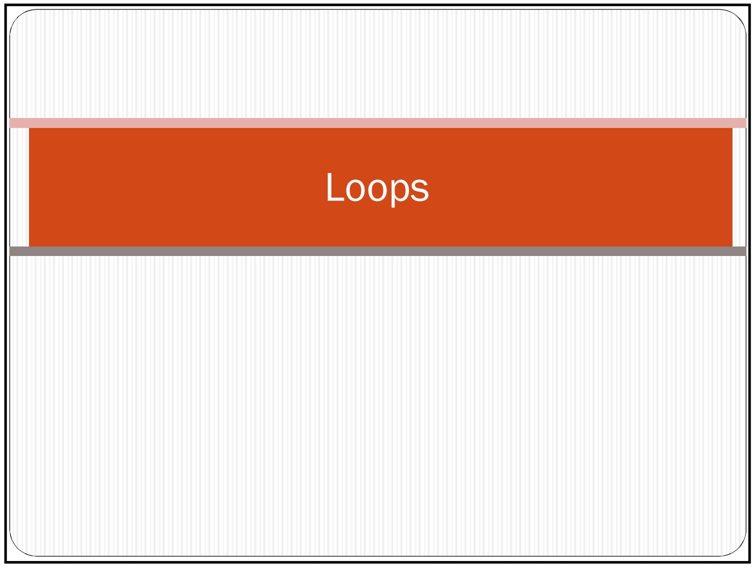
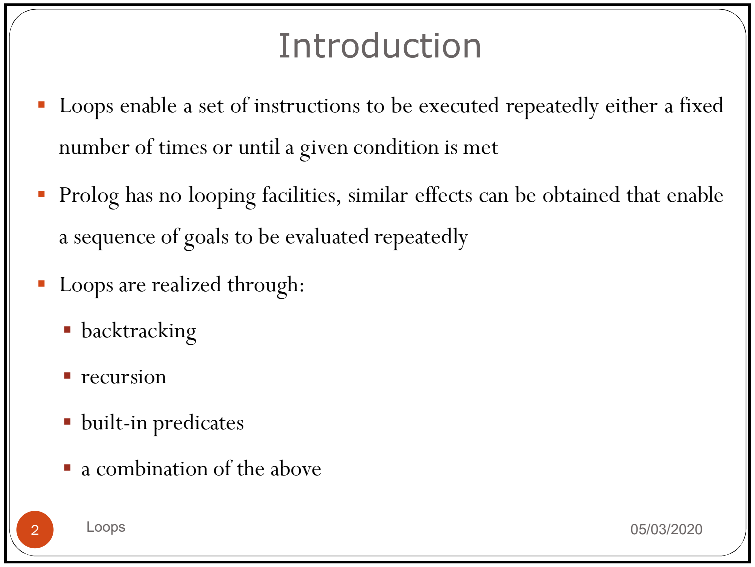
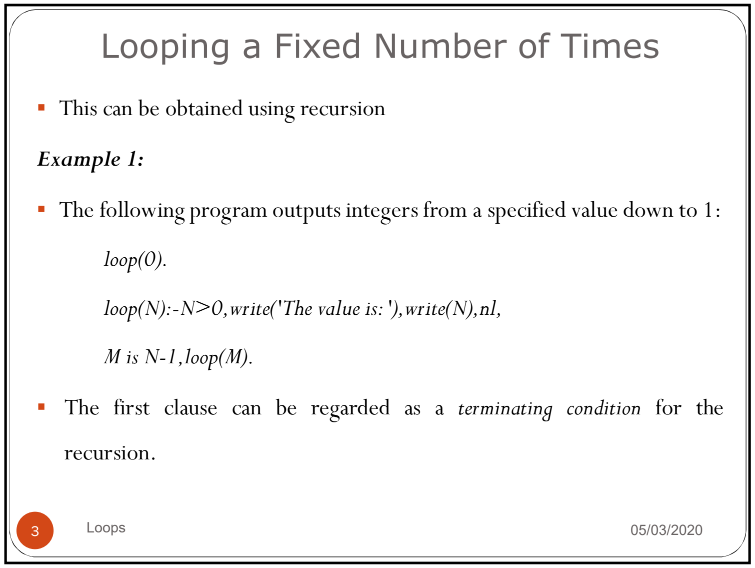


