

Search:  Go

Articles Converting numbers to strings and string

register


Not logged in

log in

Published by **Bazzy**  
Oct 9, 2009 (last update: Nov 22, 2012)

C++
<a href="#">Information</a> <a href="#">Tutorials</a> <a href="#">Reference</a> <a href="#">Articles</a> <a href="#">Forum</a>
Articles
<a href="#">Algorithms</a> <a href="#">C++ 11</a> <a href="#">Graphics and multimedia</a> <a href="#">How-To</a> <a href="#">Language Features</a> <a href="#">Unix/Linux programming</a> <a href="#">Source Code</a> <a href="#">Standard Library</a> <a href="#">Tips and Tricks</a> <a href="#">Tools and Libraries</a> <a href="#">Visual C++</a> <a href="#">Windows API</a>

## Converting numbers to strings and strings to numbers

 Score: 4.1/5 (610 votes)

Converting numbers to text and vice versa is a common issue as it can be useful in many different situations and C++98 doesn't provide a tool designed specifically to solve this problem.

Luckily C++ is a general purpose language so it allows to solve this quite easily and, as most things, you have many ways of accomplishing this task.

Here are listed some

### Contents:

- C++ - stringstream
  - Number to String
    - Custom Formatting
  - String to Number
  - Simple Sample Functions
- C++11
- C++ - boost library
- C - stdio
- C - stdlib
- Writing your own function

## C++ - stringstream

The C++ stream library is powerful and it allows easy formatted input output operations. With [stringstream](#) you can perform this input/output to string, this allows you to convert numbers ( or any type with the << >> stream operators overloaded ) to and from strings. With [stringstream](#) you can use the same syntax to convert the different numeric types.

To use [stringstream](#) you need to `#include <sstream>`

### Number to String

Converting a number to a string takes two steps using [stringstream](#):

1. Outputting the value of the number to the stream
2. Getting the string with the contents of the stream

As with this conversion needs only output operation with the stream, an [ostringstream](#) ( output string stream ) can be used instead of the stream for both input and output ( [stringstream](#) )

Here is an example which shows each step:

```
1 int Number = 123;           // number to be converted to a string
2
3 string Result;              // string which will contain the result
4
5 ostringstream convert;      // stream used for the conversion
6
7 convert << Number;          // insert the textual representation of 'Number' in the characters in the stream
8
9 Result = convert.str();      // set 'Result' to the contents of the stream
10
11 // 'Result' now is equal to "123"
```

This operation can be shorten into a single line:

```
1 int Number = 123;
2 string String = static_cast<ostringstream*>( &(ostringstream() << Number) )->str();
```

Here is constructed an unnamed [stringstream](#) object and performed the output `ostringstream() << Number` Then, since the << returns a reference to an [ostream](#) ( a base of [ostringstream](#) ) the result of the operation needs to be casted back to a [stringstream](#) `static_cast<ostringstream*>` Finally, we get the contents of the resulting stream as a string `->str()` and we assign that value to the string `string String =`

### Custom formatting

[Stringstreams](#) allow [manipulators](#) and [locales](#) to customize the result of these operations so you can easily change the format of the resulting string

Example: - This is not a complete program -

```
1 // Headers needed:
2
3 #include <iomanip>
4 #include <locale>
5 #include <sstream>
6 #include <string> // this should be already included in <sstream>
7
8
9 // Defining own numeric facet:
10
11 class WithComma: public numpunct<char> // class for decimal numbers using comma instead of point
12 {
13     protected:
14         char do_decimal_point() const { return ','; } // change the decimal separator
15 };
16
17
18 // Conversion code:
19
20 double Number = 0.12;           // Number to convert to string
21
22 ostringstream Convert;
23
24 locale MyLocale( locale(), new WithComma); // Create customized locale
25
26 Convert.imbue(MyLocale);         // Imbue the custom locale to the stringstream
27
28 Convert << fixed << setprecision(3) << Number; // Use some manipulators
29
30 string Result = Convert.str();    // Give the result to the string
31
32 // Result is now equal to "0,120"
```

## String to Number

Also converting a string to a number via stringstream takes two steps:

1. Constructing the stream from the string
2. Reading the value into the variable

For this ( as you need to read input from the stream ) an `istringstream` will be used

While a number can always be converted in a string, a string must be valid to be converted to a number ( eg: An attempt of converting "hello" to an integer would certainly fail ) so on this conversion, some checking must be done

Here is the code example:

```
1 string Text = "456"; // string containing the number
2
3 int Result;           //number which will contain the result
4
5 istringstream convert(Text); // stringstream used for the conversion constructed with the contents of 'Text'
6                               // ie: the stream will start containing the characters of 'Text'
7
8 if ( !(convert >> Result) ) //give the value to 'Result' using the characters in the stream
9     Result = 0;           //if that fails set 'Result' to 0
10
11 // 'Result' now equal to 456
```

This conversion is even easier to reduce to a single line:

```
1 string Text = "456";
2 int Number;
3 if ( ! ( istringstream(Text) >> Number ) ) Number = 0;
```

In the above code an object of `istringstream` gets constructed from 'Text' `istringstream(Text)` and its contents get read into the numeric variable `>> Number`.

