Clang Concepts

how to write your first compiler feature

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)

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 Ha
    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
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_hashe
    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 (Northern Israel)
  - Like programming, graphic design and video games (Dark Souls)
- 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 7!
  - Which wasn't even out back then
    - Probably still buggy…
      - Not sure if maintained
        - Nah, it'll be fine

A slippery slope

- Built GCC 7
- 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

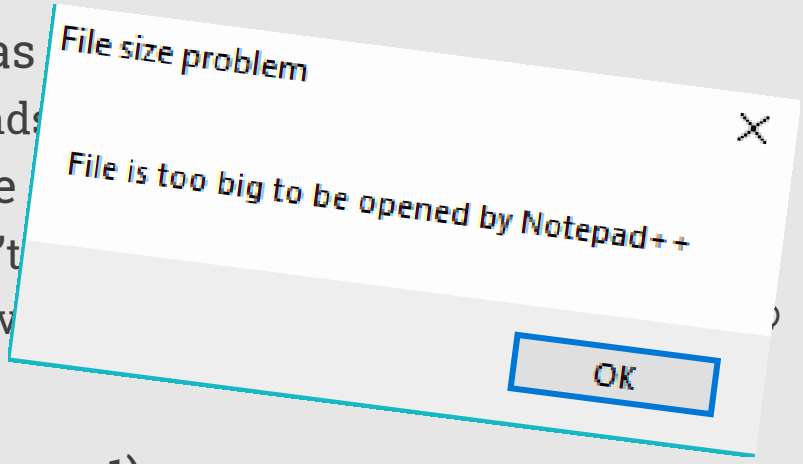
- So at one point I tried to compile the project
- It froze
  - The whole PC
    - The kernel
      - The mouse won't even move
- Maybe a very long error message?
  - -fmax-errors=1
    - Still doesn't work
  - Output the message to a file?
    - Still doesn't work
- A problem with cygwin?

Oh well it's probably a Windows problem

- Move to a Linux VM
- Freezes
  - The host as well
    - (In hindsight it was a BIOS problem)
  - Output the error message to a file
    - Doesn't freeze…
    - We have a file with an error message!!!! 🍻
- **1.2GB**
  - (-fmax-errors=1)
  - What does it say?

