



# Clang Concepts

---

and what it **requires** to get C++20

Saar Raz • 2019

# About Concepts



- 
- Make C++ Typed Again
  - Removes duck-typing from C++
  - “A type system for types”
  - The Python 3 of C++
  - 20+ years in the making



# The **requires** Clause

---

- Before

```
template<typename It>
void sort(It begin, It end) {
    // ...
}
```

- After

```
template<typename It> requires Iterator<It>
void sort(It begin, It end) {
    // ...
}
```



# Abbreviated Templates

- Before

```
template<typename T, typename U>
void foo(T t, U u) {
    // ...
}
```

- After

```
void foo(auto t, auto u) {
    // ...
}
```



# Static Requirements

---

```
template<typename T>
concept Large = sizeof(T) > 10;
```

```
template<typename T, typename U>
concept FooableWith = requires (T t, U u) {
    typename T::foo_type;
    { t.foo(u) } -> typename T::foo_type;
    t++;
};

void doFoo(FooableWith<int> auto t) {
    t.foo(3);
}
```

# Nicer Errors



## ■ Before

```
std::unordered_map<A, int> m;
```

```
22 | std::unordered_map<A, int> m;
#1 with x86-64 clang (experimental concepts) X
A ▾ □ Wrap lines

In file included from <source>:2:
In file included from /opt/compiler-explorer/clang-concepts-trunk-20190218/bin/../include/c++/v1/unordered_map:407:
/opt/compiler-explorer/clang-concepts-trunk-20190218/bin/../include/c++/v1/__hash_table:867:5: error: static_assert failed due to requirement '__check_hash_requirements<A, hash<A>>::value' "the specified hash does not meet the Hash requirements"
static_assert(__check_hash_requirements<_Key, _Hash>::value,
^ ~~~~~
/opt/compiler-explorer/clang-concepts-trunk-20190218/bin/../include/c++/v1/__hash_table:882:1: note: in instantiation of template class 'std::__1::__enforce_unordered_container_requirements<A, std::__1::hash<A>, std::__1::equal_to<A>, std::__1::allocator<std::__1::pair<const A, int> >>' requested here
typename __enforce_unordered_container_requirements<_Key, _Hash, _Equal>::type
^

/opt/compiler-explorer/clang-concepts-trunk-20190218/bin/../include/c++/v1/unordered_map:855:26: note: while substituting explicitly-specified template arguments into function template '__diagnose_unordered_container_requirements'
static_assert(sizeof(__diagnose_unordered_container_requirements<_Key, _Hash, _Pred>(0)), "");
^

<source>:22:32: note: in instantiation of template class 'std::__1::unordered_map<A, int, std::__1::hash<A>, std::__1::equal_to<A>, std::__1::allocator<std::__1::pair<const A, int> >>' requested here
std::unordered_map<A, int> m;
^

In file included from <source>:2:
/opt/compiler-explorer/clang-concepts-trunk-20190218/bin/../include/c++/v1/unordered_map:431:11: error: call to implicitly-deleted default constructor of 'std::__1::hash<A>'
: _Hash() {}
^

/opt/compiler-explorer/clang-concepts-trunk-20190218/bin/../include/c++/v1/memory:2178:39: note: in instantiation of member function 'std::__1::__unordered_map_hasher<A, std::__1::__hash_value_type<A, int>, std::__1::hash<A>, true, _LIBCPP_INLINE_VISIBILITY constexpr __compressed_pair_elem() = default;
^

/opt/compiler-explorer/clang-concepts-trunk-20190218/bin/../include/c++/v1/unordered_map:903:5: note: in instantiation of member function 'std::__1::__hash_table<std::__1::__hash_value_type<A, int>, std::__1::__unordered_map_hasher<A, std::__1::__hash_value_type<A, int>, std::__1::hash<A>, true, _LIBCPP_INLINE_VISIBILITY constexpr __compressed_pair_elem() = default, std::__1::allocator<std::__1::pair<const A, int> >>::unordered_map'
^

<source>:22:32: note: in instantiation of member function 'std::__1::unordered_map<A, int, std::__1::hash<A>, std::__1::equal_to<A>, std::__1::allocator<std::__1::pair<const A, int> >>::unordered_map' requested here
std::unordered_map<A, int> m;
^

/opt/compiler-explorer/clang-concepts-trunk-20190218/bin/../include/c++/v1/utility:1575:36: note: default constructor of 'hash<A>' is implicitly deleted because base class '__enum_hash<A>' has a deleted default constructor
struct _LIBCPP_TEMPLATE_VIS hash : public __enum_hash<_Tp>
^

/opt/compiler-explorer/clang-concepts-trunk-20190218/bin/../include/c++/v1/utility:1569:5: note: '__enum_hash' has been explicitly marked deleted here
__enum_hash() = delete;
^

2 errors generated.
```

# Nicer Errors



## ■ After

```
22     std::unordered_map<A, int> m;

#1 with x86-64 clang (experimental concepts) ×
A ▾  Wrap lines

<source>:22:10: error: constraints not satisfied for class template 'unordered_map' [with K = A, V = int]
  std::unordered_map<A, int> m;
          ^~~~~~
<source>:10:14: note: because 'A' is not 'Hashable'
  template<Hashable K, typename V>
          ^
<source>:8:43: note: because 'std::hash<T>({})(t)' would be invalid: type 'std::hash<A>' does not provide a call operator
  concept Hashable = requires (T t) { { std::hash<T>{}(t) } -> std::size_t; };
          ^
1 error generated.
Compiler returned: 1
```



# Overloading

---

```
template<Iterator It>
void sort(It begin, It end) {
    // ...
}
```

```
template<RandomAccessIterator It>
void sort(It begin, It end) {
    // ...
}
```



# About me

---

- 24 years old, from Kiryat Atta
- Fell in love with C++ ever since I relearned it in 2015
- Have been working on the Clang implementation of Concepts for the past year or so
- This is **the story of how I got around to doing this**



# A slippery slope

---

- Started writing a game engine
  - Involved a lot of generics
  - Things were getting out of hand
- Concepts had an implementation in GCC 6!
  - Which wasn't even out back then
    - Probably still buggy...
    - Not sure if maintained
    - Nah, it'll be fine



# A slippery slope

- Built GCC 6
- Wrote much code with concepts #future
  - Before:

```
template<typename Message_, typename Source_, typename PasserLocation_, typename ReceiverLocation_,
typename Context_, typename Propagate_>
auto passMessage(Message_ message, Source_ sourceFromPasser, PasserLocation_ passerLocation,
                 ReceiverLocation_ receiverLocation, Context_ receiverContext,
                 Propagate_ propagate) {
    // ...
}
```

- Which is basically like:

```
auto passMessage(auto message, auto sourceFromPasser, auto passerLocation, auto receiverLocation,
                 auto receiverContext, auto propagate) {
    // ...
}
```

# C++ with strong typing



```
auto passMessage(auto message, auto sourceFromPasser, auto passerLocation,
                 auto receiverLocation, auto receiverContext, auto propagate)
{
    // ...
}
```

- Becomes:

```
Message auto passMessage(Message auto message, MessageSource auto sourceFromPasser,
                         Location auto passerLocation, Location auto receiverLocation,
                         Context auto receiverContext, Callable auto propagate) {
    // ...
}
```

- 😊
- There's no turning back now!

