Examples of Reverse Polish Formulas —

with 0 operators:

x
293.71
5

with 1 operator:

+ 3 7
* x 62.1
+ 1 x

with 2 operators:

+ * x 2 1	≡	(x*2)+1
* x + 2 1	≡	x*(2+1)
+ + 1 2 3	≡	1+2+3
+ 7 / x 2	≡	$7+\frac{x}{2}$

with 3 operators:

+ * + x 1 2 3	≡	((x+1)*2)+3
* + x 1 + x 2	≡	(x+1)*(x+2)
+ 1 * x + 2 3	≡	1+x*(2+3)

etc. etc. etc.

Assume the existence of

```
string read_next_symbol()
```

which reads a single symbol (e.g. "+", "123.6" etc) from input and returns it.

```
bool is_number(string s)
```

true if s consists of digits, optional decimal pt., etc.

```
bool is_variable(string s)
```

true if s is "x" or any other acceptable variable name.

```
bool is_operator(string s)
```

true if s is "+", "*", etc.

How to read <u>any</u> R.P.F. with O operators:

```
bool read_rpf_O()
{ string s=read_next_symbol();
  if (is_number(s))
     return true;
  else if (is_variable(s))
     reburn true;
  else
     return false; }
```

An R.P.F. with 1 operator consists of
an operator, followed by
an R.P.F. with 0 operators, followed by
another R.P.F. with 0 operators.

So – How to read <u>any</u> R.P.F. with exactly one operator:

read an operator
read an RPF with 0 operators
read an RPF with 0 operators
done.

as a function that checks validity:

```
bool read_rpf_exactly_1()
{ string s = read_next_symbol();
  if (! is_operator(s))
     return false;
  bool ok1 = read_rpf_0();
  if (!ok1)
     return false;
  bool ok2 = read_rpf_0();
  if (!ok2)
     return false;
  return true; }
```

More Usefully~

How to read an RPF with no more than 1 operator:

Either:
read an operator, then
read an RPF with no more than 0 operators, then
read another RPF with no more than 0 operators
Or:
read a number
Or:
read a variable

```
bool read_rpf_1()
{ string s= read_next_symbol();
  if (is_number(s))
     return true;

  else if (is_variable(s))
     return true;

  else if (is_operator(s))
  { bool ok1= read_rpf_0();
     bool ok2= read_rpf_0();
     return ok1 && ok2; }

  else
     return false; }
```

An R.P.F. with no more than 2 operators consists of either:

an operator,
an RPF of no more than 1 operators,
an RPF of no more than 1 operators

or:

a number

or:

a variable.

```
bool read_rpf_2()
{ string s = read_next_symbol();
  if (is_number(s))
  {
    return true; }

  else if (is_variable(s))
  {
    return true; }

  else if (is_operator(s))
  {
     bool ok1 = read_rpf_1();
     bool ok2 = read_rpf_1();
     return ok1 && ok2; }

  else
     return false; }
```

A valid complete input line consists of an R.P.F. with an unlimited number of operators, followed by a "?" question mark.

```
bool read_input_line()
{ bool ok1 = read_rpf_oo();
  string s = read_next_symbol();
  if (s == "?")
     return ok1;
  else
     return false; }
```

Originally, we read a whole input formula and converted it into a linked list containing the individual symbols.

So to complete the implementation, ~~read_input_line~~ should first: read a line of input

```
string input;
getline (cin, input);
```

then: use last week's function to convert it into a linked list of strings,

then: proceed exactly as above.

The read_next_symbol() function becomes simply remove the first symbol from the linked list, return it.