Loops



Introduction

- Loops enable a set of instructions to be executed repeatedly either a fixed number of times or until a given condition is met
- Prolog has no looping facilities, similar effects can be obtained that enable a sequence of goals to be evaluated repeatedly
- Loops are realized through:
  - backtracking
  - recursion
  - built-in predicates
  - a combination of the above



Looping a Fixed Number of Times

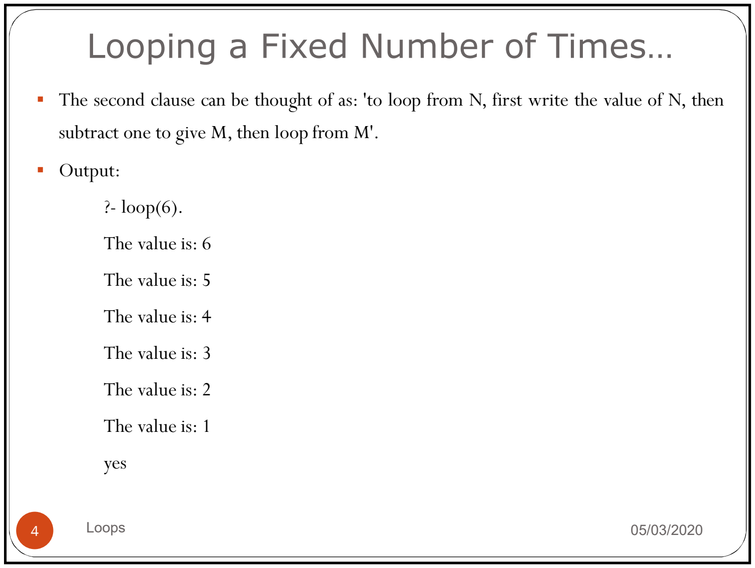
- This can be obtained using recursion

Example 1:

- The following program outputs integers from a specified value down to 1:

```
loop(0).
loop(N):-N>0,write('The value is: '),write(N),nl,
M is N-1,loop(M).
```

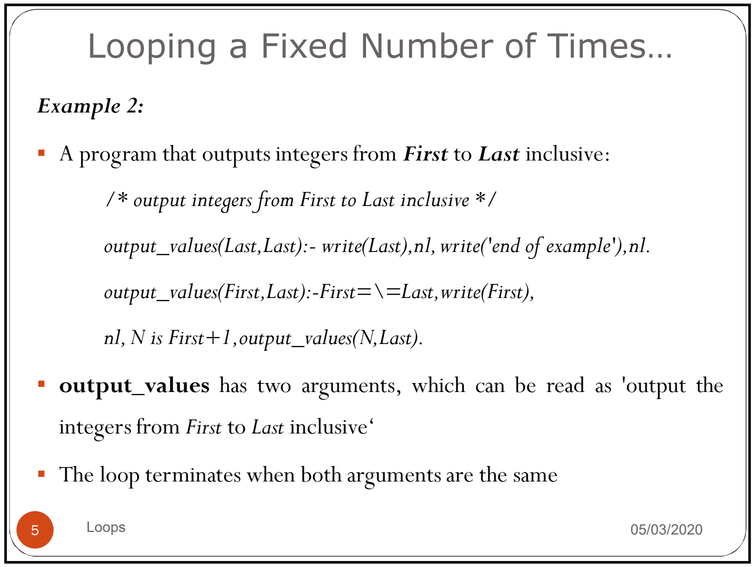
- The first clause can be regarded as a *terminating condition* for the recursion.



