

AUTOMATING TYPE CONVERSIONS WITH STRING STREAM OBJECTS

By Danny Kalev

Certain type conversions, such as long to double and signed to unsigned, are automatic in C. However, when you convert strings to numeric values and vice versa, you have to use explicit cast operations. Usually, you allocate a buffer that will store the result and call an appropriate conversion function such as `atoi()`, `sprintf()` or `scanf()`. This approach has substantial drawbacks. First, if the target buffer is too small to contain the result of the conversion operation, the onerous **buffer overflow** bug will occur and result in undefined behavior. Second, the traditional `<stdio.h>` conversion functions are type-unsafe—one can easily use the wrong format flag, inadvertently omit an ampersand before a pointer argument, or add a redundant ampersand before a non-pointer argument. Finally, you have to use a different function or format flag for each pair of types, e.g., `atoi()` for converting an ASCII string to int, `"%f"` for ASCII to float conversion and so on.

The Standard Library defines a family of string-oriented streams in the header `<sstream>`. These include `std::istringstream` for input, `std::ostringstream` for output and `std::stringstream` for both input and output operations. C++ also defines wide-character versions of these stream classes, namely `std::wistringstream`, `std::wostringstream` and `std::wstringstream`. In the following sections I will show how to use stringstream objects to perform easy, safe, automatic, and object-oriented type conversions

The Problem


Converting strings to numeric values and vice versa usually requires the use of explicit cast operations. The drawbacks are many: buffer overflow bugs, the operations are type-unsafe, and you have to use a different function or format flag for each pair of types.

The Solution

Use stringstream objects to perform easy, safe, automatic, and object-oriented type conversions.

Inserting a Value to a Stream

A stringstream-based conversion consists of two operations: writing the value to a stringstream and extracting it. The stringstream object automatically performs the low-level conversion of the source type to string when you insert a value to it. Likewise, when you extract a value from the stringstream and write it to the target variable, the opposite conversion takes place.

 Note: Do not confuse `<sstream>` classes with the outdated and **deprecated** `<strstream>` family of `char*`-oriented stream classes, namely `strstreambuf`, `istrstream`, `ostrstream`, and `strstream`. Although both libraries offer the same functionality more or less, the newer `<sstream>` library is superior in many respects.

In the following example, we define a stringstream object and insert a variable of type double to it. We use the stringstream overloaded `<<` and `>>` operators for insertion and extraction.

```
#include <sstream>
using namespace std;
int main()
```

AUTOMATING TYPE CONVERSIONS WITH STRING STREAM OBJECTS

By Danny Kalev

```
{
double d=123.45;
stringstream s;
s << d; // insert double into stream
}
```

Note that stringstream objects have a built-in memory manager that takes care of allocating memory as needed. This is a major advantage as it eliminates the drudgery of manual memory management, potential buffer overflows, and memory leaks. To extract the value stored in s to a string, use the >> operator:

```
string val;
s >> value;
Now val contains the string "123.45".
The opposite conversion, namely double to string, is just as simple:
string val="124.55";
stringstream s;
double d=0;
s << val; // insert string
s >> d; // d now equals 124.55
```

Templatizing stringstream-based Conversion

In the previous [10-Minute Solution](#) I showed how to abstract generic operations with templates. Since stringstream-based conversion uses the same interface regardless of the actual type being converted, implementing generic conversion functions is easy.

Converting an Arbitrary Type to String

Our first function template takes a value of an arbitrary type T, converts it to string, and writes the result to a user-supplied string called val. We use the stringstream::str() member function to obtain a copy of the stream's internal string:

```
#include <sstream>
#include <string>
using namespace std;
template <class T>
void string_fmt(string & val, const T & t)
{
    ostringstream oss; // create a stream
    oss << t; // insert value to stream
    val=oss.str(); // extract string and copy
}
```

In the following example, we use the string_fmt() function template to convert the value 10.76 to string and write the result to val:

```
string val;
string_fmt(val,10.76);
```

Converting from One Arbitrary Type to Another

Our second function template is called cast_stream(). It converts a variable of type in_value to a variable of type out_type (in_value and out_value are template parameters):

AUTOMATING TYPE CONVERSIONS WITH STRING STREAM OBJECTS

By Danny Kalev

```
template <class out_type, class in_value>
out_type cast_stream(const in_value & t)
{
    stringstream ss;
    ss << t; // first insert value to stream
    out_type result; // value will be converted to out_type
    ss >> result; // write value to result
    return result;
}
```

In the following example, `cast_stream()` converts a string with the value "18.67" to double using `cast_stream()`:

```
string s="18.67";
double d=cast_stream < double > (s); // assign 18.67 to d
```