



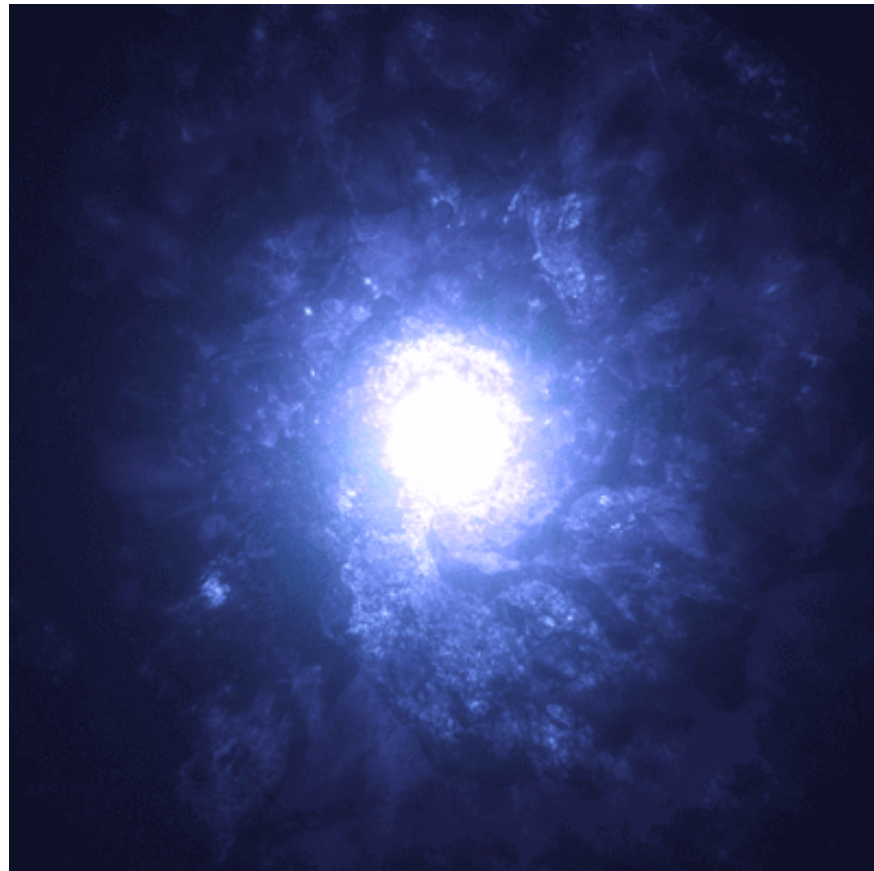
Adrian D. Finlay

@thewiprogrammer. Writer @hackernoon. Code, LOTS of it. Mangos, LOVE THEM! Barbering. Health. Travel. Business. & more! Network w/ me @ adriandavid.me/network

Nov 10, 2017 · 17 min read

## Modern C++ Part III: New Language Features in C++17

The never ending journey into learning C++ features....



Giphy: FELIKS TOMASZ KONCZAKOWSKI GIF

C++ is a general purpose, multi-paradigm, compiled language that was invented by danish computer scientist, Bjarne Stroustrup, and released in 1983. C++ marries classes, such as those found in Simula, with the majority of the C language, to create a language that is like an Object Oriented version of the C language. However, C++ is dar

divorced from the days of “C with classes”. Almost all C++ compiler vendors provide an implementation of a the standard class library, libstdc++. C++ is among the most widely implemented and widely used languages in the history of modern computing.

In my opinion, it is best suited for **systems programming, embedded programming, high performance computing, resource constrained computing** (think tiny devices), & the **development of low level APIs, language compilers, interpreters, device drivers, & the design of software infrastructure**.

Typically, the choice to use C++ is predicated by the need for performance & efficiency (little bloat, efficient use of resources and implementation constructs, getting as close to the metal as possible). For better or worse, C++ is ideologically flexible—it does not constrain you to programming in one paradigm such as many other languages. It contains a bevy of features, which is a frequent source of criticism by certain members of the programming community. C++ is an outlier of sorts in that the philosophy behind C++ embraces including good ideas from many different ideological perspectives as opposed to the KISS (*Keep It Simple Stupid*) philosophy which is more oriented towards having one simple way to do one thing.

C++ is also used for common Desktop Application Software. C++ has found widespread use in truly massive systems. For example: Google Chrome, Mozilla Firefox, Telephony Infrastructure, Chrome V8, and much, much more. Read Stroustrup’s (incomplete) list [here](#). C++ might be a better decision than using C for many reasons, the most popular of which, in my opinion, are the various abstractions that C++ provides, most notably, the class. The class allows for highly structured programs that bind data and the functions that act on such data. This often makes for more organized programs than the C equivalent.

The upcoming revision to the ISO for standard is C++17. It will ship with **35** new language features and **4** deprecated language features (**do let me know if I am wrong on any of this**). I will group some of the related features together. The industry leading compilers (GCC, MSVC, Clang) have already implemented many of the new C++17 features ahead of it’s general release [3]. The list of new features presented in this article has been generated from several sources [1][3][4][5][6][7]. C++ is expected to ship sometime this year (2017) [7]. You may

track it's current ISO approval status, here. C++17 is a major feature release, the largest such release since C++11.

## **Update: ISO Specification for C++17 is now published, work is being done on C++20.**

### **New Language Features**

1. Addition of `__has_include` macro
2. UTF 8 Character Literals
3. Hexadecimal Floating Point Literals
4. New rules for deduction of single member list using `auto`
5. Update to `__cplusplus` value
6. `inline` variables
7. New Syntax for Nested Namespace definitions
8. Initializers added to `if/switch` statements
9. `constexpr if`
10. New standard attributes `[[fallthrough]]`, `[[maybe_unused]]` & `[[nodiscard]]` ^
11. Attributes for Enumerator & Namespaces
12. Error message for `static_assert` now optional
13. Structured binding declarations
14. Keyword `typename` now allowed in lieu of `class` in a template's template parameter
15. Constant evaluation for non-type template arguments
16. Class template argument deduction
17. Extensions on over-aligned Memory Allocation
18. Fold expressions
19. List-style Initialization of Enumerations
20. Specifying non-type template parameters with `auto`

21. constexpr lambda expressions
22. Lambda this by value (\*this)
23. Extending Aggregate Initialization to Base Types
24. Unknown Attributes Required to be Ignored
25. Pack Expansions legal in using declarations
26. Generalization of Range-based for loop
27. The byte data type <sup>^^</sup>
28. Using attribute namespaces without repetition
29. Stricter Order of Evaluation Rules
30. Exception Specifications are part of type definitions
31. Template-Template Parameters match compatible arguments
32. Guaranteed Copy Elision
33. Changes to Specification on Inheriting Constructors

<sup>^</sup>These are three features grouped into one, which consequently when expanded would make the new feature list count 35.

<sup>^^</sup>This is implemented in std::byte (<cstdint>) and is not a part of the actual language such as the other primitive data types. It is considered a basic type inasmuch that std::string is considered a basic type.

C++17 also introduced a revision to Elementary string conversions which you can read about here.

### **Deprecated Language Features**

1. Removal of Trigraphs by default
2. Removal of deprecated Increment Operator (++) for bool type
3. Removal of deprecated register keyword
4. Removal of deprecated Dynamic Exception Specifications

Check with this list often to see compiler support for the various language changes! [3] You can find information about GCC C++17 support here, and LLVM/Clang C++17 Support here. Please note that I will be covering C++ language features only and **will NOT discuss the several changes to the standard library.**

Why so many features Bjarne? Maybe one day he'll tell me.



GfyCat: "Stroustrup" (Taken from BigThink YouTube Video)

**Want the source? Grab it here.**

afinlay5/Cplusplus17

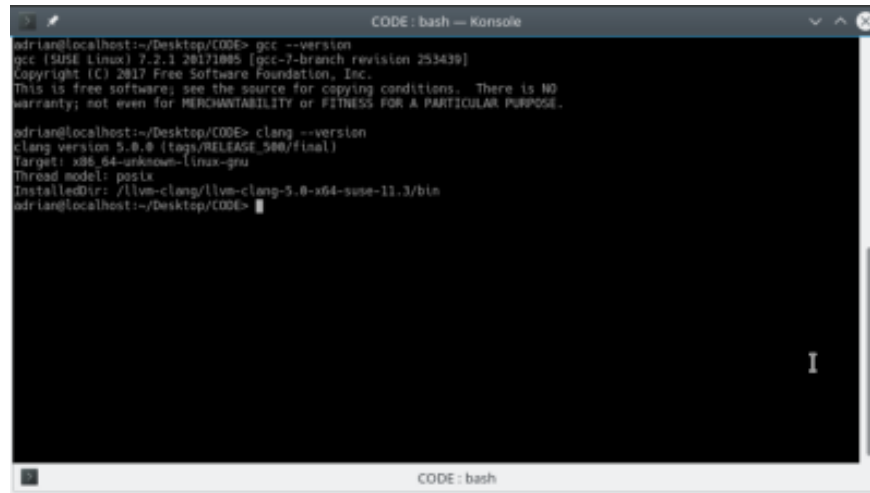
CMake source code repository for C++17 source code examples posted on personal blog...

github.com

## The Compilers I'll be using

I will be using GCC version 7.2.1 and Clang (LLVM) version 5.0.0, both the latest versions of the respective compilers as of this publications posting, to test and run my examples. **A testament to C++'s breadth, both of these compilers are themselves written in C++!** Both compilers are part of suites of tools providing **compiler support for**

several different languages on several different architectures. I will be compiling and running the code on bash on my SUSE Linux box.



```
adrian@localhost:~/Desktop/CODE> gcc --version
gcc (SUSE Linux) 7.2.1 20171005 [gcc-7-branch revision 253439]
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

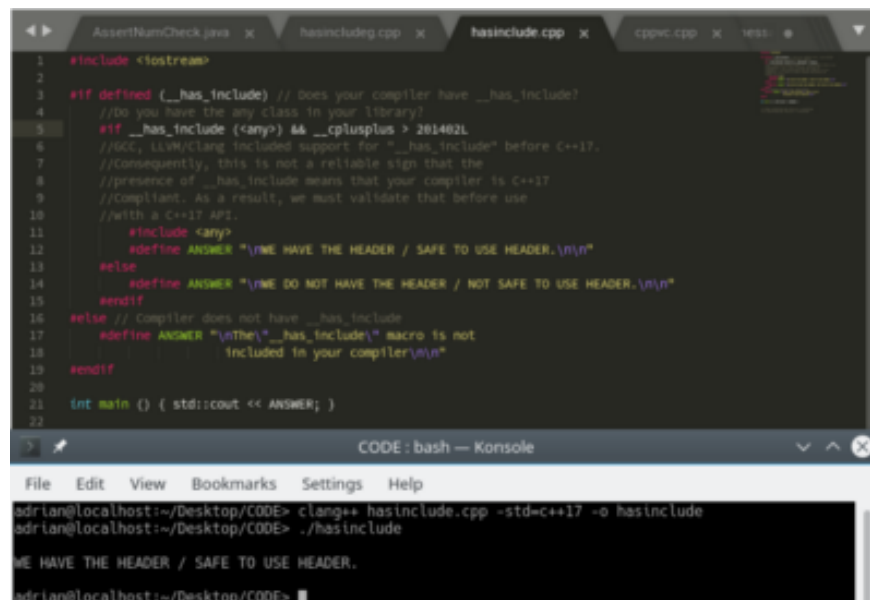
adrian@localhost:~/Desktop/CODE> clang --version
clang version 5.0.0 (tags/RELEASE_500/final)
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /llvm-clang/llvm-clang-5.0-x64-suse-11.3/bin
adrian@localhost:~/Desktop/CODE>
```

I'll post the code first and the output in bash second. Let's start with the New Language Features.

## New Language Features

### 1) Addition of `__has_include` macro

The macro `__has_include` (added in C++17) allows the programmer to check the availability of a header to be checked by a preprocessor directive.



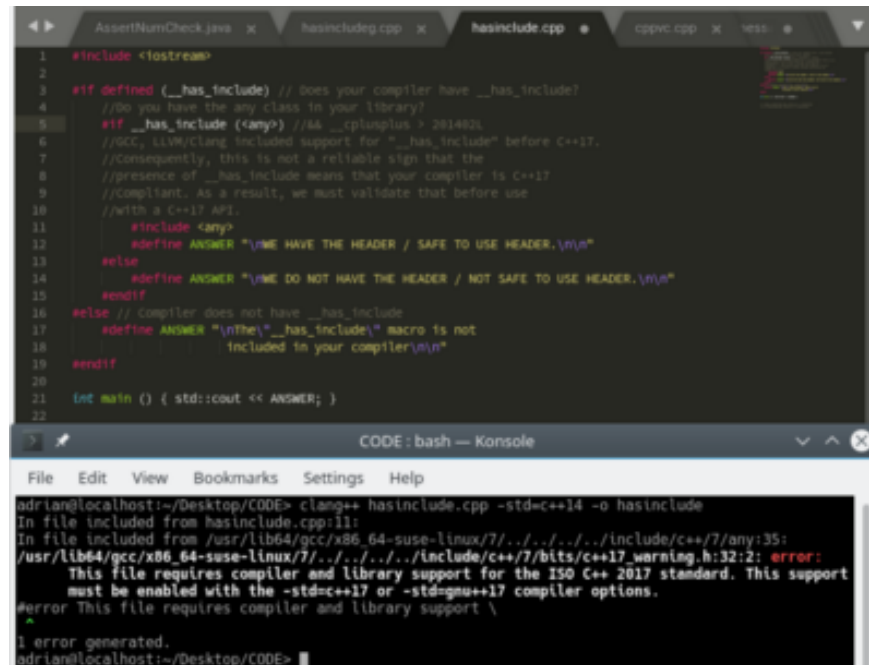
```
1 #include <iostream>
2
3 #if defined (__has_include) // does your compiler have __has_include?
4 // Do you have the any class in your library?
5 #if __has_include (<any>) && __cplusplus > 201402L
6 //GCC, LLVM/Clang included support for "__has_include" before C++17.
7 //Consequently, this is not a reliable sign that the
8 //presence of __has_include means that your compiler is C++17
9 //Compliant. As a result, we must validate that before use
10 //with a C++17 API
11 #include <any>
12 #define ANSWER "WE HAVE THE HEADER / SAFE TO USE HEADER.\n\n"
13 #else
14 #define ANSWER "WE DO NOT HAVE THE HEADER / NOT SAFE TO USE HEADER.\n\n"
15 #endif
16 #else // Compiler does not have __has_include
17 #define ANSWER "The \"__has_include\" macro is not
18               included in your compiler.\n\n"
19 #endif
20
21 int main () { std::cout << ANSWER; }
22
```

```
adrian@localhost:~/Desktop/CODE> clang++ hasinclude.cpp -std=c++17 -o hasinclude
adrian@localhost:~/Desktop/CODE> ./hasinclude

WE HAVE THE HEADER / SAFE TO USE HEADER.

adrian@localhost:~/Desktop/CODE>
```

Notice the comments from Ln. 5–10. We must check for C++17 support. For example, `std::any` is only available in C++17. Otherwise this will happen:



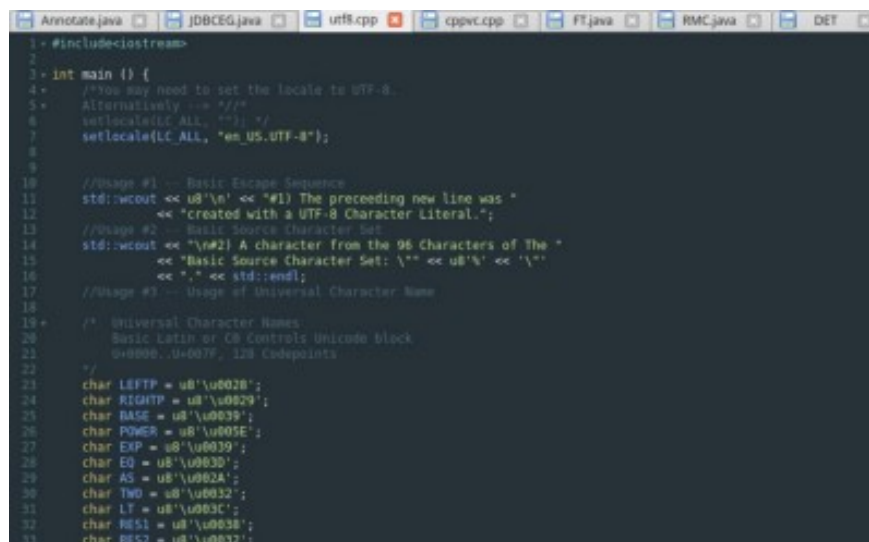
```
1 #include <iostream>
2
3 #if defined (__has_include) // Does your compiler have __has_include?
4 //Do you have the any class in your library?
5 #if __has_include (<any>) //64 __cplusplus > 201402L
6 //GCC, LLVM/clang included support for __has_include before C++17.
7 //Consequently, this is not a reliable sign that the
8 //presence of __has_include means that your compiler is C++17
9 //Compliant. As a result, we must validate that before use
10 //with a C++17 API.
11 #include <any>
12 #define ANSWER "I HAVE THE HEADER / SAFE TO USE HEADER.\n\n"
13 #else
14 #define ANSWER "I DO NOT HAVE THE HEADER / NOT SAFE TO USE HEADER.\n\n"
15 #endif
16 #else // Compiler does not have __has_include
17 #define ANSWER "\nThe __has_include macro is not
18             included in your compiler\n\n"
19 #endif
20
21 int main () { std::cout << ANSWER; }
22
```

```
adrian@localhost:~/Desktop/CODE> clang++ hasinclude.cpp -std=c++14 -o hasinclude
In file included from hasinclude.cpp:11:
In file included from /usr/lib64/gcc/x86_64-suse-linux/7/../../../../include/c++/7/any:35:
/usr/lib64/gcc/x86_64-suse-linux/7/../../../../include/c++/7/bits/c++17_warning.h:32:2: error:
This file requires compiler and library support for the ISO C++ 2017 standard. This support
must be enabled with the -std=c++17 or -std=gnu++17 compiler options.
#error This file requires compiler and library support \
^
1 error generated.
adrian@localhost:~/Desktop/CODE>
```

This works with both LLVM/Clang as well as GCC.

