Document number:	P2039R0
Date:	2020-01-01
Project:	do_until Loop
Reply-to:	Menashe Rosemberg <rosemberg@ymail.com>

I. Table of Contents

Proposal to include a new primitive loop ‘do_until’.

II. Introduction

The primitive loops in c++ are: ‘while’, ‘for’ and ‘do...while’.

a. Loop ‘while’ is a loop where the conditions is available before the inner statements. In other words, the statements may or may not be performed.
b. Loop ‘for’ is similar to while with the difference its syntax propose a flow manage by a local variable. Important to note, like while, the condition is available before the inner statements execution.
c. Loop ‘do...while’ is different from the others because the condition is available only in the end of loop. That means the inner statements are performed at least once, always.

What we don’t have is a loop ‘for’ based on do...while loop where the inner statements are performed at least once before the condition evaluation.

III. Motivation and Scope

The introduction of this new loop will give more refinement and clarification to the code and robustness to the language.

Developers of any level can benefit of it. In fact, this kind of practice is widespreaded, but it is not notice. It is not unusual to face a situation where the loop ‘do..while’ is managed by a variable like a ‘for’ loop does without the advantage of local variable declaration and an elegant syntax.

IV. Impact on the Standard

The implementation of this new loop type has no dependence and no changes to standard components are required and no external language or library features of C++11.

VI. Technical Specifications

The syntax proposal follows the loop ‘for’ syntax:

```
do_until ( [ (init-statement) ]; [ condition ]; [ increment-statement ])
{
        ...statements...
}
```

Where ‘do_until’ is a new keyword

Bellow a use case with all loops to demonstrate the advantage of this new loop characteristic.

The templated function prints the index of an array in crescent order:

```
// While ---------------------------------------------------------------
//
// Cons: The life time of variable Size doesn’t end when the loop has
//       finished. It is an ugly code. It has room for inefficacy
//       (compiler implementation)
// Pros: The variable Size is not evaluated in the first loop.
//
template <typename obj, size_t S, enable_if_t <S != 0, size_t> = 1>
void While_Example(const std::array<obj, S>& Obj) {
     size_t Size = 0;
     while (true) {
           std::cout << ‘\n’ << Size;
           ++Size;
           if (Size == Obj.size())
              Break;
     }
}


// for -----------------------------------------------------------------
//
// Cons: The variable Size is checked for the first time before the loop
//       starts unnecessarily for this use case.
// Pros: The life time of variable Size ends when the loop has finished.
//       The code is elegant and easy to read.
//
template <typename obj, size_t S, enable_if_t <S != 0, size_t> = 1>
void For_Example(const std::array<obj, S>& Obj) {
     for (size_t Size = 0; Size < Obj.size(); ++Size)
         std::cout << ‘\n’ << Size;
}


// do...while ----------------------------------------------------------
//
// Cons: The life time of the variable Size doesn’t end when the loop has
//       finished. No so easy to read and not so elegant
// Pros: Variable Size is evaluated only in the end of loop
//
template <typename obj, size_t S, enable_if_t <S != 0, size_t> = 1>
void Do_Example(const std::array<obj, S>& Obj) {
     size_t Size = 0;
     do {
         std::cout << ‘\n’ << Size;
         ++Size;
     } while (Size < Obj.size());
}


// do_until ------------------------------------------------------------
//
// Prós: Variable Size is evaluated only in the end of loop
//       The life time of variable Size ends when the loop has finished.
//       The code is elegant and easy to read.
//
template <typename obj, size_t S, enable_if_t <S != 0, size_t> = 1>
void Do_Until_Example(const std::array<obj, S>& Obj) {
     do_until (size_t Size = 0; Size == Obj.size(); ++Size)
              std::cout << ‘\n’ << Size;
}
```