# And they lived happ-



- Well it turns out GCC concepts did have bugs
- No problem! I can report them!

Saar Raz 2017-02-28 16:30:34 UTC

[Description](#)

```
The following program returns 1 with the latest gcc 7 snapshot:  
  
template<typename X>  
concept bool FalseConcept = false;  
  
template<FalseConcept Y>  
concept bool AnotherConcept = true;  
  
int main() {  
    return AnotherConcept<int>;  
}
```

- (And that was the last time I heard of this)



# Compile times

---

- As I said before, this involved a bunch of templates.
- Compile times started to get out of hand.
- Error messages started to get out of hand...



# An unindicative error message

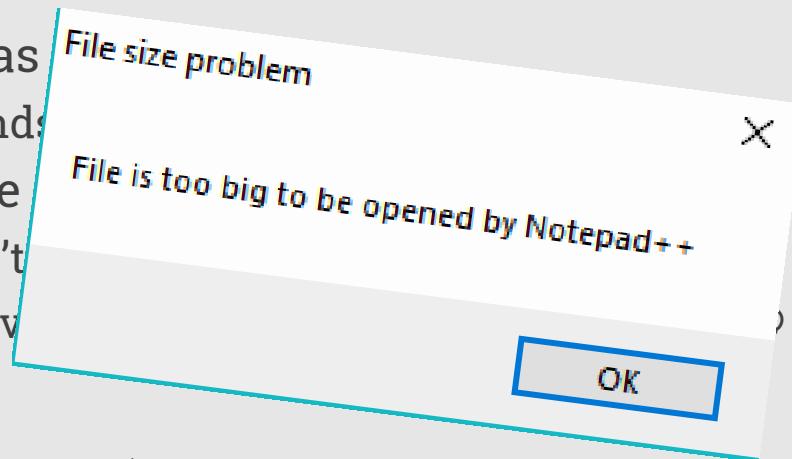
---

- So at one point I tried to compile the project
- It froze
  - The whole PC
    - The kernel
      - The mouse won't even move
- Maybe a very long error message?
  - -fmax-errors=1
    - Still doesn't work
  - Output the message to a file?
    - Still doesn't work
- A problem with cygwin?



# Oh well it's probably a Windows problem

- Move to a Linux VM
- Freezes
  - The host as
    - (In hind)
  - Output the
    - Doesn't
    - We hav
- **1.2GB**
  - (-fmax-errors=1)
  - What does it say?



# How do you read 1.2GB?



- Opens in some of the text editors
- Only 10 lines of error message!
  - Each line ~100MB
    - In instantiation of foo::bar<T, U>::bar() [with T = foo<A, B> [with A = ..., B = ...], U = ...]
    - In instantiation of foo::bar<T, U>::baz() [with T = foo<A, B> [with A = ..., B = ...], U = ...]
    - ...
- Template backtrace
  - We can limit the backtrace depth, but I needed all of it to understand what the problem was...
  - People complain C++ gives unindicative errors, I couldn't even read mine...
  - **Let us parse!**



# How do you parse 1.2GB?

---

- So I started writing a Python script -
  - In instantiation of foo::bar<T, U>::bar() [with T = <1>, U = <2>]
  - Click 1 to expand <1>, 2 to expand <2>
- Doesn't work
  - Python is too slow...
  - C++ to the rescue!
    - Works!
      - (I had to really optimize the C++ script)
    - **Got the bug!!!**
    - A few days later, the PC freezes again
    - 2.0GB
      - Script can't handle this anymore



# What now?

---

- The long-named templates are actually compile-time trees:
  - `tree<a,tree<c,tree<a>, tree<a>>, tree<a>, tree<c, tree<b>>>`

- How can we shorten their names?

```
struct my_tree : tree<a,tree<c,tree<a>, tree<a>>, tree<a>, tree<c, tree<b>>>
{
    // inherit constructors
    using tree<a,tree<c,tree<a>, tree<a>>, tree<a>, tree<c, tree<b>>>::tree;
};
```

- `my_tree` behaves just like `tree<a,tree<c,tree<a>, tree<a>>, tree<a>, tree<c, tree<b>>>`, except the fact that it's name is shorter in error messages!
  - Works! Only 400MB of error!
  - Piece of cake for the script

# A long-term solution



- Inheriting from every long template like this is a hassle
- And sometimes I don't even need all that information
- If we take a look at the error message:
  - In instantiation of foo::bar<T, U>::bar() [with T = <1>, U = <2>]

The screenshot shows three windows of a debugger displaying assembly code:

- Top Left Window:** Shows assembly code for the instantiation of a template. It includes instructions like mov, call, and test, along with comments such as "with" and "template\_parms". The assembly is color-coded with syntax highlighting.
- Top Right Window:** Shows the continuation of the assembly code, likely the next part of the function or a related function. It also includes comments and color-coded assembly.
- Bottom Window:** Shows the final part of the assembly code, which appears to be the cleanup or return section of the function. It includes instructions like mov, add, and ret, along with comments and color-coded assembly.



# Another unindictive error message

```
n.cpp:92:108:     required from here  
82:16: internal compiler error: Segmentation fault
```

# And accusations of murder



```
g++-7.1: internal compiler error: Killed (program cc1plus)
Please submit a full bug report,
with preprocessed source if appropriate.
See <https://gcc.gnu.org/bugs/> for instructions.
CMakeFiles/glop.dir/build.make:62: recipe for target
make[3]: *** [CMakeFiles/glop.dir/build.make:62: . . .]
```

# If I'm already patching GCC...

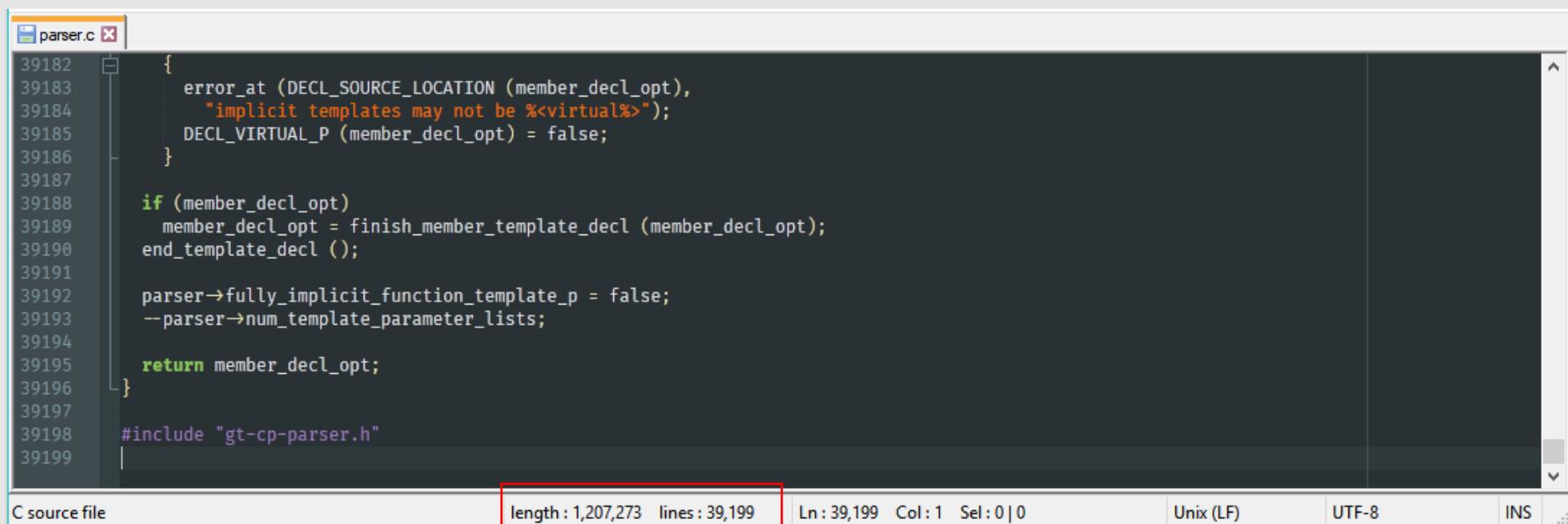


- 
- I needed to debug a lot of compile-time stuff
  - There is no print-debugging at compile time 😞
  - Let's add some!
  - Opened up GCC sources