## 2) UTF 8 Character Literals

C++17 introduced UTF-8 Character Literals.



```
1 #include <iostream>
2
3 int main () {
4     //You may need to set the locale to UTF-8.
5     //Alternatively --> */
6     setlocale(LC_ALL, "");
7     setlocale(LC_ALL, "en_US.UTF-8");
8
9     //Page #1 -- Basic Escape Sequence
10    std::wcout << u8'\n' << "The preceding new line was "
11               << "created with a UTF-8 Character Literal.";
12
13    //Page #2 -- Multi-Source Character Set
14    std::wcout << "A character from the 96 Characters of The "
15               << "Basic Source Character Set: \" << u8'\n' << "\"
16               << " << std::endl;
17
18    //Page #3 -- Usage of Universal Character Name
19
20    /* Universal Character Names
21     Basic Latin or CB Controls Unicode block
22     U+0000..U+007F, 128 Codepoints
23     */
24    char LEFTP = u8'\u0028';
25    char RIGHTP = u8'\u0029';
26    char BASE = u8'\u0038';
27    char POWER = u8'\u003E';
28    char EXP = u8'\u0039';
29    char EQ = u8'\u003D';
30    char AS = u8'\u002A';
31    char TWO = u8'\u0032';
32    char LT = u8'\u003C';
33    char RES1 = u8'\u003B';
34    char RES2 = u8'\u0037';
35
36 }
```

```

Annotate.java JDBCEG.java utf8.cpp cppvc.cpp FT.java RMC.java DET
34 char SPACE = u8'\u0020';
35 char QUES = u8'\u003F';
36
37 std::wcout << "#3) Universal Character Names: " << "\t\t" << LEFTP << BASE << POWER << EXP << RIGHTP
38 << SPACE << LT << SPACE << RES1 << RES2 << SPACE << QUES << "\t" << "\tYES." << std::endl;
39
40
41 //How to do Emojis
42 std::wcout << "\nCurious about Emojis too?\nCheck out few: " << u8'\n' << u8'\n';
43
44 //Some Emojis
45 wchar_t NERD_FACE = L'\U0001F913';
46 wchar_t MONEY_MOUTH_FACE = L'\U0001F911';
47 wchar_t SIGN_OF_THE_HORNS = L'\U0001F918';
48 wchar_t UNICORN_FACE = L'\U0001F984';
49 wchar_t MUSIC_CLEF = L'\U0001D11E';
50 wchar_t BANANA = L'\U0001F34C';
51
52 //Let's Print them
53 std::wcout << "Unicode 6.0--\t\t\t\t\t'BANANA'\t\t\t\t\t(U+1D11E): \t\t\t" <<
54 BANANA << std::endl;
55 std::wcout << "Unicode 8.0--\t\t\t\t\t'NERD_FACE'\t\t\t\t\t(U+1F913): \t\t\t" <<
56 NERD_FACE << std::endl;
57 std::wcout << "Unicode 8.0--\t\t\t\t\t'UNICORN_FACE'\t\t\t\t\t(U+1F984): \t\t\t" <<
58 UNICORN_FACE << "\n";
59 std::wcout << "Unicode 8.0--\t\t\t\t\t'MONEY-MOUTH_FACE'\t\t\t\t\t(U+1F911): \t\t\t" <<
60 MONEY_MOUTH_FACE << "\n";
61 std::wcout << "Unicode 8.0--\t\t\t\t\t'SIGN OF THE HORNS'\t\t\t\t\t(U+1F918): \t\t\t" <<
62 SIGN_OF_THE_HORNS << "\n";
63 std::wcout << "Unicode 3.1--\t\t\t\t\t'MUSICAL SYMBOL G CLEF'\t\t\t\t\t(U+1D11E): \t\t\t" <<
64 MUSIC_CLEF << "\n" << std::endl;
65 }
66

```

```

CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> clang++ utf8.cpp -std=c++17 -o utf8
adrian@localhost:~/Desktop/CODE> ./utf8

#1) The preceding new line was created with a UTF-8 Character Literal.
#2) A character from the 96 Characters of The Basic Source Character Set: "%".
#3) Universal Character Names: "Is (9*9) < 82 ?"      YES.

Curious about Emojis too?
Check out few:

Unicode 6.0--      "BANANA"                (U+1D11E):
Unicode 8.0--      "NERD_FACE"              (U+1F913):
Unicode 8.0--      "UNICORN_FACE"           (U+1F984):
Unicode 8.0--      "MONEY-MOUTH_FACE"       (U+1F911):
Unicode 8.0--      "SIGN OF THE HORNS"      (U+1F918):
Unicode 3.1--      "MUSICAL SYMBOL G CLEF"  (U+1D11E):

adrian@localhost:~/Desktop/CODE>

```

```

adrian@adrian-ThinkPad-T520 ~/Downloads/CODE
File Edit View Search Terminal Help
adrian@adrian-ThinkPad-T520 ~/Downloads/CODE $ g++-7 utf8.cpp -std=c++17 -o utf8
adrian@adrian-ThinkPad-T520 ~/Downloads/CODE $ ./utf8

#1) The preceding new line was created with a UTF-8 Character Literal.
#2) A character from the 96 Characters of The Basic Source Character Set: "%".
#3) Universal Character Names: "Is (9*9) < 82 ?"      YES.

Curious about Emojis too?
Check out few:

Unicode 6.0--      "BANANA"                (U+1D11E):
Unicode 8.0--      "NERD_FACE"              (U+1F913):
Unicode 8.0--      "UNICORN_FACE"           (U+1F984):
Unicode 8.0--      "MONEY-MOUTH_FACE"       (U+1F911):
Unicode 8.0--      "SIGN OF THE HORNS"      (U+1F918):
Unicode 3.1--      "MUSICAL SYMBOL G CLEF"  (U+1D11E):

adrian@adrian-ThinkPad-T520 ~/Downloads/CODE $

```

From Linux Mint.



### 3) Hexadecimal Floating Point Literals

C++17 introduced support for Hexadecimal Floating Point Literals. You may find out more, [here](#).

```
1 #include <iostream>
2 int main ()
3 {
4     std::cout << "\nC++17 introduced Hexadecimal Floating Point Literals "
5               << "in three different formats.\n\n"
6               << "\"0x|0X hex-digit-seq|:|t|t|t|t\" << "0x1ffp96|:|t|t\" << "0x1ffp96 << '\n'
7               << "\"0x|0X hex-digit-seq . |\"|:|t|t|t|t\" << "0xa.p-2|:|t|t\" << "0xa.p-2 << '\n'
8               << "\"0x|0X hex-digit-seq . hex-digit-seq|:|t|t\" << "0x1.743p-3|:|t|t\" << "0x1.743p-3 << '\n'
9               << '\n' << std::endl;
10 }
```

```
CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> clang++ hex.cpp -std=c++17 -o hex
adrian@localhost:~/Desktop/CODE> ./hex

C++17 introduced Hexadecimal Floating Point Literals in three different formats.

"0x|0X hex-digit-seq|:|t|t|t|t"      0x1ffp96:      4.04856e+31
"0x|0X hex-digit-seq . |\"|:|t|t|t|t"  0xa.p-2:      2.5
"0x|0X hex-digit-seq . hex-digit-seq|:|t|t"  0x1.743p-3:  0.181732

adrian@localhost:~/Desktop/CODE> g++ hex.cpp -std=c++17 -o hex
adrian@localhost:~/Desktop/CODE> ./hex

C++17 introduced Hexadecimal Floating Point Literals in three different formats.

"0x|0X hex-digit-seq|:|t|t|t|t"      0x1ffp96:      4.04856e+31
"0x|0X hex-digit-seq . |\"|:|t|t|t|t"  0xa.p-2:      2.5
"0x|0X hex-digit-seq . hex-digit-seq|:|t|t"  0x1.743p-3:  0.181732

adrian@localhost:~/Desktop/CODE> █
```

### 4) New rules for deduction of single member list using auto

C++17 introduced a more common sense deduction of types using auto for single member lists. Previously, the deduction of single member lists evaluated to `std::initializer_list<x>` where x was the actual type originally in the list. In C++17 this is more intuitively deduced directly to x for single member lists.

```

1 #include<iostream>
2 #include<cstring>
3 #include<typeinfo>
4
5 int main () {
6     auto _int {9};
7     auto _str {"Marc-Elie"};
8     auto _dbl {23.5};
9     //It's a shame std::initializer_list<> is not subscriptable!
10    auto _listinit = { "List", "Init"};
11    std::string _int_type,_str_type,_dbl_type,_listinit_type;
12
13    /*This solution is not portable across compilers--
14     The result of std::type_info.name() is mangled
15     on gcc and clang but IBM, Oracle, and MSVC
16     provide human readable names. */
17
18    if ( std::strcmp((typeid(_int).name()),"i") == 0 ) { _int_type = "int"; }
19    else { _int_type = "unknown/non-gcc/clang compiler"; }
20
21    if ( std::strcmp((typeid(_dbl).name()),"d") == 0 ) { _dbl_type = "double"; }
22    else { _dbl_type = "unknown/non-gcc/clang compiler"; }
23
24    if ( std::strcmp((typeid(_str).name()),"PKc") == 0 ) { _str_type = "string"; }
25    else { _str_type = "unknown/non-gcc/clang compiler"; }
26
27    if ( std::strcmp((typeid(_listinit).name()),"St16initializer_listIPKcE") == 0 )
28    { _listinit_type = "std::initializer_list<char const*>"; }
29    else { _listinit_type = "unknown/non-gcc/clang compiler"; }
30
31    //Print the values and their types
32    std::cout << "\n"
33    <<"\n_int\": " << _int << " is an " << _int_type << ".\n"
34    <<"\n_dbl\": " << _dbl << " is a " << _dbl_type << ".\n"
35    <<"\n_str\": " << _str << " is a " << _str_type << ".\n"
36    <<"\n_listinit\": " << "{ \"List\", \"Init\"}" << " is an " << _listinit_type << ".\n"
37    <<std::endl;
38 }

```

```

CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> clang++ auto_deduc.cpp -std=c++17 -o auto_deduc
adrian@localhost:~/Desktop/CODE> ./auto_deduc
_int: 9 is an int.
_dbl: 23.5 is a double.
_str: Marc-Elie is a string.
_listinit: {"List", "Init"} is an std::initializer_list<char const*>.

adrian@localhost:~/Desktop/CODE> g++ auto_deduc.cpp -std=c++17 -o auto_deduc
adrian@localhost:~/Desktop/CODE> ./auto_deduc
_int: 9 is an int.
_dbl: 23.5 is a double.
_str: Marc-Elie is a string.
_listinit: {"List", "Init"} is an std::initializer_list<char const*>.

adrian@localhost:~/Desktop/CODE>
CODE: bash

```

## 5) Update to \_\_cplusplus value

The value of the predefined MACRO `__cplusplus` has changed to 201703L reflecting the update in the language standard.

```
AssertNumCheck.java x cppvc.cpp Making the text message for
1 #include <iostream>
2
3 int main() {
4
5     std::cout << "\n";
6
7     switch(__cplusplus) {
8         //C++17
9         case 201500L:
10         case 201703L: {
11             std::cout << "The C++ vs. is:\t" << "C++17 Standard compliant.\n";
12             break;
13         }
14         //C++14
15         case 201402L:
16         case 201406L: {
17             std::cout << "The C++ vs. is:\t" << "C++14 Standard compliant.\n";
18             break;
19         }
20         //C++11
21         case 201103L: {
22             std::cout << "The C++ vs. is:\t" << "C++11 Standard compliant.\n";
23             break;
24         }
25         //C++98
26         case 199711L: { //Notice that this went unchanged for C++03
27             std::cout << "The C++ vs. is:\t" << "C++98 Standard compliant.\n";
28             break;
29         }
30         //Some other version
31         default: {
32             std::cout << "The C++ vs. is:\t" << "C++? Some other version.\n";
33             break;
34         }
35     }
36
37     std::cout << "The \"__cplusplus\" code is: " << __cplusplus << ".\n\n";
38 }
```

```
CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> g++ -std=c++17 cppvc.cpp -o cppvc
adrian@localhost:~/Desktop/CODE> ./cppvc
The C++ vs. is: C++17 Standard compliant.
The "__cplusplus" code is: 201703.

adrian@localhost:~/Desktop/CODE> clang++ -std=c++17 cppvc.cpp -o cppvc
adrian@localhost:~/Desktop/CODE> ./cppvc
The C++ vs. is: C++17 Standard compliant.
The "__cplusplus" code is: 201703.

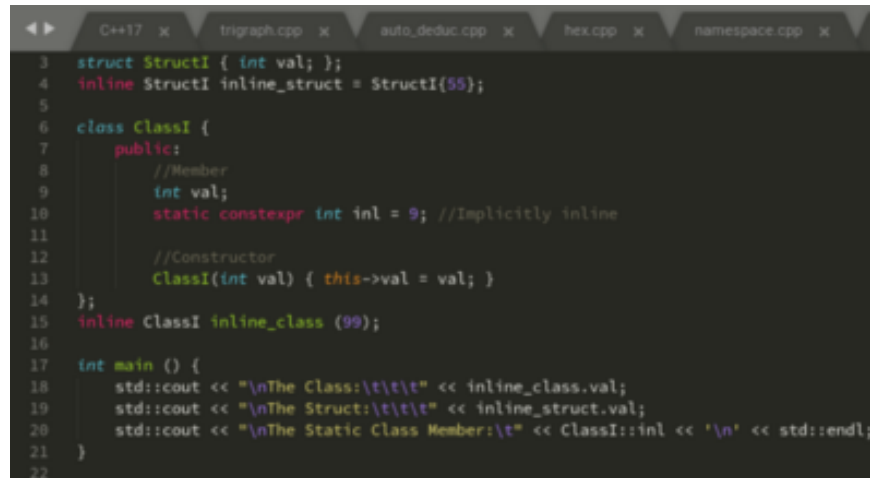
adrian@localhost:~/Desktop/CODE> █

CODE: bash
```

## 6) Inline variables

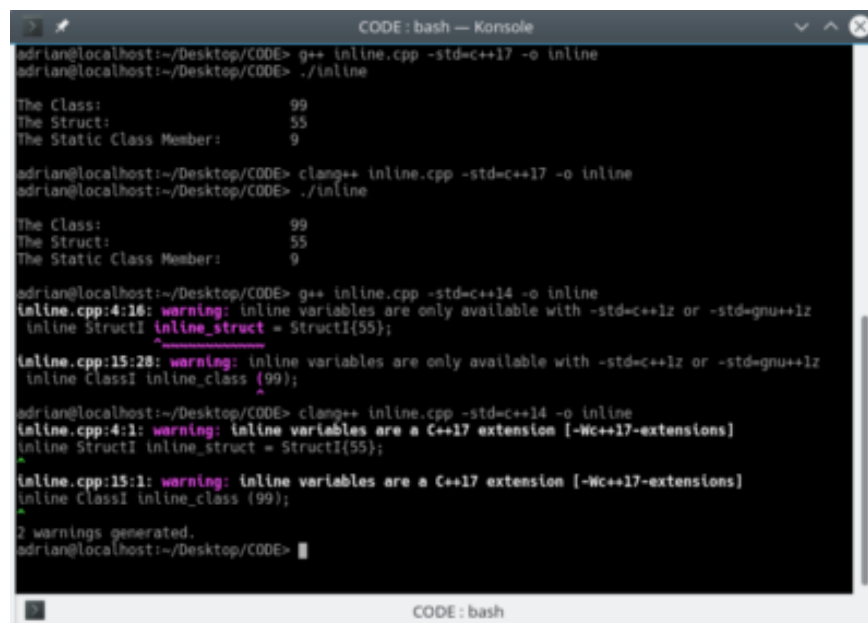
C++17 introduced additional functionality to the inline specifier. You may now use the inline specifier with static class members or namespace-scope variables to declare the variable as an inline variable. Also, a static class member variable (not a namespace-scope variable)

marked constexpr is implicitly an inline variable [8] . This functionality was previously only available for functions. You will want to read more about this here.



```
3 struct StructI { int val; };
4 inline StructI inline_struct = StructI{55};
5
6 class ClassI {
7 public:
8     //Member
9     int val;
10    static constexpr int inl = 9; //Implicitly inline
11
12    //Constructor
13    ClassI(int val) { this->val = val; }
14 };
15 inline ClassI inline_class (99);
16
17 int main () {
18     std::cout << "\nThe Class:\t\t\t" << inline_class.val;
19     std::cout << "\nThe Struct:\t\t\t" << inline_struct.val;
20     std::cout << "\nThe Static Class Member:\t" << ClassI::inl << '\n' << std::endl;
21 }
22
```

Notice the warnings with C++14.