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>]`

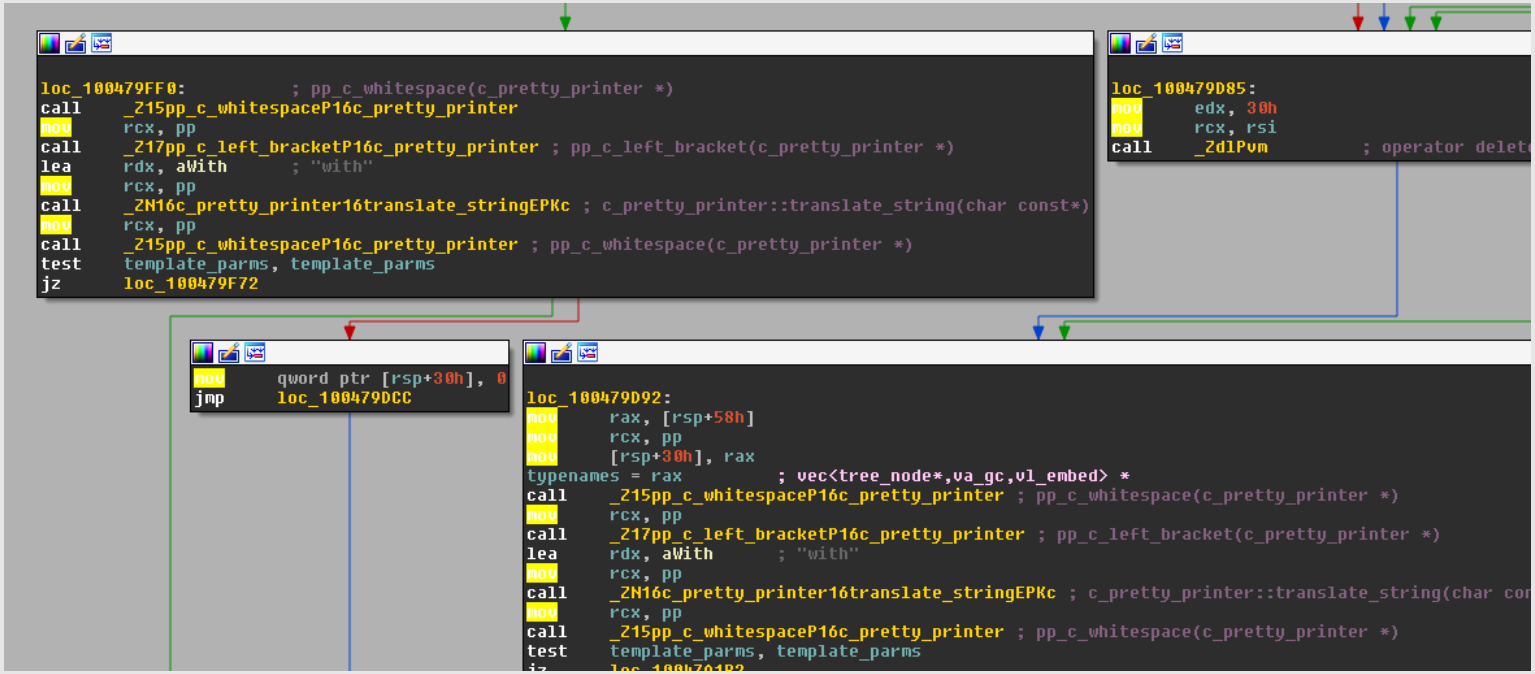
A long-term solution

- 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>]`

🤔

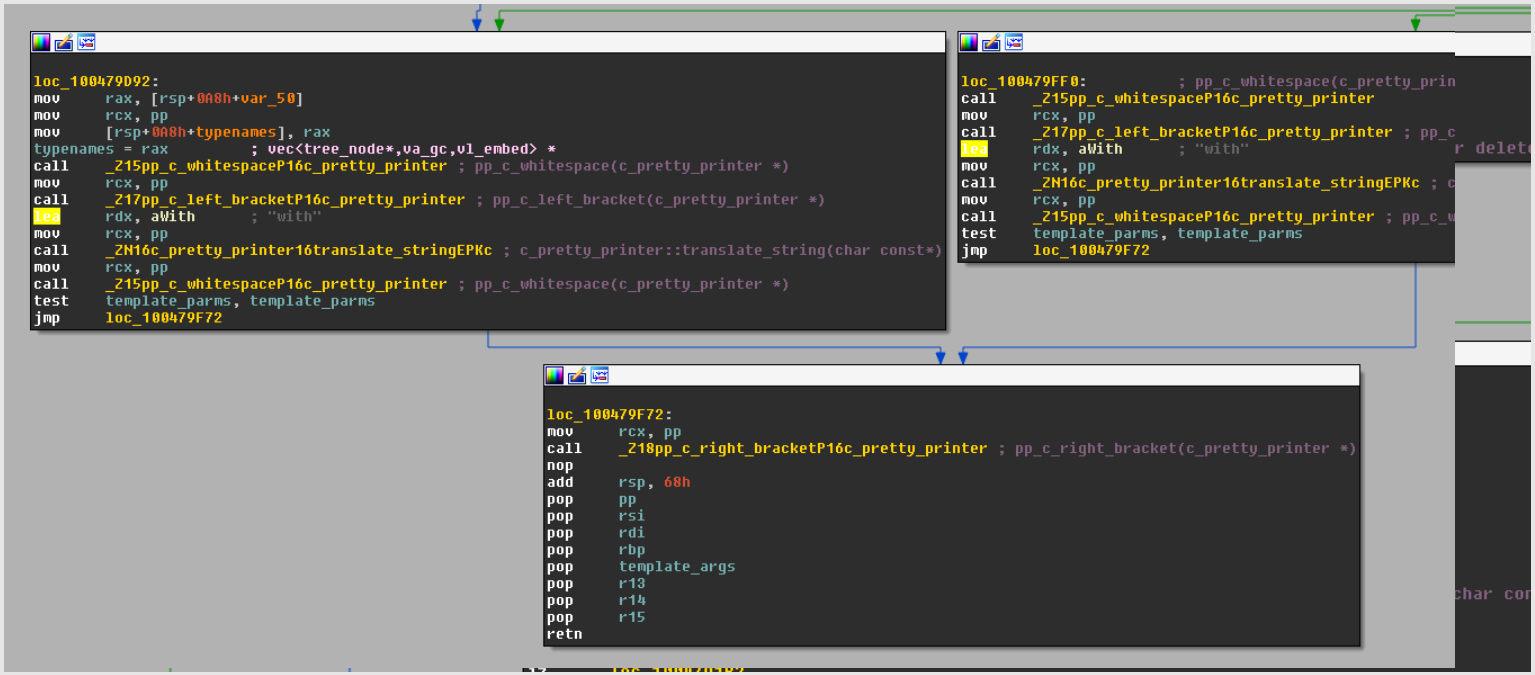
A long-term solution

- 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>]`

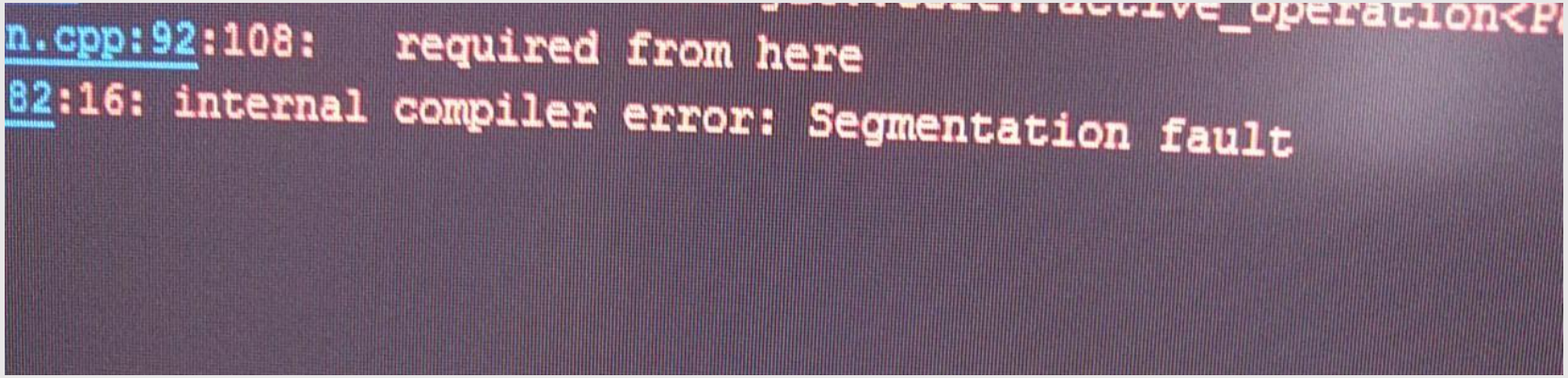


A long-term solution

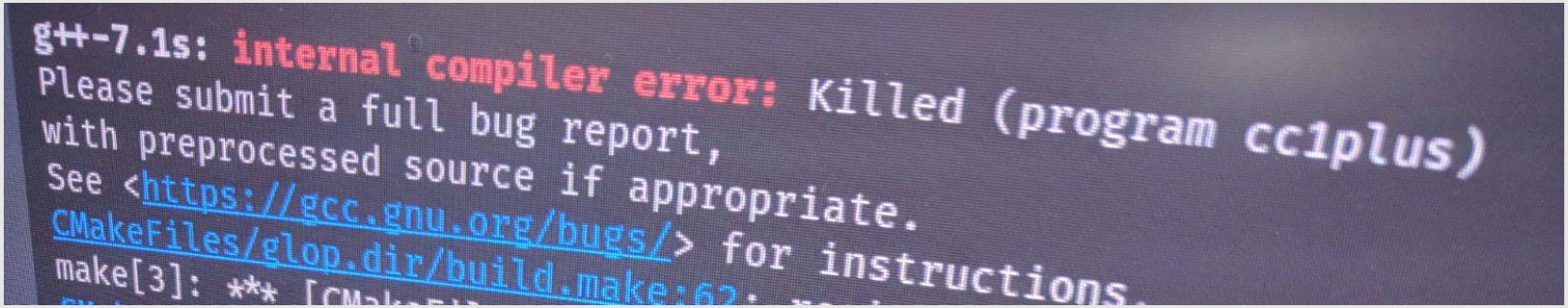
- Inheriting from every long template like this is a hassle
- And sometimes I don't even need all that information
- If we take a look at the error message:
  - `In instantiation of foo::bar<T, U>::bar() [with T = <1>, U = <2>]`



Another unindicative error message



And accusations of murder



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++:

parser.c

```
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
```

C source file | length : 1,207,273 lines : 39,199 | Ln : 39,199 Col : 1 Sel : 0 | 0 | Unix (LF) | UTF-8 | INS

- 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);
```

- I'll just leave this here:

```
/* 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);
```

More Copy & Paste

- ```
  cp_parser_static_print (parser, /*member_p=*/false);
  ```
- ```
  lse if (token1->keyword == RID_STATIC_PRINT)
  ```
- ```
  f the next token is `static_print' we have a static print statement. */
  ```

- 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_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");
      /* 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);
}
```

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);
}
```

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);
```

Mmm.. A constant expression is not enough

```
  /* 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);
```

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!

Posted by u/saarraz1 Clang Concepts dev 1 year ago

157

I created a C++ "static_print" statement! It allows printing formatted text during compilation, template instantiation and constexpr execution!

github.com/saarra...

32 Comments Share Save Hide ...

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

Hello, world!

Posted by u/saarraz1 Clang Concepts dev 1 year ago

157

I created a C++ "static_print" statement! It allows printing formatted text during compilation, template instantiation and constexpr execution!

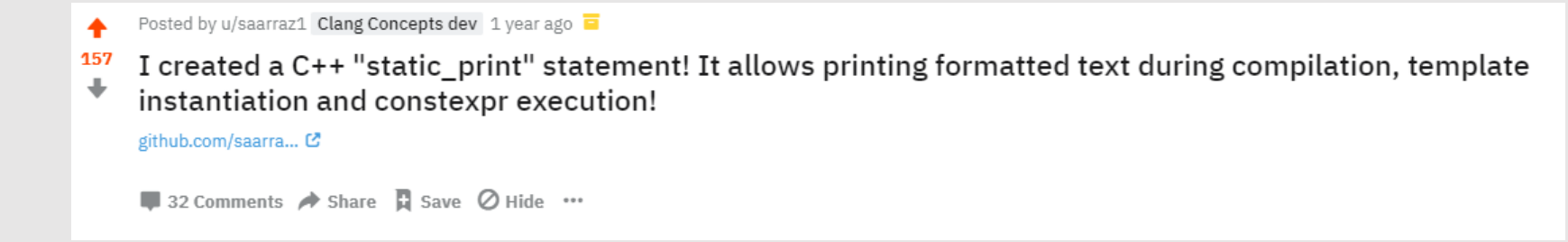
github.com/saarra...

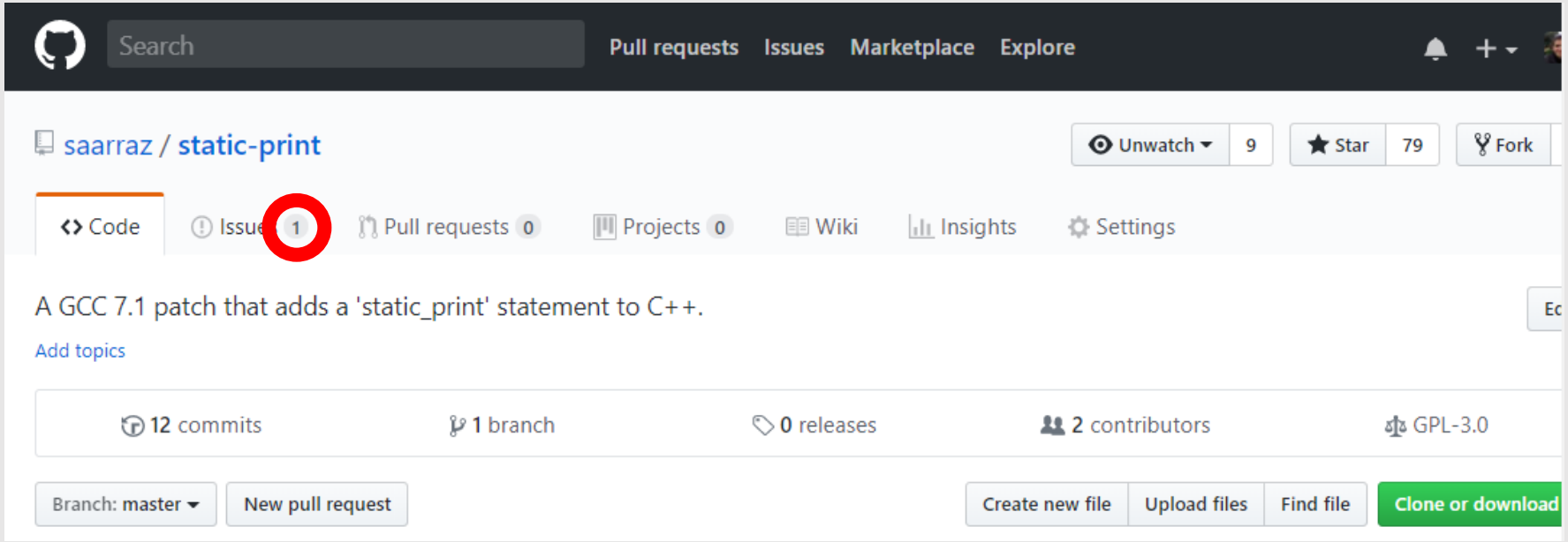
32 Comments Share Save Hide ...

Hello, world!


C++ Weekly
with Jason Turner
Episode 76
static_print
James Coulter
Jon Raymond
Léo Willian Kölln
Paul Ilardi
Pedro Henrique Linhares
Philip Whitfield
Rainer Grimm
Sergei Soloviev

Hello, world!


Posted by u/saarraz1 Clang Concepts dev 1 year ago
157
I created a C++ "static_print" statement! It allows printing formatted text during compilation, template instantiation and constexpr execution!
github.com/saarra...
32 Comments
Share
Save
Hide


Search
Pull requests
Issues
Marketplace
Explore
saarraz / static-print
Unwatch 9
Star 79
Fork
Code
Issues 1
Pull requests 0
Projects 0
Wiki
Insights
Settings
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

- Someone's probably done it already, right?
- LMGTFY

ConceptClang: an implementation of C++ concepts in Clang ...
https://www.semanticscholar.org/.../ConceptClang%3A...concepts.../982369338e3027b9...
In this paper, we present our experience implementing an infrastructure for exploring **concept** designs based on **Clang**--an **LLVM** frontend for the C family of ...

