Enhancing the C++ Basic Character Set with Standard Character Mappings

Document Number: PO3OORO
Date: 2016/04/01
Reply-To:
Robert Douglas (rwdougla@gmail.com)
Faisal Vali (faisalv@yahoo.com)
Nate Wilson (nwilson20@gmail.com)
Nevin ":-)" Liber (nevin@cplusplusguy.com)
Attention: EWG, LEWG

Motivation

Languages evolve, as the means by which people best express their ideas. As this is the domain of programming languages, it is up to us to make sure that C++ evolves in conjunction with how people can best communicate their intents to machines and co-workers. Current language has begun a transition to new forms of language.[1]

The world is additionally becoming far more social. In this new world, a competitive language must be ready to embrace modern social forums with better support for the social paradigms of the day. Unfortunately, C++ has a rather heavy syntax for expressing ideas. This is less of a problem when armed with a keyboard and large monitors, but most devices these days are smaller with shrinking keyboards, if any physical keyboard at all. This problem must be mitigated, if C++ is to be an appealing language with the newest generation of developers.

Background

Historically, C++ has limited itself to the basic source character set, and left it to the implementation to define any extended characters which may be used, and the means by which those characters are mapped to the basic source character set. This limits the ability for a software developer to best express their ideas in a cross-platform manner. In fact, to be truly standards compliant, in all practicality, only the basic source character set is viable for code.

Prior Art

There have been a number of papers seeking to deal with the intricacies of unicode support in the past. Such papers found, have been noted in the references section at the end of this paper. It is understood that unicode support can be a tricky subject. This paper operates a bit differently from the others, though, in tackling the issue of the unicode as an expression of the code, itself, rather than the values defined in the code.

Visual Studio has been shipping with support for UTF-16 and UTF-8 within “identifiers, macros, string and character literals, and in comments”[2] since Visual Studio 2015.

Clang has supported unicode in identifiers since at least 3.3. [3]

Proposal

As translation phase 1 is implementation-defined, we have a means by which to specify a standard set of conversions from unicode characters to the basic source character set, with minimal breakage. Through this, we can effectuate standard support for whatever subset of the unicode standard we choose. Thus, we propose to define a standard set of conversions, at first for all the keywords, with notice to implementors, that we may wish to extend the supported set of unicode characters in the future.

Impact on the Standard

It is expected that this proposal will impact only the core library wording. No library features are proposed at this time. All breakages would be limited to already implementation-defined behavior.

Proposed Wording

1.2 Normative references [intro.refs]

~~(1.8) — ISO/IEC 10646-1:1993, Information technology — Universal Multiple-Octet Coded Character Set (UCS)~~
~~— Part 1: Architecture and Basic Multilingual Plane~~
(1.8) — ISO/IEC 10646:2014, Information technology -- Universal Coded Character Set (UCS)

2.2 Phases of translation [lex.phases]

1 The precedence among the syntax rules of translation is specified by the following phases.11

1. Physical source file characters are mapped, in an implementation-defined manner except for those listed in Table 1, to the basic source character set (introducing new-line characters for end-of-line indicators) if necessary. For those characters in Table 1, the mapping is applied as specified, to the basic source characters. (2.3) For all other characters, t~~T~~he set of physical source file characters accepted is implementation-defined. Any source file character not in the basic source character set (2.3) is replaced by the universal-character-name that designates that character. (An implementation may use any internal encoding, so long as an actual extended character encountered in the source file, and the same extended character expressed in the source file as a universal-character-name (e.g., using the \uXXXX notation), are handled equivalently except where this replacement is reverted in a raw string literal.)

Add a new table to 2.11 [lex.phases]

