SAMPLE CODECHECK RULE FILE FOR A CORPORATE C++ STYLE GUIDELINE

```
/*      xyzrule.cc
        Copyright (c) 1996 by XYZ, Inc. All rights reserved.
=============================================================================
        Purpose:        Checks for compatibility with XYZ C++ standards.
        Written:        1 December 1995.
        Revised:        2 January 1996.
        Format: Monospaced font with 4 spaces/tab.
=============================================================================
abstract:

        These CodeCheck rules check for compatibility with the XYZ C++
        Coding Rules. These rules are based on the document "XYZ C++
        Guidelines".

        These rules are applied to all headers that are included in double
        quotes, e.g. #include "project.h". For proper use of these rules, be
        sure to include system headers in angle brackets: #include <ctypes.h.

Warning Codes:

9111    Header filename  is not in DOS 8.3 format.
9121    Class names must begin with "XYZxxx_
9131    Private member name  must not begin in uppercase.
9131    Private member name  must end with an underscore.
9211    Definition of class  belongs in a header file.
9212    File  needs a leading comment block.
9213    Header file  should be wrapped in an #ifndef.
9213     is not the correct wrapper name for this file.
9214    Definition of function  must NOT be in a header file.
9221    Public section must come first in class .
9222    Data member  of class  must be private.
9231    Function  is too long to be inlined.
9232    Do NOT use inline within a class definition.
9251    Class  needs a default constructor.
9251    Class  needs a copy constructor.
9252    Class  needs an operator=().
9253    Class  has too many constructors (limit is 3).
9254    Class  needs a destructor.
9255    Destructor for class  must be virtual.
9261    Operator  should not be a friend.
9261    Do NOT declare friend functions.
9261    Do NOT declare friend classes.
9271    Do NOT use virtual base classes.
9281    Declare  using a typedef name, not a basic C type.
9282    Define typedef name  in a base class.
9293    Define enum  in a base class.
9311    Declare parameter  to be a reference to .
9312    Operator  should not return an object.
9314    Function  should not return an object.
```

```
9315    Reference parameters must come first.
9316    Constant member functions should be avoided.
9317    Parameter  should not be const.
9411    Use // comments, not oldstyle C comments.
9421    Declare  as a const, not a macro.
9422    This enumeration needs an enum type name.
9431    Global constant  should be a class member.
9441    Function  needs an explicit return type.

================================================================================
*/

#include        <check.cch

#define DOT                     ('.')           //      period character
#define SLASH                   ('/')           //      forward slash character
#define BACKSLASH               ('\\')          //      backslash character
#define COLON                   (':')           //      colon character
#define TILDE                   ('~')           //      tilde character
#define UNDERSCORE      ('_')           //      underscore character

#define PUBLIC          0
#define PROTECTED               1
#define PRIVATE         2


int     ch, j, k, okay, level,
                is_constant, is_object,
                lin_if_depth,                   //  Holds latest value of
pp_if_depth.
                comment_needed,         //      True until a header file's
comment-block is found.
                define_needed,          //      True until a header file's wrapper
macro is #defined.
                public_needed,          //      True when a class definition begins.
                one_liner_needed,               //      True when a function is
explicitly inlined.
                no_wrap_message,                //  True if a wrapper-needed message
has NOT been given.
                class_has_virtual_function,
                destructor_is_virtual,
                non_ref_parm_found,     //      True if a non-reference fcn parameter
has been found.
                detect_virtual_base,
                bad_name,                               //  True if the wrapper name
is wrong.
                length, ch1, ch2;



// ---------- Rule 1.1.1 ----------

if ( header_name() )            //      A header is about to be opened
        {
        if ( pp_include <3 )    //      Header filename is in double quotes
```

```
                {
                j = 0;
                k = 0;
                ch = header_name()[k++];
                while ( ch != 0 )
                        {
                        ch = header_name()[k++];
                        if ( ch == DOT )
                                {
                                break;          //      beginning of extension found
                                }
                        else
                                {
                                j = k;          //      count characters before dot
                                }
                        if ( ch == BACKSLASH || ch == COLON )
                                {
                                j = 0;          //      DOS directory or disk marker
                                }
                        else if ( ch == SLASH || ch == TILDE )
                                {
                                j = 0;          //      Unix directory or disk marker
                                }
                        }
                if ( j > 8 )
                        {
                        warn( 9111, "Header filename %s is not in DOS 8.3 format.",
                                        header_name() );
                        }
                else if ( ch == DOT )
                        {
                        j = 0;
                        ch = header_name()[k+j];
                        while ( ch != 0 )
                                {
                                j++;
                                ch = header_name()[k+j];        //      count chars
after dot.