- 🤔

Concepts in Clang

ConceptClang: an implementation of C++ concepts in Clang

Larisse Voufo, Marcin Zalewski, Andrew Lumsdaine • Published 2011 in WGP@ICFP • DOI: 10.1145/2036918.2036929

Concepts are a proposed C++ extension for constraints-based poly-morphism. In this paper, we present our experience implementing an infrastructure for exploring concept designs based on Clang--an LLVM frontend for the C family of languages. We discuss how the primary proposed features of concepts (such as concept-based lookup, overloading and constrained templates) are implemented in Clang, and how our implementation can be extended to support the different approaches suggested within the C++ community. Some illustrations are presented and include a subset of the Boost Graph Library. LESS

Concepts in Clang

Share

```
1  void func();                                  // #1
2  void foo(int a, int b);                       // #2
3  template<typename T> void foo(T t);           // #3
4  concept A<typename T> {
5    void bar();                                 // #4
6    void foo(int a, int b)                      // #5
7    {
8      func();                                   // Picks #1
9      ::foo(0,1);                               // Picks #2
10     foo(0,1);                                 // Picks #5
11     bar();                                    // Picks #4
12     foo(t);                                   // Picks #3
13   }
14 }
```

Figure 4. Parsing associated functions.

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	SD-6	N/A	Clang 3.4 (N3745) Clang 3.6 (N4200) Clang 4 (P0096R3) Clang 5 (P0096R4) Clang 7 (P0096R5) WIP (P1353R0)
[TS] Concepts	P0121R0		Superseded by P0734R0
[DRAFT TS] Coroutines	N4663	-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

Hi everyone,

I wanted to ask what the current state of the Concepts TS is in Clang. I saw some older list threads last year, but also didn't see much in regard of commits, but maybe I am also looking in the wrong places.

The status page lists it as WIP, but from my own experience I know that WIP can me 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 "definetely not in the next two years".

Thank you,
Hannes
--
pgp-key: https://hannes.hauswedell.net/hannes_hauswedell_public_key.asc
fingerprint: FC35 7547 7916 DA55 DC42 27EA 1D57 8E18 A109 60BF

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 h

WIP can me 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 "definetely not in the next two years".

Thank you,
Hannes
--
pgp-key: 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 clangs 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	SD-6	N/A	Clang 3.4 (N3745) Clang 3.6 (N4200) Clang 4 (P0096R3) Clang 5 (P0096R4) Clang 7 (P0096R5) WIP (P1353R0)
[TS] Concepts	P0121R0		Superseded by P0734R0
[DRAFT TS] Coroutines	N4663	-fcoroutines-ts -stdlib=libc++	Clang 5

P0734R0

- 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*$_{opt}$

concept-definition:

`concept` *concept-name* `=` *constraint-expression*

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

How do you add concept?

- Using the only tool in my arsenal right now
  - Copy & Paste!
- Search the entire sources for a file named Template something
  - TemplateDecl.h!
  - Contains a class named `TemplateDecl`!
  - A bunch of other classes inherit from it or from `RedeclarableTemplateDecl`…
    - But it doesn't seem that the syntax allows to redeclare a template (only to declare and define it at the same time)
  - Let's inherit from TemplateDecl!

```
// \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 it and add concepts!
  - Ideas?
    - `VarTemplateDecl`!

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

What now?

- Pick something roughly similar to a concept, search all files for it and add

Find in Path Match case File mask: *.td

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

I do this for a while…

- I had to go through all manner of weird stuff
  - ASTDumper
  - ASTReader
  - ASTWriter
  - …
  - Mentions of “VarTemplate” in non-code files

I do this for a while...

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

DeclNodes.td

```
62          def 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>;
```

I do this for a while…

- I had to go through all manner of weird stuff
  - ASTDumper
  - ASTReader
  - ASTWriter
  - …
  - Mentions of “VarTemplate” in non-code files
  - A bunch of switch-cases
- Compiles!

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
```

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);
}
```

```
Decl *
Parser::ParseConceptDefinition(unsigned Context,
                               const ParsedTemplateInfo &TemplateInfo,
                               ParsingDeclRAIIObject &DiagsFromTParams,
                               SourceLocation &DeclEnd,
                               AccessSpecifier AS,
                               AttributeList *AccessAttrs) {
  assert(TemplateInfo.Kind != ParsedTemplateInfo::NonTemplate &&
         "Template information required");

  assert(Tok.is(tok::kw_concept));
  ConsumeToken();

  if (!Tok.is(tok::identifier)) {
    Diag(Tok.getLocation(), diag::err_expected) << tok::identifier;
    return nullptr;
  }

  IdentifierInfo *Id = Tok.getIdentifierInfo();
  SourceLocation IdLoc = ConsumeToken();

  if (!TryConsumeToken(tok::equal)) {
    Diag(Tok.getLocation(), diag::err_expected) << "equal";
    return nullptr;
  }

  ExprResult ConstraintExprResult = ParseExpression();
  if (ConstraintExprResult.isInvalid())
    return nullptr;

  ExpectAndConsumeSemi(diag::err_expected_semi_declaration);
  Expr *ConstraintExpr = ConstraintExprResult.get();
  return Actions.ActOnConceptDefinition(getCurScope(),
                                        *TemplateInfo.TemplateParams,
                                        Id, IdLoc, ConstraintExpr);
}
```

Works!

- Concept is parsed!
- But wait! There's a bug
- The following code compiles:

```
constexpr bool A = true;

template<typename T>

concept C = B;
```

- Huh?
- Spot the bug:

Wo

- Co
- Bu
- Th
  co
  te
  co
- Hu
- Sp

```
Decl *
Parser::ParseConceptDefinition(unsigned Context,
                               const ParsedTemplateInfo &TemplateInfo,
                               ParsingDeclRAIIObject &DiagsFromTParams,
                               SourceLocation &DeclEnd,
                               AccessSpecifier AS,
                               AttributeList *AccessAttrs) {
  assert(TemplateInfo.Kind != ParsedTemplateInfo::NonTemplate &&
         "Template information required");

  assert(Tok.is(tok::kw_concept));
  ConsumeToken();

  if (!Tok.is(tok::identifier)) {
    Diag(Tok.getLocation(), diag::err_expected) << tok::identifier;
    return nullptr;
  }

  IdentifierInfo *Id = Tok.getIdentifierInfo();
  SourceLocation IdLoc = ConsumeToken();

  if (!TryConsumeToken(tok::equal)) {
    Diag(Tok.getLocation(), diag::err_expected) << "equal";
    return nullptr;
  }

  ExprResult ConstraintExprResult = ParseExpression();
  if (ConstraintExprResult.isInvalid())
    return nullptr;

  ExpectAndConsumeSemi(diag::err_expected_semi_declaration);
  Expr *ConstraintExpr = ConstraintExprResult.get();
  return Actions.ActOnConceptDefinition(getCurScope(),
                                        *TemplateInfo.TemplateParams,
                                        Id, IdLoc, ConstraintExpr);
}
```

Wo

- Co
- Bu
- Th
  co
  te
  co
- Hu
- Sp

```
Decl *
Parser::ParseConceptDefinition(unsigned Context,
                               const ParsedTemplateInfo &TemplateInfo,
                               ParsingDeclRAIIObject &DiagsFromTParams,
                               SourceLocation &DeclEnd,
                               AccessSpecifier AS,
                               AttributeList *AccessAttrs) {
  assert(TemplateInfo.Kind != ParsedTemplateInfo::NonTemplate &&
         "Template information required");

  assert(Tok.is(tok::kw_concept));
  ConsumeToken();

  if (!Tok.is(tok::identifier)) {
    Diag(Tok.getLocation(), diag::err_expected) << tok::identifier;
    return nullptr;
  }

  IdentifierInfo *Id = Tok.getIdentifierInfo();
  SourceLocation IdLoc = ConsumeToken();

  if (!TryConsumeToken(tok::equal)) {
    Diag(Tok.getLocation(), diag::err_expected) << "equal";
    return nullptr;
  }

  ExprResult ConstraintExprResult =
      Actions.CorrectDelayedTyposInExpr(ParseExpression());
  if (ConstraintExprResult.isInvalid())
    return nullptr;

  ExpectAndConsumeSemi(diag::err_expected_semi_declaration);
  Expr *ConstraintExpr = ConstraintExprResult.get();
  return Actions.ActOnConceptDefinition(getCurScope(),
                                        *TemplateInfo.TemplateParams,
                                        Id, IdLoc, ConstraintExpr);
}
```

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>
```

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>
```

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

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) >= 4;
foo<int>();
```

- This should work right?

