$C{++}{++}{++}{++}{++}+\cdots$

Saar Raz • saar@raz.email

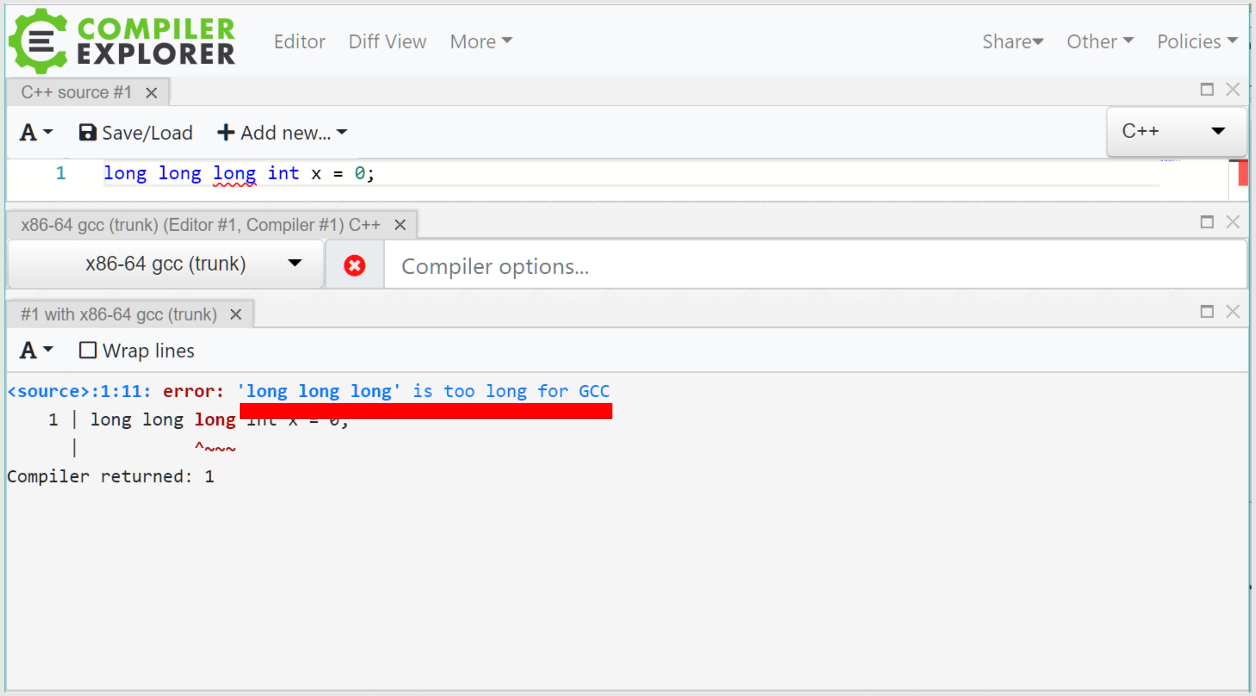
Riddle (props to Ben Deane)

- Write a C++ program with the longest sequence of keywords

Riddle (props to Ben Deane)

- Write a C++ program with the longest sequence of keywords
- Easy:

```
long long long long long long long … int x = 0;
```



```
long long long int x = 0;
```

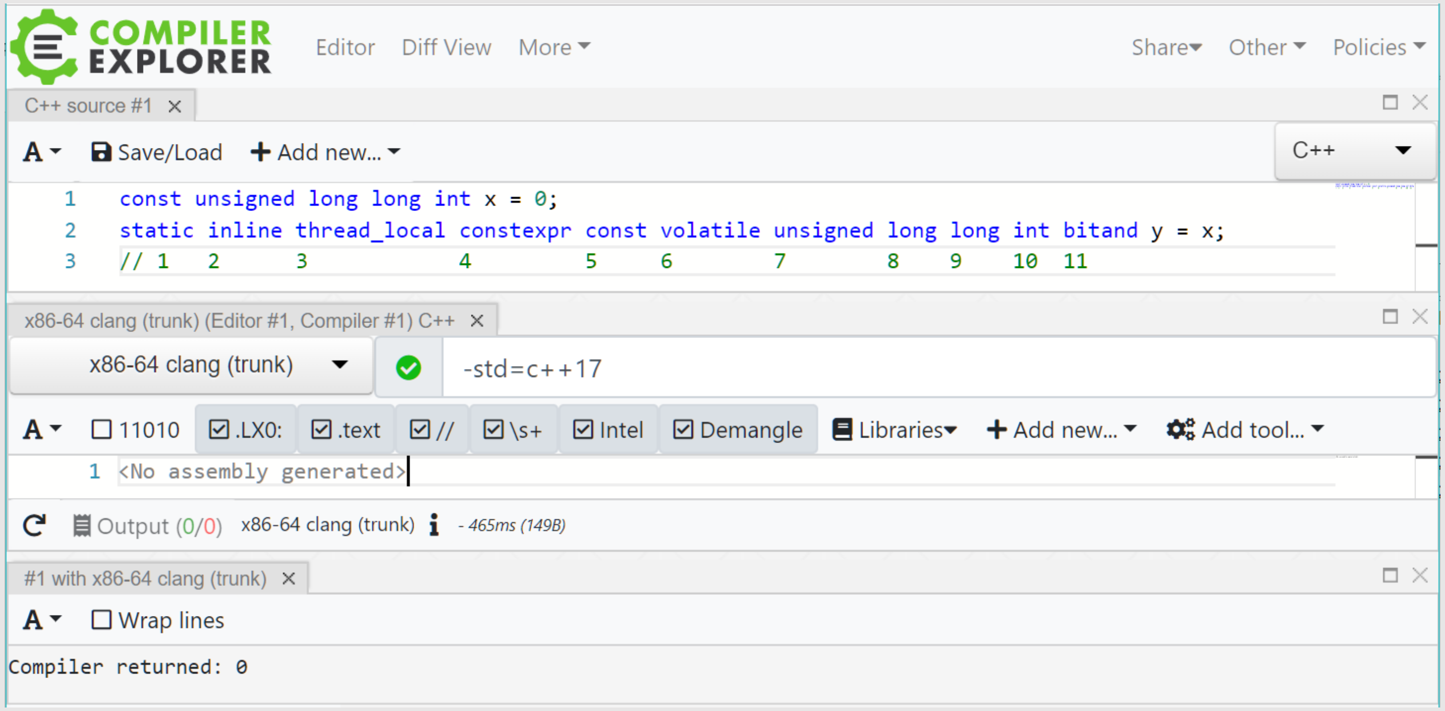
```
<source>:1:11: error: 'long long long' is too long for GCC
    1 | long long long int x = 0;
      |           ^~~~
Compiler returned: 1
```