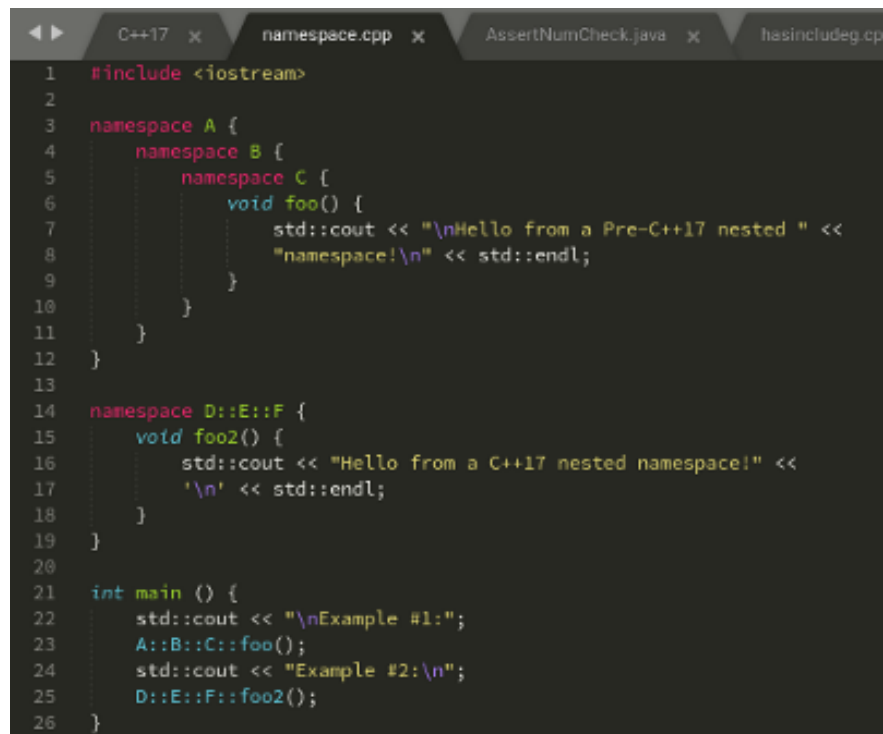
```
CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> g++ inline.cpp -std=c++17 -o inline
adrian@localhost:~/Desktop/CODE> ./inline
The Class:          99
The Struct:         55
The Static Class Member:  9

adrian@localhost:~/Desktop/CODE> clang++ inline.cpp -std=c++17 -o inline
adrian@localhost:~/Desktop/CODE> ./inline
The Class:          99
The Struct:         55
The Static Class Member:  9

adrian@localhost:~/Desktop/CODE> g++ inline.cpp -std=c++14 -o inline
inline.cpp:4:16: warning: inline variables are only available with -std=c++1z or -std=gnu++1z
    inline StructI inline_struct = StructI{55};
                   ^
inline.cpp:15:28: warning: inline variables are only available with -std=c++1z or -std=gnu++1z
    inline ClassI inline_class {99};
                           ^
adrian@localhost:~/Desktop/CODE> clang++ inline.cpp -std=c++14 -o inline
inline.cpp:4:1: warning: inline variables are a C++17 extension [-Wc++17-extensions]
inline StructI inline_struct = StructI{55};
^
inline.cpp:15:1: warning: inline variables are a C++17 extension [-Wc++17-extensions]
inline ClassI inline_class {99};
^
2 warnings generated.
adrian@localhost:~/Desktop/CODE>
```

## 7) New Syntax for Nested Namespace definitions

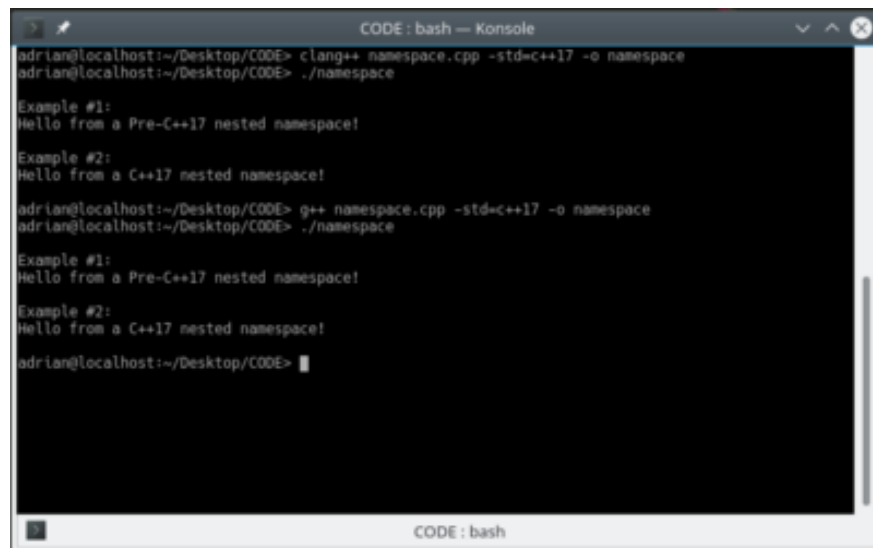
C++17 introduced the use of the scope resolution operator to create nested namespaces. This makes for much less verbose code.



```

1  #include <iostream>
2
3  namespace A {
4      namespace B {
5          namespace C {
6              void foo() {
7                  std::cout << "\nHello from a Pre-C++17 nested " <<
8                      "namespace!\n" << std::endl;
9              }
10         }
11     }
12 }
13
14 namespace D::E::F {
15     void foo2() {
16         std::cout << "Hello from a C++17 nested namespace!" <<
17             '\n' << std::endl;
18     }
19 }
20
21 int main () {
22     std::cout << "\nExample #1:";
23     A::B::C::foo();
24     std::cout << "Example #2:\n";
25     D::E::F::foo2();
26 }

```



```

CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> clang++ namespace.cpp -std=c++17 -o namespace
adrian@localhost:~/Desktop/CODE> ./namespace
Example #1:
Hello from a Pre-C++17 nested namespace!
Example #2:
Hello from a C++17 nested namespace!
adrian@localhost:~/Desktop/CODE> g++ namespace.cpp -std=c++17 -o namespace
adrian@localhost:~/Desktop/CODE> ./namespace
Example #1:
Hello from a Pre-C++17 nested namespace!
Example #2:
Hello from a C++17 nested namespace!
adrian@localhost:~/Desktop/CODE>
CODE: bash

```

## 8) Initializers added to if/switch statements

C++17 introduced for initializers in if and switch statements. This allows for more concise syntax for common coding activities such as initializing a value outside of an if statement. Often, what we really want is for the variable to be local to the if statement or switch-case statement. Initializers solve this design issue.

```
1 #include<iostream>
2
3 int main () {
4
5     //if(init; cond)
6     if (int x =9; x<9.000000001)
7         std::cout << "\nBasic Example of if initializer." << std::endl;
8     //switch(init; cond)
9     switch (long long y [] = {2198376, 7637847}; y[0]+1) {
10         case 7637847:
11             break;
12         case 2198377:
13             std::cout << "Basic Example of switch-case initializer.\n" << std::endl;
14             break;
15     }
16     return 0;
17 }
18
```

```
CODE: bash — Konsole
File Edit View Bookmarks Settings Help
adrian@localhost:~/Desktop/CODE> g++ init_if_sw.cpp -o init_if_sw
init_if_sw.cpp: In function 'int main()':
init_if_sw.cpp:6:6: warning: init-statement in selection statements only available with
      -std=c++1z or -std=gnu++1z
    if (int x =9; x<9.000000001)
        ^~~~~
init_if_sw.cpp:9:10: warning: init-statement in selection statements only available with
      -std=c++1z or -std=gnu++1z
    switch (long long y [] = {2198376, 7637847}; y[0]+1) {
            ^~~~~~
adrian@localhost:~/Desktop/CODE> ./init_if_sw
Basic Example of if initializer.
Basic Example of switch-case initializer.
adrian@localhost:~/Desktop/CODE> g++ init_if_sw.cpp -o init_if_sw
init_if_sw.cpp: In function 'int main()':
init_if_sw.cpp:6:6: warning: init-statement in selection statements only available with
      -std=c++1z or -std=gnu++1z
    if (int x =9; x<9.000000001)
        ^~~~~
init_if_sw.cpp:9:10: warning: init-statement in selection statements only available with
      -std=c++1z or -std=gnu++1z
    switch (long long y [] = {2198376, 7637847}; y[0]+1) {
            ^~~~~~
adrian@localhost:~/Desktop/CODE> ./init_if_sw
Basic Example of if initializer.
Basic Example of switch-case initializer.
adrian@localhost:~/Desktop/CODE> █
CODE: bash
```

## 9) constexpr if

C++ 17 introduced **constexpr if** statements. This allows for explicit compile time evaluation of the if condition. A list of constant expressions are available here.

```
1 #include<iostream>
2 #include<string>
3 #include<ctime>
4
5 int main (int argc, char* argv[]) {
6
7     /* An example of a an if condition that can be
8        evaluated at compile-time. This will succeed
9        with constexpr if as well as a regular if. */
10    std::string ANS;
11    if constexpr (9-9>2) ANS = "Compile-Time Evaluation with 'if constexpr'.";
12    else ANS = "Compile-Time Evaluation with 'if constexpr'.";
13    std::cout << '\n' << ANS << std::endl;
14
15    /* Two examples of runtime evaluated conditions.
16       These will fail with constexpr if but will
17       succeed with regular if. */
18
19    //The time is only known at runtime
20    std::time_t time = std::time(NULL);
21    //if constexpr (std::time (NULL) == time) {
22    if (std::time (NULL) == time) {
23        std::cout << "Runtime-Evaluated: " <<
24            std::asctime(std::localtime(&time));
25    }
26    //Args are only known at runtime
27    //if constexpr (argc == 4) {
28    if (argc == 4) {
29        std::cout << "Your CMD/Terminal Args (4) are: \"
30            << argv[0] << "\", \"\" << argv[1] << "\", \"\" << argv[2]
31            << "\", and \"\" << argv[3] << \"\".\n\" << std::endl;
32    }
33 }
```

Notice the failure if we use `if constexpr`:

```
CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> clang++ constexpr_if.cpp -std=c++17 -o constexpr_if
adrian@localhost:~/Desktop/CODE> ./constexpr_if 2 Adrian Finlay
Compile-Time Evaluation with 'if constexpr'.
Runtime-Evaluated: Thu Nov 2 11:59:59 2017
Your CMD/Terminal Args (4) are: "./constexpr_if", "2", "Adrian", and "Finlay".

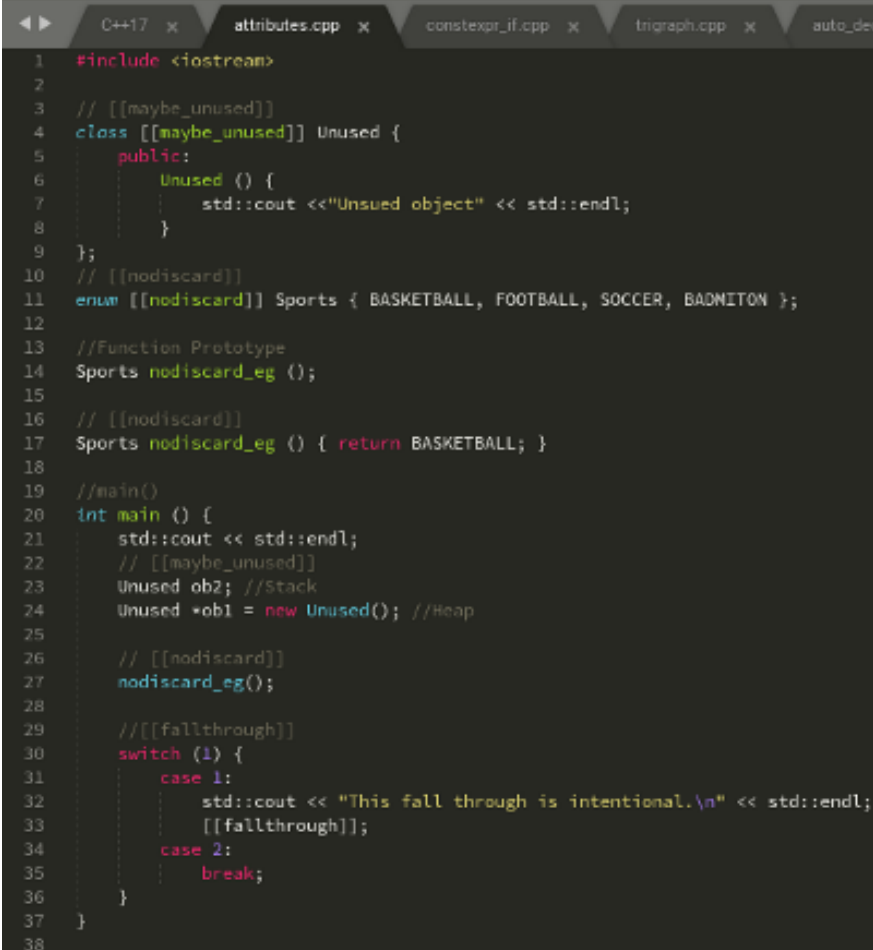
adrian@localhost:~/Desktop/CODE> g++ constexpr_if.cpp -std=c++17 -o constexpr_if
adrian@localhost:~/Desktop/CODE> ./constexpr_if 2 Adrian Finlay
Compile-Time Evaluation with 'if constexpr'.
Runtime-Evaluated: Thu Nov 2 12:00:16 2017
Your CMD/Terminal Args (4) are: "./constexpr_if", "2", "Adrian", and "Finlay".

adrian@localhost:~/Desktop/CODE> clang++ constexpr_if.cpp -std=c++17 -o constexpr_if
constexpr_if.cpp:21:16: error: constexpr if condition is not a constant expression
    if constexpr (std::time (NULL) == time) {
                   ^
constexpr_if.cpp:21:16: note: non-constexpr function 'time' cannot be used in a constant
expression
/usr/include/time.h:75:15: note: declared here
extern time_t time (time_t *__timer) __THROW;
                   ^
constexpr_if.cpp:27:16: error: constexpr if condition is not a constant expression
    if constexpr (argc == 4) {
                   ^
constexpr_if.cpp:27:16: note: read of non-const variable 'argc' is not allowed in a constant
expression
constexpr_if.cpp:5:15: note: declared here
int main (int argc, char* argv[]) {
                   ^
2 errors generated.
adrian@localhost:~/Desktop/CODE> g++ constexpr_if.cpp -std=c++17 -o constexpr_if
constexpr_if.cpp: In function 'int main(int, char*)':
constexpr_if.cpp:21:26: error: call to non-constexpr function 'time_t time(time_t*)'
    if constexpr (std::time (NULL) == time) {
                           ^
constexpr_if.cpp:27:25: error: 'argc' is not a constant expression
    if constexpr (argc == 4) {
                        ^
adrian@localhost:~/Desktop/CODE> █

CODE: bash
```