The Same Expression

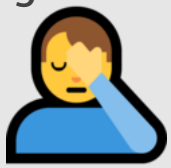
```
template<typename T>
void foo() requires sizeof(T) > 1;
template<typename T>
void foo() requires sizeof(T) > 1 && sizeof(T) >= 4;
foo<int>();
```

- 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:

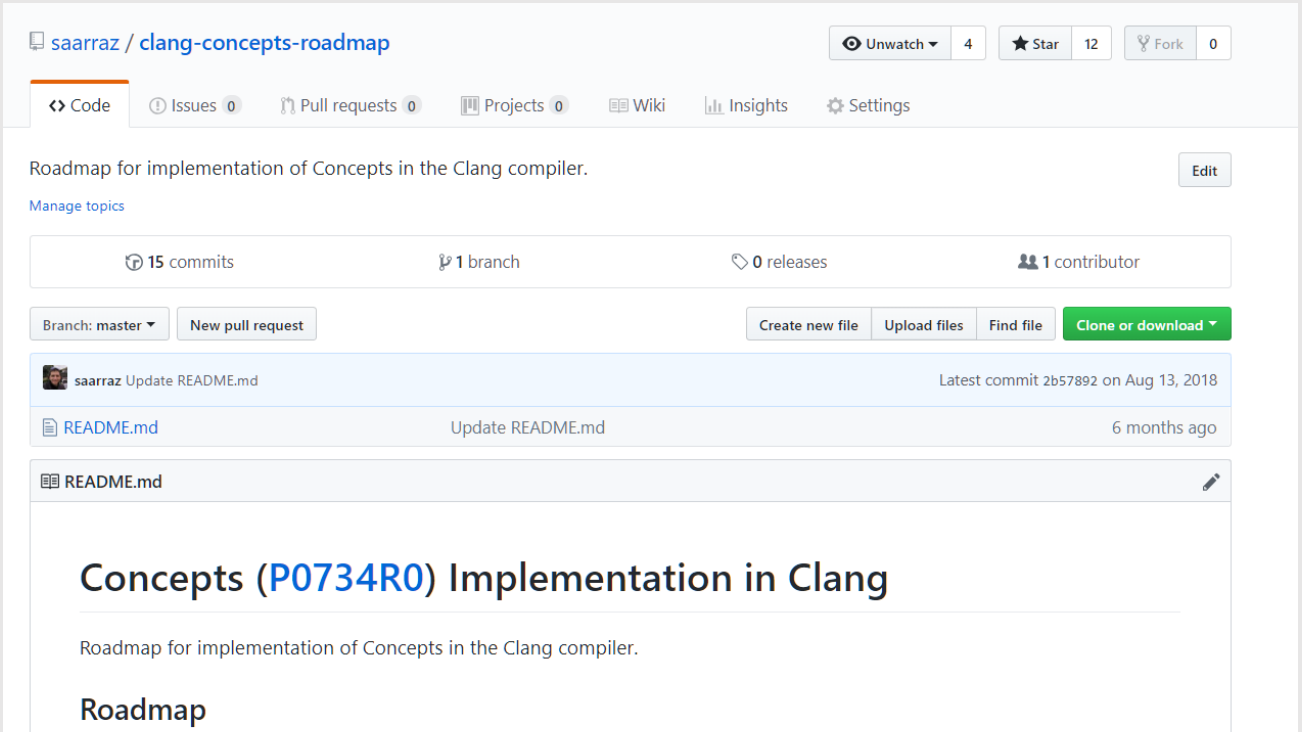
Incremental Development

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.

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!



The Moment of Truth

- The most stressful email I've ever sent

Roadmap for a Concepts implementation P0734R0, currently merged into C++20 draft

S

Saar Raz to cfe-dev, hubert.reinterpretcast@gmail.com 11/18/17

Hi all,

As I've read and seen in Clang's code, a Concepts implementation in Clang hasn't really been pushed forward in the past year, with only very, very little code in place right now regarding Concepts, placed by Hubert Tong, CC'd, back in February, and very little to non-existent discussions in the mailing lists.

As of the Toronto meeting back in July, a subset of the original Concepts TS was merged into draft C++20, which omits some shorthand notations for concepts but keeps the core parts of the language feature. Most notably - function concepts were removed, as well as the 'bool' keyword after the 'concept' keyword, and the 'introducer' and "terse/natural syntax" were omitted.

I believe we should **go ahead and implement P0734R0 now** for the following reasons:
- P0734R0 hasn't yet been implemented in any compiler (excluding GCC which implements a different proposal), and furthermore **no Concepts implementation was made from specification** - Andrew Sutton who worked on the GCC implementation also wrote the proposal. Doing a concepts implementation now would, I believe, help the committee discover defects and in general be more confident in the proposal.
- to be honest, it is **really not that hard** to implement, and can, I believe, be done in a month or two.
- Concepts will be present in C++2a unless something extraordinary happens - if not in their current form, then in a very similar form with maybe a few more terse syntaxes introduced **which would not break existing code** but just allow for nicer code - in any case, I believe it is a good time to lay a foundation for the core of the language feature which seems pretty stable already and were agreed upon by the committee.

I took a look at the relevant code and the proposal and hereby propose **a roadmap for implementing** Concepts in clang as it stands today: https://github.com/saarraz/clang-concepts-roadmap
I broke it up into commit-sized chunks, which should take us to a working implementation of the proposal. I'm of course willing to implement all of this if needed.

Please tell me what you think and post issues/pull requests to the roadmap repo or post here.
I'd really like to see this get going.

Thanks!
...

The Moment of Truth

- The most stressful email I've ever sent

Mail Delivery Subsystem to saar 11/18/17

Address not found

Your message wasn't delivered to **cfe-dev@llvm.org** because the address couldn't be found, or is unable to receive mail.

LEARN MORE

The response was:

```
550-5.1.1 The email account that you tried to reach does not exist. Please try 550-5.1.1 double-checking the recipient's email address for typos or
550-5.1.1 unnecessary spaces. Learn more at 550 5.1.1 https://support.google.com/mail/?p=NoSuchUser 29sor933400iog.70 - gsmtp
```

---------- Forwarded message ----------
From: Saar Raz <saar@raz.email>
To: cfe-dev@llvm.org
Cc: "hubert.reinterpretcast@gmail.com" <hubert.reinterpretcast@gmail.com>
Bcc:
Date: Sat, 18 Nov 2017 00:22:18 +0000

The Moment of Truth

- The most stressful email I've ever sent

Mail Delivery Subsystem to saar 11/18/17

Richard Smith to Saar, Clang, Hubert 11/18/17

On 17 November 2017 at 16:24, Saar Raz via cfe-dev <cfe-dev@lists.llvm.org> wrote:

> Hi all,
>
> As I've read and seen in Clang's code, a Concepts implementation in Clang hasn't really been pushed forward in the past year, with only very, very little code in place right now regarding Concepts, placed by Hubert Tong, CC'd, back in February, and very little to non-existent discussions in the mailing lists.
>
> As of the Toronto meeting back in July, a subset of the original Concepts TS was merged into draft C++20, which omits some shorthand notations for concepts but keeps the core parts of the language feature. Most notably - function concepts were removed, as well as the 'bool' keyword after the 'concept' keyword, and the 'introducer' and "terse/natural syntax" were omitted.
>
> I believe we should **go ahead and implement P0734R0 now** for the following reasons:
> - P0734R0 hasn't yet been implemented in any compiler (excluding GCC which implements a different proposal), and furthermore **no Concepts implementation was made from specification** - Andrew Sutton who worked on the GCC implementation also wrote the proposal. Doing a concepts implementation now would, I believe, help the committee discover defects and in general be more confident in the proposal.
> - to be honest, it is **really not that hard** to implement, and can, I believe, be done in a month or two.
> - Concepts will be present in C++2a unless something extraordinary happens - if not in their current form, then in a very similar form with maybe a few more terse syntaxes introduced **which would not break existing code** but just allow for nicer code - in any case, I believe it is a good time to lay a foundation for the core of the language feature which seems pretty stable already and were agreed upon by the committee.
>
> I took a look at the relevant code and the proposal and hereby propose **a roadmap for implementing** Concepts in clang as it stands today: https://github.com/saarraz/clang-concepts-roadmap
> I broke it up into commit-sized chunks, which should take us to a working implementation of the proposal. I'm of course willing to implement all of this if needed.

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

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.

The Moment of Truth

- The most stressful email I've ever sent

Mail Delivery Subsystem to saar 11/18/17

Richard Smith to Saar, Clang, Hubert 11/18/17

On 17 November 2017 at 16:24, Saar Raz via cfe-dev <cfe-dev@lists.llvm.org> wrote:

> Hi all,
>
> As I've read and seen in Clang's code, a Concepts implementation in Clang hasn't really been pushed forward in the past year, with only very, very little code in place right now regarding Concepts, placed by Hubert Tong, CC'd, back in February, and very little to non-existent discussions in the mailing lists.

Hubert Tong to Richard, Saar, Changyu, Nathan, Clang 11/18/17

Hi Saar,

As Richard mentioned, the only reason for the delay has been an issue with finding people with available time to work on the project. I'm adding Changyu and Nathan to the CC because they've also been active in this area.

…

> **which would not break existing code** but just allow for nicer code - in any case, I believe it is a good time to lay a foundation for the core of the language feature which seems pretty stable already and were agreed upon by the committee.
>
> I took a look at the relevant code and the proposal and hereby propose **a roadmap for implementing** Concepts in clang as it stands today: https://github.com/saarraz/clang-concepts-roadmap
> I broke it up into commit-sized chunks, which should take us to a working implementation of the proposal. I'm of course willing to implement all of this if needed.

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

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.

The Moment of Truth

- The most stressful email I've ever sent

Mail Delivery Subsystem to saar 11/18/17

Richard Smith to Saar, Clang, Hubert 11/18/17

On 17 November 2017 at 16:24, Saar R

Hi all,

As I've read and seen in C… very little code in place right now regarding Concepts, placed by Hub…

Hubert Tong to Richard, 11/18/17

Hi Saar,

As Richard mentioned, the… Changyu and Nathan to the CC because they've also been activ…

…

Great Success!

