IMPROVED
COMPILE TIME CONFIGURATION

Proposal for a new work item

Author : Jan Kristoffersen, Denmark, jkristof@ramtex.dk

Background

Ten years ago programmers typically wrote code which should only be compiled with one specific compiler using one specific operating system and be running at one specific hardware platform.

Today the situation is the nearly the opposite. It is common that programmers write source modules which should be compiled with different compilers, should be compiled for different operating systems and/or for different hardware platforms.

This, together with increasing demands for shorter time to market, reduced development and maintenance cost, plus the fact that systematic software reuse had become more widespread, creates a growing demand for more flexible compile time configuration possibilities in C.

In particular there is a requirement for that multiple configuration parameters can be used to “shape” a source module at compile time for a particular environment without the risk of making the source module incomprehensive or difficult to maintain.

The current standard only give a programmer two choices, to use multiple (nearly identical) C modules or to use preprocessing directives like **#if #else #endif**

As an increasing number of compiler, system and environment combinations have to be supported maintenance become a nightmare if multiple parallel versions of source files must exist to handle the configuration cases. If preprocessing directives are used for configuration management the source code tend to become much more unreadable and incomprehensive.

It may be possible to handle multi environment configuration by use of third party configuration or preprocessing tools, however in practice this is often a very cumbersome solution and definitely a very poor substitute for having more flexible compile time configuration methods supported by the C language itself.

Therefore it is proposed that the committee adapt “Improved compile time configuration” as a new work item.

A proposed solution

A relatively simple way to improve compile time configuration considerably is to introduce a new kind of selection statements were the controlling expression is evaluated after translation phase 4 and before the existing translation phase 5. Code blocks not selected are skipped from further processing.

For the sake of the discussion let us call the new selection statements *compile time selectors* : *_if* and *_switch*.

Example:

```
_if ( <const expression at compile time> )
   {
   /* If expression is true this code block is visible */
   }
_else
   {
   /* If expression is true this code block is skipped */
   }


_switch ( <const expression at compile time> )
   {
   _case <const value> :
       {
       /* If switch expression is equal to the case label this code
          block is visible, otherwise it is skipped */
       }
   _default:
       {
       /* This code block is visible if no other case labels match,
          otherwise it is skipped */
       }
   }
```

To handle *compile time selectors* section 5.1.1.2, Translation phase 4 should be modified to:

4a. Preprocessing directives are executed, macro invocations are expanded, and **_Pragma** unary operator expressions are executed. If a character sequence that matches the syntax of a universal character name is produced by token concatenation (6.10.3.3), the behavior is undefined. A **#include** preprocessing directive causes the named header or source file to be processed from phase 1 through phase 4a, recursively.

4b. All preprocessing directives are then deleted. Constant expressions in *compile time selectors* are evaluated and code blocks not selected are deleted.

There is several advantage with this solution:

- It can be used both inside function bodies and inside macro bodies (unlike *#if*)
    - Configurable C modules can be made more readable by use of macro functions without the runtime penalties associated with function calls.
    - Macros can now use configuration parameters to select between different flavors of implementations at compile time.
- It promotes the use of parameter based configuration to make configuration simpler and more easily portable.
    - It solve the problem we have today that a compile time configuration become unnecessary complex if the number of instances where a macros is used grow significantly.
    - It promotes the idea with parameter based configuration which is to move the burden from the person who are doing the configuration (and may have little expertise) to the programmer / implementer (which have the expertise).
- The _switch statement is much more readable than the use of various combinations of *#if* preprocessing directives which are the alternative today. Maintenance become easier.
- It is possible to have compile time configuration in macros and still write strongly conforming code. Functions and identifiers used in the code block not selected by the constant expression need not be declared nor defined.
  (If *if (<constant expression>)* was used for the same purpose one will get code that only works if the compiler does *not* produce diagnostics in the presence of several items of undefined behavior).
- It does not break existing code.

This solution example above is just one way that compile time configuration can be improved, other solutions may be just as versatile. The important issue here is that the committee should take a step towards codifying an existing and growing requirement from the market place.