## 10) New standard attributes `[[fallthrough]]`, `[[maybe_unused]]` & `[[nodiscard]]`

C++17 introduced three new standard attributes. You will see a demonstration of their use below. A list of the standard attributes, including an explanation of the three new standard attributes are available [here](#).



```
1  #include <iostream>
2
3  // [[maybe_unused]]
4  class [[maybe_unused]] Unused {
5      public:
6          Unused () {
7              std::cout << "Unused object" << std::endl;
8          }
9  };
10 // [[nodiscard]]
11 enum [[nodiscard]] Sports { BASKETBALL, FOOTBALL, SOCCER, BADMITON };
12
13 //Function Prototype
14 Sports nodiscard_eg ();
15
16 // [[nodiscard]]
17 Sports nodiscard_eg () { return BASKETBALL; }
18
19 //main()
20 int main () {
21     std::cout << std::endl;
22     // [[maybe_unused]]
23     Unused ob2; //Stack
24     Unused *ob1 = new Unused(); //Heap
25
26     // [[nodiscard]]
27     nodiscard_eg();
28
29     //[[fallthrough]]
30     switch (1) {
31         case 1:
32             std::cout << "This fall through is intentional.\n" << std::endl;
33             [[fallthrough]];
34             case 2:
35                 break;
36     }
37 }
38
```



```
CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> clang++ attributes.cpp -o attributes -std=c++17 -Weverything -Wl
mplicit-fallthrough
attributes.cpp:4:7: warning: C++11 attribute syntax is incompatible with C++98 [-Wc++98-compat]
class [[maybe_unused]] Unused {
      ^
attributes.cpp:11:6: warning: C++11 attribute syntax is incompatible with C++98 [-Wc++98-compat]
enum [[nodiscard]] Sports { BASKETBALL, FOOTBALL, SOCCER, BADMITON };
      ^
attributes.cpp:33:4: warning: C++11 attribute syntax is incompatible with C++98 [-Wc++98-compat]
[[fallthrough]];
   ^
attributes.cpp:27:2: warning: ignoring return value of function declared with 'nodiscard'
attribute [-Wunused-result]
  nodiscard_eg();
  ^
attributes.cpp:24:10: warning: unused variable 'obl' [-Wunused-variable]
  Unused *obl = new Unused(); //Heap
         ^
5 warnings generated.
adrian@localhost:~/Desktop/CODE> g++ attributes.cpp -o attributes -std=c++17 -Wunused -Wextra -Wl
mplicit-fallthrough
attributes.cpp: In function 'int main()':
attributes.cpp:27:14: warning: ignoring returned value of type 'Sports', declared with attribute
nodiscard [-Wunused-result]
  nodiscard_eg();
  ^
attributes.cpp:17:8: note: in call to 'Sports nodiscard_eg()', declared here
  Sports nodiscard_eg () { return BASKETBALL; }
  ^
attributes.cpp:11:20: note: 'Sports' declared here
enum [[nodiscard]] Sports { BASKETBALL, FOOTBALL, SOCCER, BADMITON };
                   ^
attributes.cpp:24:10: warning: unused variable 'obl' [-Wunused-variable]
  Unused *obl = new Unused(); //Heap
         ^
adrian@localhost:~/Desktop/CODE> █
```

## 11) Attributes for Enumerator & Namespaces

C++17 introduced support for support for Attributes on Enumerators & Namespaces, which were formerly illegal.

```
C++17  attributes2.cpp  attributes.cpp  constexpr_if.cpp  loggraph.cpp  auto_deduc.cpp
1 //Attribute [[deprecated]] with a namespace, C++17
2 namespace [[deprecated]] attributes_namespace { }
3 //Attribute [[deprecated]] with an enumeration, C++17
4 enum [[deprecated]] attributes_enum { EMPTY };
5
6 int main () {}
7
8
9
10 adrian@localhost:~/Desktop/CODE> clang++ attributes2.cpp -o attributes2 -std=c++17 -Weverything
attributes2.cpp:2:11: warning: C++11 attribute syntax is incompatible with C++98 [-Wc++98-compat]
namespace [[deprecated]] attributes_namespace { }
          ^
attributes2.cpp:4:6: warning: C++11 attribute syntax is incompatible with C++98 [-Wc++98-compat]
enum [[deprecated]] attributes_enum { EMPTY };
     ^
2 warnings generated.
adrian@localhost:~/Desktop/CODE> g++ attributes2.cpp -o attributes2 -std=c++17 -Wextra
attributes2.cpp:2:47: warning: 'deprecated' attribute directive ignored [-Wattributes]
namespace [[deprecated]] attributes_namespace { }
              ^
adrian@localhost:~/Desktop/CODE> █
```

## 12) Error message for static\_assert now optional

With C++17, the error message in the keyword **static\_assert** is now optional. Notice the warning in C++14.

```
C++17 x static_assert.cpp x attributes2.cpp e attributes.cpp x constexpr_if.cpp x
1 constexpr bool is10 (double x) {
2     if (x == 10) return true;
3     else return false;
4 }
5
6 int main () {
7     //Illegal before C++17
8     static_assert(is10(10.00001));
9     //Only legal way before C++17
10    static_assert(is10(10.00001), "No, it is not 10.");
11 }
12
13
14 CODE: bash — Konsole
15 adrian@localhost:~/Desktop/CODE> clang++ static_assert.cpp -o static_assert -std=c++14
16 static_assert.cpp:8:30: warning: static_assert with no message is a C++17 extension
17     static_assert(is10(10.00001));
18         ^
19 static_assert.cpp:8:2: error: static_assert failed
20     static_assert(is10(10.00001));
21     ^
22 static_assert.cpp:10:2: error: static_assert failed "No, it is not 10."
23     static_assert(is10(10.00001), "No, it is not 10.");
24     ^
25
26 1 warning and 2 errors generated.
27 adrian@localhost:~/Desktop/CODE> g++ static_assert.cpp -o static_assert -std=c++14
28 static_assert.cpp: In function 'int main()':
29 static_assert.cpp:8:2: error: static assertion failed
30     static_assert(is10(10.00001));
31     ^
32 static_assert.cpp:10:2: error: static assertion failed: No, it is not 10.
33     static_assert(is10(10.00001), "No, it is not 10.");
34     ^
35 adrian@localhost:~/Desktop/CODE>
```

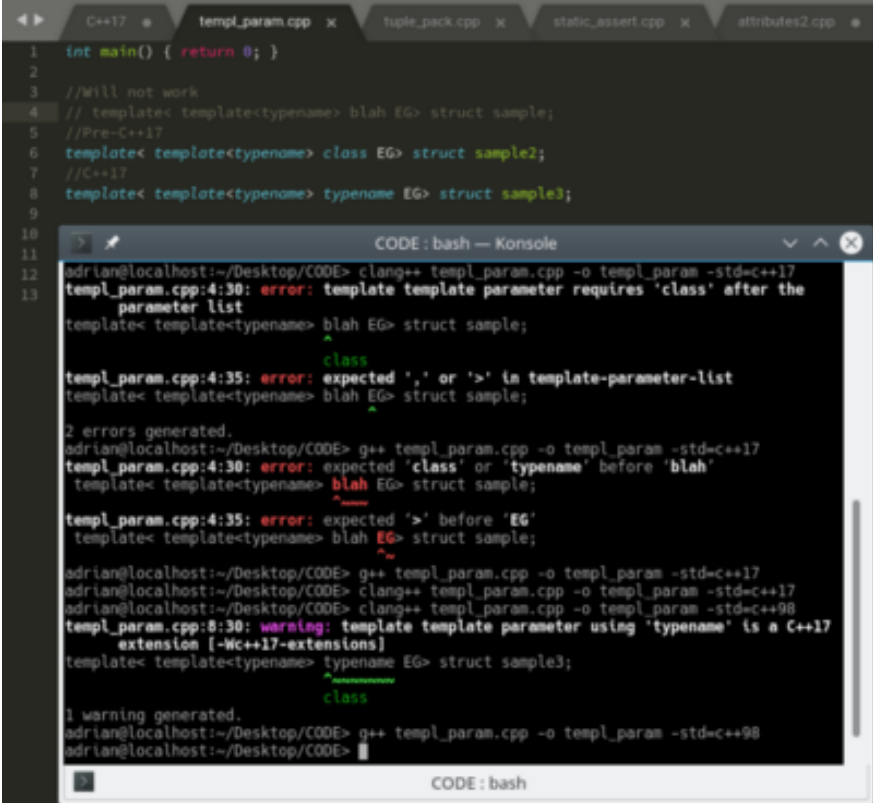
### 13) Structured binding declarations

C++17 introduced initialization by deconstruction of a tuple like object with auto. The values are bound to the original object.

```
C++17 e tuple_pack.cpp x static_assert.cpp x attributes2.cpp e attributes.cpp x
1 #include <iostream>
2 #include <tuple>
3
4 int main () {
5     //A tuple with four elements
6     std::tuple <const char *, const char *,
7         const char *, int> _tuple = std::make_tuple ("Dog", "Cat", "Hyena", 99);
8
9     //C++17
10    auto [a,b,c,d] = _tuple;
11    std::cout << std::endl;
12    std::cout << "Contents of the tuple:\t" << a << ", " << b << ", " << c
13        << ", " << d << ".\n" << std::endl;
14    std::cout << std::endl;
15 }
16
17
18 CODE: bash — Konsole
19 adrian@localhost:~/Desktop/CODE> clang++ tuple_pack.cpp -o tuple_pack -std=c++17
20 adrian@localhost:~/Desktop/CODE> ./tuple_pack
21
22 Contents of the tuple: Dog, Cat, Hyena, 99.
23
24 adrian@localhost:~/Desktop/CODE> g++ tuple_pack.cpp -o tuple_pack -std=c++17
25 tuple_pack.cpp:10:7: warning: decomposition declaration only available with -std=c++1z
26     or -std=gnu++1z
27     auto [a,b,c,d] = _tuple;
28         ^
29 adrian@localhost:~/Desktop/CODE> ./tuple_pack
30
31 Contents of the tuple: Dog, Cat, Hyena, 99.
32
33 adrian@localhost:~/Desktop/CODE>
```

## 14) Keyword typename now allowed in lieu of class in a template's template parameter

C++17 now allows the use of the keyword `typename` in lieu of `class` in a template's template parameter. Curiously enough, while Clang, by default, will warn you about the potential illegal use of the keyword `template` in the aforementioned situation in C++98, GCC does not do so by default.



```
1 int main() { return 0; }
2
3 //Will not work
4 // template< template<typename> blah EG> struct sample;
5 //Pre-C++17
6 template< template<typename> class EG> struct sample2;
7 //C++17
8 template< template<typename> typename EG> struct sample3;
9
10
11
12
13
CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> clang++ templ_param.cpp -o templ_param -std=c++17
templ_param.cpp:4:30: error: template template parameter requires 'class' after the
      parameter list
template< template<typename> blah EG> struct sample;
                        ^
                        class
templ_param.cpp:4:35: error: expected ',' or '>' in template-parameter-list
template< template<typename> blah EG> struct sample;
                        ^
2 errors generated.
adrian@localhost:~/Desktop/CODE> g++ templ_param.cpp -o templ_param -std=c++17
templ_param.cpp:4:30: error: expected 'class' or 'typename' before 'blah'
template< template<typename> blah EG> struct sample;
                        ^~~~~
templ_param.cpp:4:35: error: expected '>' before 'EG'
template< template<typename> blah EG> struct sample;
                        ^
adrian@localhost:~/Desktop/CODE> g++ templ_param.cpp -o templ_param -std=c++17
adrian@localhost:~/Desktop/CODE> clang++ templ_param.cpp -o templ_param -std=c++17
adrian@localhost:~/Desktop/CODE> clang++ templ_param.cpp -o templ_param -std=c++98
templ_param.cpp:8:30: warning: template template parameter using 'typename' is a C++17
      extension [-Wc++17-extensions]
template< template<typename> typename EG> struct sample3;
                        ^~~~~~
                        class
1 warning generated.
adrian@localhost:~/Desktop/CODE> g++ templ_param.cpp -o templ_param -std=c++98
adrian@localhost:~/Desktop/CODE>
```

## 15) Constant evaluation for non-type template arguments

C++17 now allows constant evaluation for non-type template arguments. You should read more about this, [here](#).

```
1  #include<iostream>
2
3  static int k = 0;
4
5  //An Example of a Template Class
6  //with a pointer to a member as a template argument
7  template<int *i> class EG {
8      public:
9          void talk() {
10              if (k>0) {
11                  std::cout << "Hello, again. #" << k << std::endl;
12              }
13              else std::cout << "\nHello." << std::endl;
14              k++;
15          }
16      };
17
18  //EG #1, #3
19  int i = 9;
20  //EG #2
21  constexpr int *x () { return &i; }
22
23
24  int main () {
25
26      EG<i> eg;
27      eg.talk();
28
29      EG<x()> eg2; //Why is this an issue for GCC???
30      eg2.talk();
31
32      EG<i> eg3;
33      eg3.talk();
34
35      std::cout<<std::endl;
36
37  }
38
```

```
CODE: bash — Konsole
File Edit View Bookmarks Settings Help
adrian@localhost:~/Desktop/CODE> clang++ const_nontype.cpp -o const_nontype -std=c++17
adrian@localhost:~/Desktop/CODE> ./const_nontype
Hello.
Hello, again. #1
Hello, again. #2
adrian@localhost:~/Desktop/CODE> g++ const_nontype.cpp -o const_nontype -std=c++17
const_nontype.cpp: In function 'int main()':
const_nontype.cpp:29:8: error: 'x()' is not a valid template argument for 'int*' because it is not the address
of a variable
    EG<x()> eg2; //Why is this an issue for GCC???
    ^
const_nontype.cpp:30:6: error: request for member 'talk' in 'eg2', which is of non-class type 'int'
    eg2.talk();
    ^~~~~
adrian@localhost:~/Desktop/CODE>
```


For some reason GCC does not play well when I call the function within the template parameter declaration. However it works on other compilers, and clang has no problem with it. I tried many things (providing a public, default constructor/destructor) to no avail. GCC is

reputed as being a strict compiler. If you can figure out the issue, please inform me in the comments.

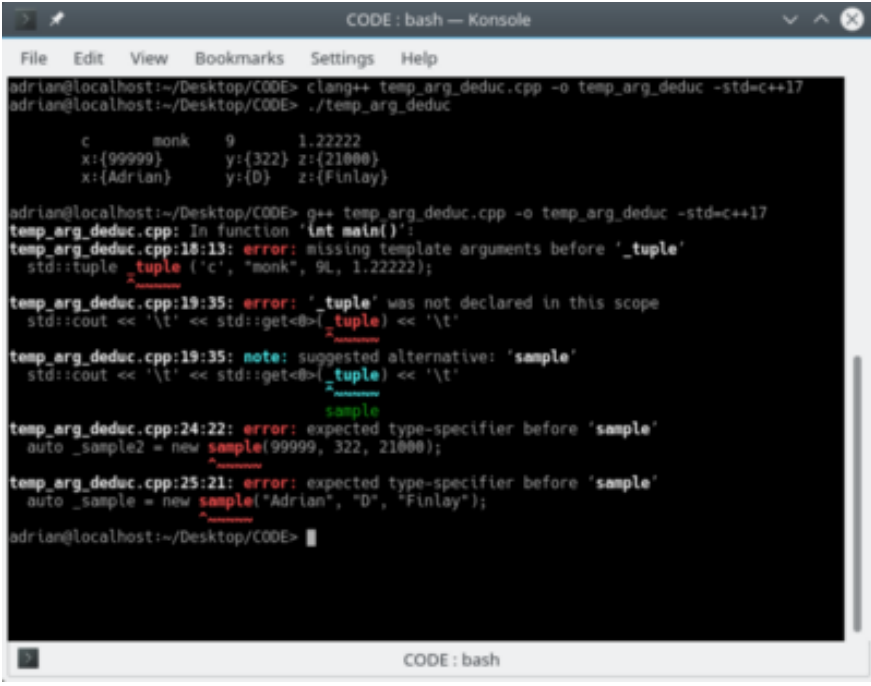
**UPDATE:** Many thanks to the people who have tested the code with GCC, including @Ciel who ran it on SUSE Linux with GCC 7.2.1. It turns out the issue may be local to my installation.

