1 Files and I/O

- By default, the users *terminal* is the general input/output stream - i.e. all input is read from the terminal and all output directed to the terminal
- For instance, using the *read* and *write* commands, all interaction is via the user terminal

```
do :-
 write('command: '),
 read(Command),
 perform(Command).

perform( stop) :- !.
perform( Command) :-
 execute( Command),
 do.
```

Until *stop* is encountered, the program executes user commands.

Possible to write multiple commands to the output stream (i.e. the terminal - unless specified otherwise).

```
write_colour(X) :-
 colour(X, Colour),
 write('The colour of'),
 write(X),
 write('is'),
 write(Colour),
 write('.'),
```

```
nl.
```

- Where *colour* is a already present in the program, and associates colours with objects
- *nl* is a built-in predicate - which provides a *new line*

Once a result to any input and output operation has been obtained, all further results are ignored. Therefore, *I/O* operations are deterministic in that (while backtracking) they cannot be *re-satisfied*. Therefore, writing :

```
write('hello'), fail.
```

will only produce one output *hello* and stop.

Possible to read and write a term by doing :

```
read(X), write(X), fail.
```

Only *one* term may be read from the current input stream - the conjunction of goals, i.e. read and write, will then fail because backtracking cannot take place.

1.1 Changing I/O streams

- Useful to direct I/O to files, instead of the terminal
- This is equivalent to changing the current input stream to the new one by a goal
- Also, for writing a result, changing the current output stream to a new one via a goal

1.2 Reading from a file

```
see('data.txt')
```

opens the file for *reading*. To switch back to the default input stream (i.e. the user, instead of a file), use the command *seen*.

```
display(Filename) :-
 see(Filename),
 read(Term),
 write(Term), nl,
 see(user),
 write('More terms: (y/n)'),
 read( Yes_No),
 perform( Yes_No, Filename).

perform( n,_).
perform( _,Filename) :-
 display( Filename).
```

- The goal : *see(Filename)* succeeds only once, and cannot be *re-satisfied* on backtracking
- The first time it is satisfied for some file, that file - *Filename* is opened
- Reading of terms from file, in a top down manner, until *seen* is encountered
- If *Filename* is not instantiated or does not exist - then *goal* fails

1.3 Writing to file

- Assign current output stream to file name
- Done via the *tell(Filename)* command
- If *ANY* file already opened for writing - tell will write to that file
- *told* closes the file and switches output stream back to default value - the user
- In *MacProlog* also have *seeing/0* - which returns the current input stream - (file currently opened)

```
copy(Filename) :-
 tell(user),
 write( 'Term: '),
 read(Term),
 tell(Filename),
 perform(Term, Filename).

perform( stop,_) :- told.
perform( Term, Filename) :-
 write(Term), nl,
 copy(Filename).
```

- If arguments of *tell* not instantiated, *tell(Filename)* fails
- If argument instantiated to a filename that does not exist, a new file of that name is created and output stream switched
- Over writes an existing file

1.4 Reading Characters

```
get0(X)
get(X)

put(X)
```

- *get0(X)* reads a single character from the input stream and instantiates it to X
- *get1(X)* reads a non-white space character from the input stream and instantiates it to X (white character has ASCII value less than or equal to 32)

```
Q: get(X)
   P
A: X = 80
```

- *MacProlog* also provides *getb/1* and *getx/2* - used to get a *byte* or *word* from the keyboard

```
Q: getb(Byte)
(user then hits the left mouse button)
A: Byte=-1
```

- *put(X)* writes a single character to the output stream with the ASCII code X
- Multiple options of *put* also available in *MacProlog*
- No backtracking allowed - only a single character and then the goal fails

```
copy_sentence(Input,Output) :-
 see(Input),
 tell(Output),
 copy,
 seen,
 told.

copy :-
 get0(Ch),
 put(Ch),
 treat([Ch]).

treat(".").
treat( _) :-
 copy.
```

- *copy_sentence* copies from a file *Input* to a file *Output* - until terminated by a period
- Instead of procedures *seen* and *told*, also possible to close the input/output stream with *close(Filename)*
- When reading, if end of file reached, in the *read(X)* goal, X is instantiated to *end_of_file*, and the next *read(X)* fails
- *get0(X)* fails immediately if end of file is reached

2 Consulting Files

Contains three clauses *consult*, *reconsult* and *listing*

- Possible to place all predicates with the same name in a separate file (module)

- This module is then loaded into a complete program - done via *consult(Filename)*
- Possible to *consult* a list of file
- *reconsult(Filename)* - same as consult, except that all previous predicate definitions (of the same name) are deleted from the program and Filename
- *listing(P)* displays predicates with name P - regardless of the arity
- *listing* - without argument, shows the complete program

```
plays(X,Y) :-
 plays(X, Y, _).

plays(X, piano) :-
 plays(X, keyboard).

plays(X, keyboard) :-
 plays(X, guitar).
```

also, consider a file named 'beatles', which contains the following :

```
plays(john,guitar,rhythm).
plays(george, guitar, solo).
plays(paul, guitar, bass).
plays(ringo, drums).
```

and consider the dialog (queries) with Prolog

```
Q: listing
```

```
A: yes

Q: consult(beatles)
A: yes

Q: listing
A: plays(john,guitar,rhythm).
plays(george, guitar, solo).
plays(paul, guitar, bass).
plays(ringo, drums).
yes

Q: reconsult(plays)
A: yes

Q: listing
A: plays(john,guitar,rhythm).
plays(george, guitar, solo).
plays(paul, guitar, bass).

plays(X,Y) :-
 plays(X, Y, _).
plays(X, piano) :-
 plays(X, keyboard).
plays(X, keyboard) :-
 plays(X, guitar).
yes
```

- Note that *reconsult(plays)* resulted in the removal of *plays(ringo,drums)*
- Also, only one occurrence of the required *play* clause was

removed - hence, the *reconsult* clause only needs to be satisfied once (i.e. no backtracking)