Aw, Snap!

Riddle

- Solution (11 keywords)

```
static inline thread_local constexpr const volatile unsigned long long int bitand
```



```
const unsigned long long int x = 0;
static inline thread_local constexpr const volatile unsigned long long int bitand y = x;
// 1    2       3           4         5     6        7        8    9    10  11
```

x86-64 clang (trunk) (Editor #1, Compiler #1) C++

x86-64 clang (trunk)   -std=c++17

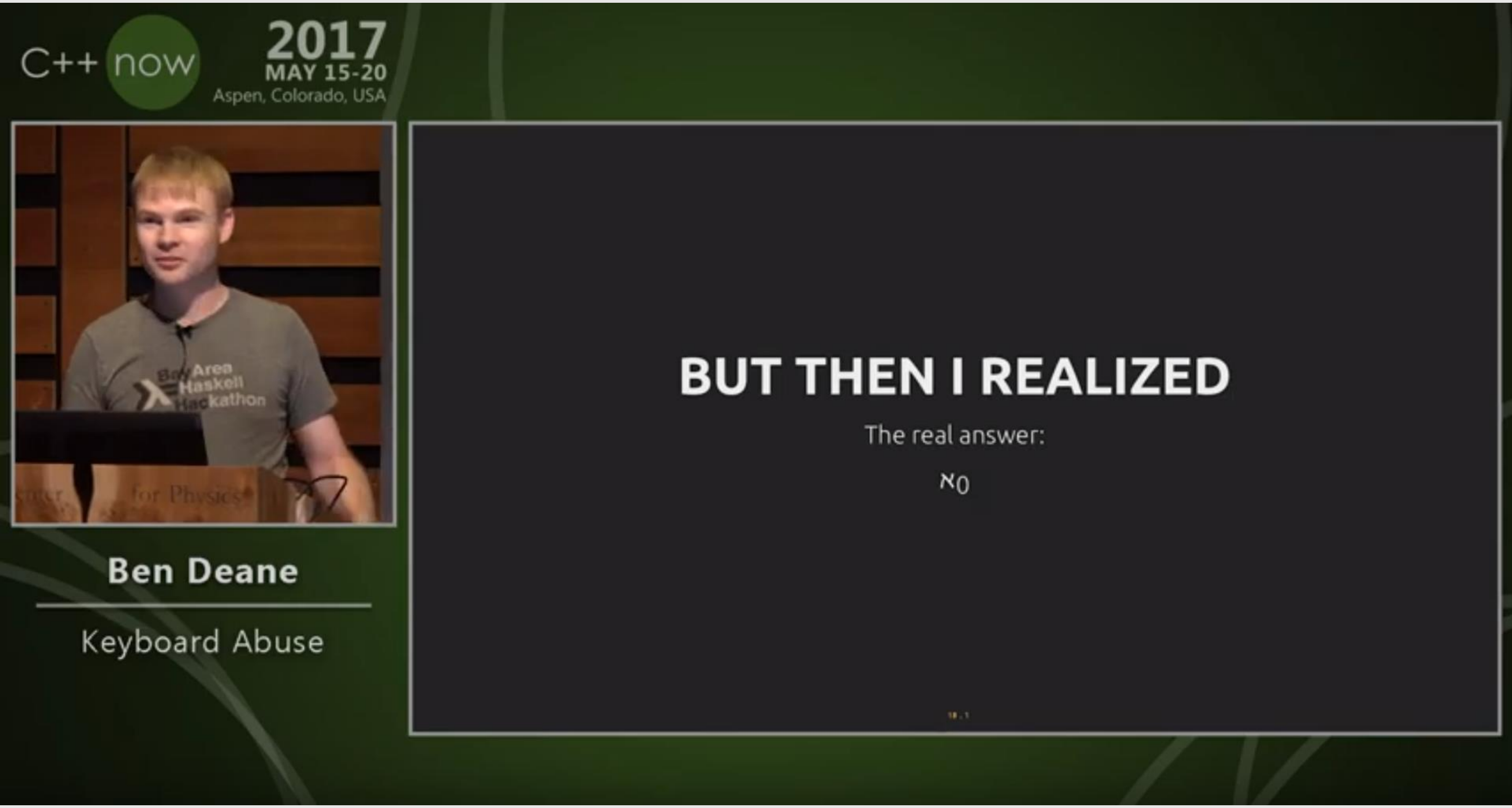
```
<No assembly generated>
```

Output (0/0) x86-64 clang (trunk) - 465ms (149B)