# Good thing GCC's code is so nice



- Meet parser.c, which parses all of C++:



A screenshot of a code editor window titled "parser.c". The code is written in C and handles template declarations. It includes error handling for implicit templates and manages template parameter lists. The code editor shows line numbers from 39182 to 39198. A red box highlights the status bar at the bottom, which displays "length : 1,207,273 lines : 39,199". Other status bar details include "Ln : 39,199 Col : 1 Sel : 0 | 0", "Unix (LF)", "UTF-8", and "INS".

```
39182 {  
39183     error_at (DECL_SOURCE_LOCATION (member_decl_opt),  
39184         "implicit templates may not be %<virtual%>");  
39185     DECL_VIRTUAL_P (member_decl_opt) = false;  
39186 }  
39187  
39188 if (member_decl_opt)  
39189     member_decl_opt = finish_member_template_decl (member_decl_opt);  
39190 end_template_decl ();  
39191  
39192 parser->fully_implicit_function_template_p = false;  
39193 --parser->num_template_parameter_lists;  
39194  
39195 return member_decl_opt;  
39196 }  
39197  
39198 #include "gt-cp-parser.h"  
39199
```

- Yes, there are bigger files (52k lines)



# static\_print

---

- I wanted to add a new keyword to C++:

```
int main() {
    test<int, 3> y;
    static_print("y's type is ", decltype(y));
    return 0;
}
```

- While compiling the above code, the compiler will print:
  - y's type is test<int, 3>

# How do you add a keyword?



- Well I did take a compilers class back in university...
- There's probably a nice little file that defines the grammar declaratively
  - I only need to add my new keyword and I'm done, right??
- The real world isn't as pretty
- It's functions all the way down
- What now?
  - Copy & Paste!



# Copy & Paste

---

- `static_print` behaves awfully similar to `static_assert`
  - Can appear in the same places
  - It also parses string literals
  - It also starts with `static_` and is also colored pink in the slides
- The plan:
  - Search the whole source for the string “`static_assert`”
    - Find where keyword is parsed
  - Wherever it is, duplicate it and change to “`static_print`”
  - If that string is assigned to any variables/constants – do the same thing recursively



# Copy & Paste

---

- Found this:

```
const struct c_common_resword c_common_reswords[] =
{
{ "_Alignas",           RID_ALIGNAS,   D_CONLY },
{ "_Alignof",            RID_ALIGNOF,   D_CONLY },

// a bunch more like these...

{ "static_assert",      RID_STATIC_ASSERT, D_CXXONLY | D_CXX11 | D_CXXWARN },
// ...
```

- Jackpot! Add this:

```
{ "static_print",      RID_STATIC_PRINT, D_CXXONLY | D_CXX11 | D_CXXWARN },
```

- But now we have RID\_STATIC\_PRINT



# More Copy & Paste

---

- Then

```
enum rid{
    RID_STATIC = 0,
    // ...
    RID_NULLPTR, RID_STATIC_ASSERT,
    RID_STATIC_PRINT,
    // ...
};
```

- Search for usage of RID\_STATIC\_ASSERT

```
/* If the next token is `static_assert' we have a static assertion. */
else if (token1->keyword == RID_STATIC_ASSERT)
    cp_parser_static_assert (parser, /*member_p=*/
    false);
    ^
/* If the next token is `static_print' we have a static print statement. */
else if (token1->keyword == RID_STATIC_PRINT)
    cp_parser_static_print (parser, /*member_p=*/
    false);
```

# The business logic



```
static void
cp_parser_static_assert(cp_parser *parser, bool member_p)
{
    tree condition;
    tree message;
    cp_token *token;
    location_t saved_loc;
    bool dummy;

    /* Peek at the 'static_assert' token so we can keep track of exactly
     * where the static assertion started. */
    token = cp_lexer_peek_token(parser->lexer);
    saved_loc = token->location;

    /* Look for the 'static_assert' keyword. */
    if (!cp_parser_require_keyword(parser, RID_STATIC_ASSERT,
                                  RT_STATIC_ASSERT))
        return;

    /* We know we are in a static assertion; commit to any tentative
     * parse. */
    if (cp_parser_parsing_tentatively(parser))
        cp_parser_commit_to_tentative_parse(parser);

    /* Parse the `(` starting the static assertion condition. */
    cp_parser_require(parser, CPP_OPEN_PAREN, RT_OPEN_PAREN);

    /* Parse the constant-expression. Allow a non-constant expression
     * here in order to give better diagnostics in finish_static_assert. */
    condition =
        cp_parser_constant_expression(parser,
                                       /*allow_non_constant_p=*/true,
                                       /*non_constant_p=*/&dummy);

    if (cp_lexer_peek_token(parser->lexer)->type == CPP_CLOSE_PAREN)
    {
        if (cxx_dialect < cxx1z)
            pedwarn (input_location, OPT_Wpedantic,
                     "static_assert without a message "
                     "only available with -std=c++1z or -std=gnu++1z");
        /* Eat the `)` */
        cp_lexer_consume_token(parser->lexer);
        message = build_string(1, "");
        TREE_TYPE (message) = char_array_type_node;
        fix_string_type(message);
    }
    else
    {
        /* Parse the separating `,` */
        cp_parser_require(parser, CPP_COMMA, RT_COMMA);

        /* Parse the string-literal message. */
        message = cp_parser_string_literal(parser,
                                           /*translate=*/false,
                                           /*wide_ok=*/true);

        /* A `)` completes the static assertion. */
        if (!cp_parser_require(parser, CPP_CLOSE_PAREN, RT_CLOSE_PAREN))
            cp_parser_skip_to_closing_parenthesis(parser,
                                                 /*recovering=*/true,
                                                 /*or_comma=*/false,
                                                 /*consume_paren=*/true);
    }

    /* A semicolon terminates the declaration. */
    cp_parser_require(parser, CPP_SEMICOLON, RT_SEMICOLON);

    /* Complete the static assertion, which may mean either processing
     * the static assert now or saving it for template instantiation. */
    finish_static_assert(condition, message, saved_loc, member_p);
}
```

