[OGIS] ®
OatGrain
InnovationS

The Compound-Group
”LOOP“

https://github.com/F-Haferkorn/ogis-modern-cxx-future-cpp

by Frank Haferkorn 2020-10-22T1900

Lightning Talk at **meetup.com** Group **MUC++**

The Compound-Group “LOOP”
Table-of-Content

- **Introduction**
- Syntax
- Implementation
- Examples
- Discussion: Pro / Contra

The Compound-group "LOOP"

- Target is a **core-level extension** for the **C++** programming language.
- Start a discussion about a new ***compound statement***
  - related to *simple iteration*
  - based on the *compound for(;;){}*
  - Implemented with the *cpp preprocessor*

The Idea:

- Introduce **new Compound loop(){}**
  - to reduce the DEGREES OF FREEDOM
  - of the **for(;;){}** compound statement
  - to allow simpler forms of iterations.

```
loop(4)                          // iterate over 4 rows
    loop(10)                     // iterate over 10 columns
        *tgt++ = *src++ ;        // copy *source to *target
```

The Compound-Group “LOOP”
Table-of-Content

- Introduction
- **Syntax**
- Implementation
- Examples
- Discussion: Pro / Contra

Syntax:
The loop(){} compound

loop(<rep> [, <postexpr>]...**){}**

<rep> **loop** repeats the body {} <rep>-times using hidden index of same type as <rep>

<postexpr> **loop** may have an **optional,** comma separated list of post-expressions.

```
loop(N) loop(M, tgt++, src++) { *tgt += *src; }
```

Syntax:
The named_loop_up/down(){} compounds

named_loop_up(<id>**,** <rep> [, <postexpr>]...**){}**
named_loop_down(<id>**,** <rep> [, <postexpr>]...**){}**

<rep>	repeat the <rep>-times
<postexpr>	optional comma separated list of post-expressions.
<id>	symbol-name of the known index-variable.

```
named_loop_up(index, noRepetitions)  value+=func(index);
```

Syntax:
The loop(){} compound

typed_loop(<type>, <rep> [, <postexpr>]...**){}**

<rep> repeat the loop <rep>-times of
<postexpr> an optional comma separated list of post-expressions.
<type> the type of the (hidden) index-variable

```
typed_loop(char, 64) *tgt++ = *src++;
```

The Compound-Group “LOOP”
Table-of-Content

- Introduction
- Syntax
- **Implementation**
- Examples
- Discussion: Pro / Contra

Implementation:
Prerquisite #define(s)

```
/// create an unique symbol id...
#define CPPMACRO_UNIQUE_ID()  \
            CPPMACRO_UNIQUE_ID_LINE_##__LINE__##_##__COUNTER__

// C++ macro to loop upwards from 0 to nbrOfRepetitions-1
#define CPPMACRO_NTIMES_UP(the_type, indexName, nbrOfRepetitions, ...) \
            for(the_type indexName = 0;
                  indexName<nbrOfRepetitions;   ++indexName, ##__VA_ARGS__)

// C++ macro to loop downwards from nbrOfRepetitions-1 to 0
// be aware if infinite loops as any unsigned integral-type cannot be <0
#define CPPMACRO_NTIMES_DOWN(the_type, indexVarName, nbrOfRepetitions, ...) \
            for(std::make_signed<the_type>::type indexVarName = nbrOfRepetitions ;
                  --indexVarName >= 0 ;  __VA_ARGS__)
```

Implementation:
#define loop(){} and typed_loop(){}

```
// loop(): iterate  nbrOfRepetitions times
#define loop(nbrOfRepetitions, ...)            \
                CPPMACRO_NTIMES_UP(decltype(nbrOfRepetitions) ,
                                                CPPMACRO_UNIQUE_ID(),  \
                                               nbrOfRepetitions,   ##__VA_ARGS__)

// typed_loop(): apply a type  and iterate  nbrOfRepetitions times
#define typed_loop(type, nbrOfRepetitions, ...)    \
                CPPMACRO_NTIMES_UP(type, CPPMACRO_UNIQUE_ID(), \
                                               nbrOfRepetitions, ## __VA_ARGS__)
```