                                }
                        if ( j  3 )
                                {
                                warn( 9111, "Header filename %s is not in DOS 8.3
format.",
                                                header_name() );
                                }
                        }
                }
        }



// ---------- Rule 1.2.1 ----------

if ( tag_begin )
        {
```

```
        //      Apply this rule to global classes only:

        if ( tag_global && (tag_kind == CLASS_TAG) )
                {
                if ( ! prefix("XYZ") )
                        warn( 9121, "Class names must begin with \"XYZ_\"" );
                else if ( strstr(tag_name(), "_") == 0 )
                        warn( 9121, "Use an underscore after the class name prefix."
);
                }
        }



// ---------- Rule 1.3.1 ----------

if ( dcl_member == 3 )
        {
        if ( dcl_access == PRIVATE )
                {
                if ( dcl_variable )
                        {
                        if ( isupper(dcl_name()[0]) )
                                warn( 9131, "Private data member name %s must not
begin in uppercase.",
                                                dcl_name() );
                        if ( ! suffix("_") )
                                warn( 9131, "Private data member name %s must end
with an underscore.",
                                                dcl_name() );
                        }
                else if ( dcl_function )
                        {
                        if ( isupper(dcl_name()[0]) )
                                warn( 9131, "Private function name %s must not begin
in uppercase.",
                                                dcl_name() );
                        if ( ! suffix("_") )
                                warn( 9131, "Private function name %s must end with
an underscore.",
                                                dcl_name() );
                        }
                }
        }



// ---------- Rule 2.1.1 ----------

if ( tag_begin )
        {
        if ( tag_global && (tag_kind == CLASS_TAG) )
                {
                if ( lin_source )
                        warn( 9211, "Definition of class %s belongs in a header
```

```
file.",
                                        tag_name() );
                }
        }



// ---------- Rules 2.1.2 ----------

if ( mod_begin )
        {
        comment_needed = FALSE;
        }

if ( lin_end )
        {
        if ( lin_number == 1 )
                {
                if ( lin_is_comment )
                        {
                        comment_needed = FALSE;
                        }
                else if ( lin_header )
                        {
                        comment_needed = TRUE;
                        }
                }

        if ( comment_needed )
                {
                if ( lin_is_comment )
                        {
                        comment_needed = FALSE;
                        }
                else if ( lin_preprocessor )
                        {
                        warn( 9212, "File %s needs a leading comment block.",
                                        file_name() );
                        comment_needed = FALSE;
                        }
                else if ( lin_has_code )
                        {
                        warn( 9212, "File %s needs a leading comment block.",
                                        file_name() );
                        comment_needed = FALSE;
                        }
                }
        }



// ---------- Rule 2.1.3 ----------

if ( pp_if_depth || pp_endif )
        {
```

```
        lin_if_depth = pp_if_depth;
        }

if ( pp_include )
        {
        if ( lin_header )
                {
                if ( no_wrap_message )
                        {
                        warn( 9213, "Header file %s should be wrapped in an
#ifndef.",
                                        file_name() );
                        }
                }

        no_wrap_message = TRUE;
        define_needed = TRUE;
        }

if ( lin_dcl_count )
        {
        if ( lin_header && lin_if_depth == 0 )
                {
                if ( no_wrap_message )
                        {
                        warn( 9213, "Header file %s should be wrapped in an
#ifndef.",
                                        file_name() );
                        no_wrap_message = FALSE;
                        define_needed = FALSE;
                        }
                }
        }

if ( pp_macro )
        {
        if ( lin_header )
                {
                if ( no_wrap_message && lin_if_depth == 0 )
                                {
                                warn( 9213, "Header file %s should be wrapped in an
#ifndef.",
                                                file_name() );
                                no_wrap_message = FALSE;
                                define_needed = FALSE;
                                }
                if ( define_needed && lin_if_depth == 1 )
                        {
                        bad_name = FALSE;
                        length = strlen( pp_name() );
                        k = 0;
                        while ( k  1 )
        {
        if ( one_liner_needed )
                {
```

```