Looping a Fixed Number of Times…

- The second clause can be thought of as: 'to loop from N, first write the value of N, then subtract one to give M, then loop from M'.
- Output:

```
?- loop(6).
The value is: 6
The value is: 5
The value is: 4
The value is: 3
The value is: 2
The value is: 1
yes
```



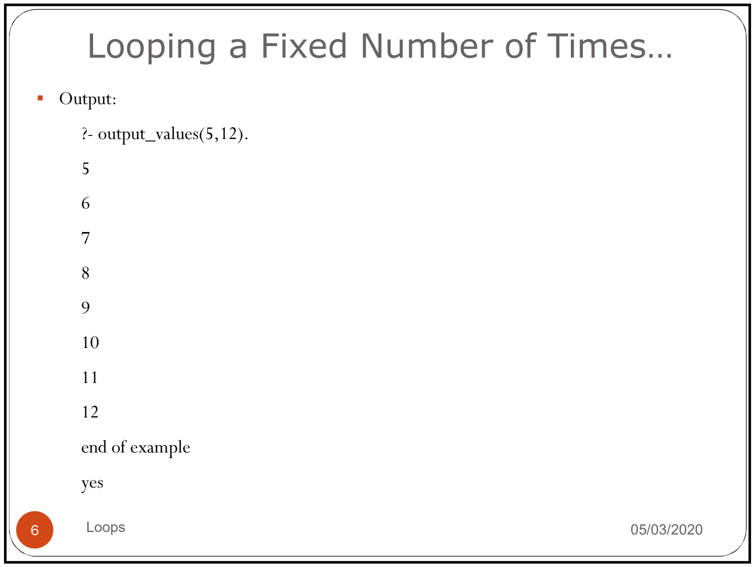
Looping a Fixed Number of Times…

Example 2:

- A program that outputs integers from ***First*** to ***Last*** inclusive:

```
/* output integers from First to Last inclusive */
output_values(Last,Last):- write(Last),nl, write('end of example'),nl.
output_values(First,Last):-First=\=Last,write(First),
nl, N is First+1,output_values(N,Last).
```

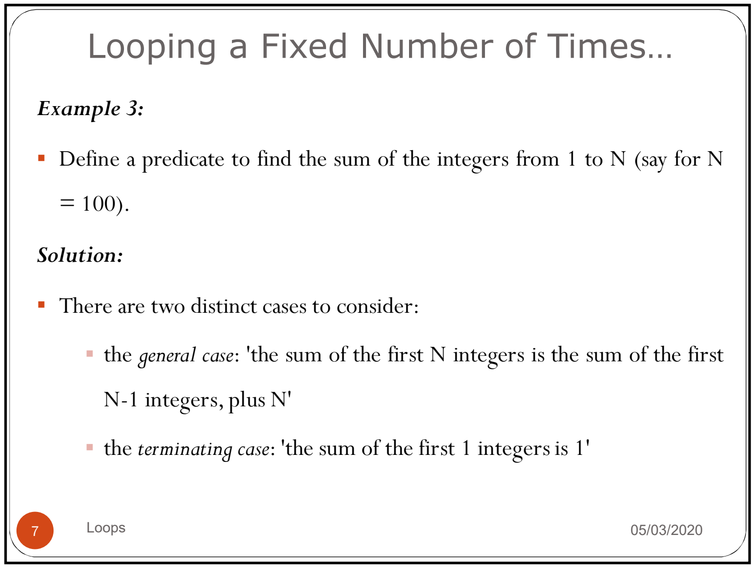
- **output_values** has two arguments, which can be read as 'output the integers from *First* to *Last* inclusive‘
- The loop terminates when both arguments are the same



Looping a Fixed Number of Times…

- Output:

```
?- output_values(5,12).
5
6
7
8
9
10
11
12
end of example
yes
```



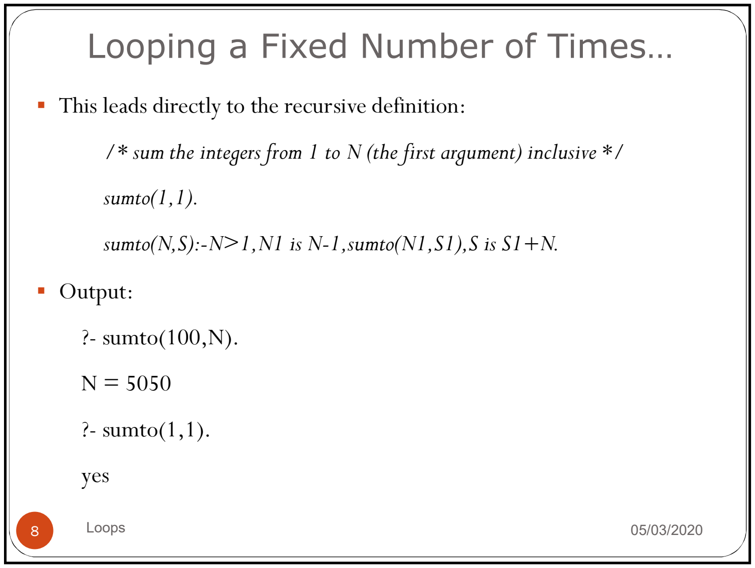
Looping a Fixed Number of Times…

Example 3:

- Define a predicate to find the sum of the integers from 1 to N (say for N = 100).

Solution:

- There are two distinct cases to consider:
  - the *general case*: 'the sum of the first N integers is the sum of the first N-1 integers, plus N'
  - the *terminating case*: 'the sum of the first 1 integers is 1'



Looping a Fixed Number of Times…

- This leads directly to the recursive definition:

```
/* sum the integers from 1 to N (the first argument) inclusive */
sumto(1,1).
sumto(N,S):-N>1,N1 is N-1,sumto(N1,S1),S is S1+N.
```

- Output:

```
?- sumto(100,N).
N = 5050
?- sumto(1,1).
yes
```

Looping a Fixed Number of Times…

- Note that using the additional variable **N1** for holding the value of **N-1** is essential
- Writing **sumto(N-1,S1)** etc. instead would not work correctly. **N-1** is a term, not a numerical value

Looping a Fixed Number of Times…

Example 4:

- Define a predicate to output the squares of the first N integers, one per line.
- Solution:
  - This can most easily be programmed if first recast in a recursive form
  - The general case is 'to output the squares of the first N integers, output the squares of the first N-1 and then output N2‘
  - The terminating case is 'to output the squares of the first 1 integers, output the number 1'

Looping a Fixed Number of Times…

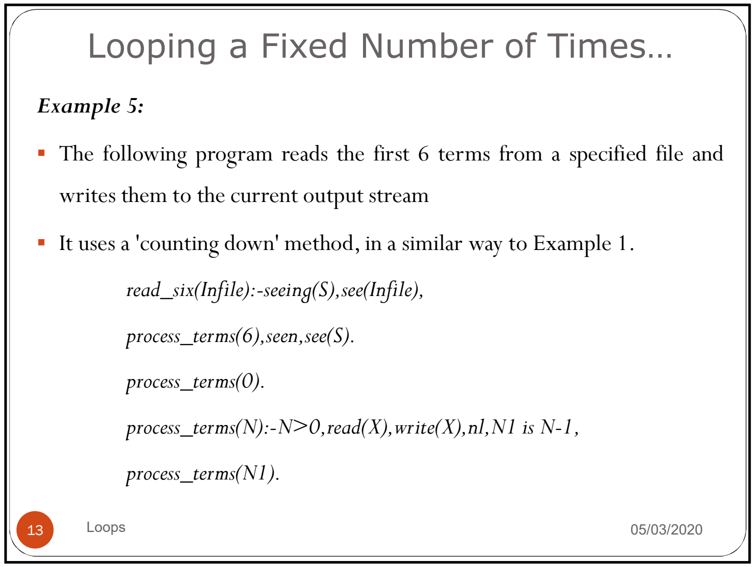
- This leads to the following two-clause program:

```
/* output the first N squares, one per line */
writesquares(1):-write(1),nl.
writesquares(N):-N>1,N1 is N-1,writesquares(N1),
Nsq is N*N,write(Nsq),nl.
```