…ore of the language feature which seems pretty stable already and were agreed upon…

I took a look at the relevant code and the prop… as it stands today: https://github.com/saarraz/clang-concepts-roadmap

I broke it up into commit-sized chunks, which should take… of course willing to implement all of this if needed.

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

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 do concepts and just needed somebody to do it!

Well That Escalated Quickly

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

Changyu Li to hubert.reinterpretcast, cfe-dev, nwilson20, richard, saar 11/19/17

Hi Saar,

I'd like to help you implement your road map. I've finished point 1 and am working on points 2 and 3. I may have some questions in the future, not right now though.

...

Well That Escalated Quickly

- I was preaching to the choir
- They wanted to [illegible] just needed somebody to do it!

Whoops!


4
MONTHS
LATER

Why Small Commits are Good

- Well turns out implementing (and testing) this properly takes longer than , and can, I believe, be done in a month or two.
- After 4 months, I reached the same point I had before I sent that email
  - But properly this time
  - A few people on reddit suggested that I get a version of that up on Compiler Explorer…

Why Small Commits are Good

Clang Concepts godbolt

Saar Raz to Matt 1/2/18

Yo
I'm working on the Clang implementation of concepts, and thought it might be nice to upload the WIP build online for people to play with.
What do you think?
Thanks :)

Matt Godbolt to Saar, compiler-explorer-development 1/2/18

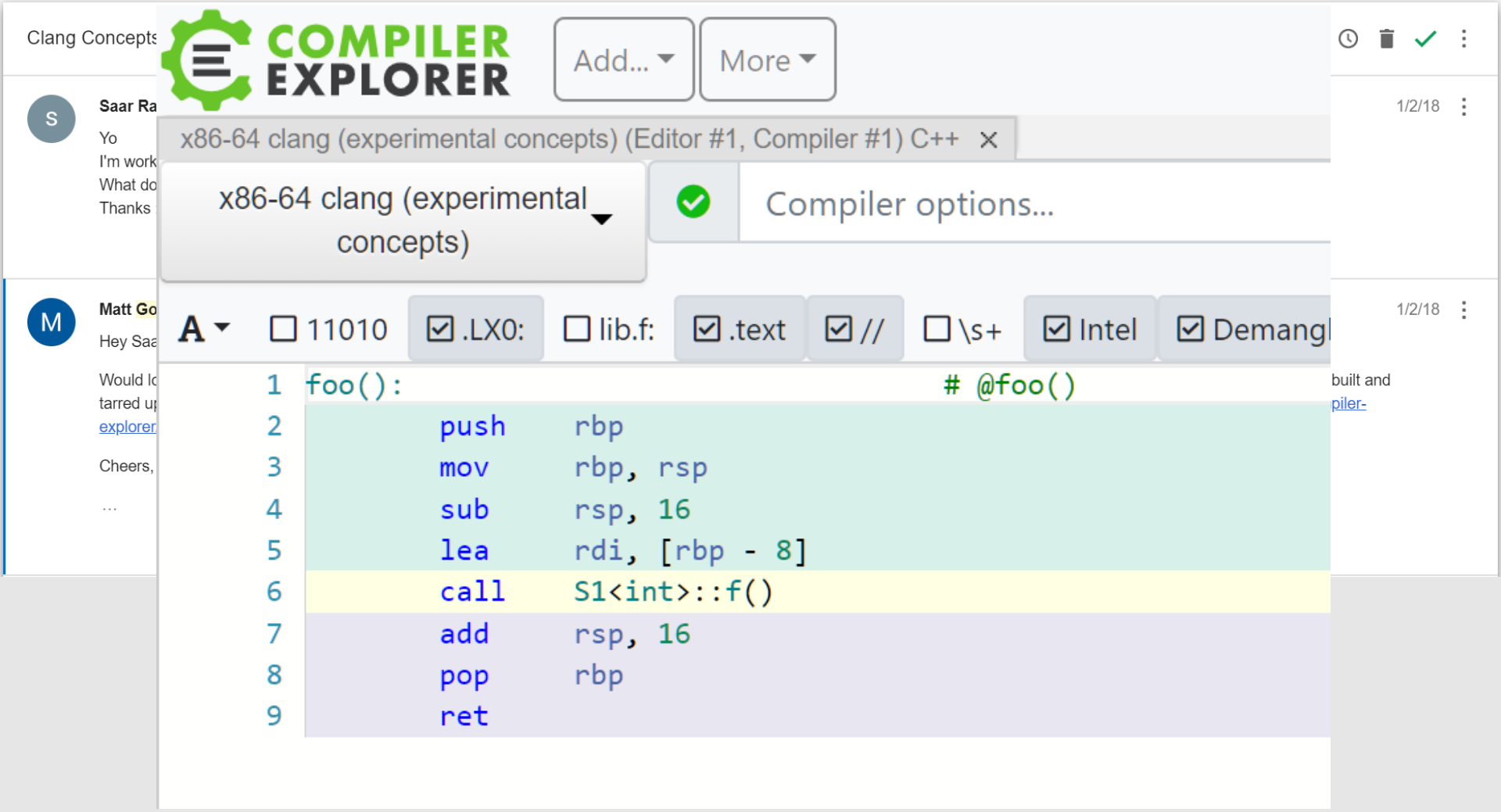
Hey Saar! And CCing the other devs

Would love to have the concepts branch around. Is there a GitHub for it? I build all the compilers from source using some scripts I've made along the way. Once we have a binary built and tarred up it's usually pretty straightforward to get into the list of compilers. There's even some docs on how to do that yourself if you feel brave! https://github.com/mattgodbolt/compiler-explorer/blob/master/docs/AddingACompiler.md

