Predefined Symbols

There are several predefined symbols which your toolchain supplies, and which you can use in conditional compilation, like this example from StackOverflow, which tests for compiling on different platforms:

```
#ifdef _WIN32
   //define something for Windows (32-bit)
#elif __APPLE__
  // define something for OSX
#elif __linux
    // linux
#endif
```

These predefined symbols include those that are standard on every version of C++ and those that are common to GCC on every platform. There are also platform-specific symbols for other toolchains (such as the operating system). You can get a list of those by running `cpp -dM` from the shell.

For this problem, we care about is a **particular version of** C++. In the list of predefined standard constants, you'll see that `__cplusplus` (double leading underscores) contains version numbers for each release of C++. You can use that to bracket your own versions of the `stoi()` and `stod()` functions.

Go back to your test program and use this facility to define the functions **only** if the symbol `__cplusplus` is `<= 199711L`. Now you can compile and run with C++98 and with C++11/14/17/20 using the same source.

To implement the functions, just use code like this:

```
function stoi <- input str -> output int
  set result to 0
  construct an input string stream using str
  read from str into result
  return result
```

The `stod()` function will be identical, except `result` will be `double` instead of `int`.