#1 with x86-64 clang (trunk)

```
Compiler returned: 0
```

But wait!



BUT THEN I REALIZED

The real answer:

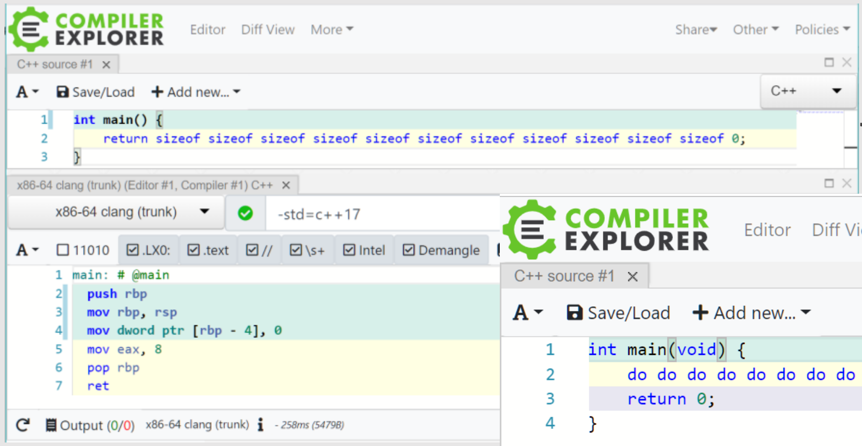
$\aleph_0$

The Real Solution

- In C as well
- An unbounded amount of keywords:

The Real Solution

words:

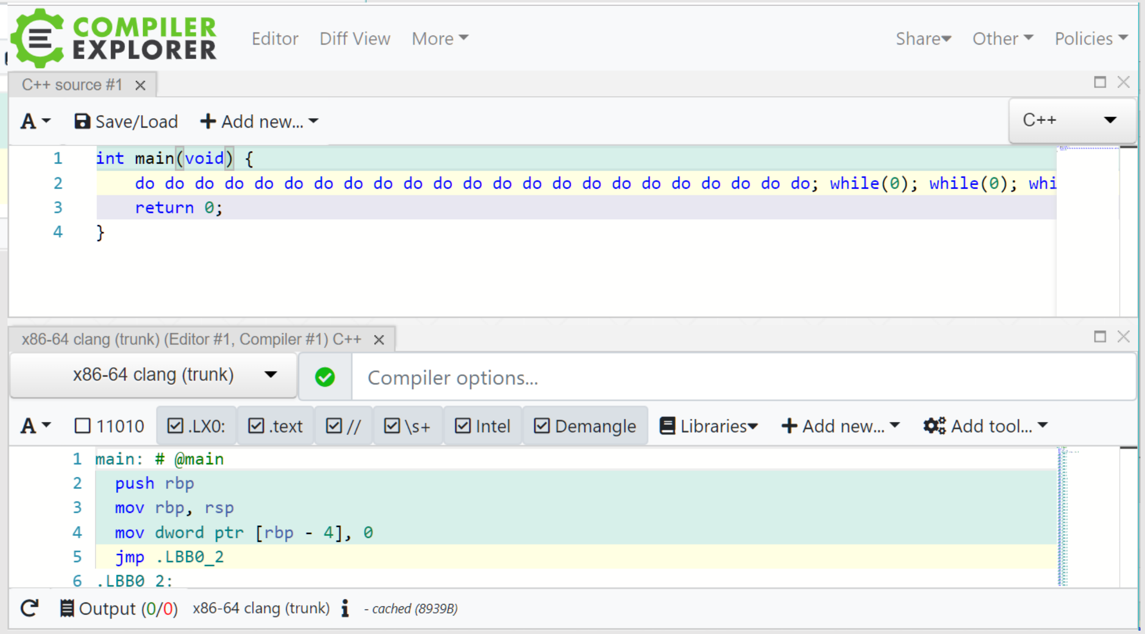


```
int main() {
    return sizeof sizeof sizeof sizeof sizeof sizeof sizeof sizeof sizeof sizeof sizeof 0;
}
```

x86-64 clang (trunk) -std=c++17

```
main: # @main
  push rbp
  mov rbp, rsp
  mov dword ptr [rbp - 4], 0
  mov eax, 8
  pop rbp
  ret
```