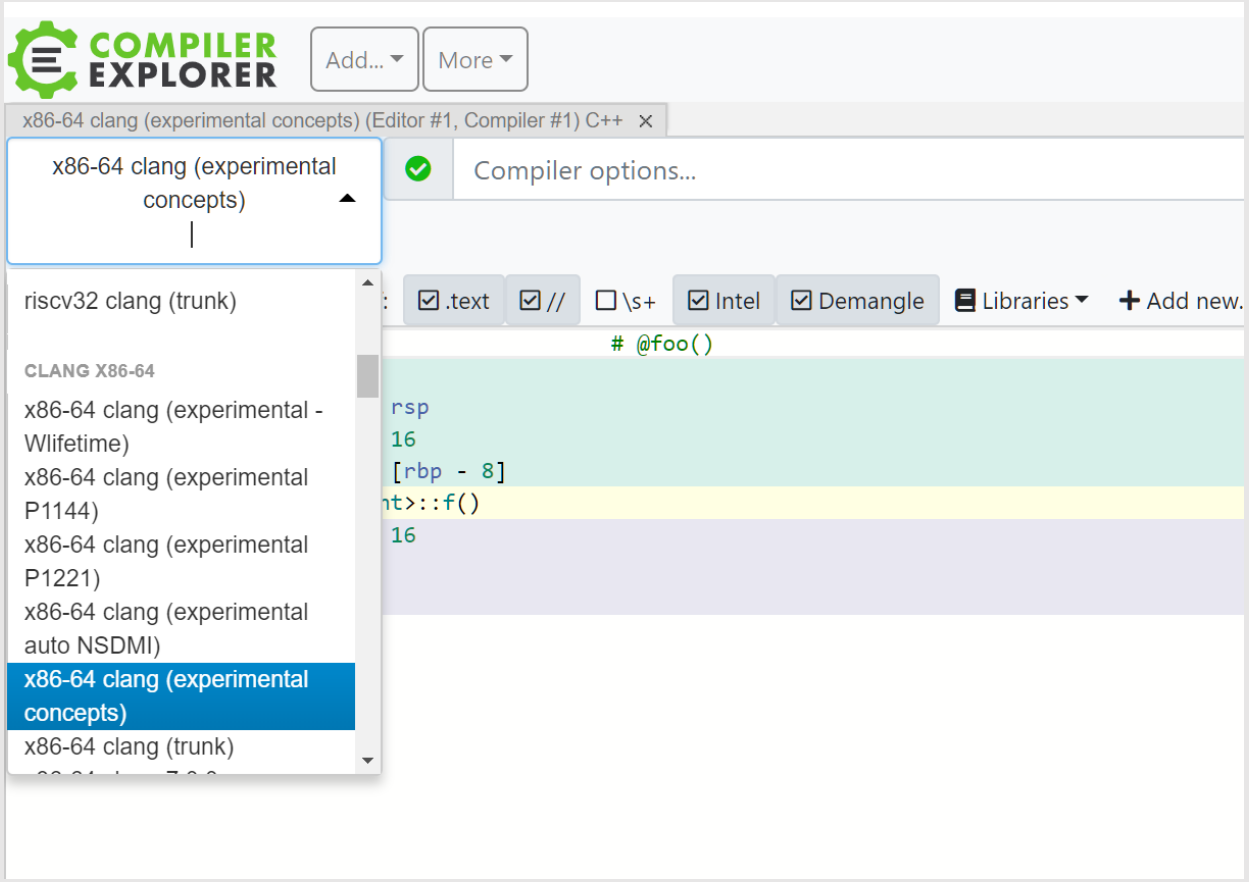
Cheers, Matt

…

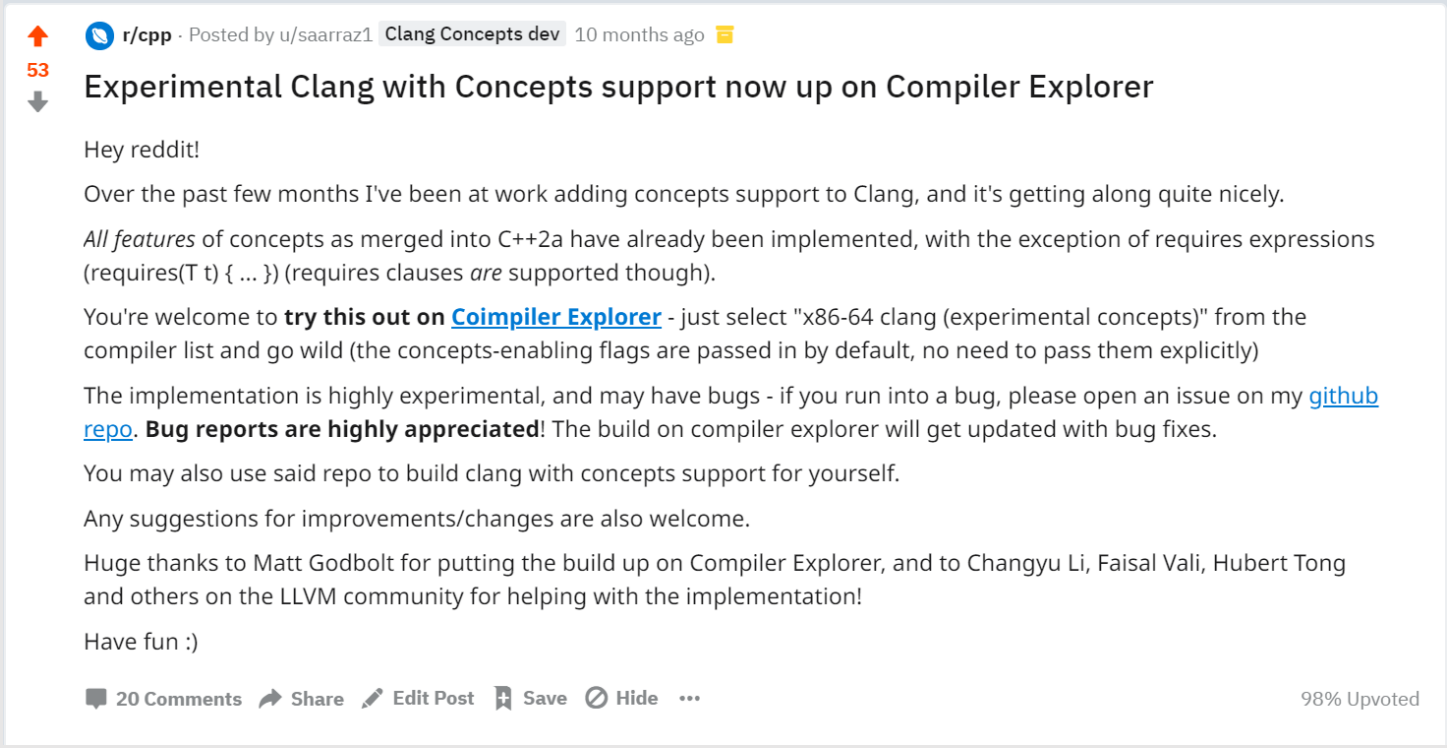
Why Small Commits are Good



Then Others Learned the Trick As Well...



Let the world know!



r/cpp · Posted by u/saarraz1 Clang Concepts dev 10 months ago

53

Experimental Clang with Concepts support now up on Compiler Explorer

Hey reddit!

Over the past few months I've been at work adding concepts support to Clang, and it's getting along quite nicely.

All features of concepts as merged into C++2a have already been implemented, with the exception of requires expressions (requires(T t) { ... }) (requires clauses *are* supported though).

You're welcome to **try this out on Coimpiler Explorer** - just select "x86-64 clang (experimental concepts)" from the compiler list and go wild (the concepts-enabling flags are passed in by default, no need to pass them explicitly)

The implementation is highly experimental, and may have bugs - if you run into a bug, please open an issue on my github repo. **Bug reports are highly appreciated**! The build on compiler explorer will get updated with bug fixes.

You may also use said repo to build clang with concepts support for yourself.

Any suggestions for improvements/changes are also welcome.

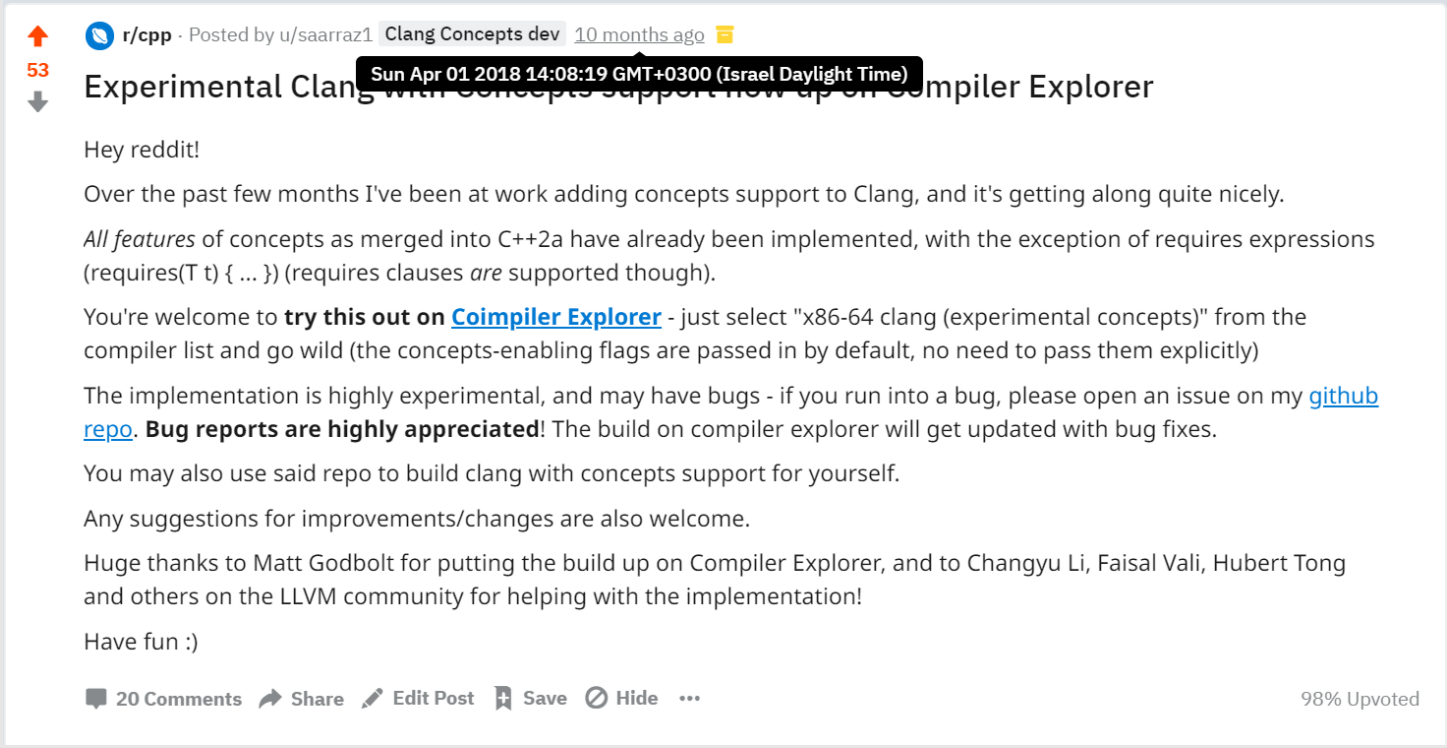
Huge thanks to Matt Godbolt for putting the build up on Compiler Explorer, and to Changyu Li, Faisal Vali, Hubert Tong and others on the LLVM community for helping with the implementation!

Have fun :)

20 Comments Share Edit Post Save Hide ••• 98% Upvoted

- See the bug?

Let the world know!



53

r/cpp · Posted by u/saarraz1 Clang Concepts dev 10 months ago

Sun Apr 01 2018 14:08:19 GMT+0300 (Israel Daylight Time)

Experimental Clang with Concepts support now up on Compiler Explorer

Hey reddit!

Over the past few months I've been at work adding concepts support to Clang, and it's getting along quite nicely.

All features of concepts as merged into C++2a have already been implemented, with the exception of requires expressions (requires(T t) { ... }) (requires clauses *are* supported though).

You're welcome to **try this out on Coimpiler Explorer** - just select "x86-64 clang (experimental concepts)" from the compiler list and go wild (the concepts-enabling flags are passed in by default, no need to pass them explicitly)

The implementation is highly experimental, and may have bugs - if you run into a bug, please open an issue on my github repo. **Bug reports are highly appreciated**! The build on compiler explorer will get updated with bug fixes.

You may also use said repo to build clang with concepts support for yourself.

Any suggestions for improvements/changes are also welcome.