Implementation:
#define named_loop_up(){} , named_loop_down(){}

```
// loop upwards with a  named (, not hidden)  index-variable
#define named_loop_up(indexVarName, nbrOfRepetitions, ...)   \
            CPPMACRO_NTIMES_UP( decltype(nbrOfRepetitions), indexVarName, \
                                                nbrOfRepetitions, ##__VA_ARGS__)

// loop downwards with a  named (, not hidden)  index-variable
 #define named_loop_down(indexVarName, nbrOfRepetitions, ...)  \
            CPPMACRO_NTIMES_DOWN(decltype(nbrOfRepetitions), indexVarName, \
                                                nbrOfRepetitions, ## __VA_ARGS__)
```

The Compound-Group “LOOP”
Table-of-Content

- Introduction
- Syntax in a Nutshell
- Implementation
- **Example I**
  - **loop(){}**
- Discussion: Pro / Contra

Example I: loop(){}
A regular for(;;){} compound (0)

```
#include “ascii_print.h”
// print a square using regular  for(;;){}
void square(short nRows, short nColumns)
{
    for(short row=0; row<nRows; row++)
    {
        for(short col=0; col<nColumns; col++)
        {
            star();
        }
        newline();
    }
}
```

```
// @file: ascii_print.h

#include <iostream>
void star()    { std::cout.put('*');  }
void space()   { std::cout.put(' ');  }
void newline() { std::cout.put('\n'); }
```

Example I: loop(){}
A regular for(;;){} compound (1)

```
#include “ascii_print.h”
// print a square using regular  for(;;){}
void square(short nRows, short nColumns)
{
      for(short row=0; row<nRows; row++)
      {
          for(short col=0;col<nColumns; col++)
          {
               star();
          }
          newline();
      }
}
```

Example I: loop(){}
Reducing for(;;){} → while(){} (2)

```
#include “ascii_print.h”
// print a square using regular  for(;;){}
void square(short nRows, short nColumns)
{
    for(short row=0; row<nRows; row++)
    {
        for(short col=0;col<nColumns; col++)
        {
            star();
        }
        newline();
    }
}
```

```
#include “ascii_print.h”
// print a square reduced to while()
void square(short nRows, short nColumns)
{
    while(nRows—)
    {
        while(nColumns--)
        {
            star();
        }
        newline();
    }
}
```

Example I: loop(){}
Reducing for(;;){} → loop(){} (3)

```
#include “ascii_print.h”
// print a square using regular  for(;;){}
void square(short nRows, short nColumns)
{
      for(short row=0; row<nRows; row++)
      {
          for(short col=0;col<nColumns; col++)
          {
               star();
          }
          newline();
      }
}
```

```
// print square using loop(){}
#include “ascii_print.h”
#include <loop>

void square(short  nRows, short nColumns)
{
      loop(nRows,  newline())
            loop(nColumns,  star())
                  ;
}
```

The Compound-Group “LOOP”
Table-of-Content

- Introduction
- Syntax in a Nutshell
- Implementation
- **Example II**
  - **named_loop_up(){}, named_loop_down/){}**
- Discussion: Pro / Contra

Example II
Using named_loop_up(){} , named_loop_down(){}

```
*
**
***
****
*****
******

******
*****
****
***
**
*
```

```
#include “ascii_print.h”
#include <loop>
void triangular_upwards(short nRows)
{
   named_loop_up(row, nRows, newline())
         loop(row + 1,  star() )
               ;
}
void triangular_downwards(short nRows)
{
   named_loop_down(row, nRows, newline())
         loop(row + 1,  star() )
               ;
}
main()
{      triangular_upwards(6); newline();
       triangular_downwards(6);
}
```

The Compound-Group “LOOP”
Table-of-Content

- Introduction
- Syntax in a Nutshell
- Implementation
- **Example III**
  - **typed_loop(){}**
- Discussion: Pro / Contra

Example III
Using typed_loop(){}

```
// prints out
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
```

```
///  force hidden-index to type  unsigned char
#include ”ascii_print.h”

#include <loop>
#include <cstdint>
main()
{
      typed_loop(uint8_t,  10,  newline() )
                  typed_loop(uint8_t, 20)
                      star();
}
```

