The new type `mystream` is introduced to make the extraction of symbols and numbers embedded within a string simple and non-confusing.

```
struct mystream
{ string line;
  int pos; };
```

A `mystream` is a bit like the familiar `istreams` (`cin`, etc). It remembers how far you've got with running through a string, so you can always ask it for the next thing without having to remember any details yourself. Seems trivial, but in many circumstances, it really does help a lot.

A `mystream`'s `line` holds the whole string being processed, including the portions already dealt with and the portions not yet looked at. The `pos` component just records how many characters from that string have already been used.

Typically, you might read a whole formula from the keyboard or a file, and use it to set up a `mystream`:

```
{ string form;
  cin >> form;
  mystream str;
  initialise(str, form);
  ...
```

So you already know what one function must do

```
void initialise(mystream & ms, string x)
{ ms.line = x;
  ms.pos = 0; }
```

Obviously useful operations include asking if the stream is empty yet, with nothing left to read, taking the next character, and just sneaking a look at the next character without actually committing to take it. They are all easy to write.

```
bool empty(mystream ms)
{ if (ms.pos >= ms.line.length())
    return true;
  else
    return false; }

char sneak_next_char(mystream ms)
{ if (empty(ms))
    return '.';
  else
    return ms.line[ms.pos]; }
```

```
char get_next_char(mystream & ms)
{ if (empty(ms))
    return '.';
  else
  { char answer = ms.line[ms.pos];
    ms.pos += 1;
    return answer; } }
```

It would be useful to have some sensible utility functions for identifying what kind of character we are looking at, and the standard C++ library has them all predefined (you just have to `#include <ctype.h>`)

`bool isdigit(char c);`	returns true only for '0' to '9'.
`bool islower(char c);`	returns true only for 'a' to 'z'.
`bool isupper(char c);`	returns true only for 'A' to 'Z'.
`int digittoint(char c);`	returns 0 for '0', 1 for '1', ..., 9 for '9'.

Now we can perform some higher level operations very simply:

To read the next element symbol (capital letter, maybe followed by a little letter) from a `mystream`:

```
string get_next_symbol(mystream & str)
{ if (isupper(sneak_next_char(str)))
  { string answer = "";
    answer += get_next_char(str);
    if (islower(sneak_next_char(str)))
      answer += get_next_char(str);
    return answer; }
  else
    return "Error!!!!!!!!"; }
```

To read the next number (just a sequence of digits) from a `mystream`:

```
int get_next_number(mystream & str)
{ int total = 0;
  while (isdigit(sneak_next_char(str)))
  { char c = get_next_char(str));
    total = total * 10 + digittoint(c); }
  return total; }
```

The two step procedure for making a string from a single character is so irritating that I usually just give myself a little helper function like this:

```
string tostring(char c)
{ string answer = "";
  answer += c;
  return answer; }
```

Then the `get_next_symbol` function can look a little less peculiar.

```
string get_next_symbol(mystream & str)
{ if (isupper(sneak_next_char(str)))
  { string answer = tostring(get_next_char(str));
    if (islower(sneak_next_char(str)))
      answer += get_next_char(str);
    return answer; }
  else
    return "Error!!!!!!!!"; }
```