# The business logic



```
static void
cp_parser_static_assert(cp_parser *parser, bool member_p)
{
    tree condition;
    tree message;
    cp_token *token;
    location_t saved_loc;
    bool dummy;

    /* Peek at the `static_assert` token so we can keep track of exactly
     * where the static assertion started. */
    token = cp_lexer_peek_token(parser->lexer);
    saved_loc = token->location;

    /* Look for the `static_assert` keyword. */
    if (!cp_parser_require_keyword(parser, RID_STATIC_ASSERT,
                                   RT_STATIC_ASSERT))
        return;

    /* We know we are in a static assertion; commit to any tentative
     * parse. */
    if (cp_parser_parsing_tentatively(parser))
        cp_parser_commit_to_tentative_parse(parser);

    /* Parse the `(` starting the static assertion condition. */
    cp_parser_require(parser, CPP_OPEN_PAREN, RT_OPEN_PAREN);

    /* Parse the constant-expression. Allow a non-constant expression
     * here in order to give better diagnostics in finish_static_assert. */
    condition =
        cp_parser_constant_expression(parser,
                                      /*allow_non_constant_p=*/true,
                                      /*non_constant_p=*/&dummy);

    if (cp_lexer_peek_token(parser->lexer)->type == CPP_CLOSE_PAREN)
    {
        if (cxx_dialect < cxx1z)
            pedwarn (input_location, OPT_Wpedantic,
                     "static_assert without a message "
                     "only available with -std=c++1z or -std=gnu++1z");
        /* Eat the `)` */
        cp_lexer_consume_token(parser->lexer);
        message = build_string(1, "");
        TREE_TYPE (message) = char_array_type_node;
        fix_string_type (message);
    }
    else
    {
        /* Parse the separating `,`. */
        cp_parser_require(parser, CPP_COMMA, RT_COMMA);

        /* Parse the string-literal message. */
        message = cp_parser_string_literal(parser,
                                           /*translate=*/false,
                                           /*wide_ok=*/true);

        /* A `)` completes the static assertion. */
        if (!cp_parser_require(parser, CPP_CLOSE_PAREN, RT_CLOSE_PAREN))
            cp_parser_skip_to_closing_parenthesis (parser,
                                                   /*recovering=*/true,
                                                   /*or_comma=*/false,
                                                   /*consume_paren=*/true);
    }

    /* A semicolon terminates the declaration. */
    cp_parser_require(parser, CPP_SEMICOLON, RT_SEMICOLON);

    /* Complete the static assertion, which may mean either processing
     * the static assert now or saving it for template instantiation. */
    finish_static_assert (condition, message, saved_loc, member_p);
}
```

# The business logic



```
static void
cp_parser_static_assert(cp_parser *parser, bool member_p)
{
    tree condition;
    tree message;
    cp_token *token;
    location_t saved_loc;
    bool dummy;

    /* Peek at the `static_assert` token so we can keep track of exactly
     where the static assertion started. */
    token = cp_lexer_peek_token(parser->lexer);
    saved_loc = token->location;

    /* Look for the `static_assert` keyword. */
    if (cp_parser_require_keyword(parser, RID_STATIC_ASSERT,
                                  RT_STATIC_ASSERT))
        return;

    /* We know we are in a static assertion; commit to any tentative
     parse. */
    if (cp_parser_parsing_tentatively(parser))
        cp_parser_commit_to_tentative_parse(parser);

    /* Parse the `(` starting the static assertion condition. */
    cp_parser_require(parser, CPP_OPEN_PAREN, RT_OPEN_PAREN);

    /* Parse the constant-expression. Allow a non-constant expression
     here in order to give better diagnostics in finish_static_assert. */
    condition =
        cp_parser_constant_expression(parser,
                                      /*allow_non_constant_p= */true,
                                      /*non_constant_p= */&dummy);

    if (cp_lexer_peek_token(parser->lexer)->type == CPP_CLOSE_PAREN)
    {
        if (cxx_dialect < cxx1z)
            pedwarn(input_location, OPT_Wpedantic,
                    "static_assert without a message "
                    "only available with -std=c++1z or -std=gnu++1z");
        cp_lexer_consume_token(parser->lexer);
        message = build_string(1, "");
        TREE_TYPE(message) = char_array_type_node;
        fix_string_type(message);
    }
    else
    {
        /* Parse the separating ','. */
        cp_parser_require(parser, CPP_COMMA, RT_COMMA);

        /* Parse the string-literal message. */
        message = cp_parser_string_literal(parser,
                                           /*translate= */false,
                                           /*wide_ok= */true);

        /* A `)` completes the static assertion. */
        if (!cp_parser_require(parser, CPP_CLOSE_PAREN, RT_CLOSE_PAREN))
            cp_parser_skip_to_closing_parenthesis(parser,
                                                 /*recovering= */true,
                                                 /*or_comma= */false,
                                                 /*consume_paren= */true);
    }

    /* A semicolon terminates the declaration. */
    cp_parser_require(parser, CPP_SEMICOLON, RT_SEMICOLON);

    /* Complete the static assertion, which may mean either processing
     the static assert now or saving it for template instantiation. */
    finish_static_assert(condition, message, saved_loc, member_p);
}
```

Amazing!

# The business logic



```
static void
cp_parser_static_assert(cp_parser *parser, bool member_p)
{
    tree condition;
    tree message;
    cp_token *token;
    location_t saved_loc;
    bool dummy;

    /* Peek at the `static_assert` token so we can keep track of exactly
     where the static assertion started. */
    token = cp_lexer_peek_token(parser->lexer);
    saved_loc = token->location;

    /* Look for the `static_assert` keyword. */
    if ((cp_parser_require_keyword(parser, RID_STATIC_ASSERT,
                                  RT_STATIC_ASSERT)))
        return;

    /* We know we are in a static assertion; commit to any tentative
     parse. */
    if (cp_parser_parsing_tentatively(parser))
        cp_parser_commit_to_tentative_parse(parser);

    /* Parse the `(` starting the static assertion condition. */
    cp_parser_require(parser, CPP_OPEN_PAREN, RT_OPEN_PAREN);

    /* Parse the constant-expression. Allow a non-constant expression
       here in order to give better diagnostics in finish_static_assert. */
    condition =
        cp_parser_constant_expression (parser,
                                       /*allow_non_constant_p=*/true,
                                       /*non_constant_p=*/&dummy);

    if (cp_lexer_peek_token (parser->lexer)->type == CPP_CLOSE_PAREN)
    {
        if (cxx_dialect < cxx1z)
            pedwarn (input_location, OPT_Wpedantic,
                     "static_assert without a message "
                     "only available with -std=c++1z or -std=gnu++1z");
        /* Eat the `)` */
        cp_lexer_consume_token (parser->lexer);
        message = build_string (1, "");
        TREE_TYPE (message) = char_array_type_node;
        fix_string_type (message);
    }
    else
    {
        /* Parse the separating `,` */
        cp_parser_require (parser, CPP_COMMA, RT_COMMA);

        /* Parse the string-literal message. */
        message = cp_parser_string_literal (parser,
                                            /*translate=*/false,
                                            /*wide_ok=*/true);

        /* A `)` completes the static assertion. */
        if (!cp_parser_require (parser, CPP_CLOSE_PAREN, RT_CLOSE_PAREN))
            cp_parser_skip_to_closing_parenthesis (parser,
                                                   /*recovering=*/true,
                                                   /*or_comma=*/false,
                                                   /*consume_paren=*/true);
    }

    /* A semicolon terminates the declaration. */
    cp_parser_require (parser, CPP_SEMICOLON, RT_SEMICOLON);

    /* Complete the static assertion, which may mean either processing
     the static assert now or saving it for template instantiation. */
    finish_static_assert (condition, message, saved_loc, member_p);
}
```

Mmm.. A constant expression is not enough



# Parsing the `static_print` arguments

- I wanted `static_print` to accept any compile time thing, not only expressions
  - Types, template names
- How in the world am I going to parse this?
  - Ideas?
    - Template arguments!
    - `cp_parser_template_argument`!