Looping a Fixed Number of Times…

- Output:

```
?- writesquares(6).
1
4
9
16
25
36
yes
```

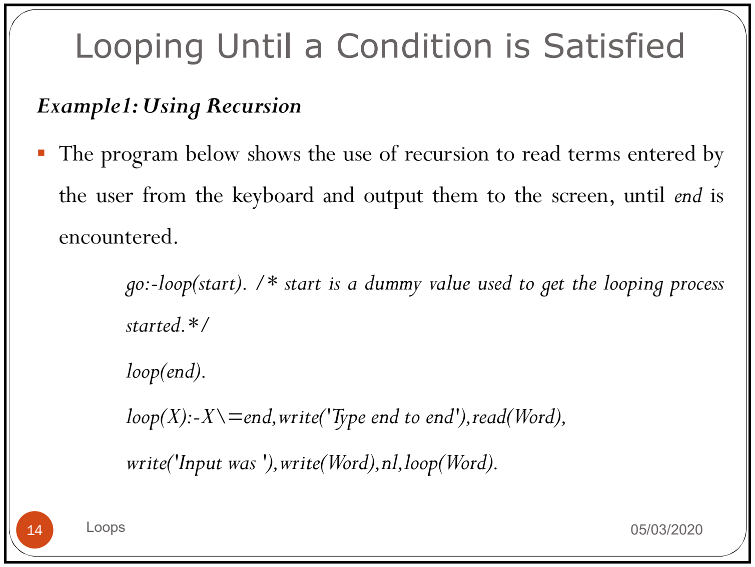


Looping a Fixed Number of Times…

Example 5:

- The following program reads the first 6 terms from a specified file and writes them to the current output stream
- It uses a 'counting down' method, in a similar way to Example 1.

```
read_six(Infile):-seeing(S),see(Infile),
process_terms(6),seen,see(S).
process_terms(0).
process_terms(N):-N>0,read(X),write(X),nl,N1 is N-1,
process_terms(N1).
```

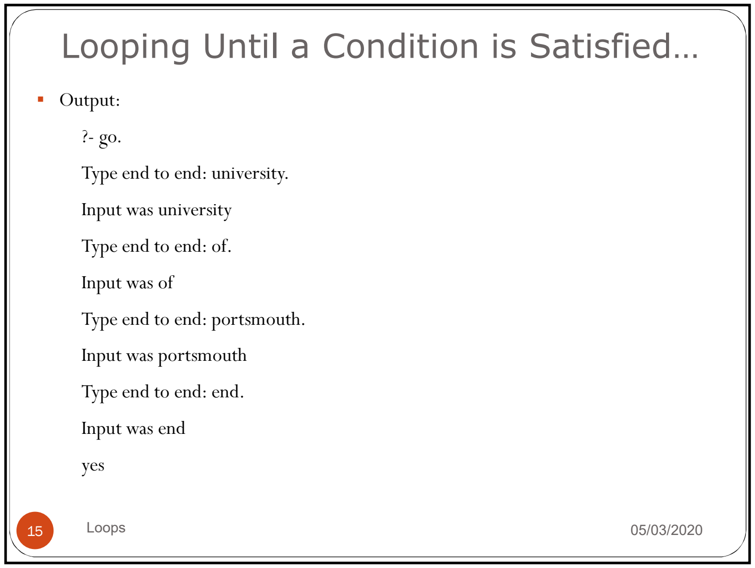


Looping Until a Condition is Satisfied

Example1: Using Recursion

- The program below shows the use of recursion to read terms entered by the user from the keyboard and output them to the screen, until *end* is encountered.

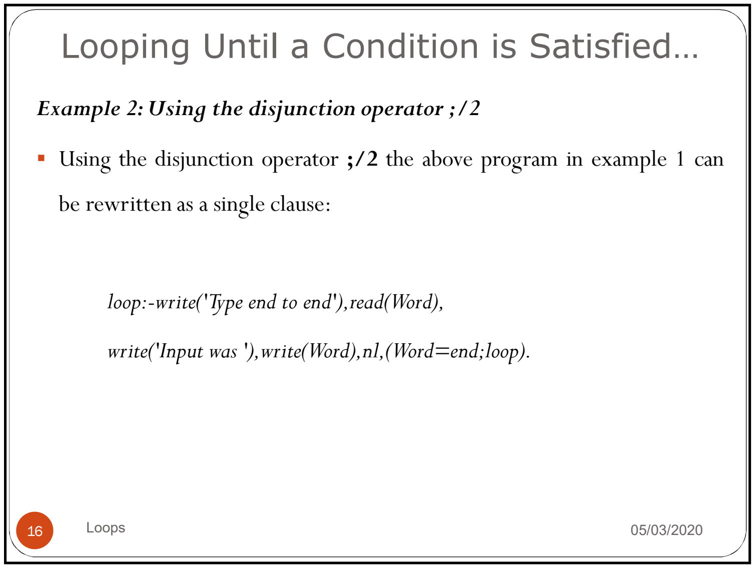
```
go:-loop(start). /* start is a dummy value used to get the looping process
started.*/
loop(end).
loop(X):-X\=end,write('Type end to end'),read(Word),
write('Input was '),write(Word),nl,loop(Word).
```