If that operation fails `if ( !, 'Number' is set to zero Number = 0;`

Locales and manipulators can be used as well as with any stream

### More complex cases

A generic `stringstream` ( which could be used both for input and for output ) can be useful in some more complex situations and in almost any situation you need to perform operations not provided by `string`

## Simple Sample Functions

Here are listed some functions to perform these conversion using `stringstream`s:

```
1 template <typename T>
2 string NumberToString ( T Number )
3 {
4     ostringstream ss;
5     ss << Number;
6     return ss.str();
7 }
```

Usage: `NumberToString ( Number );`

```
1 template <typename T>
2 T StringToNumber ( const string &Text )
3 {
4     istringstream ss(Text);
5     T result;
6     return ss >> result ? result : 0;
7 }
```

Usage: `StringToNumber<Type> ( String );`

**Notice:** In the code examples `std::` was omitted to make the code simpler

Using the last functions, there is no way of detecting whether the conversion succeeded or failed

## C++11

C++11 introduced some standard library functions that can directly convert basic types to `std::string` objects and vice-versa.

`std::to_string` Converts basic numeric types to strings.

The set of functions

`stoi`, `stol`, `stoll` convert to integral types, the functions

`stof`, `stod`, `stold` to floating-point values.

These functions are declared in `<string>`.

Note that since these functions were added in the latest version of the C++ standard, they may not be available unless your implementation is very recent.

```
1 int number = 123;
2 string text = to_string(number);
3
4 text = "456"
5 number = stoi(text);
```

## C++ - Boost Library

Using `stringstream`s is the standard C++ way of doing these conversions but they usually need a few lines of code

Among the `Boost libraries` there is `lexical_cast` which allows to perform the `stringstream` conversions through simple function call To make this library working, just include the header, it doesn't need to be linked

```
1 // Boost header needed:
2 #include <boost/lexical_cast.hpp>
3
4 // Number to string conversion:
5 Text = boost::lexical_cast<string>(Number);
6
7 // String to number conversion:
```

```
8 Number = boost::lexical_cast<Type>(Text);
```

The above examples don't handle eventual conversion failures

When `boost::lexical_cast` fails, it throws `boost::bad_lexical_cast` ( derived from `std::bad_cast` )

```
1 try
2 {
3     Number = boost::lexical_cast<Type>(Text);
4 }
5 catch ( const boost::bad_lexical_cast &exc ) // conversion failed, exception thrown by lexical_cast and caught
6 {
7     Number = 0; // give 'Number' an arbitrary value ( in this case zero )
8               // if you don't give it any value, it would maintain the value it had before the conversion
9
10    // A string containing a description of the exception can be found in exc.what()
11 }
```

## C - stdio

### Number to String

In C there is no stream library, but the function `sprintf` can be used for conversion

It works in a similar way to `printf` but it will put the characters in a C string ( a character array ) instead of `stdout` Using this is not as easy as with streams as the format string changes depending on the type of the number which needs to be converted

Example:

```
1 int Number = 123; // number to convert
2
3 char Result[16]; // string which will contain the number
4
5 sprintf ( Result, "%d", Number ); // %d makes the result be a decimal integer
```

### String to Number

As `printf`, also `scanf` has a related function which can read from a character array, `sscanf`

```
1 char Text[] = "456"; // string to be converted
2
3 int Result; // number which will contain the result
4
5 sscanf ( Text, "%d", &Result );
```

If `sscanf` fails ( ie: the string is not a number ), the value of the variable passed remains unchanged, in that case the function should return zero as no argument were read successfully, if the string passed is so bad that nothing can be read from it, it would return `E0F`:

```
1 char Text[] = "456"; // string to be converted
2
3 int Result; // number which will contain the result
4
5 int Succeeded = sscanf ( Text, "%d", &Result );
6
7 if ( !Succeeded || Succeeded == E0F ) // check if something went wrong during the conversion
8     Result = 0;
```

## C - stdlib

The `stdlib` header contains some functions to convert text and numbers

Notice that some of these functions are not standard! These functions are:

- `itoa`
- `atoi`
- `atol`
- `atof`
- `strtol`
- `strtoul`
- `strtod`

- For examples refer to the individual reference pages -

### Writing your own function

Using already existing libraries is easier and better but, just to show how some of the above solutions work, here are some examples of how to write functions to convert text to numbers and numbers to text using only the core language, the following examples are from the book *"The C Programming Language"*

Here is `itoa` ( Integer TO Alphabet )

```
1 /* itoa: convert n to characters in s */
2 void itoa(int n, char s[])
3 {
4     int i, sign;
5
6     if ((sign = n) < 0) /* record sign */
7         n = -n;        /* make n positive */
8     i = 0;
9     do {               /* generate digits in reverse order */
10        s[i++] = n % 10 + '0'; /* get next digit */
11    } while ((n /= 10) > 0); /* delete it */
12    if (sign < 0)
13        s[i++] = '-';
14    s[i] = '\0';
15    reverse(s);
16 }
```

Here is the function `reverse` used in `itoa`:

```
1 /* reverse: reverse string s in place */
2 void reverse(char s[])
3 {
4     int i, j;
5     char c;
6 }
```

```

7   for (i = 0, j = strlen(s)-1; i<j; i++, j--) {
8       c = s[i];
9       s[i] = s[j];
10      s[j] = c;
11  }
12 }

```

reverse uses the function strlen from the header cstring ( string.h in C )  
This is easy to implement, here is an example:

```

1  /* strlen: return length of s */
2  int strlen(char s[])
3  {
4      int i = 0;
5      while (s[i] != '\0')
6          ++i;
7      return i;
8  }

```

As you can see, is possible to create a ( bad ) conversion function with just some basic C  
The same applies to the opposite conversion:

```

1  /* atoi: convert s to integer */
2  int atoi(char s[])
3  {
4      int i, n;
5      n = 0;
6      for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
7          n = 10 * n + (s[i] - '0');
8      return n;
9  }

```

Of course these functions are bad for many reasons and should not be used in actual code  
They just show the idea behind the conversion between an integer value and a character sequence