Software Tools: STLFilt

Adam Hasselbalch Hansen
adhh@diku.dk

Christian Riis
criis@diku.dk

Joakim Hovard
chuy@diku.dk

Abstract. This essay describes the usefulness of the tool "STLFilt" for processing and sanitizing error messages from particularly the GCC C++ compiler.

1. Purpose

STLFilt simplifies and/or reformats long-winded C++ error and warning messages, with a focus on STL-related diagnostics. The result renders many of even the most cryptic diagnostics comprehensible[3].

This is particularly useful with errors originating from template use, as compilers notoriously generate confusing and unhelpful error messages when errors are detected in template code.

Comparing the output of a sample test program, designed to produce template errors, shows a significant improvement in legibility, as illustrated in the example in Appendix A.

1.1 Compatibility

According to the website[3], STLFilt supports a wide variety of compilers, including the Microsoft Visual C++ compiler in versions 6.0, 7.0 (.NET) and 8.0, although the latter is somewhat broken when using Miscrosoft's IDE, due to a particularity in the way input is piped to the compiler.

Also, the GNU Compiler Collection, particularly g++, is supported in versions 2.9x and 3.x. GCC 4.0 support is on the way, but due to a radical change in the way GCC provides error message output in version 4, the author needs to rewrite the regular expressions used[4].

Although a wide variety of compilers are supported, we have focused on g++ in versions below 4.0.

A couple of runs, forcing STLFilt to use g++ 4.0 with supplied testing programs, however, suggests that even though unupported, the filter yields results comparable to results with an officially supported version.

2. Evaluation

As it is evident from our example, STLFilt does not really do much but filter error messages, which the name indeed implies. While this does provide somewhat prettier (i.e. shorter) output, it is in itself pretty useless as the software does not rewrite error messages to something more readable.

If one does not understand some error message from `g++` one will not be able to understand STLFilt's output either, but STLFilt can help one find what exactly *is* important by weeding out what is not.

Furthermore, if one, after having seen the filtered output from STLFilt, wants to see the unfiltered or partially filtered output, one would need to rerun STLFilt which would recompile the source code; potentially a time consuming process. Though fixing that would require only a semi-trivial rewrite, it is still somewhat inelegant.

As a last thing it is pretty horrible style-wise to write production ready Perl code without using the `strict` pragma or at the very least `strict 'subs'`. In our opinion, this makes for an inherently untrustworthy piece of software.

Whether this piece of software is useful or not is entirely up to each individual person. Our opinion is that it is not helpful enough to warrant general use. It is, however, useful for template debugging purposes or other situations when the compiler generates long and incomprehensible error messages.

3. The code

We have made a slight change to the code which is worth a mention. For reasons unknown STLFilt outputs to `stdout` which is wrong since it outputs errors. We have changed the output file handle to be user defined, but defaulting to `stderr`.

As we are not allowed for some reason to have the ports needed for anonymous `svn` to DIKU's DMZ (where our svn-server is located) opened the source code (of which there's precious little in this assignment) is available upon request.

All we have done for this assignment is add the following two lines somewhere near the beginning of `gSTLFilt.pl`:

```
1 my $out_thingy = "STDERR";
2
3 open (STDOUT, ">&$out_thingy")
4         or die "Can't redirect to $out_thingy, $!\n"
5         if (defined $out_thingy);
```

4. Other Tools

Another tool, “TextFilt”, exists, which, like STLFilt, uses regular expressions to simplify the errors output by the compiler, but uses “Regex++” found in the Boost-libraries[1]. As such, it is programmed in C++ rather than Perl, providing at least a theoretical performance increase.

TextFilt started its life as a specialized tool for error message processing, but has since been developed to be a general purpose text processing tool. Its homepage has not been updated since September 2002[2], and the tool lacks support for other compilers than GCC.

References

[1] Boost, *Peer-reviewed portable C++ source libraries*. http://www.boost.org.

[2] D. Frey, *TextFilt*. http://textfilt.sourcefourge.net.

[3] L. Zolman, *STL Filt*. http://www.bdsoft.com/tools/stlfilt.html.

[4] L. Zolman, *STL Filt Forums*, Internet Forum. http://bdsoft.proboards34.com/index.cgi?board=talk&action=display&thread=1144527430.