## 16) Class template argument deduction

C++17 introduces argument deduction for class template instantiation. Users of languages such as Java, C# may find this familiar. You will want to read more about this feature here. Some basic usage is demonstrated below.



```
1  #include<iostream>
2  #include<tuple>
3
4  template<typename Tn> class sample {
5      public:
6          sample (Tn x, Tn y, Tn z) {
7              std::cout << "\tx:{" << x << "}" << "\t"
8                  << "y:{" << y << "}" << "\t" << "z:{"
9                  << z << "}" << "\t" << std::endl;
10             }
11 };
12
13 int main () {
14     std::cout << std::endl;
15
16     // Declaration that specifies initialization
17     // of a variable and variable template
18     std::tuple _tuple ('c', "monk", 9L, 1.22222);
19     std::cout << '\t' << std::get<0>(_tuple) << '\t'
20         << std::get<1>(_tuple) << '\t' << std::get<2>(_tuple)
21         << '\t' << std::get<3>(_tuple) << '\t' << std::endl;
22
23     //Dynamic allocator Expression
24     auto _sample2 = new sample(99999, 322, 21000);
25     auto _sample = new sample("Adrian", "D", "Finlay");
26     // This won't compile. Types must be the same.
27     // auto _sample = new sample(99, "Monk", 32);
28
29     std::cout << std::endl;
30     return 0;
31 }
32
```



```
CODE: bash — Konsole
File Edit View Bookmarks Settings Help
adrian@localhost:~/Desktop/CODE> clang++ temp_arg_deduc.cpp -o temp_arg_deduc -std=c++17
adrian@localhost:~/Desktop/CODE> ./temp_arg_deduc

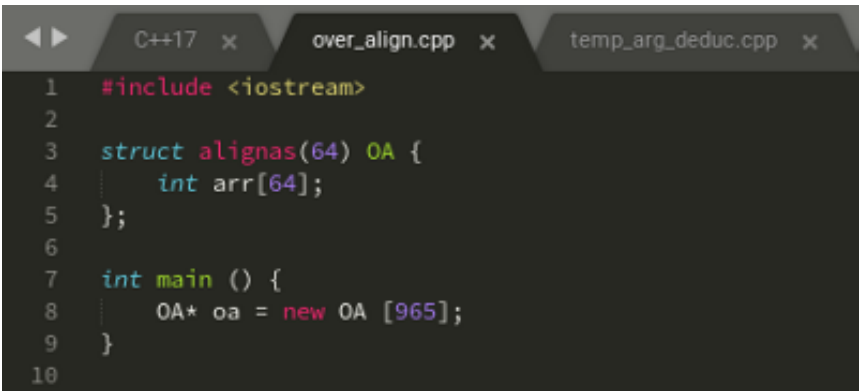
c      monk      9      1.22222
x:{99999}  y:{322}  z:{21000}
x:{Adrian}  y:{D}    z:{Finlay}

adrian@localhost:~/Desktop/CODE> g++ temp_arg_deduc.cpp -o temp_arg_deduc -std=c++17
temp_arg_deduc.cpp: In function 'int main()':
temp_arg_deduc.cpp:18:13: error: missing template arguments before '_tuple'
    std::tuple ('c', "monk", 9L, 1.22222);
                ^~~~~~
temp_arg_deduc.cpp:19:35: error: '_tuple' was not declared in this scope
    std::cout << '\t' << std::get<0>(_tuple) << '\t'
                                   ^~~~~~
temp_arg_deduc.cpp:19:35: note: suggested alternative: 'sample'
    std::cout << '\t' << std::get<0>(_tuple) << '\t'
                                   ^~~~~~
                                   sample
temp_arg_deduc.cpp:24:22: error: expected type-specifier before 'sample'
    auto _sample2 = new sample(99999, 322, 21000);
                        ^~~~~~
temp_arg_deduc.cpp:25:21: error: expected type-specifier before 'sample'
    auto _sample = new sample("Adrian", "D", "Finlay");
                        ^~~~~~
adrian@localhost:~/Desktop/CODE> █
```

It appears that GCC hasn't yet implemented this feature. I could be wrong, but it appears that way. Perhaps there is a switch that needs to be enabled. If you can get this code to run on GCC, do let me know.

## 17) Extensions on over-aligned Memory Allocation


C++17 overloads the new operator to provide support for correctly dynamically allocating over-aligned data. Intel's compiler had supported this feature by way of their own work `<aligned_new>`. You may view the paper [here](#). I also strongly recommend checking out these pages for more understanding as to the changes in memory management C++17: `std::aligned_alloc`, `std::align_val_t`, `std::align`. Lastly, you should look at the description about how the new operator has been overloaded to reflect these changes.



```
C++17 x over_align.cpp x temp_arg_deduc.cpp x
1  #include <iostream>
2
3  struct alignas(64) OA {
4      int arr[64];
5  };
6
7  int main () {
8      OA* oa = new OA [965];
9  }
10
```

## 18) Fold Expressions

One of the major new features of C++, C++17 introduces Fold Expressions, a mechanism which reduces a parameter pack over a binary operator. While we will not cover all the aspects of fold expressions (you should **absolutely** do so yourself, see this page), we will show some basic use. There are binary and unary folds.

A screenshot of a C++ code editor with a dark theme. The editor shows a file named 'fold\_expr.cpp' with several tabs open at the top: 'fold\_expr.cpp', 'using\_constr.cpp', 'SUBLINE.LICENSE.txt', and 'templ\_match.cpp'. The code is a C++ program demonstrating fold expressions. It includes `<iostream>` and defines several function prototypes for unary and binary folds. The function definitions use `std::cout` to print the results of the folds. The code is as follows:

```
1 #include<iostream>
2
3 /* Function Prototypes */
4
5 //Unary Right Fold ( pack op ... )
6 template<typename ...vargs> void UR_MULT (vargs... x);
7 //Unary Left Fold ( ... op pack )
8 template<typename ...vargs> void UL_DIVIDE (vargs... x);
9 //Binary Right Fold ( pack op ... op init )
10 template<typename ...vargs> void BR_XOR (vargs... x);
11 //Binary Left Fold ( op init ... pack op )
12 template<typename ...vargs> void BL_X (vargs... x);
13
14 //Function Definitions
15 template<typename ...vargs>
16 void UR_MULT (vargs... x) {
17     std::cout<< "\nUNARY RIGHT FOLD:\tUL_MULT():\t" << (... * x) << '.';
18 }
19 template<typename ...vargs>
20 void UL_DIVIDE (vargs... x) {
21     std::cout<< "\nUNARY LEFT FOLD:\tUL_DIVIDE():\t" << (... / x) << '.';
22 }
23 template<typename ...vargs>
24 void BR_XOR (vargs... x) {
25     std::cout<< "\nBINARY RIGHT FOLD:\tBR_ADD():\t"
26         << (x ^ ... ^ 12) << '.';
27 }
28 template<typename ...vargs>
29 void BR_SOR (vargs... x) {
30     std::cout<< "\nBINARY LEFT FOLD:\tBR_ADD():\t";
31     bool result = (false || ... || x);
32     if (result == 1)
33         std::cout<< "true" << '.';
34     else
35         std::cout<< "false" << '.';
36 }
37
```

```

37
38 //main()
39 int main () {
40     std::cout<<std::endl;
41
42     //UNARY RIGHT FOLD
43     UR_MULT(9,3,4,5);
44
45     //UNARY LEFT FOLD
46     UL_DIVIDE(99,2,9,1); //Note that this will fail if it were a right fold.
47
48     //BINARY RIGHT FOLD
49     BR_XOR(12,3,5);
50
51     //BINARY LEFT FOLD
52     // Empty Pack is false by default so {false || true} = true
53     BR_SOR();
54
55     std::cout<< '\n' <<std::endl;
56     return 0;
57 }
58

```

```

CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> g++ fold_expr.cpp -o fold_expr -std=c++17
adrian@localhost:~/Desktop/CODE> clang++ fold_expr.cpp -o fold_expr -std=c++17
adrian@localhost:~/Desktop/CODE> ./fold_expr

UNARY RIGHT FOLD:      UL_MULT():      540.
UNARY LEFT FOLD:       UL_DIVIDE():     5.
BINARY RIGHT FOLD:     BR_ADD():        6.
BINARY LEFT FOLD:      BR_ADD():        false.

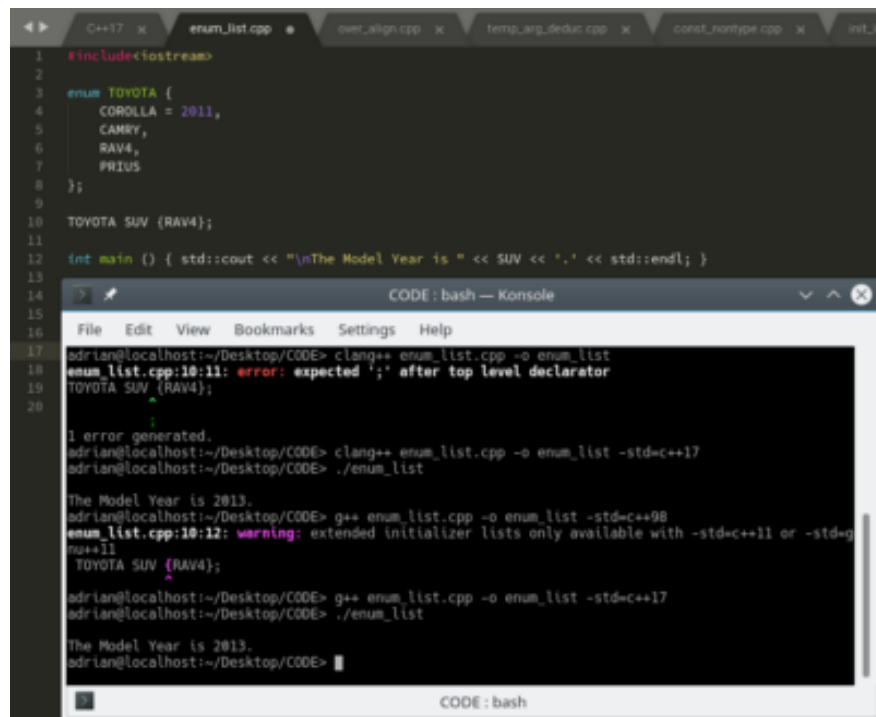
adrian@localhost:~/Desktop/CODE> clang++ fold_expr.cpp -o fold_expr -std=c++14
fold_expr.cpp:17:54: warning: pack fold expression is a C++17 extension [-Wc++17-extensions]
    std::cout<< "\nUNARY RIGHT FOLD:\tUL_MULT():\t" << {... * x} << '\n';
                                           ^
fold_expr.cpp:21:55: warning: pack fold expression is a C++17 extension [-Wc++17-extensions]
    std::cout<< "\nUNARY LEFT FOLD:\tUL_DIVIDE():\t" << {... / x} << '\n';
                                           ^
fold_expr.cpp:26:11: warning: pack fold expression is a C++17 extension [-Wc++17-extensions]
    << (x ^ ... ^ 12) << '\n';
              ^
fold_expr.cpp:31:26: warning: pack fold expression is a C++17 extension [-Wc++17-extensions]
    bool result = (false || ... || x);
                        ^
4 warnings generated.
adrian@localhost:~/Desktop/CODE> g++ fold_expr.cpp -o fold_expr -std=c++14
fold_expr.cpp: In function 'void UR_MULT(vargs ...)':
fold_expr.cpp:17:60: warning: fold-expressions only available with -std=c++1z or -std=gnu++1z
    std::cout<< "\nUNARY RIGHT FOLD:\tUL_MULT():\t" << {... * x} << '\n';
                                                           ^
fold_expr.cpp: In function 'void UL_DIVIDE(vargs ...)':
fold_expr.cpp:21:61: warning: fold-expressions only available with -std=c++1z or -std=gnu++1z
    std::cout<< "\nUNARY LEFT FOLD:\tUL_DIVIDE():\t" << {... / x} << '\n';
                                                           ^
fold_expr.cpp: In function 'void BR_XOR(vargs ...)':
fold_expr.cpp:26:17: warning: fold-expressions only available with -std=c++1z or -std=gnu++1z
    << (x ^ ... ^ 12) << '\n';
                  ^
fold_expr.cpp: In function 'void BR_SOR(vargs ...)':
fold_expr.cpp:31:33: warning: fold-expressions only available with -std=c++1z or -std=gnu++1z
    bool result = (false || ... || x);
                          ^
adrian@localhost:~/Desktop/CODE>
CODE: bash

```

## 19) List-style Initialization of Enumerations

C++17 now introduces an optional attribute specifier sequence in initializing enumerations.





```
1 #include<iostream>
2
3 enum TOYOTA {
4     COROLLA = 2011,
5     CAMRY,
6     RAV4,
7     PRIUS
8 };
9
10 TOYOTA SUV (RAV4);
11
12 int main () { std::cout << "\nThe Model Year is " << SUV << '.' << std::endl; }
13
14
15
16
17
18
19
20
```

CODE: bash — Konsole

File Edit View Bookmarks Settings Help

```
adrian@localhost:~/Desktop/CODE> clang++ enum_list.cpp -o enum_list
enum_list.cpp:10:11: error: expected ';' after top level declarator
TOYOTA SUV (RAV4);
          ^
1 error generated.
adrian@localhost:~/Desktop/CODE> clang++ enum_list.cpp -o enum_list -std=c++17
adrian@localhost:~/Desktop/CODE> ./enum_list
The Model Year is 2013.
adrian@localhost:~/Desktop/CODE> g++ enum_list.cpp -o enum_list -std=c++98
enum_list.cpp:10:12: warning: extended initializer lists only available with -std=c++11 or -std=g
nu++11
    TOYOTA SUV (RAV4);
               ^
adrian@localhost:~/Desktop/CODE> g++ enum_list.cpp -o enum_list -std=c++17
adrian@localhost:~/Desktop/CODE> ./enum_list
The Model Year is 2013.
adrian@localhost:~/Desktop/CODE>
```

## 20) Specifying non-type template parameters with auto

C++17 now allows you to specify non-type template parameters with auto.

```
C++17 • auto_temp.cpp x enum_list.cpp
1  #include<iostream>
2
3  template <auto pm> class X {
4      public:
5          X () {
6              std::cout << "\n" << "Hiya, "
7                  << pm << '.' << std::endl;
8          }
9  };
10
11 template <int size, auto ... pm> class Y {
12     public:
13         Y () {
14             std::cout << "Hello, "
15                 << "with size of " << size
16                 << '.' << std::endl;
17             //Expansion of Variadic Arguments
18             //Not allowed here.
19         }
20 };
21
22 int main () {
23     X eg1 = X <'D'> ();
24     Y eg2 = Y <3, 9, 1, 3> ();
25 }
26
```

```
CODE: bash — Konsole
File Edit View Bookmarks Settings Help
adrian@localhost:~/Desktop/CODE> g++ auto_temp.cpp -o auto_temp -std=c++14
auto_temp.cpp:3:16: error: 'auto' parameter not permitted in this context
template <auto pm> class X {
               ^
auto_temp.cpp:11:30: error: 'auto' parameter not permitted in this context
template <int size, auto ... pm> class Y {
                             ^
auto_temp.cpp: In function 'int main()':
auto_temp.cpp:23:4: error: missing template arguments before 'eg1'
  X eg1 = X <'D'> ();
  ^
auto_temp.cpp:24:4: error: missing template arguments before 'eg2'
  Y eg2 = Y <3, 9, 1, 3> ();
  ^
adrian@localhost:~/Desktop/CODE> clang++ auto_temp.cpp -o auto_temp -std=c++14
auto_temp.cpp:3:11: error: 'auto' not allowed in template parameter until C++17
template <auto pm> class X {
          ^
auto_temp.cpp:11:21: error: 'auto' not allowed in template parameter until C++17
template <int size, auto ... pm> class Y {
                    ^
auto_temp.cpp:23:2: error: use of class template 'X' requires template arguments
  X eg1 = X <'D'> ();
  ^
auto_temp.cpp:3:26: note: template is declared here
template <auto pm> class X {
                         ^
auto_temp.cpp:24:2: error: use of class template 'Y' requires template arguments
  Y eg2 = Y <3, 9, 1, 3> ();
  ^
auto_temp.cpp:11:40: note: template is declared here
template <int size, auto ... pm> class Y {
                                       ^
4 errors generated.
adrian@localhost:~/Desktop/CODE> g++ auto_temp.cpp -o auto_temp -std=c++17
adrian@localhost:~/Desktop/CODE> ./auto_temp
Hiya, D.
Hello, with size of 3.
adrian@localhost:~/Desktop/CODE> clang++ auto_temp.cpp -o auto_temp -std=c++17
adrian@localhost:~/Desktop/CODE> ./auto_temp
Hiya, D.
Hello, with size of 3.
adrian@localhost:~/Desktop/CODE>
```

## 21) constexpr lambda expressions

C++17 now allows you to explicitly specify a lambda expression as constexpr. In the absence of constexpr, had a lambda qualified, it would have been treated as constexpr anyway. This is still true. Read more about this, [here](#).

```
C++17 x constexpr_lambda.cpp bashrc x const_nontype.cpp x
1  #include <iostream>
2  #include <string>
3
4  /*
5   constexpr: explicitly specifies that the function call operator
6   is a constexpr function. When this specifier is not present, the
7   function call operator will be constexpr anyway, if it happens to
8   satisfy all constexpr function requirements (cppreference.com)
9
10 */
11
12 constexpr auto lambda = [] (std::string x) {
13     std::cout << '\n' << x << '\n' << std::endl;
14 };
15
16 constexpr auto lambda2 = [] (int x, int y) {
17     return x < y;
18 };
19
20
21 int main () {
22
23     //A static lambda expressions
24     lambda("Hello Janee :)");
25
26     //To demonstrate that this is done statically, at compile time.
27     static_assert( (lambda2(7,9)), "False");
28     //static_assert( (lambda2(9,7)), "False");
29 }
30
```

```
CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> clang++ constexpr_lambda.cpp -o constexpr_lambda -std=c++17
adrian@localhost:~/Desktop/CODE> ./constexpr_lambda
Hello Janee :)
adrian@localhost:~/Desktop/CODE> g++ constexpr_lambda.cpp -o constexpr_lambda-std=c++17
constexpr_lambda.cpp: In function 'int main()':
constexpr_lambda.cpp:27:2: error: non-constant condition for static assertion
    static_assert( (lambda2(7,9)), "False");
    ^~~~~~
constexpr_lambda.cpp:27:25: error: call to non-constexpr function '<lambda(int, int)>'
    static_assert( (lambda2(7,9)), "False");
    ^~~~~~
constexpr_lambda.cpp:16:42: note: '<lambda(int, int)>' is not usable as a constexpr function because
e:
    constexpr auto lambda2 = [] (int x, int y) {
adrian@localhost:~/Desktop/CODE>
```

For some reason GCC gave me errors when attempting to compile/link/run but it seems to be a local issue, as other SUSE Linux users with similar configurations have built an executable with no errors. Many thanks to [Ciel](#) for assistance with this.