The Compound-Group “LOOP”
Table-of-Content

- Introduction
- Syntax in a Nutshell
- Implementation
- **Example IV**
  - **matrix_copy_w_stride()**
- Discussion: Pro / Contra

Example IV: loop(): matrix_copy_with_stride()

```
#include <loop>
template<typename TPtr,
                typename TRowSize,     typename TColSize,       typename TStrideSize>
void matrix_copy_with_stride( TPtr  tgt, TPtr src,
                                        TRowSize nRows,  TColSize nColumns,
                                        TStrideSize stride)
{
     loop(nRows,  tgt+=stride, src+=stride)        // apply stride-offset after each row.
          loop(nColumns, tgt++, src++)              // increment addresses  after each copy.
                *tgt = *src ;                       // copy source to target.
      return ;
}
```

The Compound-Group “LOOP”
Table-of-Content

- Introduction
- Syntax in a Nutshell
- Implementation
- Examples
- **Discussion: Pro / Contra**

Discussion
Basic Facts

- New compounds
  - **loop**(){}, **typed_loop**(){}, **named_loop_up/down**(){}
- Implementation bases solely on the ***cpp preprocessor*.**
- *The loop iteration* has as *reduced degree of freedom* and is mapped to a **for(;;){}** compound.
- Is a tiny extension,
- It is already implemented (but see compilation caveats)

Discussion
Advantages: Readability / Algorithmics / Teachability

- **READABILITY**:
  - It reduces C++ source code size and improve its readability.
- **ALGORITHMICS**:
  - It allows/leads the developer(s) to notate code that is NOT depending on the iteration index.
- **TEACHABILITY:**
  - it can improve the way to teach C++ especially for a younger audience (for details → https://github.com/F-Haferkorn).

Discussion
Advantages: Same or better performance

- **The LOOP compound allows further OPTIMIZATION**
  - Loop(){} is **(at least) as fast** as a regular for(;;){} iteration.
  - Has more iterative flexibility (for the compiler).
    - due to the reduced DEGREE of FREEDOM of loop(){}
  - *Allows* ***Hardware "accelerated" Loops :***
    - e.g.: DSP TMS320: *Software Pipelined Loop: (*"**SPLOOP")**
  - Allows **Fast Register Post-Operations :**
    - e.g.: DSP ADSP218x "Data Address Generators" (DAG1/2)

Discussion: *Compilation Caveat*
Disadvantages: Problem with templates with comma

- Current preprocessor Implementation
  has a **compilation problem**
  at **arguments containing commas** (like some **templates**)

```
loop(std::integral_constant<int, 10>::value)    /// compiler error
        do_something();

/// workaround: embrace with "()"
 loop( (std::integral_constant<int, 10>::value) )   /// works:
        do_something();
```

Discussion
As expected: Will not compile for iterations on enums

- As expected
  - **Looping** over **enums** does not compile

```
enum {RED, GREEN, BLUE} rgb=BLUE;
loop(rgb)                          /// compiler error.
    do_something();
```

The Compound Group “LOOP”

For more details, some code examples and references

have a look at:

https://github.com/F-Haferkorn/ogis-modern-cxx-future-cpp

Thank You!

These are the slides of a lightning talk hold at 2020-10-22 for the meetup.com group MUC++

https://www.meetup.com/de-DE/MUCplusplus/events/273223910/

Presenter: Frank Haferkorn : Frank is a physicist and senior software-developer form Ottobrunn, near Munich, Germany. He uses C++ for 27 years now and gives thoughts on physics, algorithmics and C++. When not doing computer stuff you can find Frank drawing or making music/sound-design.

In his lightning talk Frank will present a **C/C++ core-language** extension in form of the **compound-group "LOOP".** The new compounds ***loop(){}***, ***typed_loop(){}***, ***named_loop_up(){}*** and ***named_loop_down()*** allow easier coding of simple iterations and can improve the ***teachability***, the ***readability*** *of* ***C/C++*** and can lead to **simpler algorithms** and maybe **optimized code.**

Frank shows the **syntax**, explains the ***basic usage***, *discusses the* ***cons and pros*** and *presents an* ***implementation*** that provides reduced functionality of the well known ***for(;;){}*** compound statement