I + 1,
        for_count(NewI, N).
goal
    readint(N),
    for_count(0, N)
```

Note

- ☺ In prolog code, line 12, we give the *initValue* for I, that is 0.
- ☺ In prolog code, line 5, the negation of i<n is I>=N . However, the condition I = N is enough .

(9) DO WHILE / REPEAT UNTIL

do **{** *do_body*; **}while** (condition);	**repeat** **{** *do_body*; **}until**(*neg_condition*);

```
predicates
        do
        while
clauses
        do.
        do :- do.

        while :- do,
                 do_body ,
                 neg_condition,    !, .
```

Where

- ☺ *neg_condition* is the negation of *condition* . while *condition* is a looping condition, *neg_condition* is stopping condition.
- ☺ *do* predicate is used to generate virtual alternatives (untried passes).

Example : Typing

idea: continue

```
1   #include <iostream.h>
2   void Main()
3   {
4       char c;
5
6       do
7       {
8           cin >> c;
9           cout << c;
10      }while( c != '\r')
11
12  }
13
```

```
predicates
        repeat
        typing
clauses
        repeat.
        repeat :- repeat.

        typing :- repeat,
                  readchar(C),
                  write(C) ,
                  C = '\r',   !.
goal
        typing
```

Note

☺ In prolog code, line 11, the negation of c != '\r' is C = '\r'.

(10) SQL select

students table

id	name
1	ali
2	ahmed
⋮	⋮

```
create table students (id int, name text);

insert into students(1, "ali");
insert into students(2, "ahmed");
⋮
```

```
predicates
    students(integer, symbol)
clauses
    students(1, ali).
    students(2, ahmed).
    ⋮
```

Now, let's try to execute a **select** query

```
select id, name from students;
```

```
predicates
    query1
clauses
    query1 :-
        students(Id, Name),
        write(Id, " ", Name, "\n"),
        fail.
    query1.
```

Note

- ☺ In prolog code, **fail** is used to tell prolog that the current solution is not good, and forces it to try other solutions .
- ☺ The last clause, i.e. query1, is put for satisfaction purpose . If you remove it, you will get the table printed then followed with "No solution".

(11) Arrays → List

```
void f ( int[] x)
{
    int[] a = {1,2,3};
}
```

```
domains
    my_list = integer*
predicates
    f(my_list)
clauses
    f(X) :-
        A= [1,2,3].
```

A list consists of two parts:
- **Head** : the first element,
- **Tail** : all the subsequent elements.

The empty list can't be broken into head and tail.
A list can be written as **[**Head **|** Tail**]**

List	Head	Tail
['a' , 'b' , 'c']	'a'	['b' , 'c']
['a']	'a'	[]
[]	Undefined	Undefined
[[1,2,3] , [2,3,4] , []]	[1,2,3]	[[2,3,4], []]

Unification of Lists

List 1	List 2	Variable Binding
[X, Y, Z]	[egg, eats, bob]	X=egg, Y=eats, Z=bob
[c]	[X|Y]	X=c , Y=[]
[1,2,3,4]	[X , Y| Z]	X=1 ,Y=2 , Z=[3,4]
[1,2]	[3 |X]	fail

Manipulating Lists
Arrays are manipulated using for loop, but lists are manipulated as follows :

"If the list becomes empty, Stop.
Else,
    manipulate the head (a single element),
    then manipulate the tail (a list)"

which is much similar to **foreach** in c#

```
    Type[] a = {...};
    foreach (Type x in a)
    {
        f(x);
    }
```

```
domains
    list = type*
predicates
    foreach(list)
clauses
    foreach(A) :- A=[], !.
    foreach([X|Rest]):-
        f(X),
        foreach(Rest).
```

Example : Writing out a list

```
1   #include <iostream.h>
2   void Main()
3   {
4       int[] a;
5       a = {1,2,3};
6
7       foreach(int x in a)
8           cout << x;
9   }
10
11
12
```

```
domains
    list = integer*
predicates
    foreach(list)
clauses
    foreach( [] ).
    foreach([X|Rest]):-
        write(X),
        foreach(Rest).
goal
    A= [1,2,3],
    foreach(A)
```

Note

☺ In prolog code, line 6, it should be written as foreach(A) :- A=[], !.

Generally, if a variable in the head appears once in the body giving it a value or testing it for a value, it is preferred to be replaced in the head.

Some most-used list operations

member

```
DOMAINS
    list = element*
    element = symbol
PREDICATES
    member(name, namelist)
CLAUSES
    member(Name, [Name|_] ).
    member(Name [_|Tail]) :-
        member (Name, Tail).
```

append

```
DOMAINS
    list = element*
    element = integer
PREDICATES
    append(list, list, list)
CLAUSES
    append( [] , List, List) .
    append( [H|L1] ,L2, [H|L3] ) :-
        append(Ll,L2,L3).
```

length_of

```
DOMAINS
    list = element*
    element = integer
PREDICATES
    length_of (list, integer)
    length_of (list, integer, integer)
CLAUSES
    length_of(List, Length):-
        length_of(List, Length, 0).

    length_of ( [] , N, N) .
    length_of( [_|T] , N, I) :-
        NewI = I + 1,
        length_of(T, N, NewI)  .
```