## 22) Lambda this by value (\*this)

C++17 introduced the ability by a lambda expression to capture its invoking object by value in addition to by reference.

```
1  #include <iostream>
2  #include <string>
3
4  class Obj {
5  private:
6      std::string name = "Adrian";
7  public:
8      //Lambda by Reference, Pre-C++17
9      auto byRef () {
10         return [this] { return name; };
11     }
12
13     //Lambda by Value, C++17
14     auto byVal () {
15         return [+this] { return name; };
16     }
17
18     //Set name
19     void setName(std::string name) {
20         std::cout<<"Let's change the value.\n";
21         this->name=name;
22     }
23 };
24
25
26 int main () {
27     Obj ob;
28
29     auto val = ob.byRef();
30     std::cout<< '\n' << "The Name, by value: " << val() << ".\n";
31     //Let's change name.
32     ob.setName("Sydni");
33
34     auto ref = ob.byVal();
35     std::cout<< "The Name, by reference: " << ref() << ".\n";
36
37     std::cout<< std::endl;
38 }
```

```
CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> g++ lambda_this.cpp -o lambda_this -std=c++17
adrian@localhost:~/Desktop/CODE> ./lambda_this

The Name, by value: Adrian.
Let's change the value.
The Name, by reference: Sydni.

adrian@localhost:~/Desktop/CODE> clang++ lambda_this.cpp -o lambda_this -std=c++17
adrian@localhost:~/Desktop/CODE> ./lambda_this

The Name, by value: Adrian.
Let's change the value.
The Name, by reference: Sydni.

adrian@localhost:~/Desktop/CODE> █
```

## 23) Extending Aggregate Initialization to Base Types

C++17 introduced aggregate initialization of base types. You may want to read more about this, [here](#).

```
aggregate.cpp x register.cpp x bool++_de
1  #include<iostream>
2
3  struct B {
4      int x,y;
5  };
6
7  struct D : B {
8      short z[3];
9  };
10
11 int main () {
12     D EG[ {42, 63}, {4,6,9} ];
13     std::cout << "\nThese are the contents.\n";
14
15     std::cout << "EG.x = \t" << EG.x << '\n';
16     std::cout << "EG.y = \t" << EG.y << '\n';
17
18     int ix =0;
19     for (short _z : EG.z) {
20         std::cout<<"Z[" << ix << "] = \t" << _z << '\n';
21         ix++;
22     }
23     std::cout<< std::endl;
24 }
25
```

```
CODE: bash — Konsole
File Edit View Bookmarks Settings Help

aggregate.cpp: In function 'int main()':
aggregate.cpp:12:26: error: no matching function for call to 'D::D(<brace-enclosed initializer list>)'
    D EG( {42, 63}, {4,6,9} );
                        ^
aggregate.cpp:7:8: note: candidate: D::D()
    struct D : B {
    ^
aggregate.cpp:7:8: note:   candidate expects 0 arguments, 2 provided
aggregate.cpp:7:8: note: candidate: constexpr D::D(const D&)
aggregate.cpp:7:8: note:   candidate expects 1 argument, 2 provided
aggregate.cpp:7:8: note: candidate: constexpr D::D(D&)
aggregate.cpp:7:8: note:   candidate expects 1 argument, 2 provided
adrian@localhost:~/Desktop/CODE> clang++ aggregate.cpp -o aggregate -std=c++14
aggregate.cpp:12:4: error: no matching constructor for initialization of 'D'
    D EG( {42, 63}, {4,6,9} );
    ^
aggregate.cpp:7:8: note: candidate constructor (the implicit copy constructor) not viable: requires 1
argument, but 2 were provided
struct D : B {
    ^
aggregate.cpp:7:8: note: candidate constructor (the implicit move constructor) not viable: requires 1
argument, but 2 were provided
aggregate.cpp:7:8: note: candidate constructor (the implicit default constructor) not viable: requires 0
arguments, but 2 were provided
1 error generated.
adrian@localhost:~/Desktop/CODE> clang++ aggregate.cpp -o aggregate -std=c++17
adrian@localhost:~/Desktop/CODE> g++ aggregate.cpp -o aggregate -std=c++17
adrian@localhost:~/Desktop/CODE> ./aggregate

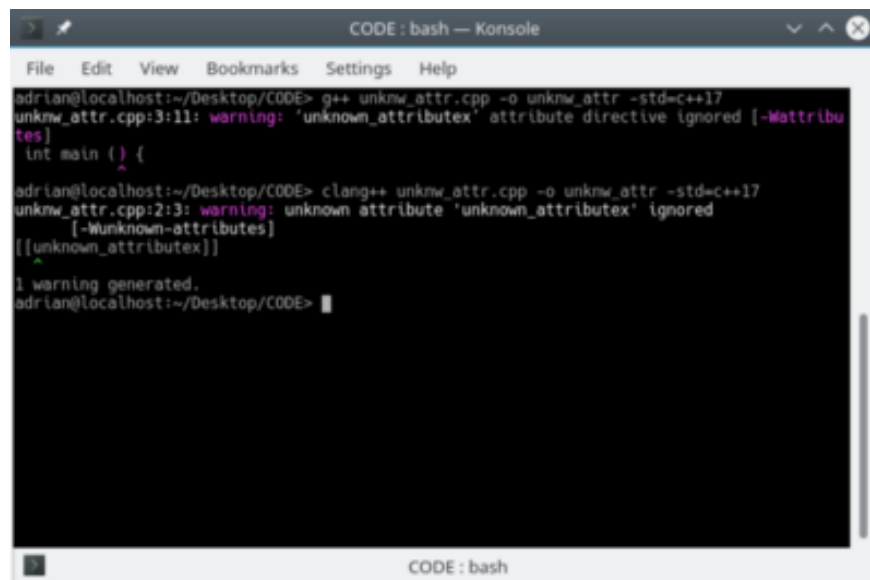
These are the contents.
EG.x = 42
EG.y = 63
Z[0] = 4
Z[1] = 6
Z[2] = 9

adrian@localhost:~/Desktop/CODE>
```

## 24) Unknown Attributes Required to be Ignored

C++17 now mandates that unknown attributes are required to be ignored.

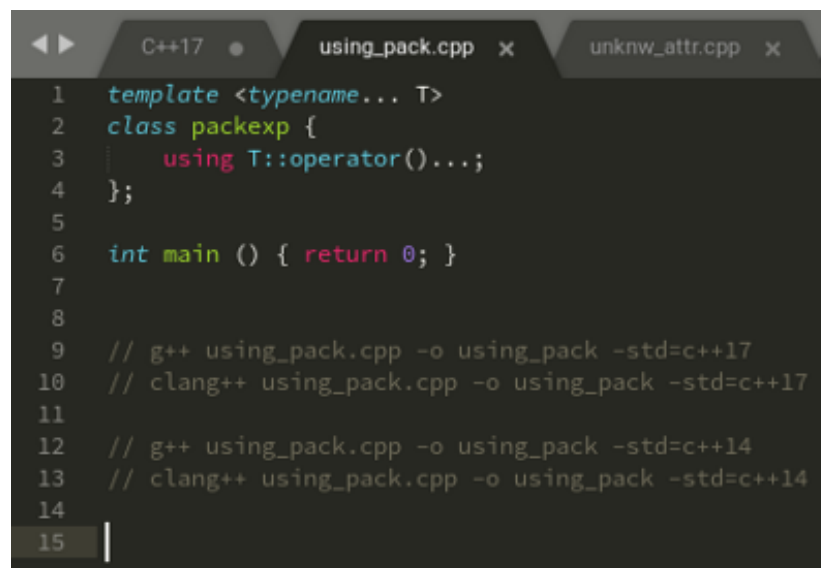
```
C++17  unknw_attr.cpp  aggregate.cpp
1
2  [[unknown_attribute]]
3  int main () {
4      return 0;
5  }
6
7  // g++ unknw_attr.cpp -o unknw_attr -std=c++17
8  // clang++ unknw_attr.cpp -o unknw_attr -std=c++17
```



```
CODE : bash — Konsole
File Edit View Bookmarks Settings Help
adrian@localhost:~/Desktop/CODE> g++ unknw_attr.cpp -o unknw_attr -std=c++17
unknw_attr.cpp:3:11: warning: 'unknown_attributex' attribute directive ignored [-Wattribu
tes]
int main () {
^
adrian@localhost:~/Desktop/CODE> clang++ unknw_attr.cpp -o unknw_attr -std=c++17
unknw_attr.cpp:2:3: warning: unknown attribute 'unknown_attributex' ignored
[-Wunknown-attributes]
[[unknown_attributex]]
^
1 warning generated.
adrian@localhost:~/Desktop/CODE>
```

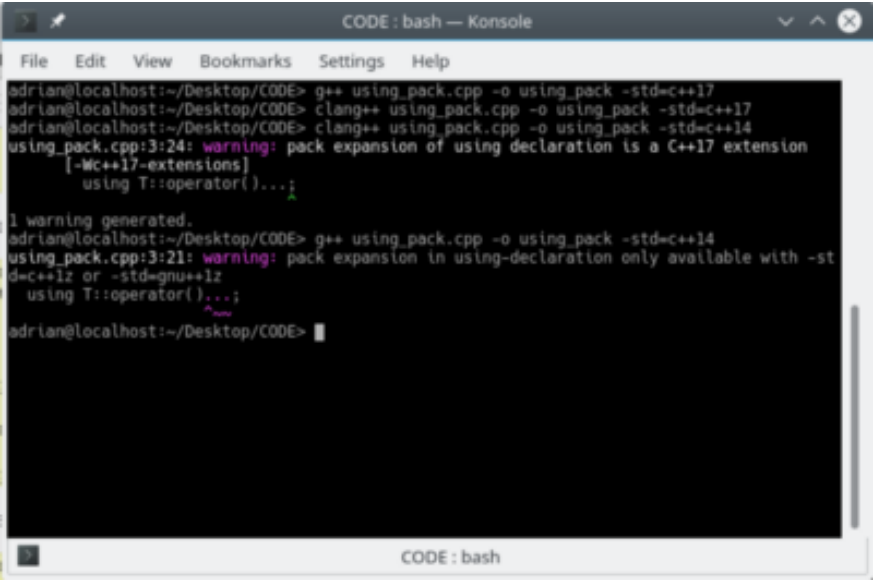
## 25) Pack Expansions legal in using declarations

C++17 now allows using declarations to make use of pack expansions. You should read more about the motivations of this feature, [here](#), as I will only show its basic use.



```
C++17 using_pack.cpp x unknw_attr.cpp x
1 template <typename... T>
2 class packexp {
3     using T::operator()...;
4 };
5
6 int main () { return 0; }
7
8
9 // g++ using_pack.cpp -o using_pack -std=c++17
10 // clang++ using_pack.cpp -o using_pack -std=c++17
11
12 // g++ using_pack.cpp -o using_pack -std=c++14
13 // clang++ using_pack.cpp -o using_pack -std=c++14
14
15
```



A screenshot of a terminal window titled "CODE: bash — Konsole". The window has a menu bar with "File", "Edit", "View", "Bookmarks", "Settings", and "Help". The terminal shows the following commands and output:

```
adrian@localhost:~/Desktop/CODE> g++ using_pack.cpp -o using_pack -std=c++17
adrian@localhost:~/Desktop/CODE> clang++ using_pack.cpp -o using_pack -std=c++17
adrian@localhost:~/Desktop/CODE> clang++ using_pack.cpp -o using_pack -std=c++14
using_pack.cpp:3:24: warning: pack expansion of using declaration is a C++17 extension
    [-Wc++17-extensions]
    using T::operator()...;
                        ^
1 warning generated.
adrian@localhost:~/Desktop/CODE> g++ using_pack.cpp -o using_pack -std=c++14
using_pack.cpp:3:21: warning: pack expansion in using-declaration only available with -std=c++1z or -std=gnu++1z
    using T::operator()...;
                    ^~~~~
adrian@localhost:~/Desktop/CODE>
```

## 26) Generalization of Range-based for loop

C++17 generalizes the range-based for loop by relaxing the requirement that the beginning and ending types be of the same type. Consequently, You might want to see how this expands to a normal for loop, here. You may also find the changes in `std::for_each` and `for_each_n`.

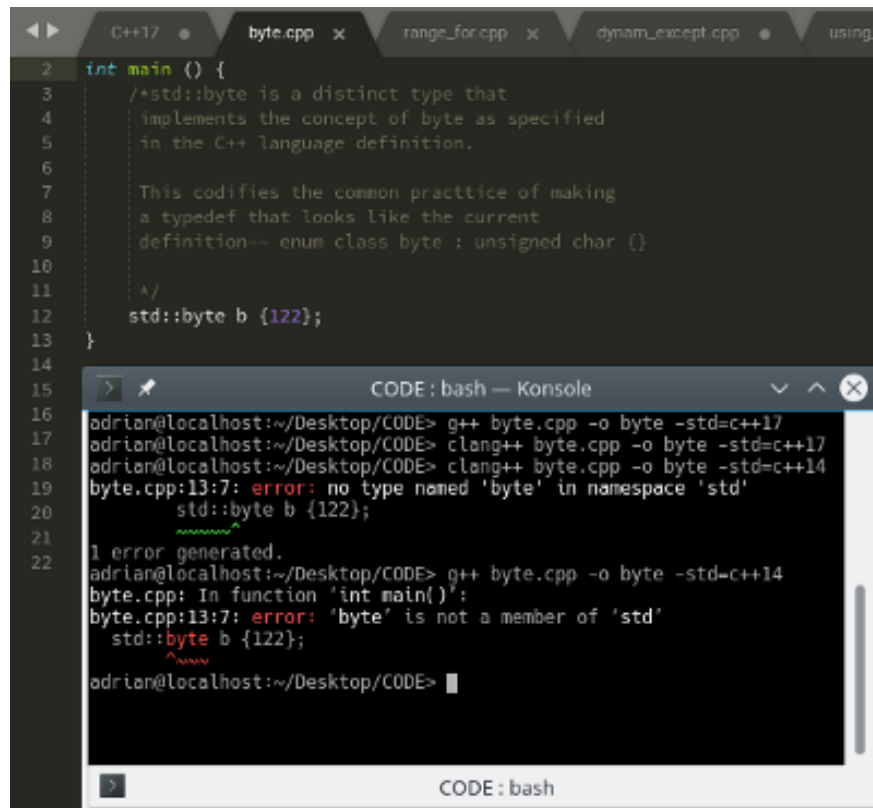
```
C++17 range_for.cpp x dynam_except.cpp using_pack.c
1 #include<map>
2 #include<iostream>
3
4 //Let's Declare a Map
5 std::map <const char*, short> person;
6
7 int main () {
8
9     //Let's give the map values;
10    person["Adrian"] = 991;
11    person["Kerrie"] = 912;
12    person["Nicholas"] = 105;
13    person["Elizabeth"] = 231;
14    person["Wilson"] = 874;
15    person["Joyce"] = 228;
16
17    std::cout << std::endl;
18    for (auto[x,y] : person) {
19        std::cout << "[" << x << "]" << " =\t " << y << '\n';
20    }
21    std::cout << std::endl;
22 }
23
24
25 // ./range_for
26
27 // g++ range_for.cpp -o range_for -std=c++17
28 // clang++ range_for.cpp -o range_for -std=c++17
29
30 // g++ range_for.cpp -o range_for -std=c++14
31 // clang++ range_for.cpp -o range_for -std=c++14
32
```

```
CODE: bash — Konsole
File Edit View Bookmarks Settings Help
adrian@localhost:~/Desktop/CODE> g++ range_for.cpp -o range_for -std=c++14
range_for.cpp: In function 'int main()':
range_for.cpp:18:11: warning: decomposition declaration only available with -std=c++1z or -std-g
nu++1z
    for (auto[x,y] : person) {
        ^
adrian@localhost:~/Desktop/CODE> clang++ range_for.cpp -o range_for -std=c++14
range_for.cpp:18:11: warning: decomposition declarations are a C++17 extension
    [-Wc++17-extensions]
    for (auto[x,y] : person) {
        ^
1 warning generated.
adrian@localhost:~/Desktop/CODE> g++ range_for.cpp -o range_for -std=c++17
adrian@localhost:~/Desktop/CODE> clang++ range_for.cpp -o range_for -std=c++17
adrian@localhost:~/Desktop/CODE> ./range_for

["Adrian"] = 991
["Kerrie"] = 912
["Nicholas"] = 105
["Elizabeth"] = 231
["Wilson"] = 874
["Joyce"] = 228
adrian@localhost:~/Desktop/CODE>
```