Appendix A. Sample output

Appendix A.1 Using g++

```
testvec.cpp: In function ‘int main(int, char**)’:
testvec.cpp:13: error: invalid conversion from ‘const char*’ to ‘unsigned int’
testvec.cpp:13: error:   initializing argument 1 of ‘std::vector<_Tp,
   _Alloc>::vector(unsigned int) [with _Tp = int, _Alloc = std::allocator<int>]
   ’
testvec.cpp:14: error: no matching function for call to ‘std::vector<Widget,
   std::allocator<Widget> >::vector(int, int, int, int, int, int)’
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/stl_vector
.h:265: error: candidates
   are: std::vector<_Tp, _Alloc>::vector(const std::vector<_Tp, _Alloc>&) [with
   _Tp = Widget, _Alloc = std::allocator<Widget>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/stl_vector
.h:252: error:
                  std::vector<_Tp, _Alloc>::vector(unsigned int) [with _Tp =
   Widget, _Alloc = std::allocator<Widget>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/stl_vector
.h:240: error:
                  std::vector<_Tp, _Alloc>::vector(unsigned int, const _Tp&,
   typename std::_Vector_base<_Tp, _Alloc>::allocator_type&) [with _Tp =
   Widget, _Alloc = std::allocator<Widget>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/stl_vector
.h:229: error:
                  std::vector<_Tp, _Alloc>::vector(typename
   std::_Vector_base<_Tp, _Alloc>::allocator_type&) [with _Tp = Widget, _Alloc
   = std::allocator<Widget>]
testvec.cpp:15: error: ‘foobar’ undeclared (first use this function)
testvec.cpp:15: error: (Each undeclared identifier is reported only once for
   each function it appears in.)
testvec.cpp:16: error: no matching function for call to ‘
```

```
   std::basic_string<char, std::char_traits<char>, std::allocator<char> >::
   basic_string(int, int, int, int, int)'
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/basic_stri
ng.tcc:233: error: candidates
   are: std::basic_string<_CharT, _Traits, _Alloc>::basic_string(typename
   _Alloc::size_type, _CharT, const _Alloc&) [with _CharT = char, _Traits =
   std::char_traits<char>, _Alloc = std::allocator<char>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/basic_stri
ng.tcc:226: error:
                 std::basic_string<_CharT, _Traits,
   _Alloc>::basic_string(const _CharT*, const _Alloc&) [with _CharT = char,
   _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/basic_stri
ng.tcc:220: error:
                 std::basic_string<_CharT, _Traits,
   _Alloc>::basic_string(const _CharT*, typename _Alloc::size_type, const
   _Alloc&) [with _CharT = char, _Traits = std::char_traits<char>, _Alloc =
   std::allocator<char>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/basic_stri
ng.tcc:213: error:
                 std::basic_string<_CharT, _Traits,
   _Alloc>::basic_string(const std::basic_string<_CharT, _Traits, _Alloc>&,
   typename _Alloc::size_type, typename _Alloc::size_type, const _Alloc&) [with
   _CharT = char, _Traits = std::char_traits<char>, _Alloc =
   std::allocator<char>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/basic_stri
ng.tcc:205: error:
                 std::basic_string<_CharT, _Traits,
   _Alloc>::basic_string(const std::basic_string<_CharT, _Traits, _Alloc>&,
   typename _Alloc::size_type, typename _Alloc::size_type) [with _CharT = char,
   _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/basic_stri
ng.tcc:192: error:
                 std::basic_string<_CharT, _Traits,
   _Alloc>::basic_string(const std::basic_string<_CharT, _Traits, _Alloc>&)
   [with _CharT = char, _Traits = std::char_traits<char>, _Alloc =
   std::allocator<char>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/basic_stri
ng.tcc:199: error:
                 std::basic_string<_CharT, _Traits,
   _Alloc>::basic_string(const _Alloc&) [with _CharT = char, _Traits =
   std::char_traits<char>, _Alloc = std::allocator<char>]
/usr/lib/gcc-lib/i686-pc-linux-gnu/3.3.5-20050130/include/g++-v3/bits/basic_stri
ng.h:862: error:
                 std::basic_string<_CharT, _Traits, _Alloc>::basic_string()
   [with _CharT = char, _Traits = std::char_traits<char>, _Alloc =
   std::allocator<char>]
```

Appendix A.2 Using STLFilt

```
BD Software STL Message Decryptor v2.47a for gcc
testvec.cpp: In function `int main(int, char **)':
testvec.cpp:13: error: invalid conversion from `const char *' to `unsigned int'
testvec.cpp:13: error:   initializing argument 1 of
   `vector<int>::vector(unsigned int)'
testvec.cpp:14: error: No match for
   `vector<Widget>::vector(int, int, int, int, int)'
testvec.cpp:15: error: `foobar' undeclared (first use this function)
testvec.cpp:16: error: No match for 'string(int, int, int, int, int)'
```

```
STL Decryptor reminder:
    Use the /cand:L option to see all suppressed template candidates
```