Output (0/0) x86-64 clang (trunk) - 258ms (5479B)



```
int main(void) {
    do do do do do do do do do do do do do do do do do do do do do do do; while(0); while(0); whi
    return 0;
}
```

x86-64 clang (trunk) Compiler options...

```
main: # @main
  push rbp
  mov rbp, rsp
  mov dword ptr [rbp - 4], 0
  jmp .LBB0_2
.LBB0_2:
```

Output (0/0) x86-64 clang (trunk) - cached (8939B)

Is there a better one?

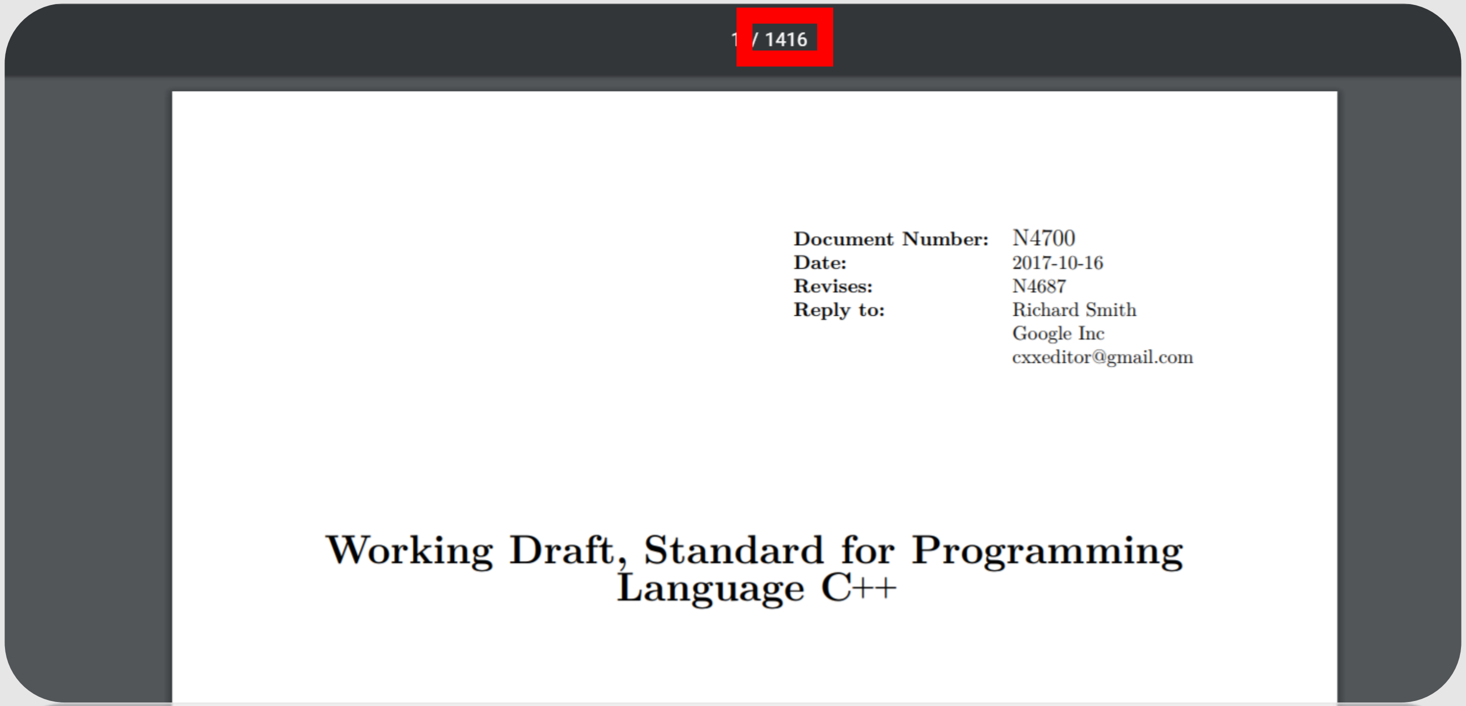
- We already have an unbounded no. of keywords
- What about an **infinite** amount?