# It works!

---

- Compiled the first program using `static_print`!
- But then, a bug:
  - This doesn't work:
  - `static_print("Check this out: ", sizeof(T) > 3);`
    - Ideas why?
    - I used `cp_parser_template_argument`, which knows it is inside a template argument list
    - When it sees the '>', it terminates the argument!
    - Lesson learned:
      - **Copy & Pasting may break some hidden code assumptions**
      - Make sure to scan the code you use for those assumptions



# Hurray!

---

- Now it really works!
- I can print-debug my own code at compile time!
- Maybe others will like to use this as well?
  - 4 options:
    1. Just use this for myself
      - (no work)
    2. Publish the .patch file
      - (a day's work)
    3. Try getting this merged this into GCC
      - (a month's work? Might not be accepted)
    4. Propose this to the standard
      - (two years work? There's already some proposal in circulation)
  - Went with #2

# Hello, world!



[↑](#) lonkamikaze 23 points · 1 year ago

[↓](#) I thought this would be an article about a really cool metaprogramming hack. 😞

[Share](#) [Report](#) [Save](#) [Give Award](#)

[↑](#) saarraz1 Clang Concepts dev [→](#) 14 points · 1 year ago

[↓](#) Doesn't hacking the compiler count as metaprogramming? ;)

[Share](#) [Save](#) [Edit](#) [...](#)

[↑](#) lonkamikaze 13 points · 1 year ago

[↓](#) Nope. Extending the language is cheating.

[Share](#) [Report](#) [Save](#) [Give Award](#)

A GCC 7.1 patch that adds a 'static\_print' statement to C++.

[Add topics](#)

[12 commits](#)

[1 branch](#)

[0 releases](#)

[2 contributors](#)

[GPL-3.0](#)

Branch: master ▾

[New pull request](#)

[Create new file](#)

[Upload files](#)

[Find file](#)

[Clone or download](#)

# A bug?! Impossible!



## build fails at stage2 #2

Closed

abigagli opened this issue on Jul 4, 2017 · 2 comments



abigagli commented on Jul 4, 2017

Contributor



On macos 10.12.5, native compiler being clang-4.0, the first stage succeeds and xg++ gets successfully built, but then compilation fails **during stage2** with the following error:



# Bootstrapping

- A new version of GCC comes out, with new optimizations
- Compile it – and get a compiler that builds **faster code**
  - But the compiler itself was compiled with a worse compiler
- Compile again – and get a **fast compiler** that builds fast code
  - “Stage 2”
  - But maybe the optimizations broke something?
- Compile again – and check you got **the same result** as in stage 2
  - “Stage 3”
- **TL;DR** – Compilers compiler compilers compile compile compilers

# So what was the issue?



- How did compilation fail on stage 1 succeed but stage 2 failed?

```
cp-tree.h:1090:5: error: expected unqualified-id before 'static_print'  
    static_print;  
    ^~~~~~
```

- Ideas?
  - We added a new keyword
  - In stage one we used a compiler without this keyword
  - I had a local variable named `static_print`
  - In stage 2, `static_print` is a keyword and using it as a variable name is a syntax error!
  - Changed the variable name – solved ☺☺



# What now?

---

- Given `static_print`, I could “profile” my long compilation times

```
static_print("Before big template");
funcThatInstantiatesHugeTemplates();
static_print("After big template");
    ■ (clang has a template profiler...)
```
- It's still too slow
  - GCC also had a page on their website saying they know the compiler is too slow and they need to take care of that
  - Some say Clang is faster...



# 4 options (Reprise)

---

1. Drop the project
  - (no work)
2. Stop using concepts
  - (a week's work + a lifetime of regret)
3. Optimize GCC
  - (no)
4. Implement Concepts in Clang myself
  - (a month or two maybe?)
  - Went with the last one



# Concepts in Clang

## The C++0x "Remove Concepts" Decision

By Bjarne Stroustrup, July 22, 2009

[Post a Comment](#)

### Concepts were to have been the central new feature in C++0x

Bjarne Stroustrup designed and implemented the C++ programming language. He can be contacted [here](#).

*At the July 2009 meeting in Frankfurt, Germany, the C++ Standards Committee voted to remove "concepts" from C++0x. Although this was a big disappointment for those of us who have worked on concepts for years and are aware of their potential, the removal fortunately will not directly*

Figure 4. Parsing associated functions.

# Concepts in Clang



## Technical specifications and standing documents

ISO C++ also publishes a number of documents describing additional language and library features that are not part of standard C++.

### ▼ List of features and minimum Clang version with support

Document	Latest draft	Compiler flag	Available in Clang?
SD-6: SG10 feature test recommendations	<a href="#">SD-6</a>	N/A	<a href="#">Clang 3.4 (N3745)</a> <a href="#">Clang 3.6 (N4200)</a> <a href="#">Clang 4 (P0096R3)</a> <a href="#">Clang 5 (P0096R4)</a> <a href="#">Clang 7 (P0096R5)</a> WIP ( <a href="#">P1353R0</a> )
[TS] Concepts	<a href="#">P0121R0</a>		Superseded by <a href="#">P0734R0</a>
[DRAFT TS] Coroutines	<a href="#">N4663</a>	-fcoroutines-ts -stdlib=libc++	Clang 5

# Concepts in Clang



## The state of Concepts in Clang

[Classic](#)   [List](#)   [Threaded](#)

Feb 05, 2017; 3:43pm [David Blaikie via cfe-dev](#) The state of Concepts in Clang

Feb 05, 2017; 7:57pm [David Blaikie via cfe-dev](#)

The Concepts TS implementation for Clang is occurring on trunk; so you are looking in the right place. Regardless of the implementation status in Clang, the TS remains an experimental design, which may be

WIP can mean anything from "we are running the last tests before release" to "I have put it on the TODO list".

The background is that we are currently redesigning a large template library and would really like to make use of concepts. Working with GCC during development is not a problem, but when we start distributing first release candidates maybe a year from now, it would be important to have Clang support, too.

If someone could shed light on the current status and whether there is an ETA that would help us a lot. Note that I am not implying that anyone should do anything for us, it's just important for us to know whether it's something we can likely expect for e.g. clang-6 or "definitely not in the next two years".

Thank you,  
Hannes

--

pgp-key: [https://hannes.hauswedell.net/hannes\\_hauswedell\\_public\\_key.asc](https://hannes.hauswedell.net/hannes_hauswedell_public_key.asc)  
fingerprint: FC35 7547 7916 DA55 DC42 27EA 1D57 8E18 A109 60BF



# Concepts in Clang

---

- Turns out clang even had a `-fconcepts-ts` flag!
  - But it seemed to just parse some requires-clauses and ignore them...
- Anyway, it seems no substantial work had been done at the time



# Getting It Merged

---

- I'm not a compiler engineer
- Why would the clang gods even let me work on their compiler?
- The plan:
  - Implement the whole feature without asking anyone
  - Show up at clang's door with everything implemented and then they'll accept me!