Looping Until a Condition is Satisfied…

- Output:

```
?- go.
Type end to end: university.
Input was university
Type end to end: of.
Input was of
Type end to end: portsmouth.
Input was portsmouth
Type end to end: end.
Input was end
yes
```



Looping Until a Condition is Satisfied…

Example 2: Using the disjunction operator ;/2

- Using the disjunction operator **;/2** the above program in example 1 can be rewritten as a single clause:

```
loop:-write('Type end to end'),read(Word),
write('Input was '),write(Word),nl,(Word=end;loop).
```

Looping Until a Condition is Satisfied…

- Output:

```
?- loop.
Type end to end: university.
Input was university
Type end to end: of.
Input was of
Type end to end: portsmouth.
Input was portsmouth
Type end to end: end.
Input was end
yes
```

Looping Until a Condition is Satisfied…

Example 3: Using Recursion

- The recursive program below repeatedly prompts the user to enter a term until either *yes* or *no* is entered.

```
get_answer(Ans):-write('Enter answer to question'), nl,get_answer2(Ans).
get_answer2(Ans):- write('answer yes or no'), read(A),
((valid(A),Ans=A,write('Answer is '),
write(A),nl);get_answer2(Ans)).
valid(yes). valid(no).
```

Looping Until a Condition is Satisfied…

- Output:

```
?- get_answer(Myanswer).
Enter answer to question
answer yes or no: maybe.
answer yes or no: possibly.
answer yes or no: yes.
Answer is yes
Myanswer = yes
```

Looping Until a Condition is Satisfied…

Example 4: Using the 'repeat' Predicate

- Although it can often be used to great effect, recursion is not always the easiest way to provide the types of looping required in Prolog programs
- Another method that is often used is based on the built-in predicate **repeat**.
- The name of this predicate is really a misnomer. The goal **repeat** does not repeat anything; it merely succeeds whenever it is called
- The great value of **repeat** is that it also succeeds (as many times as necessary) on backtracking

Looping Until a Condition is Satisfied…

- The effect of this, as for any other goal succeeding, is to change the order of evaluating goals from 'right to left' (i.e. backtracking) back to 'left-to-right'.
- This can be used to create a looping effect, as shown in the examples below.
- The program below repeatedly prompts the user to enter a term until either *yes* or ***no*** is entered. It

Looping Until a Condition is Satisfied…

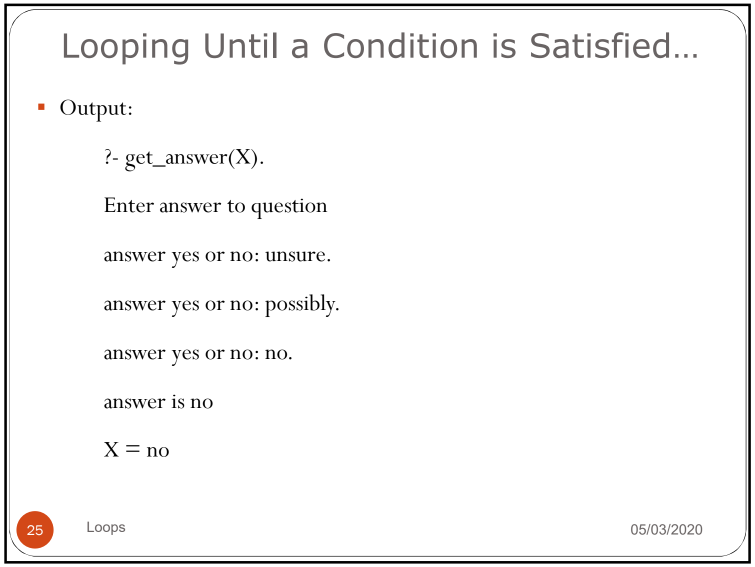
```
get_answer(Ans):-

write('Enter answer to question'),nl,

repeat,write('answer yes or no'),read(Ans),

valid(Ans),write('Answer is '),write(Ans),nl.

valid(yes). valid(no).
```