The Question

- Does an infinite C++ program exist?
  - **Does**: We're gonna try and answer this
  - **An**: One is enough
  - **Infinite**: That contains an infinite number of characters.
  - **C++**: A well-formed program, as defined by the standard
  - **Program**: What *is* a program anyway?
  - **Exist**: In the theoretical sense

C++

- C++ is an abstract term, but is (mostly) well-defined:
    - **ISO/IEC 14882:2017**: Standard for Programming Language C++



1 / 1416

Document Number:	N4700
Date:	2017-10-16
Revises:	N4687
Reply to:	Richard Smith
	Google Inc
	cxxeditor@gmail.com

Working Draft, Standard for Programming Language C++

C++

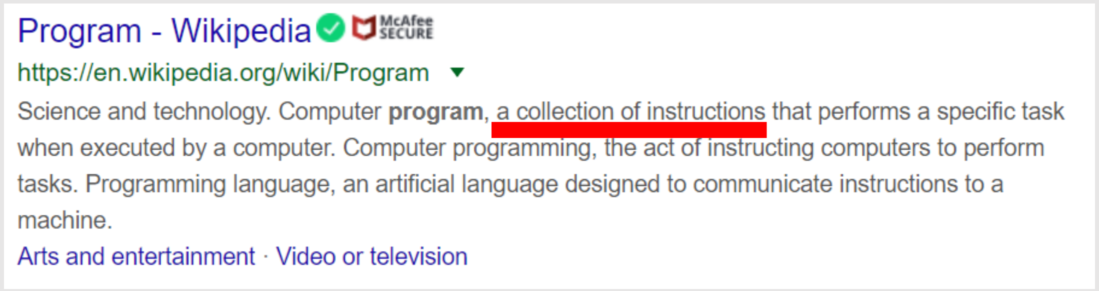
- C++ is an abstract term, but is (mostly) well-defined:
  - **ISO/IEC 14882:2017**: Standard for Programming Language C++

> **§[intro.scope]/1**
>
> This document specifies requirements for implementations of the C++ programming language. The first such requirement is that they implement the language, **so this document also defines C++**. Other requirements and relaxations of the first requirement appear at various places within this document.

  - We don’t care about implementations, just if it’s part of ‘C++’

Program

- Can a program be infinite?
- What is the definition of a program?



Program - Wikipedia McAfee SECURE

https://en.wikipedia.org/wiki/Program ▾

Science and technology. Computer **program**, a collection of instructions that performs a specific task when executed by a computer. Computer programming, the act of instructing computers to perform tasks. Programming language, an artificial language designed to communicate instructions to a machine.

Arts and entertainment · Video or television

- Somewhat unclear…

Wikipedia can’t be trusted

- That’s what we have a standard for!

What does the standard have to say?

§[defns.well.formed]

C++ **program** constructed according to the syntax rules, diagnosable semantic rules, and the one-definition rule

- But what is a program!?!?!?!?

§[intro.refs]/1

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document.

— [...]

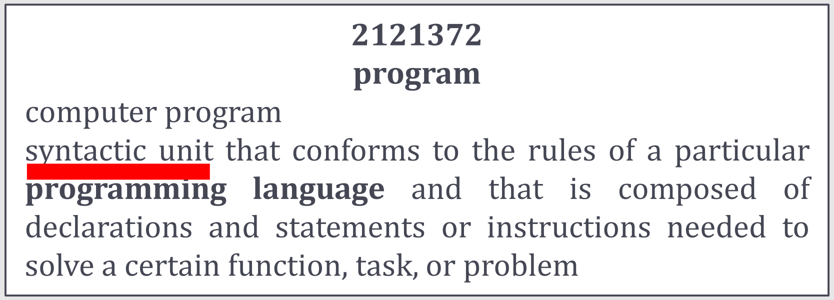
— [...]

— ISO/IEC 2382 (all parts), Information technology — **Vocabulary**

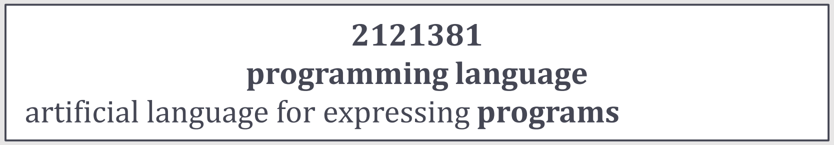
— [...]

- Awesome! There's a standard for CS Vocabulary!

ISO/IEC 2382



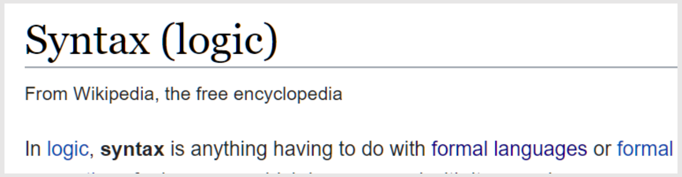
2121372
program
computer program
syntactic unit that conforms to the rules of a particular **programming language** and that is composed of declarations and statements or instructions needed to solve a certain function, task, or problem



2121381
programming language
artificial language for expressing **programs**

- Neither ‘syntactic unit’ nor ‘syntax’ is defined...