## 27) The byte data type

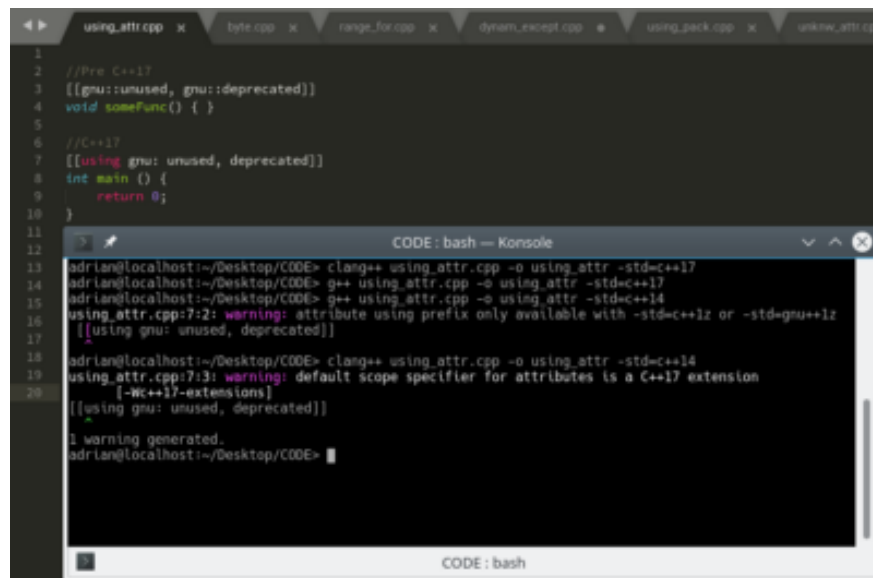
C++17 introduces the byte data type. `std::byte` is included in the `cstdint` header and is not an actual built in data type as `byte` and `short` are, for example. It is analogous to `std::string`, in a way.



```
2 int main () {
3     /*std::byte is a distinct type that
4     implements the concept of byte as specified
5     in the C++ language definition.
6
7     This codifies the common practice of making
8     a typedef that looks like the current
9     definition-- enum class byte : unsigned char {}
10
11     */
12     std::byte b {122};
13 }
14
15
16 CODE: bash — Konsole
17 adrian@localhost:~/Desktop/CODE> g++ byte.cpp -o byte -std=c++17
18 adrian@localhost:~/Desktop/CODE> clang++ byte.cpp -o byte -std=c++17
19 adrian@localhost:~/Desktop/CODE> clang++ byte.cpp -o byte -std=c++14
byte.cpp:13:7: error: no type named 'byte' in namespace 'std'
    std::byte b {122};
    ~~~~~^
1 error generated.
adrian@localhost:~/Desktop/CODE> g++ byte.cpp -o byte -std=c++14
byte.cpp: In function 'int main()':
byte.cpp:13:7: error: 'byte' is not a member of 'std'
    std::byte b {122};
    ~~~~~^
adrian@localhost:~/Desktop/CODE>
```

## 28) Using attribute namespaces without repetition

C++17 cleaned up attribute namespace syntax, to make for cleaner expression. For attributes in the same namespace, one can specify several.



The image shows a code editor with a file named `using_attr.cpp` and a terminal window below it. The code in the editor is as follows:

```
1 //Pre C++17
2 [[gnu:unused, gnu:deprecated]]
3 void someFunc() { }
4
5
6 //C++17
7 [[using gnu: unused, deprecated]]
8 int main () {
9     return 0;
10 }
```

The terminal window shows the following commands and output:

```
adrian@localhost:~/Desktop/CODE> clang++ using_attr.cpp -o using_attr -std=c++17
adrian@localhost:~/Desktop/CODE> g++ using_attr.cpp -o using_attr -std=c++17
adrian@localhost:~/Desktop/CODE> g++ using_attr.cpp -o using_attr -std=c++14
using_attr.cpp:7:2: warning: attribute using prefix only available with -std=c++1z or -std=gnu++1z
[[using gnu: unused, deprecated]]
^
adrian@localhost:~/Desktop/CODE> clang++ using_attr.cpp -o using_attr -std=c++14
using_attr.cpp:7:2: warning: default scope specifier for attributes is a C++17 extension
[[using gnu: unused, deprecated]]
^
1 warning generated.
adrian@localhost:~/Desktop/CODE>
```

## 29) Stricter Order of Evaluation Rules

C++17 puts into law practices that have been expressed in code in the past but never explicitly stated in the standard, in an effort to avoid undefined behavior. Longtime C++ expert Herb Sutter is an author of the paper describing this in more detail, which you should **definitely** read.

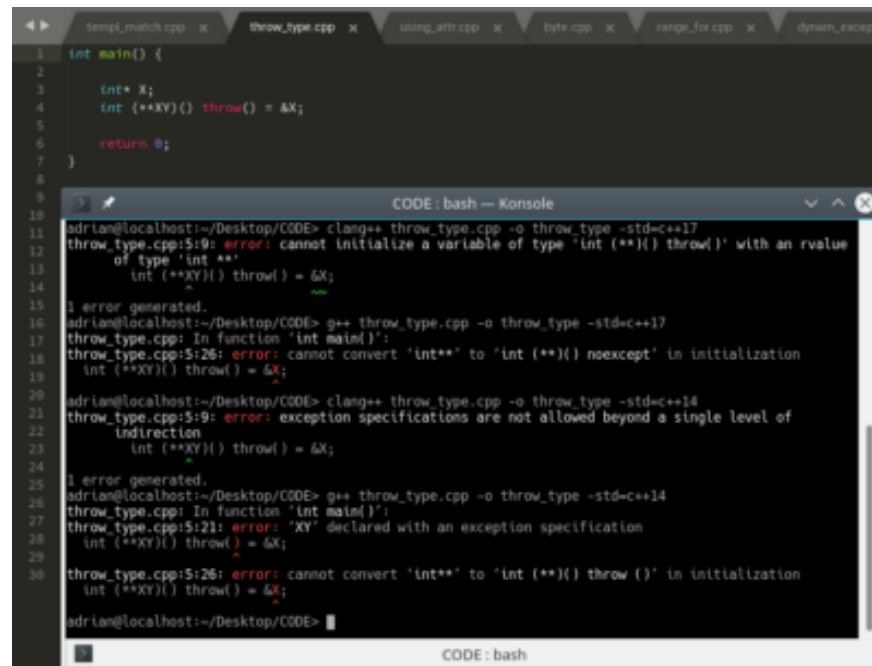
The new changes are [11]:

- “Postfix expressions are evaluated from left to right. This includes functions calls and member selection expressions.”
- “Assignment expressions are evaluated from right to left. This includes compound assignments.”
- “Operands to shift operators are evaluated from left to right.”
- “The order of evaluation of an expression involving an overloaded operator is determined by the order associated with the corresponding built-in operator, not the rules for function calls. This rule is to support generic programming and extensive use of overloaded operators, which are distinctive features of modern C++.”
- “The function is evaluated before all its arguments, but any pair of arguments (from the argument list) is indeterminately sequenced; meaning that one is evaluated before the other but the order is not

specified; it is guaranteed that the function is evaluated before the argument”

### 30) Exception Specifications are part of type definitions

C++17 modified the language of the standard to allow exception specifications to be part of the type system, thus differentiating types by their exception specifiers, something that was not previously a part of the standard.



```
1 int main() {
2
3     int* X;
4     int (**XY)() throw() = &X;
5
6     return 0;
7 }
8
9
10
11
12 adrian@localhost:~/Desktop/CODE> clang++ throw_type.cpp -o throw_type -std=c++17
13 throw_type.cpp:5:9: error: cannot initialize a variable of type 'int (**)() throw()' with an rvalue
14       of type 'int **'
15       int (**XY)() throw() = &X;
16                               ^
17 1 error generated.
18 adrian@localhost:~/Desktop/CODE> g++ throw_type.cpp -o throw_type -std=c++17
19 throw_type.cpp: In function 'int main()':
20 throw_type.cpp:5:26: error: cannot convert 'int**' to 'int (**)() noexcept' in initialization
21       int (**XY)() throw() = &X;
22                          ^
23 1 error generated.
24 adrian@localhost:~/Desktop/CODE> clang++ throw_type.cpp -o throw_type -std=c++14
25 throw_type.cpp:5:9: error: exception specifications are not allowed beyond a single level of
26       indirection
27       int (**XY)() throw() = &X;
28               ^
29 1 error generated.
30 adrian@localhost:~/Desktop/CODE> g++ throw_type.cpp -o throw_type -std=c++14
31 throw_type.cpp: In function 'int main()':
32 throw_type.cpp:5:21: error: 'XY' declared with an exception specification
33       int (**XY)() throw() = &X;
34                   ^
35 throw_type.cpp:5:26: error: cannot convert 'int**' to 'int (**)() throw()' in initialization
36       int (**XY)() throw() = &X;
37                          ^
38 2 errors generated.
39 adrian@localhost:~/Desktop/CODE>
```

Notice that in C++17 the compiler complains because the rvalue and lvalue don't match. This is because they now represent different types. C++14, however, **accepted this**, and instead complained about XY being declared with an exception specification. Notice, lastly, that g++14 gave us the same error stating that this feature change is supposed to change in C++17, while clang++ did not. The likely situation is that this feature had long been implemented before it's introduction into the ISO C++17 standard.

### 31) Template-Template Parameters match compatible arguments

C++17 provides support for matching template template parameters to a compatible argument. It's important to pay attention to the details of this paper. **The consequence of this is that any template**

argument list can be applied to the template-template parameter is also applicable to the argument template.

*“This paper allows a template template-parameter to bind to a template argument whenever the template parameter is at least as specialized as the template argument. This implies that any template argument list that can legitimately be applied to the template template-parameter is also applicable to the argument template.” [11]*

```
templ_match.cpp x throw_type.cpp x using_attr.cpp x byte.cpp x ra
1  template <class T1> class A { };
2  template <class T2, class Y = T2> class B { };
3  template <class... T3> class C { };
4
5  template< template<class> class Z> class D { };
6  // template<class> class Z {} is the parameter.
7  // template <class T1> class A matches it perfectly, for EG.
8
9  // Takes a bit of eye stretching, follow it closely.
10 // Now that the parameter is separated (above), it should
11 // be easier to see how they match.
12
13 int main() {
14
15     D <A> eg; // This is the way you are accustomed to.
16     D <B> eg2; // B is at LEAST as compatible with D's parameters.
17     D <C> eg3; // Same thing with C, that accepts several as a variadic arg
18
19     return 0;
20 }
```

It does not appear that Clang/LLVm supports this feature fully, but I could be wrong. If I am, **do let me know in the comments below**. Notice that before C++17, this was an error, and after it, it is OK.


```
CODE: bash — Konsole
scritan@scritan:~/Desktop/CXX6$ g++ templ_match.cpp -std=c++16
templ_match.cpp:16:6: error: type/value mismatch at argument 1 in template parameter list for 'template<template<class> class Z> class D'
   B <= eg; // B is at LEAST as compatible with D's parameters.
   ^
templ_match.cpp:16:6: note:   expected a template of type 'template<class> class Z', got 'template<class T2, class Y= class B'
templ_match.cpp:17:6: error: type/value mismatch at argument 1 in template parameter list for 'template<template<class> class Z> class D'
   B <= eg3; // Same thing with C, that accepts several as a variadic arg
   ^
templ_match.cpp:17:6: note:   expected a template of type 'template<class> class Z', got 'template<class ... T3> class C'
scritan@scritan:~/Desktop/CXX6$ clang++ templ_match.cpp -std=c++16
templ_match.cpp:16:6: error: template template argument has different template parameters than its corresponding template template parameter
   B <= eg; // B is at LEAST as compatible with D's parameters.
   ^
templ_match.cpp:21:1: note: too many template parameters in template template argument
template <class T2, class Y = T2> class B { };
^
templ_match.cpp:5:11: note: previous template template parameter is here
template <template<class> class Z> class D { };
          ^
templ_match.cpp:17:6: error: template template argument has different template parameters than its corresponding template template parameter
   B <= eg3; // Same thing with C, that accepts several as a variadic arg
   ^
templ_match.cpp:17:6: note: template type parameter pack does not match template type parameter in template argument
template <class... T3> class C { };
          ^
templ_match.cpp:5:20: note: previous template type parameter declared here
template <template<class> class Z> class D { };
          ^
0 errors generated.
scritan@scritan:~/Desktop/CXX6$ g++ templ_match.cpp -std=c++17
scritan@scritan:~/Desktop/CXX6$ clang++ templ_match.cpp -std=c++17
templ_match.cpp:16:6: error: template template argument has different template parameters than its corresponding template template parameter
   B <= eg; // B is at LEAST as compatible with D's parameters.
   ^
templ_match.cpp:21:1: note: too many template parameters in template template argument
template <class T2, class Y = T2> class B { };
^
templ_match.cpp:5:11: note: previous template template parameter is here
template <template<class> class Z> class D { };
          ^
templ_match.cpp:17:6: error: template template argument has different template parameters than its corresponding template template parameter
   B <= eg3; // Same thing with C, that accepts several as a variadic arg
   ^
templ_match.cpp:17:6: note: template type parameter pack does not match template type parameter in template argument
template <class... T3> class C { };
          ^
templ_match.cpp:5:20: note: previous template type parameter declared here
template <template<class> class Z> class D { };
          ^
CODE: bash
```

## 32) Guaranteed Copy Elision

C++17 now guarantees copy elision, standardizing a practice by certain compilers. That is, it “*omits copy constructors, resulting in zero-copy pass-by-value semantics.*”[12] Consequently, if an expression within a call to the dynamic allocator returns a value compatible with the type invoked by the dynamic allocator (`new X (new X())`) then only **one call** rather than several, are made to the constructor of the type in question. The net effect is that less copies of objects are made, which could otherwise result in expensive operations. This tightens (in the standard) the behavior of objects when they are copied by value. This is similar to the behavior under the hood in languages like Java. This, among other things, have changed and you should read more about this, [here](#).

## 33) Changes to Specification on Inheriting Constructors

C++17 introduces many updates to the wording regarding constructors. You should read about them, [here](#), [here](#), & [here](#). This addition fixes eight core issues. Many of these changes involve the `using` keyword. One such example is:



```
1 struct BASE1 {
2     BASE1(double x){}
3 };
4
5 struct BASE2 {
6     BASE2(double x) {}
7 };
8
9 struct DERIVED : BASE1, BASE2 {
10     using BASE1::BASE1;
11     using BASE2::BASE2;
12     /*
13      * Per ordinary overloading rules,
14      * DERIVED(double) overload BASE1(double)
15      * and BASE2(double). In C++17 this call is not
16      * ambiguous. using CLASS::CONSTRUCTOR is different
17      * because instead of declaring a new set of constructors
18      * the base class constructors are now visible, to make
19      * overloading possible.
20      */
21     DERIVED (double);
22 };
23
```

```

24 int main () {
25     //DERIVED::DOUBLE() CALLED
26     DERIVED d(1.2);
27     /*Because DERIVED::DOUBLE() is not defined, linking fails.*/
28     /*
29     GCC:
30     g++ using_constr.cpp -o using_constr -std=c++17
31     /tmp/ccW6csd.o: In function 'main':
32     using_constr.cpp:(.text+0x20): undefined reference to 'DERIVED::DERIVED(double)'
33     collect2: error: ld returned 1 exit status
34
35     CLANG:
36     clang++ using_constr.cpp -o using_constr -std=c++17
37     /tmp/using_constr-d68afa.o: In function 'main':
38     using_constr.cpp:(.text+0x1c): undefined reference to 'DERIVED::DERIVED(double)'
39     clang-5.0: error: linker command failed with exit code 1 (use -v to see invocation)
40     */
41     return 0;
42 }
43

```

Notice that linking failed. Read more about this in the comments. This example was adapted from this [paper](#).

## Deprecated Language Features

### 1) Removal of Trigraphs

C++17 brought in the removal of Trigraphs from the standard, though some compilers continue to provide support. MSVC, GCC, & Clang are among those that provide support with a command line switch. The switch to enable them in C++17 via GCC/Clang is **-trigraphs**.