# How Hard Could It Be?

---



- It's just a bunch of error messages, right?



# Implementing a C++ feature

- Where do you even start?

## Technical specifications and standing documents

ISO C++ also publishes a number of documents describing additional language and library features that are not part of standard C++.

### ▼ List of features and minimum Clang version with support

Document	Latest draft	Compiler flag	Available in Clang?
SD-6: SG10 feature test recommendations	<a href="#">SD-6</a>	N/A	<a href="#">Clang 3.4 (N3745)</a> <a href="#">Clang 3.6 (N4200)</a> <a href="#">Clang 4 (P0096R3)</a> <a href="#">Clang 5 (P0096R4)</a> <a href="#">Clang 7 (P0096R5)</a> WIP ( <a href="#">P1353R0</a> )
[TS] Concepts	<a href="#">P0121R0</a>		Superseded by <a href="#">P0734R0</a>
[DRAFT TS] Coroutines	<a href="#">N4663</a>	-fcoroutines-ts -stdlib=libc++	Clang 5



- Changes to the standard are “diffs” to the standard text

## 6 Basic concepts

[basic]

### 6.1 Basic concepts

[gram.basic]

Add concepts to the list of entities.

- 3 An *entity* is a value, object, reference, function, enumerator, type, class member, bit-field, template, concept, template specialization, namespace, or parameter pack.

### 6.2 One-definition rule

[basic.def.odr]

Modify paragraph 1.

- 1 No translation unit shall contain more than one definition of any variable, function, class type, enumeration type, ~~or~~ template, or concept.

- The concepts diff is **36 pages long**
  - (to put things in perspective, the standard is 1400 pages)



# Here goes nothing

- Well let's start slowly
  - We'll add the notion of a "concept" declaration

A *template* defines a family of classes, functions, or variables, or a concept, or an alias for a family of types.

*template-declaration:*

template-head ~~template < template-parameter-list >~~ declaration  
template-head concept-definition

*template-head:*

template < template-parameter-list > requires-clause<sub>opt</sub>

*concept-definition:*

concept concept-name = constraint-expression

- Seems simple enough
  - "concept" + name + "=" + "constraint-expression"

```
// \brief Definition of concept, not just declaration actually.
class ConceptDecl : public TemplateDecl {
protected:
    Expr *ConstraintExpr;

    ConceptDecl(DeclContext *DC, SourceLocation L, DeclarationName Name,
                TemplateParameterList *Params, Expr *ConstraintExpr)
        : TemplateDecl(Concept, DC, L, Name, Params),
          ConstraintExpr(ConstraintExpr) {}

public:
    static ConceptDecl *Create(ASTContext &C, DeclContext *DC,
                               SourceLocation L, DeclarationName Name,
                               TemplateParameterList *Params,
                               Expr *ConstraintExpr);
    static ConceptDecl *CreateDeserialized(ASTContext &C, unsigned ID);

    Expr *getConstraintExpr() const {
        return ConstraintExpr;
    }

    void setConstraintExpr(Expr *CE) {
        ConstraintExpr = CE;
    }

    // Implement isa/cast/dyncast/etc.
    static bool classof(const Decl *D) { return classofKind(D->getKind()); }
    static bool classofKind(Kind K) { return K == Concept; }

    friend class ASTReader;
    friend class ASTDeclReader;
    friend class ASTDeclWriter;
};
```

# What now?



- Pick something roughly similar to a concept. search all files for

**Find in Path**     Match case »     File mask: `*.td`

**Q:** `VarTemplateDecl` 100+ matches in 10+ files

**In Project** [Module](#) [Directory](#) [Scope](#)

```
void VisitVarTemplateDecl(const VarTemplateDecl *D ASTDumper.cpp 479
void ASTDumper::VisitVarTemplateDecl(const VarTe ASTDumper.cpp 1622
DeclResult CheckVarTemplateId(VarTemplateDecl *Template Sema.h 6305
```

- Oh boy



# I do this for a while...

- I had to go through all manner of weird stuff
  - ASTDumper

```
DeclNodes.td
62     aet OMPCapturedExpr : DDecl<var>;
63     def NonTypeTemplateParm : DDecl<Declarator>;
64     def Template : DDecl<Named, "templates", 1>;
65     def RedeclarableTemplate : DDecl<Template, "redeclarable templates", 1>;
66     def FunctionTemplate : DDecl<RedeclarableTemplate>;
67     def ClassTemplate : DDecl<RedeclarableTemplate>;
68     def VarTemplate : DDecl<RedeclarableTemplate>;
69     def TypeAliasTemplate : DDecl<RedeclarableTemplate>;
70     def TemplateTemplateParm : DDecl<Template>;
71     def BuiltinTemplate : DDecl<Template>;
72     def Concept : DDecl<Template>;
```



# Where does this get parsed?

- Following VarTemplateDecl turned out to be a bit complicated
  - Let's just follow the template keyword!

```
ParseTemplate.cpp  *
405  /// \returns true if an error occurred, false otherwise.
406 bool Parser::ParseTemplateParameters(
407     unsigned Depth, SmallVectorImpl<NamedDecl *> &TemplateParams,
408     SourceLocation &LAngleLoc, SourceLocation &RAngleLoc) {
409     // Get the template parameter list.
410     if (!TryConsumeToken(tok::less, LAngleLoc)) {
411         Diag(Tok.getLocation(), diag::err_expected_less_after) << "template";
412         return true;
413     }
414 }
```

- Close enough!



# How parsing works(?)

- This looks promising!

```
// Consume the 'template', which should be here.
SourceLocation TemplateLoc;
if (!TryConsumeToken(tok::kw_template, TemplateLoc)) {
    Diag(Tok.getLocation(), diag::err_expected_template);
    return nullptr;
}

// Parse the '<' template-parameter-list '>'
SourceLocation LAngleLoc, RAngleLoc;
SmallVector<NamedDecl*, 4> TemplateParams;
if (ParseTemplateParameters(CurTemplateDepthTracker.getDepth(),
                            TemplateParams, LAngleLoc, RAngleLoc)) {
    // Skip until the semi-colon or a '}'.
    SkipUntil(tok::r_brace, StopAtSemi | StopBeforeMatch);
    TryConsumeToken(tok::semi);
    return nullptr;
}
```



# Down the line...

- Jackpot!

```
// Parse the actual template declaration.  
return ParseSingleDeclarationAfterTemplate(Context,  
                                         ParsedTemplateInfo(&ParamLists,  
                                         isSpecialization,  
                                         LastParamListWasEmpty),  
                                         ParsingTemplateParams,  
                                         DeclEnd, AS, AccessAttrs);
```

- I'm just gonna leave this here...

```
if (Tok.is(tok::kw_concept)) {  
    return ParseConceptDefinition(Context, TemplateInfo, DiagsFromTParams,  
                                 DeclEnd, AS, AccessAttrs,  
                                 prefixAttrs);  
}
```



Wc

- Co
  - Bu
  - Th  
  - Hu
  - Sp



# Typos

---

- It turns out when you use ParseExpression, it might encounter a non-existent identifier
- Which it will treat as a typo!
- So here, it recognized the “typo”, and returned the expression “A”
- I should've known (somehow) to call CorrectDelayedTyposOnExpr
- Which will issue error messages for all typos and still return “A”...