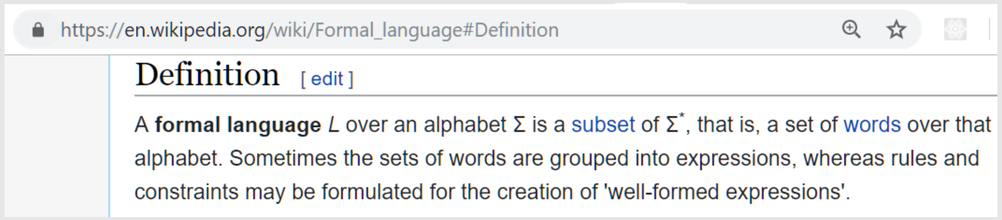
Syntactic Unit?



Syntax (logic)

From Wikipedia, the free encyclopedia

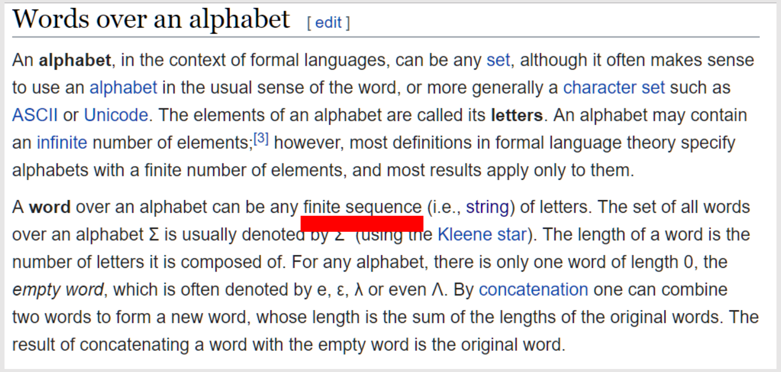
In logic, **syntax** is anything having to do with formal languages or formal



https://en.wikipedia.org/wiki/Formal_language#Definition

Definition [edit]

A **formal language** *L* over an alphabet Σ is a subset of Σ^*, that is, a set of words over that alphabet. Sometimes the sets of words are grouped into expressions, whereas rules and constraints may be formulated for the creation of 'well-formed expressions'.



Words over an alphabet [edit]

An **alphabet**, in the context of formal languages, can be any set, although it often makes sense to use an alphabet in the usual sense of the word, or more generally a character set such as ASCII or Unicode. The elements of an alphabet are called its **letters**. An alphabet may contain an infinite number of elements;[3] however, most definitions in formal language theory specify alphabets with a finite number of elements, and most results apply only to them.

A **word** over an alphabet can be any finite sequence (i.e., string) of letters. The set of all words over an alphabet Σ is usually denoted by Σ^* (using the Kleene star). The length of a word is the number of letters it is composed of. For any alphabet, there is only one word of length 0, the *empty word*, which is often denoted by e, ε, λ or even Λ. By concatenation one can combine two words to form a new word, whose length is the sum of the lengths of the original words. The result of concatenating a word with the empty word is the original word.

Wikipedia can’t be trusted

- Wikipedia’s no standard!
- What does the standard have to say?

WHAT IS A PROGRAM?!?!

- Hey!

> **§[basic.link]/1**
>
> A *program* consists of $\underline{\text{one or more}}$ translation units ([lex]) linked together. [...]

$\aleph_0 > 1!$

- OK, what's a translation unit?

> **§[basic.link]/1**
>
> [...] A translation unit consists of a sequence of declarations.
>
> translation-unit:
>
> declaration-seq$_{opt}$
>
> declaration-seq:
>
> declaration
>
> declaration-seq$_{opt}$ declaration

- It seems the number of declarations must be bounded...
- So each translation unit is finite

Infinite translation units?

- The translation units have to be “linked together”…
- There’s no mention of a linking phase!

§[lex.phases]/3

[…] Implementations must **behave as if** these separate phases occur, although in practice different phases might be folded together.

- Profit!
- But this is less interesting…

Parsing

§[lex.separate]

The text of the program is kept in units called **source files** in this document. A source file together with all the headers and source files included via the preprocessing directive #include, less any source lines skipped by any of the conditional inclusion preprocessing directives, is called a translation unit. [Note: A C++ program need not all be translated at the same time. — end note]

- Damn – source **files** – and files are finite…
- **Are they though?**

§[fs.general]/3

A *file* is an object within a file system that holds user or system data. Files can be written to, or read from, or both. A file has certain attributes, including type. File types include […].

§[fs.general]/2

A *file system* is a collection of files and their attributes.

Parsing

§[lex.phases]/3

The **precedence** among the syntax rules of translation is specified by the following phases.6

1.**Physical source file characters are mapped**, in an implementation-defined manner, to the basic source character set (introducing new-line characters for end-of-line indicators) if necessary. The set of physical source file characters accepted is implementation-defined. […].

2.[…] A source file that is not empty and **that does not end in a new-line character**, or that ends in a new-line character immediately preceded by a backslash character before any such splicing takes place, shall be processed as if an additional new-line character were appended to the file.

3.The source file is decomposed into preprocessing tokens and sequences of white-space characters […]

4.Preprocessing directives are executed, […].

5. […]

6.Adjacent string literal tokens are concatenated.

