Tampere (Finland) / Offenburg (Germany), 1 February 2013

Please be informed that a new **CTC++ version 7.1** has been released.

This revision 7.1 of CTC++ has the following version numbers in its components:

Preprocessor	7.1	(was 7.0.2; seen by -h option)
Run-time libraries	7.1	(was 7.0.2; seen by 'ident' command applied on the library in some environments)
Postprocessor	7.1	(was 7.0.2; seen by -h option and in the listings)
Header file ctc.h	7.1	(was 7.0; seen in the file)
Configuration file ctc.ini	7.1	(was 7.0.2; seen in the file)
CTC++ to HTML Converter	3.2	(was 3.1; seen by -h option)
CTC++ to Excel Converter	2.0	(unchanged; seen by -h option)
CTC++ Merger utility	2.0	(unchanged; seen by -H option and in the merged listings)
ctc2dat receiver utility	3.1	(was 3.0; seen by -h option)

and the following version numbers in its Windows platform specific components:

Visual Studio IDE Integration	4.1	(was 4.0; seen by clicking the Tw-icon in the dialog program and selecting "About...")
CTC++ Wrapper for Windows	3.0	(unchanged; seen by -h option)

and the following version numbers in its Unix platform (Linux, Solaris, HPUX) specific components:

CTC++ Wrapper for Unix	1.3	(unchanged; seen by -h option)

This CTC++ v7.1 version contains enhancements and bug fixes:

In the CTC++ preprocessor (ctc):

- New: Various new C++11 standard constructs are now properly handled
  (and instrumented where appropriate)
  -- New keywords: alignas, alignof, char16_t, char32_t, constexpr,
     decltype, noexcept, nullptr, static_assert, thread_local
     (In particular, keywords char16_t, char32_t and decltype are
     recognized in a function parameter list and in a declaration
     in condition.)
  -- Trailing return type of a function, e.g.,
        auto f() -> int { ... }

```
  -- The noexcept or noexcept(...) specification in functions, e.g.,
        auto bar() noexcept(f(v.at(0))) -> int { ... }
  -- New kind of member initializers, e.g.,
        S(int x, double y) : m{x}, n{y} {}
  -- Defaulted or deleted functions, e.g.,
        NonCopyable() = default;
  -- The specifiers 'new', 'final' and 'override' in functions, e.g.,
        virtual void f() const final;
        void g(int) override { ... }
        virtual void h(char *) new { ... }
  -- The specifiers 'final' and 'explicit' in classes, e.g.,
        class C final { ... };
        struct D explicit : public B { ... };
  -- Attribute specifiers: [[...]]
  -- Scoped and strongly typed enums (enum class/struct; underlying
     type), e.g.,
        enum class Color : unsigned int { black, white };
  -- Lambdas (lambda functions):
     - lambdas in global scope are instrumented like normal functions;
       in reports they are shown with the name "lambda-[]"
     - lambdas inside normal functions: the outermost lambdas are
       instrumented while inner ones as well as lambdas in return
       statements etc. are not instrumented; shown "lambda-[]()"
       in reports
  -- The 'constexpr' specifier prevents all instrumentation
     (of a declaration or definition that it is applied to)
  -- Templates: the '>>' token is treated as two right angle brackets
     instead of a right shift operator in the following kind of
     cases
         template <typename T = X<int>> class A ...
     (In nested instantiations, e.g., T1<T2<T3>>, '>>' has been
     supported already as of v6.5.2)
  -- Range-based loop statement, e.g.,
        for (auto x : v) { ...
     is instrumented with a counter in the beginning of the loop body
  -- Suffixes (syntactically identifiers) in string and character
     literals, e.g.,
        "1234"ssuffix, '?'csuffix

- Bug fix: Now accepting extension __restrict (by VC++ and GCC) and
  __restrict__ (by GCC) after a function's parameter list, e.g.,
     void C::memb() __restrict { ... }
  (Previously such functions were not instrumented.)

- New: the loop constructs 'while' and 'for' declaration in condition,
  'for each' (by VC++), and 'foreach' (by C#), e.g.
      while (int i = f()) { ...           // C++
      for (...;int i = f();...) { ...     // C++
      for each (Char c in MyString) { ... // VC++ extension
      foreach (String str in list) { ...  // C#
  These are now instrumented in the same way as the C++11 range-based
  loop having one counter in the beginning of the loop body.
```

- New: The following kind of template declaration/definition is now handled correctly
  ```
  template <class T, bool B = 1 < 2> class X ...
  ```
  Note the relational operator '<' between the angle brackets!
  (This seems to be accepted by some compilers, not by some others!)

- Change: Single-line comments are now supported in compiler/linker response files and in ctc's own response files. If the very first character in the response file line is '#', the whole line is treated as a comment.

- New: New configuration parameters EXT_JAVA and EXT_CSHARP. These are partly "future reservations", except that C#'s foreach loop is recognized only if the source file extension is listed in EXT_CSHARP.

- New: Java's operator '>>>' is now correctly recognized.

- Bug fix: If there were, after the keyword class, struct, or union, consecutively two or more such specifiers that take arguments in parentheses, e.g., __declspec(...) or __attribute__((...)), ctc did not recognize the class correctly, and its inline member functions were not instrumented. For example,
  ```
  struct __declspec(...) __declspec(...) S { ... };
  class __attribute__((...)) __attribute__((...)) C { ... };
  ```

- Bug fix: When ctc sees a command like
  ```
  ctc possible_ctc_options command options
  ```
  i.e. the compile or link 'command' has no file arguments, ctc just executes the command with its options. Previously, ctc tried to link the CTC++ run-time library into the "target" (unless the configured command TYPE was just 'compiler').

- Change: A misplaced #pragma CTC COUNT/APPEND/INIT/QUIT causes only a warning. Previously this was a syntax error.

In the CTC++ run-time library:

- New: On Windows platform, added libctc64.a which is a 64-bit GCC import library for the CTC++ run-time library, ctcmsnt64.dll (MinGW/Cygwin 64-bit code use at Windows).

In the CTC++ postprocessor (ctcpost):

- Bug fix: In statement coverage calculation corrected a bug when at the last "}" of a "switch() {}" or "try {}...catch(e) {}" construct it was tried to determine if the statements after the last "}" are flown-to in execution. Previously a function could get statement coverage hits even if the function was not called at all.

- Bug fix: In statement coverage calculation corrected a bug in goto statement execution flow analyses. Previously too high TER% could be reported under certain conditions.

- Change: In structural and statement coverage TER%, the 0.5 rounding is done in a new way at 0.0-0.5 and at 99.5-100.0. Now 0% is reported, when there are no hits whatsoever, and 100% is reported only when everything is covered.

In CTC++ to HTML converter (ctc2html):

- Bug fix: Corrected HTML report starting problem in 64-bit Windows.

In Visual Studio IDE integration (Windows version only):

- Enhancement: Improvements in installation
  -- support on VS2010 and VS2012 (no more manual step needed)
  -- CTC++ is set usable on all VS platforms (previously only on Win32)
  -- support on some Visual Studio Express variants
  See details from Vs_integ folder.

In Sym_cw subfolder (Windows version only):

- Removed: CTC++ support for Codewarrior compiler (Symbian EPOC emulator use) is dropped off. The whole folder is removed.

General:

- Compatibility: The CTC++ files that have been created with CTC++ v7.0.x can be further used with CTC++ v7.1.

- CTC++ User's Guide upgraded to v7.1 level (ctcug.pdf).

Version 7.0.2 (29 February 2012)

For this version, please have a look to
http://www.verifysoft.com/ctcpp702.pdf