                warn( 9231, "Function %s is too long to be inlined.",
                                fcn_name() );
                }
        }



// ---------- Rule 2.3.2 ----------


if ( dcl_inline )
        {
        if ( lin_within_class == 1 )
                {
                warn( 9232, "Do NOT use inline within a class definition. (%d)",
lin_within_class );
                }
        }



// ---------- Rule 2.4.1 ----------

//      *** This needs a new trigger in CodeCheck: dcl_override. ***



// ---------- Rules 2.5.1, 2.5.2, and 2.5.4 ----------

if ( tag_end )
        {
        if ( tag_global && (tag_kind == CLASS_TAG) )
                {
                if ( ! tag_has_default )
                        {
                        warn( 9251, "Class %s needs a default constructor.",
class_name() );
                        }
                if ( ! tag_has_copy )
                        {
                        warn( 9251, "Class %s needs a copy constructor.",
class_name() );
                        }
                if ( ! tag_has_assign )
                        {
                        warn( 9252, "Class %s needs an operator=(%s&).",
class_name(),
                                        class_name() );
                        }
                }
        }



// ---------- Rule 2.5.3 ----------
```

```
if ( tag_constructors  3 )
        {
        warn( 9253, "Class %s has too many constructors (limit = 3).",
                        class_name() );
        }



// ---------- Rules 2.5.4 and 2.5.5 ----------

if ( tag_begin )
        {
        if ( tag_global && (tag_kind == CLASS_TAG) )
                {
                class_has_virtual_function = FALSE;
                destructor_is_virtual = FALSE;
                }
        }

if ( dcl_function )
        {
        if ( dcl_virtual )
                {
                if ( dcl_base == DESTRUCTOR_TYPE )
                        {
                        destructor_is_virtual = TRUE;
                        }
                else
                        {
                        class_has_virtual_function = TRUE;
                        }
                }
        }

if ( tag_end )
        {
        if ( tag_global && (tag_kind == CLASS_TAG) )
                {
                if ( tag_has_destr )
                        {
                        if ( class_has_virtual_function )
                                {
                                if ( ! destructor_is_virtual )
                                        {
                                        warn( 9255, "Destructor %s::~%s() must be
virtual.",
                                                        class_name(), class_name() );
                                        }
                                }
                        }
                else
                        {
                        warn( 9254, "Class %s needs a destructor.", class_name() );
                        }
```

```
                }
        }



// ---------- Rules 2.6.1 and 2.6.2 ----------

if ( dcl_friend )
        {
        if ( dcl_function )
                {
                if ( prefix("operator") )
                        {
                        if ( strequiv(root(), "=") )
                                ;
                        else if ( strequiv(root(), "+") )
                                ;
                        else if ( strequiv(root(), "-") )
                                ;
                        else if ( strequiv(root(), "*") )
                                ;
                        else if ( strequiv(root(), "/") )
                                ;
                        else if ( strequiv(root(), "%") )
                                ;
                        else if ( strequiv(root(), "") )
                                ;
                        else if ( strequiv(root(), "<<") )
                                ;
                        else
                                warn( 9261, "%s should not be a friend.", dcl_name()
);
                        }
                else
                        warn( 9261, "Do NOT declare friend functions." );
                }
        else
                {
                warn( 9262, "Do NOT declare friend classes." );
                }
        }



// ---------- Rule 2.7.1 ----------

if ( keyword("class") )
        {
        detect_virtual_base = TRUE;
        }

if ( keyword("virtual") )
        {
        if ( detect_virtual_base )
                {
```

```
                warn( 9271, "Do NOT use virtual base classes." );
                }
        }

if ( dcl_base == CLASS_TYPE )
        {
        detect_virtual_base = FALSE;
        }



// ---------- Rule 2.8.1 ----------

if ( dcl_member )
        {
        if ( dcl_variable && dcl_base != DEFINED_TYPE )
                {
                warn( 9281, "Declare %s using a typedef name, not a basic C type.",
                                dcl_name() );
                }
        }



// ---------- Rule 2.8.2 ----------

if ( dcl_typedef )
        {
        if ( dcl_member == 0 )
                {
                warn( 9282, "Define typedef name %s in a base class.",
                                dcl_name() );
                }
        }



// ---------- Rule 2.9.1 ----------

//      This rule is not currently enforceable with CodeCheck.



// ---------- Rule 2.9.2 ----------

//      This rule is redundant (covered by 3.1.2).