# The Unwritten Rule(s)



- 
- The codebase is full of unwritten rules
  - Things you (probably) have no way of knowing about until you don't use them and debug the consequences
    - Stack objects (instantiation)
    - Layering (Parse → Act → Create → Constructor)
    - ...
  - Which is why Copy & Paste really is a good strategy
    - Find place in code that does something like what you want
    - Notice any unfamiliar patterns used there

# Show Must Go On



- 
- The typo example is a common example of the general mindset you have to have when developing for a compiler
  - No quit-outs!
    - If the user made a mistake, fire an error message, **guess** what they actually meant and continue compiling as if that's what happened



# Defend the User!

- Which is correct?

```
template<typename T>
auto foo(T a) -> void
```

✗ 

```
template<typename T>
auto foo(T a) requires Large<T> -> void
```

✓ 

```
template<typename T>
auto foo(T a) -> void requires Large<T>
```

- But users are still gonna get confused
- In practice, I try to parse both ways and accept both forms
  - Issuing an error message if the wrong one is used
  - But code behaves the same both ways
- As the compiler you need to defend the users from the harsh standard
  - Expect the unexpected

An

# The Fine Print



- 
- Every word used in the standard is used for a reason
  - Cutting corners almost never works



# The Same Expression

```
template<typename T>
void foo() requires sizeof(T) > 1;
template<typename T>
void foo() requires sizeof(T) > 1 && sizeof(T) >= 2;
foo<short>();
```

- This should work right?
- Well, no:

entity, called the *parameter mapping* (17.10.2). Two atomic constraints are *identical* if they are formed from the same expression and the targets of the parameter mappings are equivalent

- The same *expression* – not the same expression!
  - Italics *expression* == the grammar rule expression

*expression:*  
    *assignment-expression*  
    *expression , assignment-expression*



- In practice, I try both ways and give an error message explaining the difference



# Anyway,

---

- I continue copying and pasting my way around the feature
- For example, how would you find the place to check whether the constraints are satisfied?
  - Ideas?
    - Search for the error message produced when a wrong no. of template arguments is given -> leads you to the function that checks template arguments for a given template
- I finish most of the feature in about a month's work
- What now?

# Aw, Snap!



- I have most of the thing implemented already (or at least, that's what I thought at the time)
- Was about to show up with the ready to merge patch to the clang community
- Then I saw this:

## Incremental Development

In the LLVM project, we do all significant changes as a series of incremental patches. We have a strong dislike for huge changes or long-term development branches. Long-term development branches have a number of drawbacks:

1. Branches must have mainline merged into them periodically. If the branch development and mainline development occur in the same pieces of code, resolving merge conflicts can take a lot of time.
- A friend also warned me that getting stuff merged to LLVM is really hard

# Plan B



- Instead of coming in with a patch ready to merge,
- Break what I did into commit-size “steps” of how I “would” “theoretically” implement concepts in clang
- Show up with the plan instead!

saarraz / clang-concepts-roadmap

Unwatch 4 Star 12 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Roadmap for implementation of Concepts in the Clang compiler. Edit

Manage topics

15 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

saarraz Update README.md Latest commit 2b57892 on Aug 13, 2018

README.md Update README.md 6 months ago

README.md

## Concepts (P0734R0) Implementation in Clang

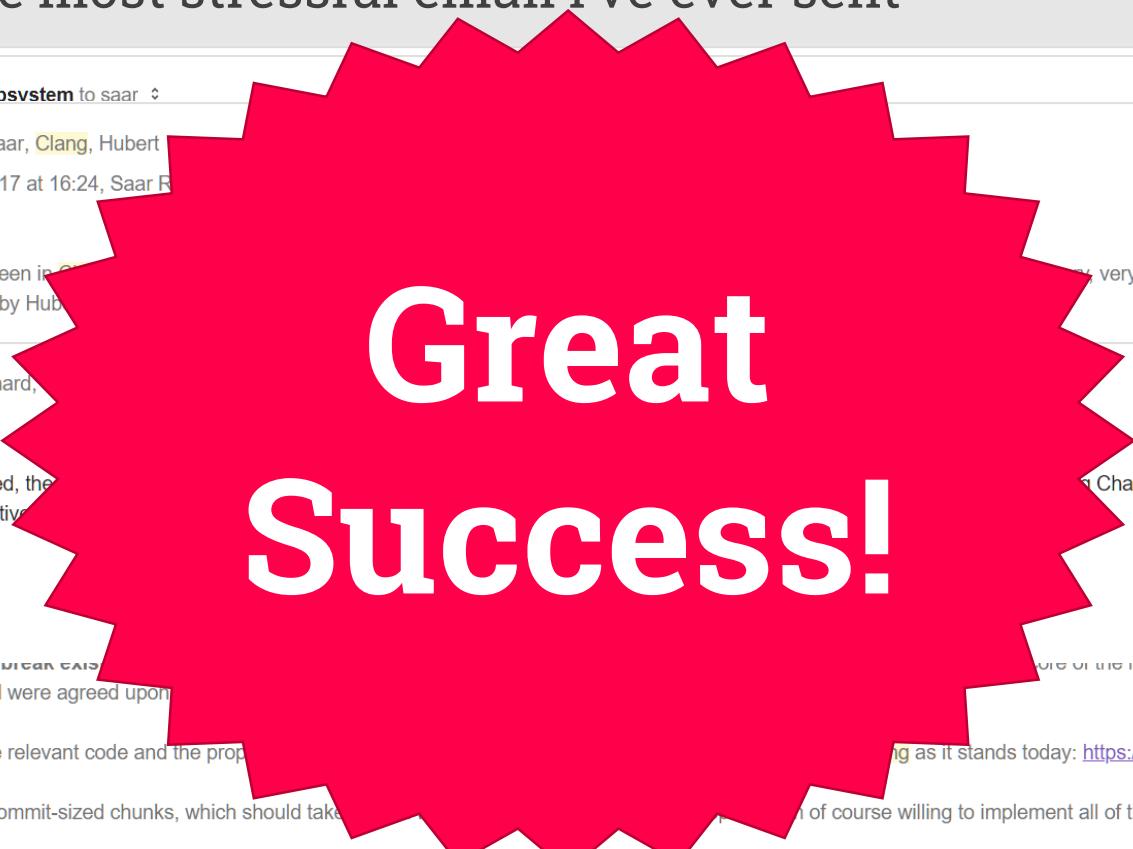
Roadmap for implementation of Concepts in the Clang compiler.

Roadmap



# The Moment of Truth

- The most stressful email I've ever sent



Mail Delivery Subsystem to saar

Richard Smith to Saar, Clang, Hubert

On 17 November 2017 at 16:24, Saar R

Hi all,

As I've read and seen in C++ Concepts, placed by Hubert Tong

Hubert Tong to Richard, Changyu and Nathan to the CC because

Hi Saar,

As Richard mentioned, they've also been active

...  
stable already and were agreed upon

I took a look at the relevant code and the proposed roadmap

I broke it up into commit-sized chunks, which should take

Thank you for your analysis and the offer to help out!

11/18/17 : 11/18/17 :

Great Success!

We are very much open to adding support for P0734R0 to Clang, along with all other features voted into the working draft for C++20. The only reason this has not already been implemented is a lack of volunteers such as yourself with the time to devote to the task.

