[OGIS]®
OATGRAIN
INNOVATIONS

The Compound-Group "LOOP"

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

by Frank Haferkorn 2021-03-30T2000

Online Talk hold at **Meetup C++ London**

info@OatGrain-InnovationS.de

The AUTHOR

Dipl.-Phys.

Frank Haferkorn **(*1968)**

[OGIS] OatGrain-InnovationS

Senior Software-Developer

Ottobrunn, Germany

- Modern C++, Physics, Sound-Design (Spatial-Audio)

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

Introduction: The Target

The Compound-group "LOOP"

- Target is a **language 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 Compound-Group “LOOP”
Introduction: The Idea

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

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(4)  loop(10,  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, 40) *tgt++ = *src++;
```

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

Implementation:
Prerquisite CPPMACRO #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

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

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

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

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 ;
}
```

Example IV: loop(): matrix_copy_with_stride()

```
#include <chrono>
#include <thread>
#include <loop>

class Foo{
	test(){/*…*/}
};
TEST_F(FooTest, StressTestCall1MillionTimes) {
	Foo foo;
	loop(1000000) {
		EXPECT_TRUE(foo.test());
		std::this_thread::sleep_for(std::chrono::milliseconds(1));
	}
}
```

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

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

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.
  -

Discussion
Reduced Degree-of-Freedom

- **TEACHABILITY:**
  - At the moment for(;;){} is taught in one of the first C/C++ lessons
  - a simple **for(int i=0; i<n; ++i){}** requires these principles
    - **types, variables**, their **assignment**, and **incrementation**
    - using **boolean expressions**
    - Like **conditions** with **comparison** and **relations (<, ==, >, >=, <=)**

Discussion
Reduced Degree-of-Freedom

- **TEACHABILITY: /// The Raspberry Generation**

  loop(){}

  - improves the way to teach C++ especially for a younger audience
  - e.g. the UK-Government decided to "force" , childern form 4-years on to learn programming
  - 5th grade (11-years old) pupils can cope with the **concept of looping**,
    - and *generates textual outputs via printing something*
    - *like squares, triangles, etc.*

Discussion
Reduced Degree-of-Freedom

- **REDUCED DEGREE OF FREEDOM:**
  - Obviously,
  - The compounds of the **LOOP-Group** reduce the degree of freedom of a **for(;;){}** iteration and allows *structuring the code* in an *easier manner*.
  - It can be used
    - To produce easier/safer/more maintainable code.

Discussion
Advantages: Same or better performance

- **The LOOP opens the door to 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!

Dipl.-Phys. Frank.Haferkorn 2021-03-30
About him: A-Short-Biograpy

R”---(
Frank Haferkorn is a graduated physicist, senior software developer and founder or Head-Of-Science of the inventors' office [OGIS] OatGrain-InnovationS. He belongs to the generation that has seen the whole development of desktop computers (from the CBM-PET) and has worked as a professional software developer in the industry since graduating from the Technical University of Munich in 1995 until today. His areas of expertise are Modern C ++ (>=2020), algorithms, parallel computing technology, physics (electrodynamics / QM / QED / SRT / ART and the Psychoacoustics of Spatial Hearing). He has also worked in the field of semantic web / linked data.

In addition to smaller publications, he also “draws” his own thoughts on the further development of C++. The use of elaborate tools of all kinds is his hobby and covers the whole spectrum from Visual Studio to Qt and Linux. To compensate, Frank can be found as an artist drawing, composing and as a sound designer for spatial audio.

mailto:info@OatGrain-InnovationS.de?subject=µ:OGIS:automotive-hmi-ux-online-confernce-2021:

https://www.google.de/search?q=frank.haferkorn+ottobrunn

)--”_@en

OGIS OatGrain-InnovationS 2021-03-30
Contact-Data && Some-Useful-Links

[OGIS] - OatGrain-InnovationS
Dipl.-Phys. Frank Haferkorn
D-85521 Ottobrunn ///! SSE-of-Munich Bavaria

Startup-Short	:	OGIS.eu@gmail.com
Startup-www	:	sorry,,,, no www-page….
Handy:	:	+49/176/70311275
Skype	:	live:F.Haferkorn

Google:		https://www.google.de/search?q=ogis+Oatgrain-innovationS&oq=ogis+Oatgrain-innovationS
Bing	:	https://www.bing.com/search?q=F.Haferkorn+Ottobrunn
LinkedIn	:	https://de.linkedin.com/in/frank-haferkorn-48ba568
Xing	:	https://www.xing.com/profile/Frank_Haferkorn

Abstract-of-this-paper

https://www.meetup.com/de-DE/CppLondon/events/276266931/

In his talk ecture scheduled for 15 minutes, Frank Haferkorn presented the compound group "LOOP" as a C / C ++ core language extension.
The original C control flow commands have changed little to nothing since, since the publication of Kernighan & Ritchie's “K&R C” in 1978(!).
These are called “Compound(s)” and are in the well-known if-else, while, do -while, for and switch. Only one really new compound in the form of the try/catch block has been added in C++. some further extensions in if and switch ware added in C++
Is it a physical law that no further compounds may ever be added?
The new compounds presented here loop() {}, typed_loop() {}, named_loop_up() {} and named_loop_down() {} enable simple coding of simple iterations.

Even if this is not going to be a major new feature, it has advantages. They reduce the complexity, improve the readability of C / C ++ and will lead to different / simpler notations (including existing) algorithms. Compilers can generate higher-performance code due to reduced complexity. An improvement in the Teachability of C/C ++ is to be expected. Thus the entry threshold teaching in C for future C/C++ developers from today's Raspberry generation drops.

Frank Haferkorn shows the syntax, explains the basic usage, explains the application using examples, discusses the advantages and disadvantages and first presents a pure C implementation based solely on the C preprocessor using variadic macros.

For a more elaborate C++ implementation a few more C ++ tricks are necessary ... Known problems with the current implementation should also not be missing.

The LOOP compounds are implemented as a single header-only include file.

#!/bin/ready to-rumble https://github.com/F-Haferkorn