3 A C program which simulates Turing machines

This is a rather simple program which (compiling to an executable tm) can be used as

```
tm [-trace] <machine> <inputs>
```

```
Note: originally, the machines were named xxx.tm, but that
extension conflicts with something, so xxx.tu was adopted
instead.

The C source code, plus several sample machines and sample
inputs, are in the module web page, subdirectory
turing-files.
```

```
        /*
         * gcc -o tm turinginC.c
         *
         * Turing machine interpreter.
         * usage: tm <quintuples> <inputs>
         * Quintuples are in the form
         *
         * qi a b V qj  .........
         *
         * i and j are zero-suppressed integers; can't have
         * q01.  V (head movement) is 'L' or 'R'.
         * For simplicity, Blank is 'B' in quintuples.
         * It is expected that every state begins with the letter q
         * and is followed by an index, a decimal integer.
         *
         * One quintuple per line.  Comments after the quintuple are
         *                               ignored.
         * Lines which are empty or begin with a space are ignored.
         * Hence more comments are possible.
         * For example the text between the hyphens defines a machine

---------------------------------

q0 0 B R q0     move right on 0
q0 1 B R q0     move right on 1
q0 B B L q0     move left on blank

        Implicitly the input alphabet is binary, and
        the machine always loops
---------------------------------
```

```
        */


  /*****************************
   * usage
   * tm  <machine> <inputs>
   * or
   * tm -trace <machine> <inputs>
   *****************************/
```

3.1 Always looping

Try the above Turing machine on the set pal of bitstrings

```
% cat pal

0
00
101
110
01

% cat looper.tu
q0 0 B R q0     move right on 0
q0 1 B R q0     move right on 1
q0 B B L q0     move left on blank

% tm looper.tu pal
q0 0 B R q0
q0 1 B R q0
q0 B B L q0

input:
output: loops

input: 0
output: loops

input: 00
output: loops

input: 101
output: loops

input: 110
```

```
output: loops

input: 01
output: loops
%
```

3.2 Input string 1^n, output $n \mod 2$

```
% cat ones

11
111
11111111
1111
1

% tm mod2.tu ones
q0 1 B R q1
q1 1 B R q0
q0 B 0 R q2
q1 B 1 R q2
q2 B B L q3

input:
output:0

input: 11
output:  0

input: 111
output:   1

input: 11111111
output:        0

input: 1111
output:    0

input: 1
output: 1

input:
output:0
```

3.3 Same machine, run with trace on 1^4

```
%
% tm -trace mod2.tu 1to4
q0 1 B R q1
q1 1 B R q0
q0 B 0 R q2
q1 B 1 R q2
q2 B B L q3

input: 1111
Configuration: state q0
1111
^
Configuration: state q1
B111
 ^
Configuration: state q0
BB11
  ^
Configuration: state q1
BBB1
   ^
Configuration: state q0
BBBB

Configuration: state q2
BBBB0

Configuration: state q3
BBBB0B
    ^
output:    0
```

3.4 Increment binary

Given a bitstring
(1) Be sure to loop on empty input
(2) Else move to the right-hand end
(3) Move left past 1s, changing to zero
(4) When zero or blank is met, change to 1
(5) Move to the left-hand end.

```
q0 B B R q0  loop on empty input
```

```
q0 0 0 R q1  nonempty: move right
q0 1 1 R q1  same again

q1 0 0 R q1
q1 1 1 R q1  Seek right-hand end

q1 B B L q2

q2 1 0 L q2 move left past 1s changing to 0

q2 0 1 L q3 rightmost 0 changed to 1
q2 B 1 L q3 --- when no 0s in input

q3 0 0 L q3
q3 1 1 L q3 Move to the left

q3 B B R q4 Ends scanning the leftmost bit
```

The C program includes a copy (no comments) of the machine. Here it runs on 11011.

```
% tm incr-220126.tu 11011
q0 B B R q0
q0 0 0 R q1
q0 1 1 R q1
q1 0 0 R q1
q1 1 1 R q1
q1 B B L q2
q2 1 0 L q2
q2 0 1 L q3
q2 B 1 L q3
q3 0 0 L q3
q3 1 1 L q3
q3 B B R q4

input: 11011
output: 11100
%
```

In all of these examples, the tape alphabet is {0,1,B}. There will be other examples which need extra tape symbols.