# Well That Escalated Quickly



- I was preaching to the choir
- They wanted to...  
I just needed somebody to do it!



Whoops!

Changyu Li to hubert.rein

Hi Saar,

I'd like to help you impl

...

11/19/17

future, not right now though.

made on imgur

# Working in the Real World



- LLVM and Clang use SVN (why???)
  - But there's a Git mirror 🤪
- I had my own Git repository which I work against
  - A branch for every step of the roadmap
- Upload diffs to Phabricator (the CR system used by LLVM)
  - Nag people until they CR
- Work on a Linux VM
  - Building the compiler in Debug mode takes 30GB and an hour to link the main executable (compared to 4 seconds for release)
  - Or 4 minutes on an SSD 😅
  - Let's get to work!

4  
MONTHS  
LATER

# Why Small Commits are Good



Clang Concepts  
Saar Ra  
Yo  
I'm work  
What do  
Thanks

Matt Go  
Hey Saar  
Would lo  
tarred up  
[explorer](#)  
Cheers,  
...

11010 .LX0: lib.f: .text // \s+ Intel Demangler

1 foo(): # @foo()  
2 push rbp  
3 mov rbp, rsp  
4 sub rsp, 16  
5 lea rdi, [rbp - 8]  
6 call S1<int>::f()  
7 add rsp, 16  
8 pop rbp  
9 ret

Compiler Explorer interface showing assembly output for a C++ function. The assembly code is color-coded by instruction type: pushes (light green), moves (medium green), subtractions (light blue), loads (yellow), calls (orange), additions (light purple), pops (dark purple), and returns (pink). The code implements a standard C-style function prologue and epilogue.

# Then Others Learned the Trick As Well...



The screenshot shows the Compiler Explorer interface for x86-64 clang (experimental concepts). The sidebar lists various compiler configurations, with "x86-64 clang (experimental concepts)" selected. The main area displays assembly code for a function named @foo(). The assembly output includes:

```
# @foo()
rsp
16
[rbp - 8]
nt>::f()
16
```

The interface includes a toolbar with compiler options like .text, //, \s+, Intel, Demangle, Libraries, and Add new.

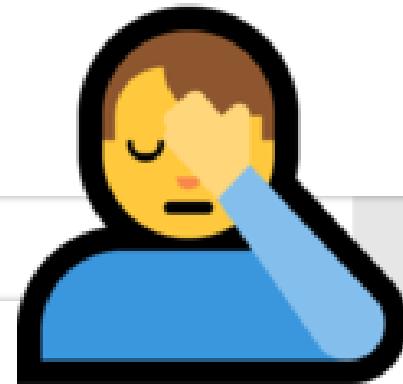
# Let the world know!



↑ r/cpp · Posted by u/saarraz1 Clang Concepts dev 10 months ago ↗  
53 Experimental Clang Sun Apr 01 2018 14:08:19 GMT+0300 (Israel Daylight Time) Compiler Explorer

blelbach NVIDIA | Thrust | HPX | C++ Committee | CppCon | C++Now 5 points · 10 months ago  
Either way, this is a huge troll.  
Share Report Save Give Award

BillyONeal MSVC STL Dev 6 points · 10 months ago  
Not sure if serious or April fool's  
Share R  
Have fun :) 20 Comments  
dpsi 2 points · 10 months ago  
I can't tell either, I keep getting compilation failures.  
Share Report Save Give Award



- See the bug?
- How about now?



# Almost done!

- All I had left were requires expressions
- e.g.

```
requires (T t) {  
    typename T::foo_result;  
    { t.foo(); } -> typename T::foo_result;  
    t++;  
    requires T::is_ok;  
}
```

- Seems easy enough

**ONE  
ETERNITY  
LATER**

# Yeah it wasn't so easy



- Probably one of the most complicated expressions in the language
- But it has been parsed!
- The compiler is now feature complete!



r/cpp · Posted by u/saarraz1 · Clang Concepts dev · 6 months ago



152 **Clang Concepts is now feature-complete!**

A few months ago I released the clang concepts build to Compiler Explorer but it was still missing some features (namely requires expressions).

I'm pleased to announce that today **the implementation is feature complete** and contains all concepts features present in the current standard working draft!

Note that the standard does not include a "terse syntax" ("void foo(const Container &s)") yet, nor does the current implementation.

Check it out on compiler explorer: <https://godbolt.org/g/Xthpfw>

Building the compiler for yourself is also possible, visit my github repo <https://github.com/saarraz/clang-concepts> for instructions.

Please do play around with this and **report any bugs** you find (open an issue on the github repo)! Any other feedback regarding the feature will also be greatly appreciated. This will *greatly* help get this merged sooner.

There's still work to do before this is merged to trunk, namely getting some more CR, finding more bugs, and solving some issues which aren't clear standard-wise.



47 Comments



Share



Edit Post



Save



Hide

...

# Bugs bugs bugs



- Ever since then I've been fixing bugs reported by the (incredible) concepts community!
  - Mostly on the C++ Slack space
- And then, in November...



Posted by u/blelbach NVIDIA | Thrust | HPX | C++ Committee | CppCon | C++Now 3 months ago



**219 2018 San Diego ISO C++ Committee Trip Report (Ranges v1 TS for C++20; consensus on modules design; new Language and Library Evolution Incubators)**

The ISO C++ Committee met in San Diego, California us last week to work on the International Standard (IS), C++20. This meeting was the last major meeting before the standard is finalized, but existing proposals like modules (on track) and coroutines (quietly merged) and ranges (not yet merged) can still make C++20. We'll make our final decisions about them at the next meeting.

This week, we added the following features to the C++20 draft:

- [Ranges](#).
- [void foo\(Concept auto x\)-style terse syntax](#).
- [constexpr functions](#).
- [std::is\\_constant\\_evaluated](#).
- [constexpr union](#).





# Current Status

- No new bugs have surfaced in a while (after fixing over 40!)
- Working towards merging this into trunk

Revision Contents				
Files	History	Commits	Stack (7 Open)	Similar
Status	Author	Revision		
● </> Needs Review	saar.raz	<a href="#">D50360 [Concepts] Requires Expressions</a>		
● </> Needs Review	saar.raz	<a href="#">D44352 [Concepts] Constrained template parameters</a>		
● </> Needs Review	saar.raz	<a href="#">D43357 [Concepts] Function trailing requires clauses</a>		
● </> Needs Review	saar.raz	<a href="#">D41910 [Concepts] Constrained partial specializations and function overloads.</a>		
● </> Needs Review	saar.raz	<a href="#">D41569 [Concepts] Constraint enforcement and diagnostics</a>		
● </> Needs Review	saar.raz	<a href="#">D41284 [Concepts] Associated constraints infrastructure.</a>		
● </> Needs Review	saar.raz	<a href="#">D41217 [Concepts] Concept Specialization Expressions</a>		
● </> Needs Review	changyu	<a href="#">D40381 Parse concept definition</a>		

# Lessons Learned



- 
- Hacking on compilers is fun!
  - Anyone with a Ctrl key can do it
  - Be naïve at first, learn from your mistakes (and from CR)
  - Search really hard for developers' manuals!
  - Everything in the standard is there for a reason
  - Take control of your compiler!
  - The fastest way to get C++20

# Questions?

requires Answerable<Q>