Trigraphs are a three character sequence beginning with ?? and ending in another element, like !. They allowed for C programs to be written in ISO 646:1983. There are 9 of them. The compiler recognizes a trigraph and replaces it with the value of a matching trigraph sequence. All of the trigraphs are punctuation symbols. C++ had inherited this feature from C.

```

C++17 x  trigraph.cpp x  auto_deduc.cpp x  hex.cpp x
1  #include<iostream>
2  int main () {
3      std::cout << "\nThere are nine trigraphs and this is one: "
4      << "\"? - ? - <\" : \"??<\".\"<< '\n' << std::endl;
5  }
6
7  // clang++ trigraph.cpp -std=c++17 -o trigraph -trigraphs
8  // g++ trigraph.cpp -std=c++17 -o trigraph -trigraphs

```

Trigraphs work just fine under GCC/Clang C++11, albeit with a warning from Clang.



```
CODE: bash — Konsole
adrian@localhost:~/Desktop/CODE> g++ trigraph.cpp -std=c++11 -o trigraph
adrian@localhost:~/Desktop/CODE> ./trigraph

There are nine trigraphs and this is one: "?-?-<" : "{".

adrian@localhost:~/Desktop/CODE> clang++ trigraph.cpp -std=c++11 -o trigraph
trigraph.cpp:3:73: warning: trigraph converted to '{' character [-Wtrigraphs]
    std::cout <<"\nThere are nine trigraphs and this is one: \"?-?-<\" : \"??<\",<\" '\n' <<
    ...
1 warning generated.
adrian@localhost:~/Desktop/CODE> ./trigraph

There are nine trigraphs and this is one: "?-?-<" : "{".

adrian@localhost:~/Desktop/CODE> █
```

Unless we flip the switch, GCC/Clang C++17 will not recognize the trigraph.

```
CODE: bash — Konsole
trigraph.cpp:4:19: warning: trigraph ignored [-Wtrigraphs]
    <<"\"?-?-<\" : \"??<\",<\" '\n' << std::endl;
    ~~~~~^
1 warning generated.
adrian@localhost:~/Desktop/CODE> g++ trigraph.cpp -std=c++17 -o trigraph
trigraph.cpp:4:19: warning: trigraph ??< ignored, use -trigraphs to enable [-Wtrigraphs]
    <<"\"?-?-<\" : \"??<\",<\" '\n' << std::endl;
    ~~~~~^
adrian@localhost:~/Desktop/CODE> ./trigraph

There are nine trigraphs and this is one: "?-?-<" : "??<".

adrian@localhost:~/Desktop/CODE> clang++ trigraph.cpp -std=c++17 -o trigraph -trigraphs
trigraph.cpp:4:19: warning: trigraph converted to '{' character [-Wtrigraphs]
    <<"\"?-?-<\" : \"??<\",<\" '\n' << std::endl;
    ~~~~~^
1 warning generated.
adrian@localhost:~/Desktop/CODE> ./trigraph

There are nine trigraphs and this is one: "?-?-<" : "{".

adrian@localhost:~/Desktop/CODE> g++ trigraph.cpp -std=c++17 -o trigraph -trigraphs
adrian@localhost:~/Desktop/CODE> ./trigraph

There are nine trigraphs and this is one: "?-?-<" : "{".

adrian@localhost:~/Desktop/CODE> █
```

## 2) Removal of deprecated Increment Operator (++) for bool type

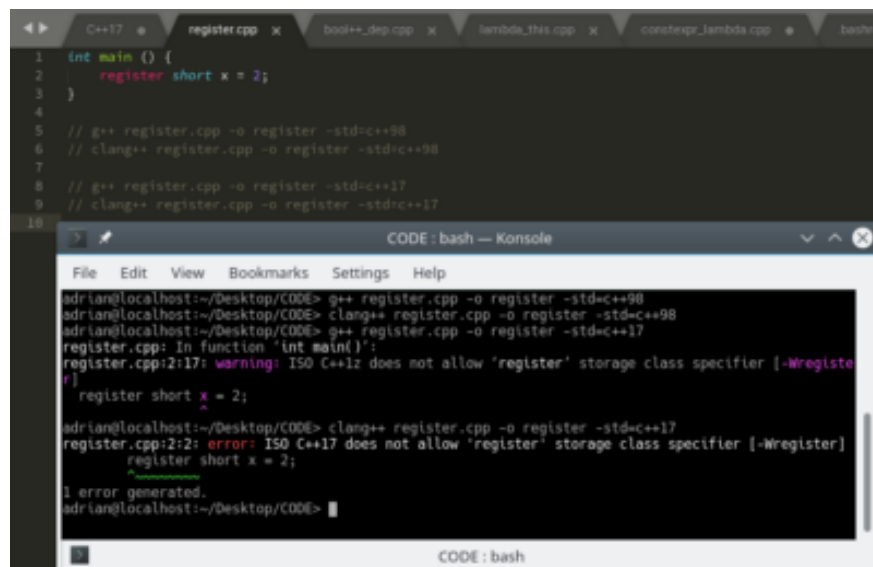
C++17 removed support for the increment operator for C++'s Boolean type. It had been deprecated since C++98 and has finally been removed. Notice that it will succeed with C++98 (albeit with a deprecation warning) but it will fail with C++17.

```
C++17 x bool++_dep.cpp x lambda_this.cpp x
1  #include <iostream>
2
3  int main () {
4      bool x = false;
5      std::cout<< "Boolean: " << x << std::endl;
6
7      x++;
8      std::cout<< "Boolean: " << x << std::endl;
9
10 }
11
12 // g++ bool++_dep.cpp -o bool++_dep -std=c++98
13 // clang++ bool++_dep.cpp -o bool++_dep -std=c++98
14
15 // g++ bool++_dep.cpp -o bool++_dep -std=c++17
16 // clang++ bool++_dep.cpp -o bool++_dep -std=c++17
17
```

```
CODE: bash — Konsole
File Edit View Bookmarks Settings Help
adrian@localhost:~/Desktop/CODE> g++ bool++_dep.cpp -o bool++_dep -std=c++98
bool++_dep.cpp: In function 'int main()':
bool++_dep.cpp:7:3: warning: use of an operand of type 'bool' in 'operator++' is deprecated [-Wdeprecated]
    x++;
    ^~
adrian@localhost:~/Desktop/CODE> ./bool++_dep
Boolean: 0
Boolean: 1
adrian@localhost:~/Desktop/CODE> clang++ bool++_dep.cpp -o bool++_dep -std=c++98
bool++_dep.cpp:7:3: warning: incrementing expression of type bool is deprecated and incompatible with C++17 [-Wdeprecated-increment-bool]
    x++;
    ^~
1 warning generated.
adrian@localhost:~/Desktop/CODE> ./bool++_dep
Boolean: 0
Boolean: 1
adrian@localhost:~/Desktop/CODE> g++ bool++_dep.cpp -o bool++_dep -std=c++17
bool++_dep.cpp: In function 'int main()':
bool++_dep.cpp:7:3: error: use of an operand of type 'bool' in 'operator++' is forbidden in C++1z
    x++;
    ^~
adrian@localhost:~/Desktop/CODE> clang++ bool++_dep.cpp -o bool++_dep -std=c++17
bool++_dep.cpp:7:3: error: ISO C++17 does not allow incrementing expression of type bool [-Wincrement-bool]
    x++;
    ^~
1 error generated.
adrian@localhost:~/Desktop/CODE>
```

### 3) Removal of deprecated register keyword

C++17 removed support for the increment operator for C++'s register keyword. It had been deprecated since C++11 and has finally been removed. Notice that it will succeed with C++98 but will fail with C++17.



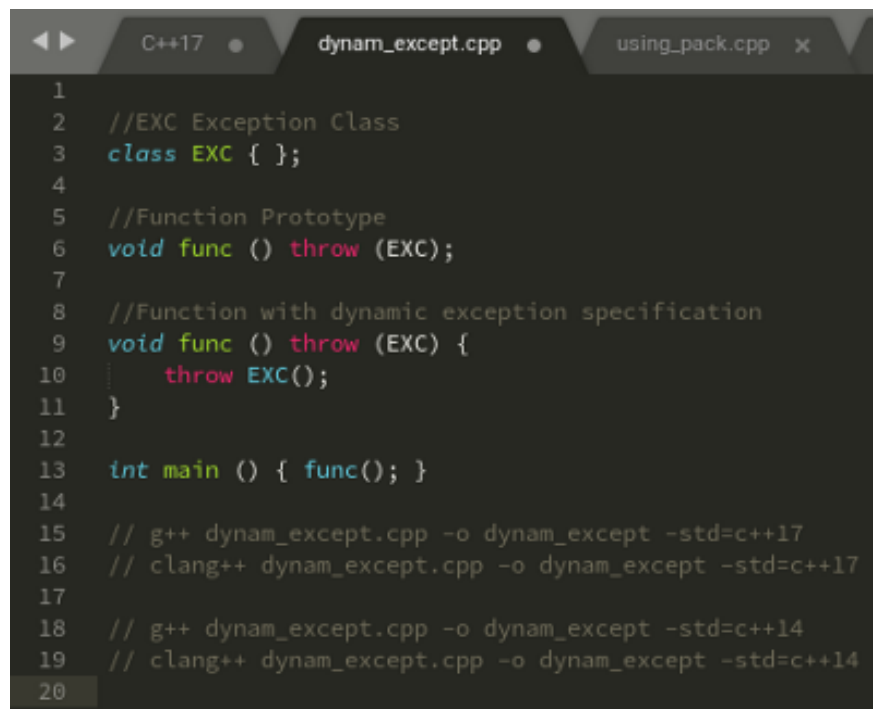
```
1 int main () {
2     register short x = 2;
3 }
4
5 // g++ register.cpp -o register -std=c++98
6 // clang++ register.cpp -o register -std=c++98
7
8 // g++ register.cpp -o register -std=c++17
9 // clang++ register.cpp -o register -std=c++17
10
```

CODE: bash — Konsole

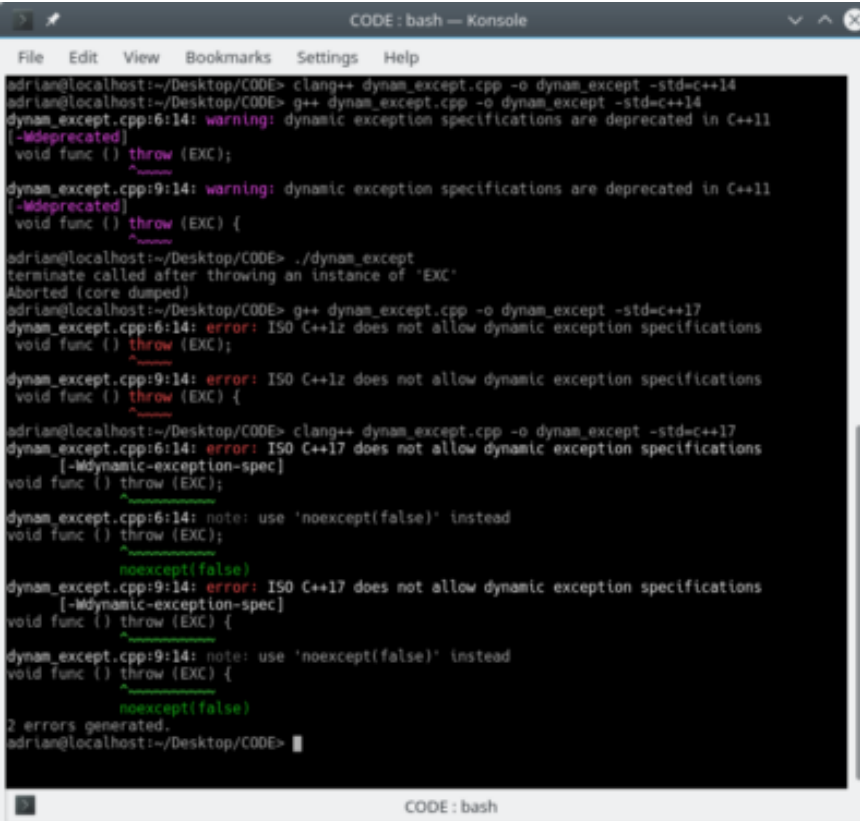
```
adrian@localhost:~/Desktop/CODE> g++ register.cpp -o register -std=c++98
adrian@localhost:~/Desktop/CODE> clang++ register.cpp -o register -std=c++98
adrian@localhost:~/Desktop/CODE> g++ register.cpp -o register -std=c++17
register.cpp: In function 'int main()':
register.cpp:2:17: warning: ISO C++1z does not allow 'register' storage class specifier [-Wregister]
    register short x = 2;
                  ^
adrian@localhost:~/Desktop/CODE> clang++ register.cpp -o register -std=c++17
register.cpp:2:2: error: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
    register short x = 2;
    ^
1 error generated.
adrian@localhost:~/Desktop/CODE>
```

#### 4) Removal of deprecated Dynamic Exception Specifications

C++17 removed support for the dynamic exception specification. It had been deprecated since C++11 and has finally been removed. Notice that it will succeed with C++03 but will fail with C++17. It is recommended to use `noexcept(true)` in lieu of `throw()` and `noexcept(false)` in lieu of `throw(...)`.



```
1
2 //EXC Exception Class
3 class EXC { };
4
5 //Function Prototype
6 void func () throw (EXC);
7
8 //Function with dynamic exception specification
9 void func () throw (EXC) {
10     throw EXC();
11 }
12
13 int main () { func(); }
14
15 // g++ dynam_except.cpp -o dynam_except -std=c++17
16 // clang++ dynam_except.cpp -o dynam_except -std=c++17
17
18 // g++ dynam_except.cpp -o dynam_except -std=c++14
19 // clang++ dynam_except.cpp -o dynam_except -std=c++14
20
```



```
CODE: bash — Konsole
File Edit View Bookmarks Settings Help
adrian@localhost:~/Desktop/CODE> clang++ dynam_except.cpp -o dynam_except -std=c++14
adrian@localhost:~/Desktop/CODE> g++ dynam_except.cpp -o dynam_except -std=c++14
dynam_except.cpp:6:14: warning: dynamic exception specifications are deprecated in C++11 [-Wdeprecated]
void func () throw (EXC);
               ^~~~~~
dynam_except.cpp:9:14: warning: dynamic exception specifications are deprecated in C++11 [-Wdeprecated]
void func () throw (EXC) {
               ^~~~~~
adrian@localhost:~/Desktop/CODE> ./dynam_except
terminate called after throwing an instance of 'EXC'
Aborted (core dumped)
adrian@localhost:~/Desktop/CODE> g++ dynam_except.cpp -o dynam_except -std=c++17
dynam_except.cpp:6:14: error: ISO C++1z does not allow dynamic exception specifications
void func () throw (EXC);
               ^~~~~~
dynam_except.cpp:9:14: error: ISO C++1z does not allow dynamic exception specifications
void func () throw (EXC) {
               ^~~~~~
adrian@localhost:~/Desktop/CODE> clang++ dynam_except.cpp -o dynam_except -std=c++17
dynam_except.cpp:6:14: error: ISO C++17 does not allow dynamic exception specifications [-Wdynamic-exception-spec]
void func () throw (EXC);
               ^~~~~~
dynam_except.cpp:6:14: note: use 'noexcept(false)' instead
void func () throw (EXC);
               ^~~~~~
dynam_except.cpp:9:14: error: ISO C++17 does not allow dynamic exception specifications [-Wdynamic-exception-spec]
void func () throw (EXC) {
               ^~~~~~
dynam_except.cpp:9:14: note: use 'noexcept(false)' instead
void func () throw (EXC) {
               ^~~~~~
noexcept(false)
2 errors generated.
adrian@localhost:~/Desktop/CODE>
```

Note that clang being the less strict compiler did not warn us in C++14 while GCC did. In C++17 it is illegal by both, with Clang giving us more useful feedback.

**Like the new features? Hate them? Did I forget some? Let me know in the comments below!**



Looney Tunes Ending [4]

## Works Cited

[1]—Open-STD: Working Draft, Standard for Programming Language C++

[2]—ISO CPP: Changes between C++14 and C++17 DIS

[3]—CppReference.com: C++ Compiler Support, C++17 Features

[4]—Stack Overflow: "What are the new features in C++17?"

[5]—ISO CPP: Changes between C++14 and C++17 DIS

[6]—GitHub: AnthonyCalandra/modern-cpp-features

[7]—Wikipedia: C++17

[8]—CppReference.com: inline specifier

[9]—ISOCPP: Current Status: Standard C++

[10]—OpenSTD: Refining Expression Evaluation Order for Idiomatic C++

[11]—OpenSTD: DR: Matching of template template-arguments excludes compatible templates

[12]—CppReference.com: copy elision