7.White-space characters separating tokens are no longer significant. **Each preprocessing token is converted into a token. The resulting tokens are syntactically** and semantically analyzed and translated as a translation unit. […]

- All tokens are defined using grammar rules…

Insignificant Whitespace

§[lex.token]/3

There are five kinds of tokens: identifiers, keywords, literals, operators, and other separators. **Blanks, horizontal and vertical tabs, newlines, formfeeds, and comments (collectively, “white space”), as described below, are ignored except as they serve to separate tokens.** [Note: Some white space is required to separate otherwise adjacent identifiers, keywords, numeric literals, and alternative tokens containing alphabetic characters. — end note]

- So you can theoretically have infinite whitespace!
- Why is this interesting?
- Ordinals!

Ordinals

- Generalization of the natural numbers
- “What’s the index of the element after all natural numbers?”

$$\underbrace{1,2,3,4,5,\ldots}_{\aleph_0},\underbrace{a}_{\omega}$$

- These are called **transfinite sequences** – are they legal C++?

Ordinals

- Generalization of the natural numbers
- “What’s the index of the element after all natural numbers?”

$$\underbrace{1,2,3,4,5,\ldots}_{\aleph_0},\underbrace{a}_{\omega}$$

- These are called **transfinite sequences** – are they legal C++?
- If so, you can have interesting stuff:

$$\texttt{int main() \{}\underbrace{\texttt{\textbackslash n\textbackslash n\textbackslash n\textbackslash n}\ldots}_{\aleph_0}\quad\underbrace{\texttt{s}}_{char\ \#\omega}\quad\texttt{td::cout << \_\_LINE\_\_; \}}$$

Ordinals

- Generalization of the natural numbers
- “What’s the index of the element after all natural numbers?”

$$\underbrace{1,2,3,4,5,\ldots}_{\aleph_0},\underbrace{a}_{\omega}$$

- These are called **transfinite sequences** – are they legal C++?
- If so, you can have interesting stuff:

$$\texttt{int main() \{}\underbrace{\texttt{\textbackslash n\textbackslash n\textbackslash n\textbackslash n}\ldots}_{\aleph_0}\quad\underbrace{\texttt{s}}_{char\ \#\omega}\quad\texttt{td::cout << \_\_LINE\_\_; \}}$$

- But!

__LINE__

§[cpp.predefined]/1/4

—__LINE__

The presumed line number (within the current source file) of the current source line (an integer literal).

- an integer literal…

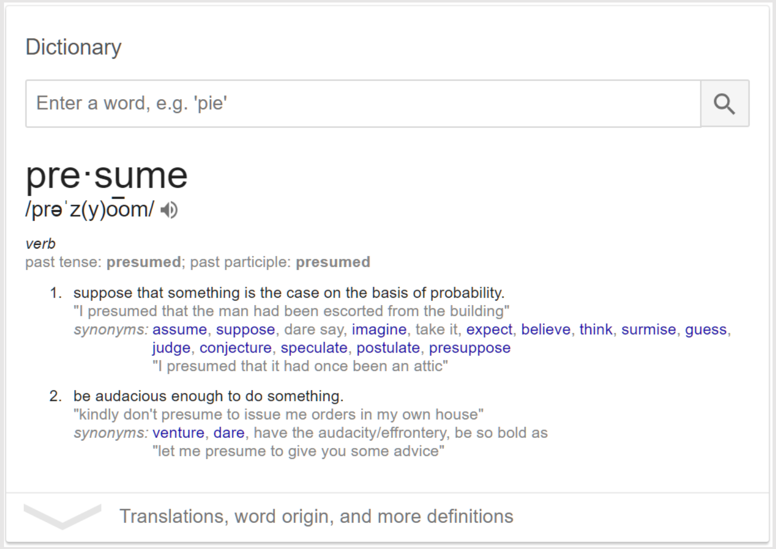
§[lex.icon]

integer-literal:
    binary-literal integer-suffix$_{opt}$
    octal-literal integer-suffix$_{opt}$
    decimal-literal integer-suffix$_{opt}$
    hexadecimal-literal integer-suffix$_{opt}$

- So, an integer literal can’t represent an infinite ordinal…
- What if it was a float literal?
  - Still no, as there is no float literal for infinity

Presumed line number?

- Can we presume the line number is 0?



Dictionary

Enter a word, e.g. 'pie'

pre·sume
/prəˈz(y)o͞om/

verb
past tense: **presumed**; past participle: **presumed**

1. suppose that something is the case on the basis of probability.
   "I presumed that the man had been escorted from the building"
   synonyms: assume, suppose, dare say, imagine, take it, expect, believe, think, surmise, guess, judge, conjecture, speculate, postulate, presuppose
   "I presumed that it had once been an attic"
2. be audacious enough to do something.
   "kindly don't presume to issue me orders in my own house"
   synonyms: venture, dare, have the audacity/effrontery, be so bold as
   "let me presume to give you some advice"

Translations, word origin, and more definitions

- The probability that the line number is 0 is exactly 0

Presumed line number!

§[cpp.line]

[...]