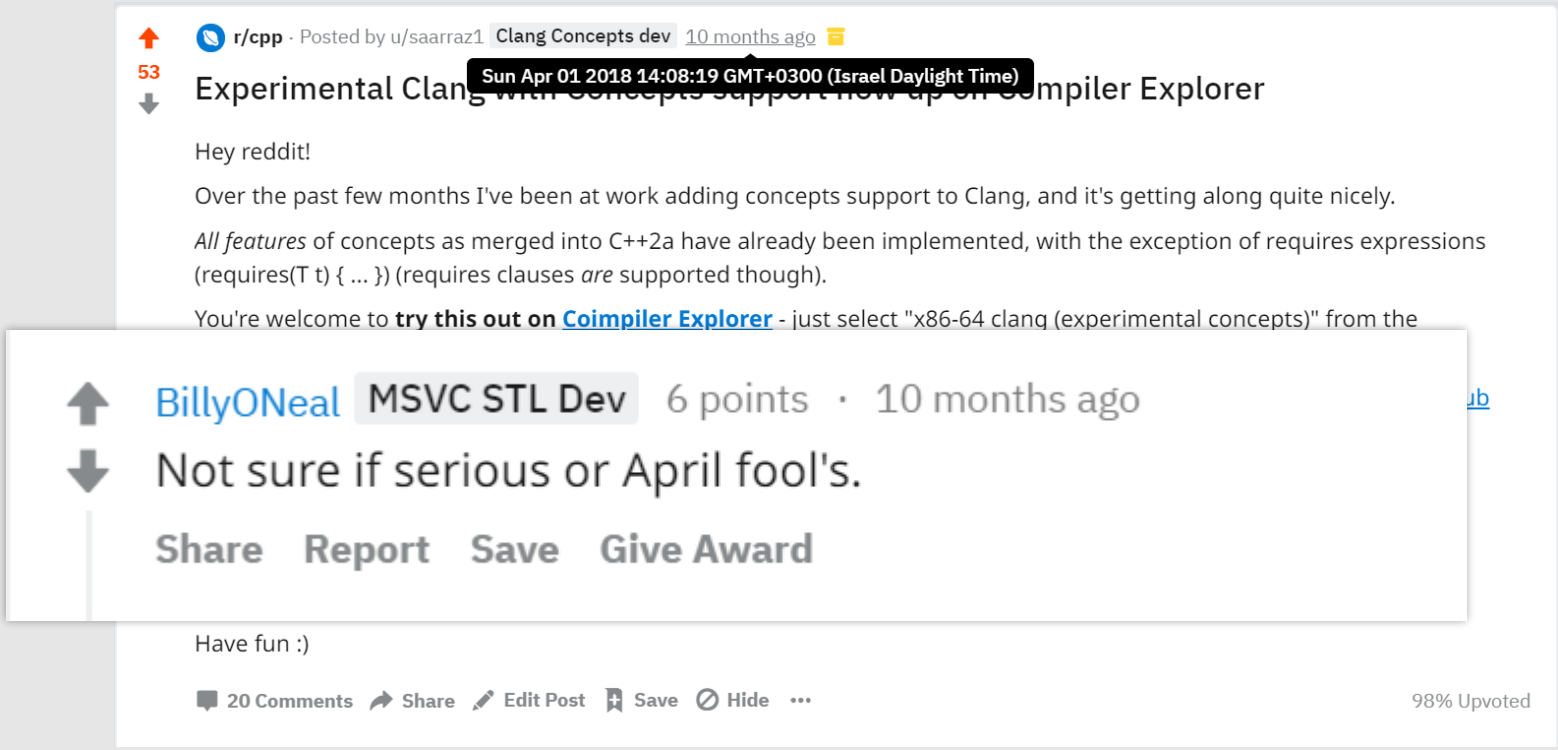
Huge thanks to Matt Godbolt for putting the build up on Compiler Explorer, and to Changyu Li, Faisal Vali, Hubert Tong and others on the LLVM community for helping with the implementation!

Have fun :)

20 Comments Share Edit Post Save Hide ••• 98% Upvoted

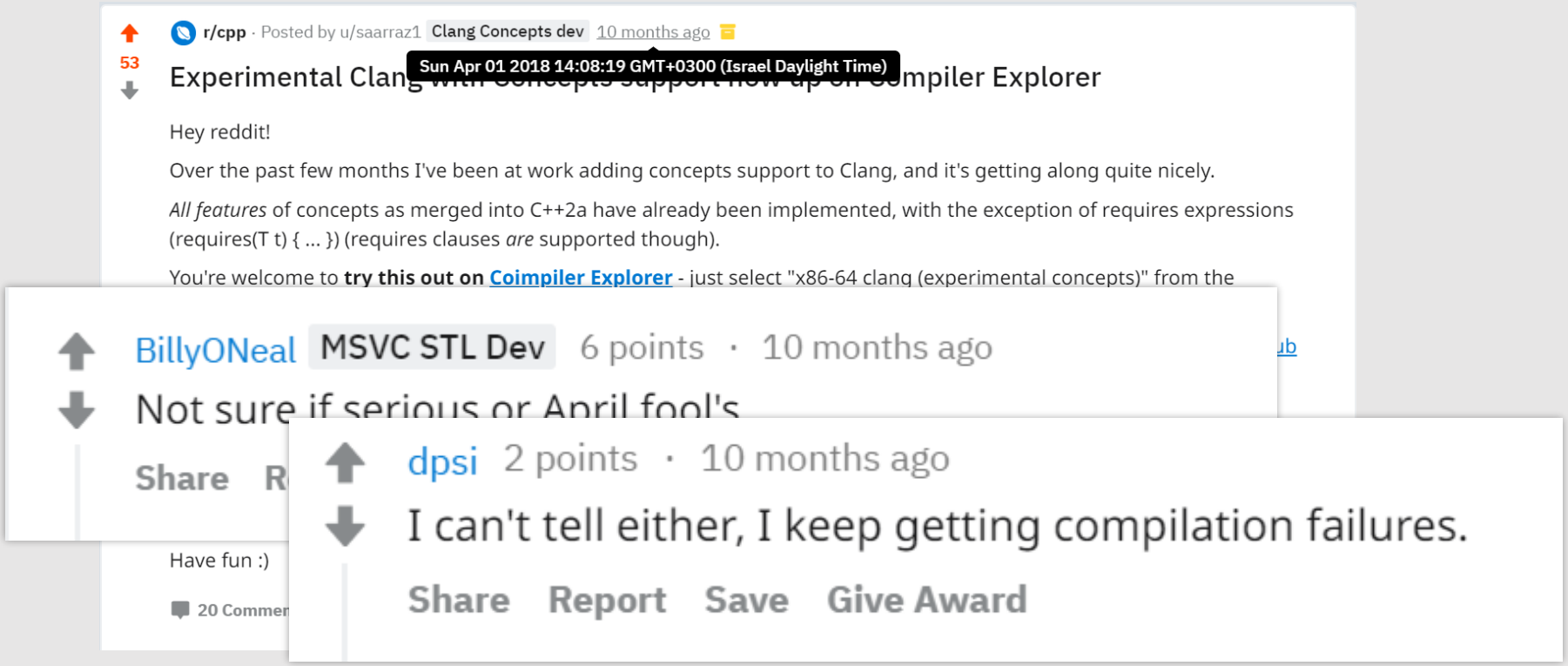
- See the bug?
- How about now?

Let the world know!



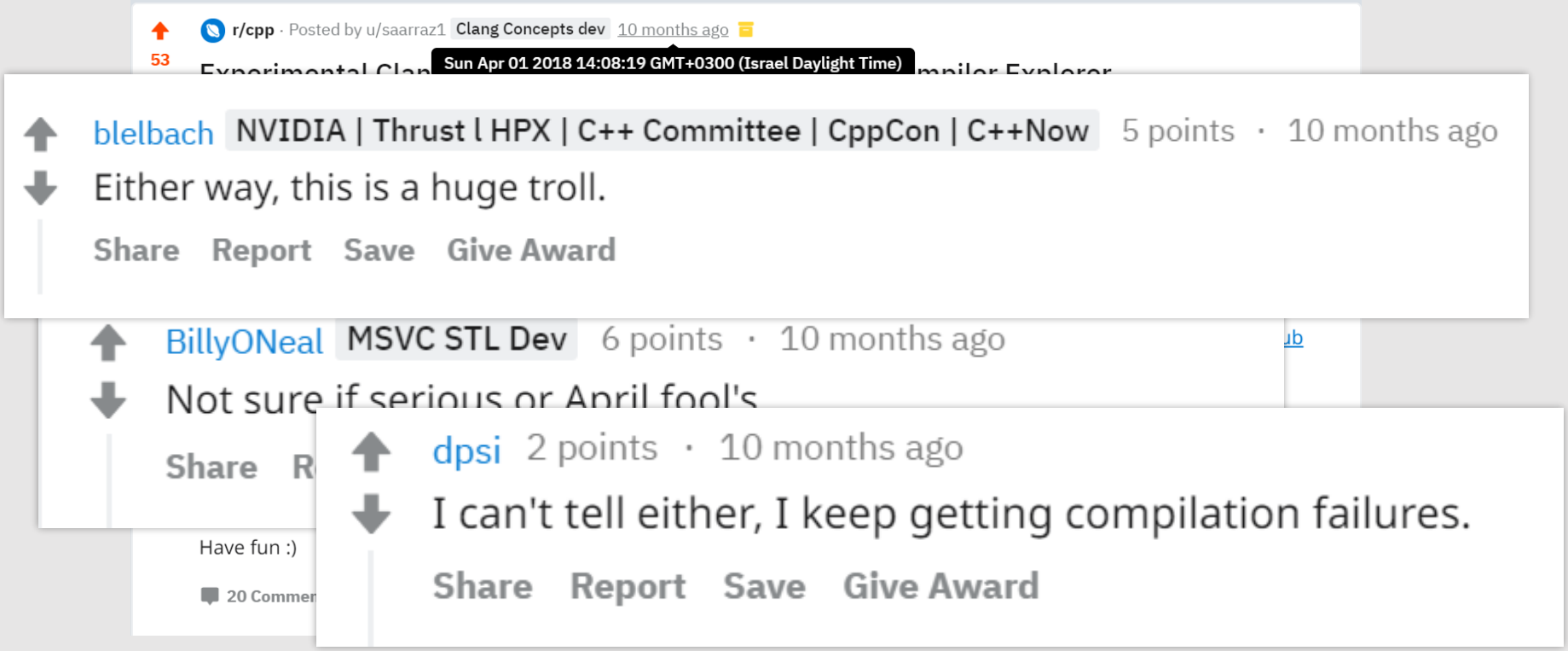
- See the bug?
- How about now?