// ---------- Rule 2.9.3 ----------

if ( tag_kind == ENUM_TAG )
        {
        if ( tag_global )
                {
```

```
                warn( 9293, "Define enum %s in a base class.", tag_name() );
                }
        }



// ---------- Rule 3.1.1 ----------

if ( dcl_parameter )
        {
        is_object = ((dcl_base == CLASS_TYPE) || (dcl_base == STRUCT_TYPE));
        if ( is_object )
                {
                if ( (dcl_levels == 0) || ((dcl_levels == 1) && (dcl_level(0) ==
POINTER)) )
                        {
                        if ( dcl_axyztract )
                                {
                                warn( 9311, "Declare parameter #%d to be a reference
to %s.",
                                                dcl_parameter, dcl_base_name() );
                                }
                        else
                                {
                                warn( 9311, "Declare parameter %s to be a reference
to %s",
                                                dcl_name(), dcl_base_name() );
                                }
                        }
                }
        }



// ---------- Rules 3.1.2 and 3.1.4 ----------

if ( dcl_function )
        {
        is_object = ((dcl_base == CLASS_TYPE) || (dcl_base == STRUCT_TYPE));
        if ( dcl_member && is_object )
                {
                if ( (dcl_levels == 1) || ((dcl_levels == 2) && (dcl_level(1) ==
REFERENCE)) )
                        {
                        if ( prefix("operator") )
                                {
                                if ( strequiv(root(), "=") )
                                        ;
                                else if ( strequiv(root(), "+") )
                                        ;
                                else if ( strequiv(root(), "-") )
                                        ;
                                else if ( strequiv(root(), "*") )
                                        ;
                                else if ( strequiv(root(), "/") )
```

```
                                ;
                        else if ( strequiv(root(), "%") )
                                ;
                        else if ( strequiv(root(), "") )
                                ;
                        else if ( strequiv(root(), "<<") )
                                ;
                        else
                                {
                                warn( 9312, "%s::%s() should not return an
object.",
                                                class_name(), dcl_name() );
                                }
                        }
                else
                        {
                        warn( 9314, "Function %s::%s() should not return an
object.",
                                        class_name(), dcl_name() );
                        }
                }
        }



// ---------- Rule 3.1.3 ----------

//      This rule cannot be enforced with this version of CodeCheck



// ---------- Rule 3.1.5 ----------

if ( dcl_parameter )
        {
        if ( dcl_parameter == 1 )
                {
                non_ref_parm_found = FALSE;
                }

        if ( dcl_level(0) == REFERENCE )
                {
                if ( non_ref_parm_found )
                        {
                        warn( 9315, "Reference parameters must come first." );
                        }
                }
        else    //      a non-reference fcn parameter has been found
                {
                non_ref_parm_found = TRUE;
                }
        }
```

```
// ---------- Rule 3.1.6 ----------

if ( dcl_function )
        {
        is_constant = (dcl_level_flags(0) & CONST_FLAG);
        if ( dcl_member && is_constant )
                {
                warn( 9316, "Constant member functions should be avoided." );
                }
        }



// ---------- Rule 3.1.7 ----------

if ( dcl_parameter )
        {
        is_object = ((dcl_base == CLASS_TYPE) || (dcl_base == STRUCT_TYPE));
        if ( is_object && (dcl_levels == 1) )
                {
                is_constant = (dcl_level_flags(1) & CONST_FLAG);
                level = dcl_level( 0 );
                if ( is_constant && ((level == POINTER) || (level == REFERENCE)) )
                        {
                        if ( dcl_axyztract )
                                {
                                warn( 9317, "Parameter #%d should not be const.",
                                                dcl_parameter );
                                }
                        else
                                {
                                warn( 9317, "Parameter %s should not be const.",
dcl_name() );
                                }
                        }
                }
        }



// ---------- Rule 4.1.1 ----------

if ( lex_c_comment )
        {
        warn( 9411, "Use /\/ comments, not \/\*...*\/ comments." );
        }



// ---------- Rule 4.2.1 ----------

if ( pp_const )
        {
        if ( lin_header && ! define_needed )
```

```
                {
                warn( 9421, "Declare %s as a const, not a macro.", pp_name() );
                }
        }



// ---------- Rule 4.2.2 ----------

if ( tag_anonymous )
        {
        if ( tag_global && tag_kind == ENUM_TAG )
                {
                warn( 9422, "This enumeration needs an enum type name." );
                }
        }



// ---------- Rule 4.3.1 ----------

if ( dcl_variable )
        {
        is_constant = (dcl_level_flags(dcl_levels) & CONST_FLAG);
        if ( dcl_global && is_constant )
                {
                warn( 9431, "Global constant %s should be a class member.",
                                dcl_name() );
                }
        }



// ---------- Rule 4.4.1 ----------

if ( dcl_no_specifier )
        {
        if ( dcl_function )
                {
                warn( 9441, "Function %s needs an explicit return type.",
                                dcl_name() );
                }
        }



// ---------- Rule 4.4.2 ----------

//      This rule cannot be enforced with this version of CodeCheck.
```