2 The line number of the current source line is one greater than the number of new-line characters read or introduced in translation phase 1 while processing the source file to the current token.

3 A preprocessing directive of the form

#line *digit-sequence new-line*

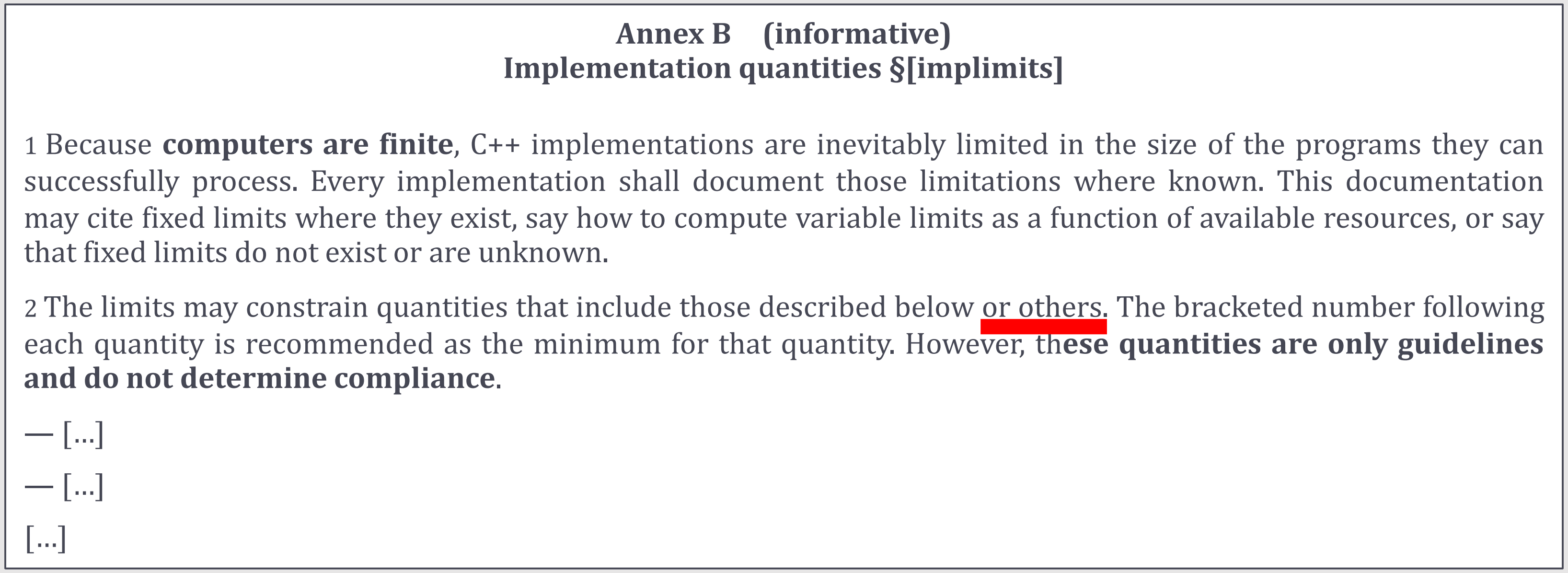
causes the implementation to behave as if the following sequence of source lines begins with a source line that has a line number as specified by the digit sequence (interpreted as a decimal integer). If the digit sequence specifies zero or a number greater than 2147483647, the behavior is undefined.
[...]

- So, theoretically, you can have:

`int main() { \n\n\n\n … \n\n\n\n …` $\aleph 0$ $\underbrace{\texttt{\textbackslash n}}_{char\ \#\omega}$ `#line 1\nstd::cout << __LINE__; }`

- But there’s no interesting semantic meaning here…

Wait a minute…



Annex B (informative)
Implementation quantities §[implimits]

1 Because **computers are finite**, C++ implementations are inevitably limited in the size of the programs they can successfully process. Every implementation shall document those limitations where known. This documentation may cite fixed limits where they exist, say how to compute variable limits as a function of available resources, or say that fixed limits do not exist or are unknown.

2 The limits may constrain quantities that include those described below or others. The bracketed number following each quantity is recommended as the minimum for that quantity. However, th**ese quantities are only guidelines and do not determine compliance**.

— […]

— […]

[…]

- But! This clause is informative, so it is not really binding!
- And in fact, the number of TUs is not listed as a limitable quantity…

The Verdict

- A program with infinite TUs can be legal
- If you allow transfinite files:
  - A transfinite file can be legal, but will not have any interesting semantic meaning over a non-transfinite file.
- If not, only C++11 or higher allow infinite programs:

C++03 §[lex.phases]1/2

... If a source file that is not empty **does not end** in a new-line character, or ends in a new-line character immediately preceded by a backslash character before any such splicing takes place, the behavior is undefined.

§[lex.phases]1/2

... A source file that is not empty and that does not end in a new-line character, or that ends in a new-line character immediately preceded by a backslash character before any such splicing takes place, **shall be processed as if an additional new-line character were appended to the file.**.

- `int main() {} \n\n\n\n...`