Table 1 - Unicode to Basic Source Character Set Conversions

Keyword	Emoji	Keyword	Emoji	Keyword	Emoji	Keyword	Emoji	Keyword	Emoji
`alignas`	↔️	`continue`	➰	`friend`	⍰	`register`	☑	`true`	👍
`alignof`	↩	`decltype`	🔎	`goto`	✈	`reinterpret_cast`	😈	`try`	🚓
`asm`	☢️	`default`	😃	`if`	❓	`return`	💩	`typedef`	📤
`auto`	🚗	`delete`	♻️	`inline`	⌛	`short`	🔬	`typeid`	🔍
`bool`	💡	`do`	👇	`int`	🔢	`signed`	➖	`typename`	⌨
`break`	💔	`double`	✌	`long`	🐟	`sizeof`	📏	`union`	💍
`case`	💼	`dynamic_cast`	🎇 `_`🎣	`mutable`	📻	`static`	⚡	`unsigned`	➕
`catch`	🚨	`else`	❔	`namespace`	📛	`static_assert`	⚡`_`💂	`using`	📥
`char`	🔥	`enum`	📠	`new`	👶	`static_cast`	⚡`_`🎣	`virtual`	👻
`char16_t`	🔥 `16_t`	`explicit`	💋	`noexcept`	🔇	`struct`	🏠	`void`	😱
`char32_t`	🔥 `32_t`	`export`	🚀	`nullptr`	☠️	`switch`	⁉	`volatile`	⛽
`class`	🏫	`extern`	🚪	`operator`	💿	`template`	💪	`wchat_t`	`w`🔥 `_t`
`const`	💎	`false`	👎	`private`	🏩	`this`	👉	`while`	🔁
`constexpr`	🗿	`float`	⛵	`protected`	🏦	`thread_local`	🎁		
`const_cast`	💣	`for`	🍀	`public`	⛪	`throw`	🔈		

Note to editor: Update *all* table numbers throughout.

Examples

Before	After

template< typename T >
bool foo() {
    return true;
}
	💪< ⌨️ T >💡 foo() { 💩 👍 ; }

#include <iostream>
using namespace std;
int main(int argc, char** argv) {
    cout << “Hello Emoji World!”;
    return 0;
}
	#include <iostream>
	📥 📛 std;
	🔢 main(🔢 argc, 🔥** argv) {
	    cout << “Hello Emoji World!”;
	    💩 0;
	}

#include <thread>
using namespace std;
bool run() {
    try {
        thread t(foo);
        thread t(bar);
        t1.join();
        t2.join();
    } catch (...) {
        cout << “Oops!”;
        return false;
    }
    return true;
}
	#include <thread>
	📥 📛 std;
	💡 run() {
	    🚓 {
	        thread t(foo);
	        thread t(bar);
	        t1.join();
	        t2.join();
	    } 🚨 (...)
	    { cout << “Oops!”; 💩 👎 ; }
	💩 👍 ;
	}

template<typename T, int Size>
class Container {
    using type = T;
    enum { size = Size };
    T data[];
};
	💪<⌨️ T, 🔢 Size>
	🏫 Container {
	    📥 type = T;
	    📠 { size = Size };
	    T data[];
	};

```
#include <string>
using namespace std;
namespace helper {
  static string ToString(int val)
{
    switch(val) {
      case 1:
        return “One”;
      case 2:
        return “Two”;
      case 3:
        return “Three”;
      default:
        return “None”;
    }
  }
}
```

```
#include <string>
📥 📛 std;
📛 helper {
  ⚡ string ToString(🔢 val) {
    ⁉(val) {
      💼 1:
        💩 “One”;
      💼 2:
        💩 “Two”;
      💼 3:
        💩 “Three”;
      😃:
        💩 “None”;
    }
  }
}
```

Future Work

We believe it would be beneficial to investigate other areas of the standard which would benefit so substantially as the basic keywords. As such, it is our recommendation that LEWG consider further papers proposing lists of mappings for common library function names.

To enable further enrichment of the unicode character set for programming in C++, we recommend committee members consider active participation in WG2, also.

Feature Test

For the purposes of SG10, this paper recommends the macro name `__cpp_emojis_rule`

Acknowledgements

A special thanks to John Mulaney for his inspiration to this endeavor.

Bibliography

1) John McWhorter: https://www.ted.com/talks/john_mcwhorter_txtng_is_killing_language_jk?language=en

2) Unicode Support in the Compiler and Linker https://msdn.microsoft.com/en-us/library/xwy0e8f2.aspx
3) Clang Release Notes: http://llvm.org/releases/3.3/tools/clang/docs/ReleaseNotes.html
4) ISO 10646:2014: http://standards.iso.org/ittf/PubliclyAvailableStandards/c063182_ISO_IEC_10646_2014.zip
5) N3572: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3572.html
6) N3336: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3336.html