```cpp
//////// Lecture 87: String Stream class

#include <iostream>
int main(){
    int a{5}, b{6};
    int sum = a+b;
    std::cout << "Sum of " << a << " & " << b << "is:
      " << sum << std::endl;
}
// This gives you a nicely formatted string.
// What if we need to display this text in a GUI
  application - like text box?
// in that case we cannot just use cout because it
  works only with console, then we may have to create
  this string formatted with the appropriate values
  and then display it in the textbox.

// So we may attempt to create something like
std::string output = "Sum of " + a + " & " + b + " is
  " + sum;
// but this doesn't happen automatically (the
  conversion from an integer to a string)
// this is where string streams come in - we can read
  some input from the keyboard and write some output
  to the console. In stringstreams, we can do the
  same thing, but instead of reading from the
  keyboard or writing to the console, we perform
  these operations on a string buffer.
// The stringsream internally maintains the string
  buffer and all the operations are performed in this
  string buffer.

// there are 3 classes that we can use with string
  streams.
#include <sstream>

std::stringstream ss;

std::istringstream is;

std::ostringstream os;

```

```cpp
// These stream classes provide appropriate overloaded operators for inserting or extracting characters from a stream. The stringstream class provides both insertion and extraction operators, that means you can read from the stringstream and you can also write to it.

// The istringstream class provides only extraction operator, which means you can ONLY read from it.

// The ostringstream class provides only insertion operator, which means you can ONLY write to it.

int main()
{
    int a{5}, b{6};
    int sum = a+b;
    std::stringstream ss;
    ss << "Sum of " << a << " & " << b << " is "<< sum << std::endl;
    // So this sentence above ^ will be inserted into the stringstream.
    // The stringstream internally manages a string buffer and we can access that buffer through a member function of the stringstream called as 'str()'.

    // This function has 2 overloads:
    // 1) One overload returns the copy of the buffer as a standard string object.
    std::string s = ss.str();
    // 2) the other overload accepts a string
    ss.str("Some string"); // if you want to assign some string to the internal buffer then this is the function that you can use. We will use this function later, for now we'll just print the string that was stored inside the stringstream.
    std::cout << s << std::endl;

    // This way we can create formatted strings and use them wherever we want.
}
```

```
52  // Using the string stream, you can also convert a
    •   primitive type into a string object:
53  int main()
54  {
55      int num = 5;
56      stringstream ss;
57      ss >> num;
58      auto ss_stream = ss.str();
59      std::cout << ss_stream << std::endl;
60  }
61  // You can see that it has also printed the earlier
    •   statement, the earlier formatted string that we had
    •   inserted into the string stream.
62
63  // So if we do not want that, before inserting sum
    •   into a stringstream, we can clear the string
    •   stream. Let's run it again.
64
65  // And this is the correct output
66
67  /////// to_string() function
68  // C++11 contains a function called to_string() and
    •   this function is overloaded for all the primitive
    •   types and it returns the primitive types and it
    •   returns the primitive types as string.
69
70  // so we can use that as well,
71  std::cout << to_string(5) << std::endl; // ==> 5
72  // And this would give us the same output.
73  // So internally it uses stringstream to convert the
    •   integer int oa string object, and it is overloaded
    •   for all the primitive types.
74
75  // READING from the string stream - this is useful
    •   for parsing strings. If we have a string that
    •   contains a bunch of numbers, then we can pass the
    •   string and extract the individual numbers.
76  int main(void)
77  {
78      std::string data = "12 89 21";
79      int num;
80
```

```
81          std::stringstream ss;
82          // Set data string into the string stream
83          ss.str(data);
84          while (!ss.fail())
85          {
86              ss >> num;
87              // So fail() is a member function in ss that
                  checks for the fail bit.  This bit is set
                  when the string stream fails to read any
                  more input, remove this and we'll also have
                  to set the data string into the
                  stringstream.
88              std::cout << a << std::endl;
89          }
90      }
91
92      // Notice that the last number is printed twice,
          because in the condition the fail bit is set only
          when it attempts to read the data string after the
          last number has been extracted. The extraction
          operator of the stringsteeam returns a reference of
          the stringstream object itself, and the
          stringstream class also contains the bool operator
          which is overloaded.
93
94      // We can use that bool operator to check for the
          fail bit.
95      int main{
96          ...
97          while (ss >> a){ // so when this expression
              extracts the last number and the next iteration
              it tries to read again, it will set the fail
              bit, and this expression will return the
              stringstream on which the bool operator will be
              invoked and that will return false, terminating
              the loop.
98              std::cout << a << std::endl;
99          }
100         // Running this now, we see the correct output.
101     }
102     // Stringstream class helps convert numbers which are
          the form of string objects into their equivalent
```

```
    the form of string objects into their equivalent
    primitive types. C++11 already provides functions
    which achieve that.
// So if you want to convert a number which is in the
    form oa string into its equivalent primitive type,
    there are functions like std::stoi.

std::stoi("5"); // 5 -> integer.

```