Looping Until a Condition is Satisfied…

- Backtracking will then reach the predicate **repeat** and succeed, causing evaluation to proceed forward (left-to-right) again, with **write('answer yes or no')** and **read(Ans)** both succeeding, followed by a further evaluation of **valid(Ans)**.
- Depending on the value of **Ans**, i.e. the user's input, the **valid(Ans)** goal will either fail, in which case Prolog will backtrack as far as **repeat**, as before, or it will succeed in which case the final three goals **write('Answer is')**, **write(Ans)** and **nl** will all succeed.

Looping Until a Condition is Satisfied…

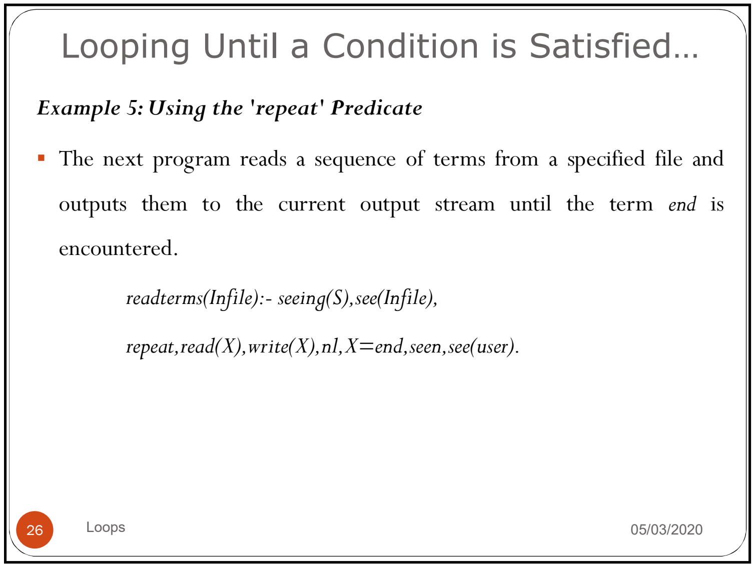
- The overall effect is that the two goals **write('answer yes or no')** and **read(Ans)** are called repeatedly until the terminating condition **valid(Ans)** is satisfied, effectively creating a loop between **repeat** and **valid(Ans)**.
- Goals to the left of **repeat** in the body of a clause will never be reached on backtracking.



Looping Until a Condition is Satisfied…

- Output:

```
?- get_answer(X).
Enter answer to question
answer yes or no: unsure.
answer yes or no: possibly.
answer yes or no: no.
answer is no
X = no
```

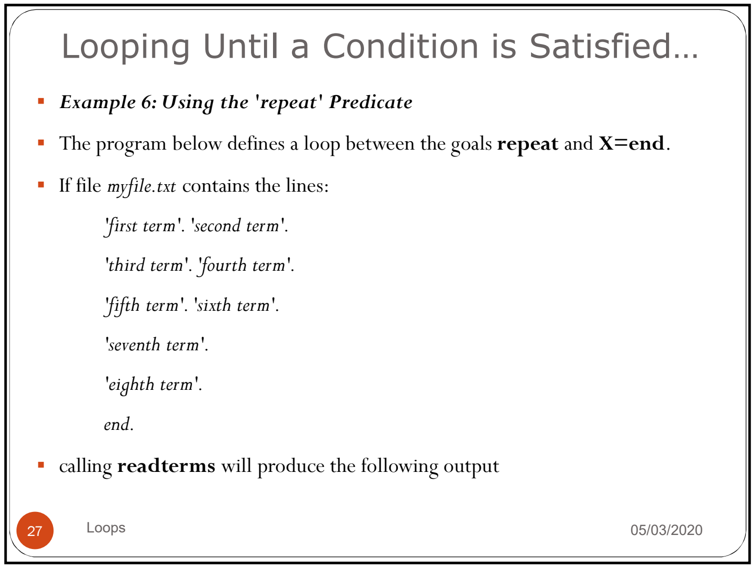


Looping Until a Condition is Satisfied…

Example 5: Using the 'repeat' Predicate

- The next program reads a sequence of terms from a specified file and outputs them to the current output stream until the term *end* is encountered.

```
readterms(Infile):- seeing(S),see(Infile),
repeat,read(X),write(X),nl,X=end,seen,see(user).
```