Let the world know!



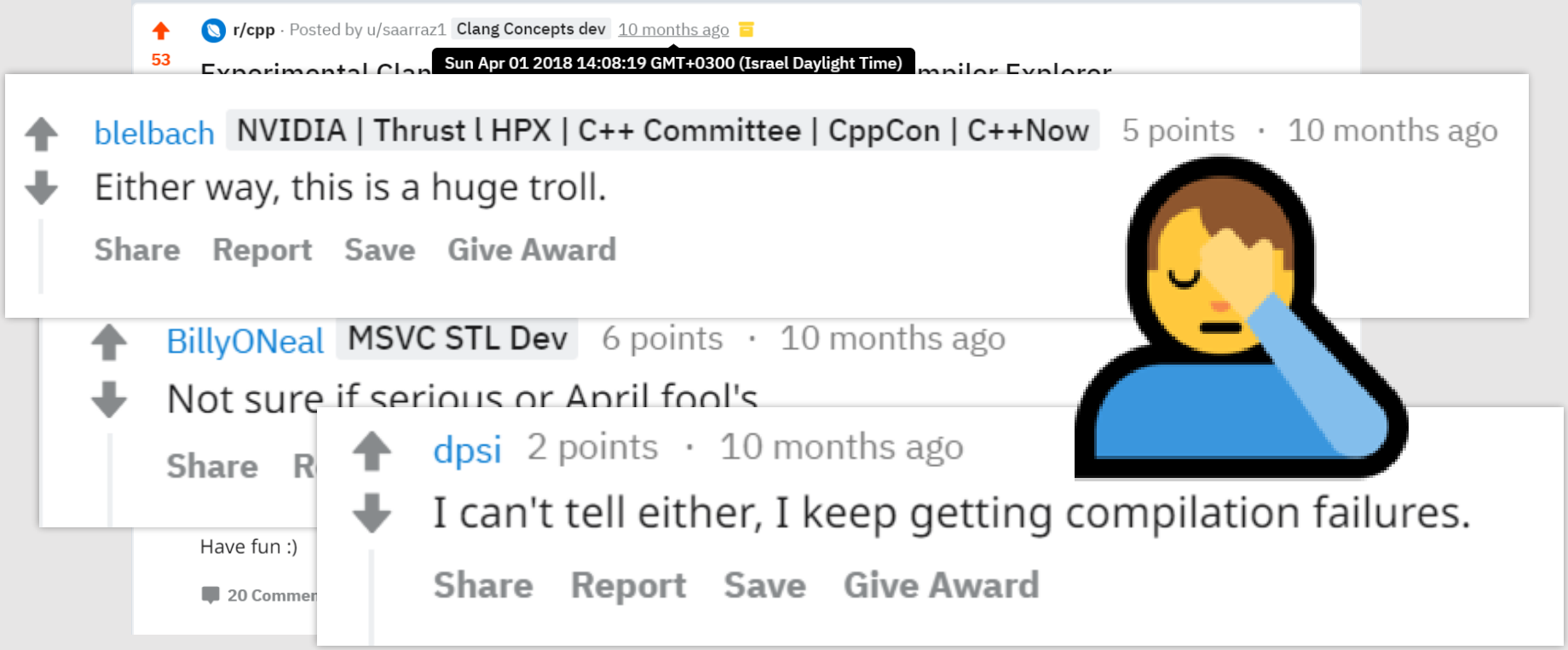
- See the bug?
- How about now?

Let the world know!



- See the bug?
- How about now?

Let the world know!



- 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;
}
```

Almost done!

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

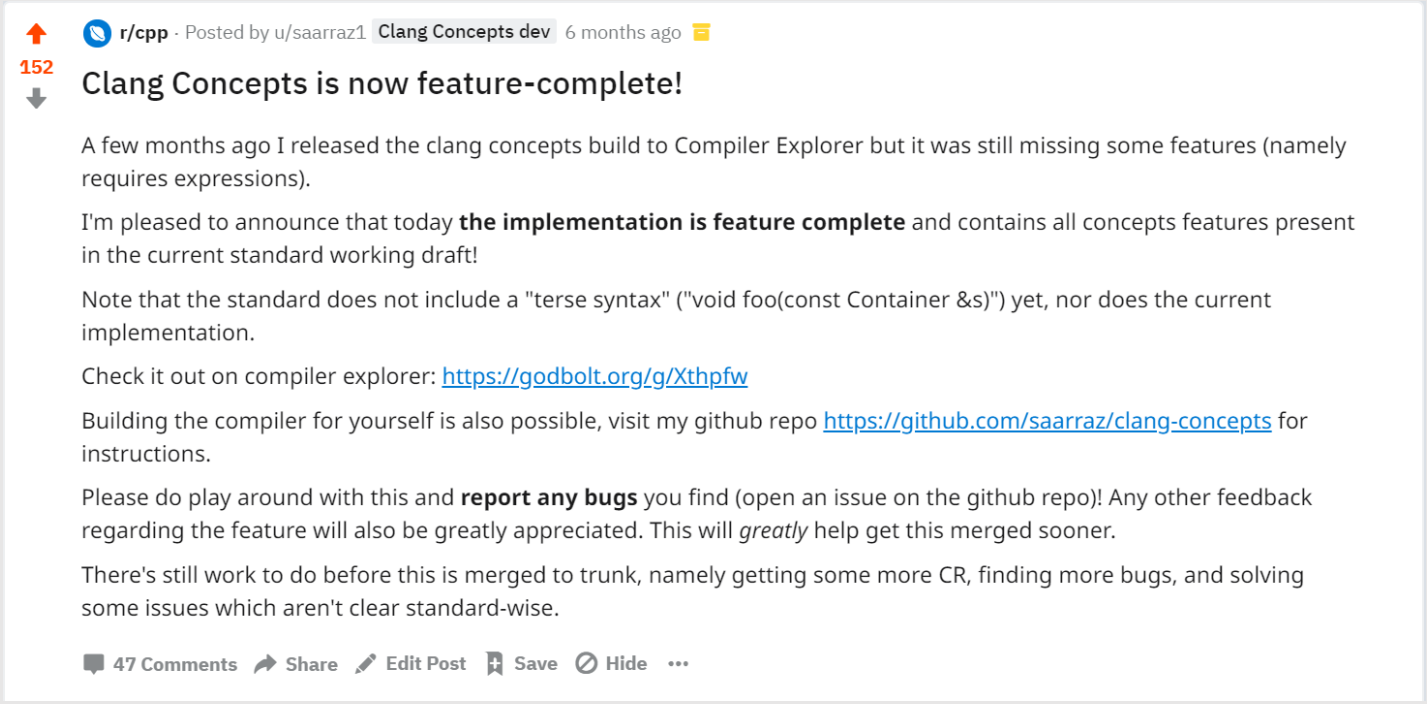
```
requires (T t) {
    typename T::foo_result;
    { t.foo(); } -> typename T::foo_result;
    t++;
    requires T::is_ok;
}
```

- 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!



152

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

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 (and ranges) 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 wee… International Standard (IS), C++20. This meeting was the last m… or C++20, but existing proposals like modules (on track) and coroutines (q… but not merged can still make C++20. We'll make our final decisions abo… next meeting.

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

- Ranges.
- `void foo(Concept auto x)` -style terse syntax.
- `consteval` functions.
- `std::is_constant_evaluated`.
- `constexpr union`.

Current Status

- No new bugs have surfaced in a while (after fixing over 50!)
- Working towards merging this into trunk
- First commit is approved but not committe

Revision Contents

Files | History | Commits | **Stack (7 Open)** | Similar

Status	Author	Revision
</> Needs Review	saar.raz	**D50360 [Concepts] Requires Expressions**
</> Needs Review	saar.raz	**D44352 [Concepts] Constrained template parameters**
</> Needs Review	saar.raz	**D43357 [Concepts] Function trailing requires clauses**
</> Needs Review	saar.raz	**D41910 [Concepts] Constrained partial specializations and function overloads.**
</> Needs Review	saar.raz	**D41569 [Concepts] Constraint enforcement and diagnostics**
</> Needs Review	saar.raz	**D41284 [Concepts] Associated constraints infrastructure.**
</> Needs Review	saar.raz	**D41217 [Concepts] Concept Specialization Expressions**
</> Needs Review	changyu	**D40381 Parse concept definition**

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>`

Saar Raz • saar@raz.email