Today

- Quiz 2
- Prolog Wrap Up (Part 2)
- The “Little” Language

Assignments

- HW-9, R-9 due
- HW-10, R-10 out
Basic Prolog I/O

Output

- The predicate `write(+Term)` prints Term

```
?- write(hi).
hi
true.

?- write('Hello World!').
Hello World!
true.

?- write('Hello'), nl, write('World!').
Hello
World!
true.
```
• Can also use the `format(+Format, +Args)` predicate

```
?- format(’~w ~w’, [’Hello’, ’World!’]).
Hello World!
ture.

?- format(’~w’, [”Hello”]). % watch out for double quotes
true

?- format(’~s’, [”Hello”]).
Hello
ture.
```
Input

- The opposite of write is read(-Term)

```prolog
?- read(X).
|: hello. % |: is the default read prompt
X = hello.

?- read(X).
|: 'Hello World!'. % Must be quoted here
X = 'Hello World!'.
```

- Can set the prompt

```prolog
?- prompt(_, '> '). % first arg is old prompt
true.

?- read(X).
> hi.
X = hi.
```

- Can also just use write ...

```prolog
?- write('Please enter something: '), read(X).
Enter somthing: hi.
X = hi.
```
Prolog supports various other I/O operations as well

- Reading and writing files (open, write)
- Reading characters, numbers, etc. (e.g., get)

Example user selection menu

go :-
    menu_choice(X),
    do_choice(X), !. \% a cut to stop backtracking

menu_choice(X) :-
    write('Enter a choice (1-continue, 2-quit): '),
    read(X).

do_choice(1) :-
    write('Good choice!'), nl, go.

do_choice(2) :-
    write('Good choice! Bye.'), nl.

do_choice(_) :-
    write('Bad choice! Try again.'), nl, go.
**Assert and retract facts.**

Used to dynamically (at runtime) add and remove facts from KB.

- `assert(+Term)` ... add term to front of KB
- `assertz(+Term)` ... add term to end of KB
- `retract(+Term)` ... remove first matching term from KB
- `retractall(+Term)` ... remove all matching terms from KB

For example:

```
?- assert( fill(1,1,red) ).
true.

?- fill(X,Y,Z).
X = 1,
Y = 1,
Z = red.

?- retract( fill(1,1,red) ).
true.

?- fill(X,Y,Z).
false.

?- retract( fill(1,1,red) ).
false.

?- assert( fill(1,1,red) ).
ture.
```

S. Bowers 6 of 11
?- assert( fill(1,2,white) ).
true.

?- retractall( fill(1,X,Y) ).
true.

?- fill(X,Y,Z).
false.

Within a file, must declare as dynamic.

:- dynamic fill/3.

go :-
    take_position(X,Y,Z),
    record_position(X,Y,Z),
    menu_choice(C),
    do_choice(C), !.

record_position(X,Y,Z) :-
    fill(X,Y,_,),
    retract(fill(X,Y,_)),
    assert(fill(X,Y,Z)),
    write('That spot is taken. Replacing.'), nl.

record_position(X,Y,Z) :-
\+ fill(X,Y,_,), % not filled
    assert(fill(X,Y,Z)),
    write('That spot is open.'), nl.

take_position(X,Y,Z) :-
    write('Enter a row: '), read(X),
write('Enter a column: '), read(Y),
write('Enter a color: '), read(Z).

menu_choice(X) :-
    write('Enter a choice (1-continue, 2-quit): '),
    read(X).

do_choice(1) :- go.

do_choice(2) :- write('Bye.'), nl.

do_choice(_) :-
    write('Bad choice! Try again.'), nl, go.
Our “Mini” Language

We’re going to implement (an interpreter for) a simple “toy” language

- the language is totally made up (and probably not very useful)

Basic aspects of the language:

- implicit string, integer, and Boolean types (no type declarations)
- lists (Python like ... more later)
- math expressions (+, -, *, /)
- comparison operators (==, <, >, <=, >=, !)
- boolean connectives (and, or, not)
- user input from stdin via readint(msg) and readstr(msg)
- variable bindings (e.g., x = 5;)
- while loops (e.g., while x < 5 do ... end)
- conditional expressions (e.g., if ... then ... elif ... else ... end)
- single-line comments with #

A program is made up of a set of statements

- all statements, except conditionals and loops, are ended by semicolons
Some super simple examples:

```
# prints Hello World! (to standard out)
println("Hello World!");

# same as above
print("Hello ");
print("World!");
print("\n");

# assignment statements
x = 5; y = 6;

# conditional
if x < y then
    println("Yes");
elif x == y then
    println("Maybe");
else
    println("No");
end

# prompt for string entered by user
var1 = readstr("enter a string: ");
# print half of number entered by user
println(readint("enter an int: ") / 2);

# divide a number by 2 until smaller than 2 and print it
var2 = readint("enter an int: ");
while var2 > 2 do
    var2 = var2 / 2;
end
println(var2);
```
Some basic constraints (we'll talk about more later):

- boolean values are either true or false (e.g., flag = true;)
- strings are always separated by double quotes
- strings can't span multiple lines
- identifiers start with a letter followed by zero or more letters, digits, or underscores (e.g., myvar1 and my_var_1)

**Exercise**: Write some programs in this “mini” language ...