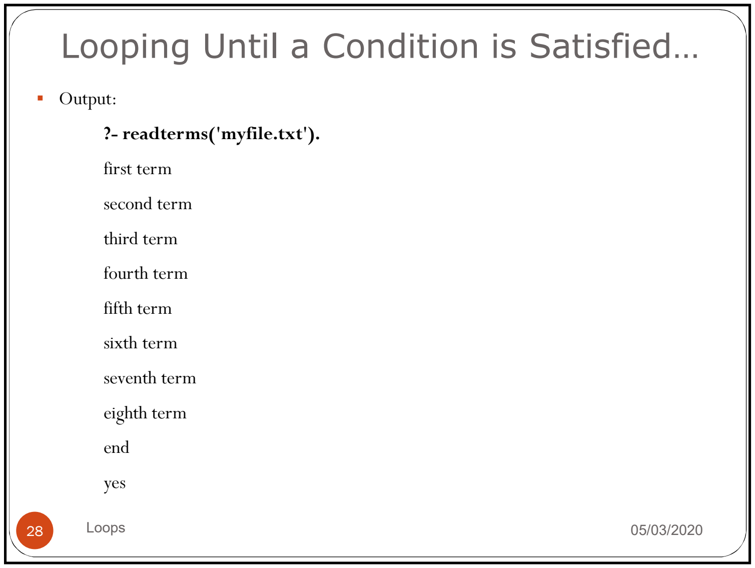


Looping Until a Condition is Satisfied…

- ***Example 6: Using the 'repeat' Predicate***
- The program below defines a loop between the goals **repeat** and **X=end**.
- If file *myfile.txt* contains the lines:

```
'first term'. 'second term'.
'third term'. 'fourth term'.
'fifth term'. 'sixth term'.
'seventh term'.
'eighth term'.
end.
```

- calling **readterms** will produce the following output



Looping Until a Condition is Satisfied…

- Output:

```
?- readterms('myfile.txt').
first term
second term
third term
fourth term
fifth term
sixth term
seventh term
eighth term
end
yes
```

Looping Until a Condition is Satisfied…

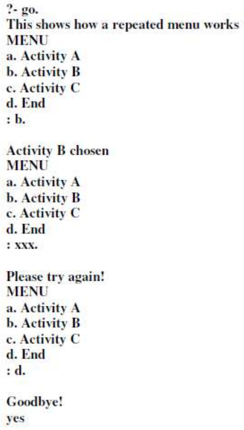
Example 7:

- This program shows how to implement a menu structure which loops back repeatedly to request more input. Entering **go** at the prompt causes Prolog to output a menu from which the user can choose activities one at a time until option *d* is chosen
- Note that all inputs are terms and so must be followed by a full stop character.

Looping Until a Condition is Satisfied…

```
go:- write('This shows how a repeated menu works'), menu.

menu:-nl,write('MENU'),nl,

write('a. Activity A'),nl,write('b. Activity B'),nl,

write('c. Activity C'),nl,write('d. End'),nl,

read(Choice),nl,choice(Choice).

choice(a):-write('Activity A chosen'),menu.

choice(b):-write('Activity B chosen'),menu.

choice(c):-write('Activity C chosen'),menu.

choice(d):-write('Goodbye!'),nl.

choice(_):-write('Please try again!'),menu.
```

Looping Until a Condition is Satisfied…



```
?- go.
This shows how a repeated menu works
MENU
a. Activity A
b. Activity B
c. Activity C
d. End
: b.

Activity B chosen
MENU
a. Activity A
b. Activity B
c. Activity C
d. End
: xxx.

Please try again!
MENU
a. Activity A
b. Activity B
c. Activity C
d. End
: d.

Goodbye!
yes
```

END