For a Few Punctuators More

Document #: P2342R0
Date: 2021-03-25
Project: Programming Language C++
Audience: SG-7, EWG
Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

Abstract

This paper argues that the dollar sign `$`, bactick ``` ` ``` and at `@` could be used as a token in C++ for any future proposal that may have a use for it, with minimal impact on the C++ ecosystem. This aim to address the concerns raised when `$` was proposed for reflection and subsequently rejected for fear of breaking either code or tools.

Goals and Scope of this paper

This is an informative paper which proposes no change to the standard.

We however hope to convince WG21 and paper authors - notably ther authors of P2320R0 [4] that using new punctuators is an option, if it would make sense in the context of a specific proposal.

Of course other consideration specific to the feature might apply - for example, ``` ` ``` and `'` are hard to distinguish, so using ``` ` ``` might not be sensible in all contextes.

We are also not trying that we must or should use new symbols, just that we can do so if we ever want to!

Motivation

It seems increasingly difficult to find syntaxes for new features that are both distinct, easily readable, and terse. Of the non-alphanumeric characters available in both ASCII and EBCDIC, C++ makes use of the following:

```
. , < > ? / : ; ' " () {} [] %  * + - + = \ # ^ & | ~
```

While the following are not used:

```
` $ @
```

Analysis of the ecosystem

@

Objective-C

`@` is used by Objective-C and Objective C++, precisely because it is not valid C or C++ syntaxes. Some keywords are introduced by `@`:

`@interface`	`@class`	`@property`	`@finally`
`@end`	`@public`	`@try`	`@synthesize`
`@implementation`	`@protected`	`@throw`	`@dynamic`
`@protocol`	`@private`	`@catch`	`@selector`

`@` is also used to construct specific objects (`NSString, NSNumbers, NSArray, NSDictionary`) from literals and expressions. As such, the following are valid constructs in Objective-C:

```
@""
@123
@[foo, bar]
@(expression )
@{foo:bar, baz:bar}
```

This does reduce the possibility of using `@` in C++ if we are about Objective-C++. We do not try to answer the question of the relevance of Objective-C++ in this paper.

Note that some syntaxes are available, for example `@</**/>` is not a valid Objective-C construct.

CMake

CMake's `configure_file` comand replaces `@VAR@` in files being generated by the corresponding cmake variable. No escape mechanism seems to be provided. This will be explored in more detail further in this document.

Bison

`@` can appear in Bison/Yacc, usually preceding a number or a dollar sign, with no mean of escaping. This can however be worked around by a macro

```
%{
#define MY_LIB_ATSIGN(x) @ ## x
%}
%%
//...

MY_LIB_ATSIGN(foo);
```

```
    //...
```

$

$ in identifiers

`$` is allowed in identifiers as a compiler extension by GCC, Clang, MSVC, ICC ([Compiler explorer]). We surveyed the VCPGK repository.

- `$` is used as a macro identifier in the Relacy library. This library does not seem to be maintained in the past couple of years and there seem to be few usages of it - although HPX is a user.
- `$` appears in system functions in OpenVMS - But not in a leading position. These functions appear listed in VSI OpenVMS Utility Routines Manual. Projects that are ported to OpenVMS call these functions.
- Similarily, `$` appears in specific symbols injected by the `armlink` linker, although never in leading position
- Similarily, `$` appears in specific system functions on Darwin platforms for example `close$NOCANCEL`
- `$inject` appears in the `boost-di` library. Although we found no use of that identifier in the surveyed projects
- `ChakraCore`, the JS engine for IE9 uses $0, $1... $9 in an enumerator (which does not appear to be part of a public interface). This projets seems to compile for C++11
- `$` can appear in asm statements or macros that are expanded in asm statements
- the C library Cello use the dollar sign as a macro extensively - This library does not appear to be widely used.
- the `.NET` framework uses `$` in reserved identifiers Github

In total, we find 2400 pp-token (excluding asm statements) with $ in them in a corpus of 64640967 lines of C++ code and 33959876 lines of C code. Relacy accounts for 900 of these uses; Most of the remaining uses are in system symbols, notably, over 1000 uses seem related to OpenVMS.

`$` appear in the leading position no more than 300 times, half of which are in `ChakraCore`.

We found a further 2900 `$` in C source files, of which 1300 are related to OpenVMS and 1100 are in the Cello library.

`${}` and `$()` never appear in the surveyed corpus except in Cello, which is a C library.

Because `$` is not portable, some library perform some gymnastics to avoid potential warning, here is an example taken from `libuv`:

```
int uv__close_nocancel(int fd) {
    #pragma GCC diagnostic push
    #pragma GCC diagnostic ignored "-Wdollar-in-identifier-extension"
    extern int close$NOCANCEL(int);
    return close$NOCANCEL(fd);
    #pragma GCC diagnostic pop
}
```

A note of methodology

Combing 10 millions lines of C++ code is a difficult exercice. A lexer was used to list pp-token containing `$`, that list was analysed manually to exclude asm syatement ad study the context of each occurence. This is an error-prone process. However it correctly model the scale of the impact on the C++ open-source ecosystem.

$ in name mangling

Darwin uses _`$` in the mangling of swift symbols for versioning

$ in code generators

CMake

Cmake's `configure_file` command (which generate a file in the build directory from a template) will replace `$VAR` by the value of VAR (or by nothing if VAR does not exist) This behavior can be disabled by passing `@ONLY` to the configure command. It is important to note that the `configure_file` command is usually run on specific files that contain little code, usually used to set a set of defines for version number or compile-time configuration. Because CMake doesn't know about C++ it will replace the content of string literals, comments and doesn't expand the preprocessor.

Build 2 uses a similar syntax.

Bison, Yacc

In bison/yacc, `$` followed by a number or another `$` is interpreted by the code generator, which doesn't provide a means to escape it. However because these tools don't expand the preprocessor, it is possible to introduce a dollar sign in the generated files by introducing a `#define` in the preamble:

```
%{
#define MY_LIB_DOLLAR $
%}
%%
```

```
//...

MY_LIB_CONCAT(MY_LIB_DOLLAR, foo);

//...
```

SWIG

SWIG is another popular generator. It is designed to support a wide range of languages, including Perl and PHP (both of which use dollar sign in identifiers), and as such offers a mechanism to escape a `$` by using `\$` instead.

Regardless of the code generator used, the conclusion is the same: Introducing a new punctuator would not change the behavior of these code generators for existing code. Using that new ponctuator in combination with a code generator that would give it a special meaning can be achieved either by an escape mechanism provided by the tool, if possible or in all cases by the introduction of a macro that would be expanded to that new punctuator. We could also consider alternative spelling for that purpose if there is a need.

`Back tick`

On the mailing list, people expressed concern about using backtick as syntax elements - The only concern (to the best of my knowledge) is that it is used in markdown.

However, we found that the following markdown properly renders the backticks (in code font) in all engines tested, including Github, Gitlab, pandoc, a VS code plugin, 3 online markdown editors, and discord. This is consistent with the CommonMark specification.

~~~~
`` `test` ``

`test`

```

`test`

```
~~~~

Therefore, using backticks in C++ would not hinder people's ability to use the most popular Markdown renderers, except the inline code block would require more than one backtick.

Other Considerations

We found no use of `$, @ and `` ` `` in CUDA, OpenCL, Qt's moc, OpenMP, SYCL, C++/CX.

Some tools, which implement a C++ parser, as is the case for MOC, might however need to be modified to not trip over the new syntax. `moc` itself seems robust enough to not complain about the following code:

```
struct X{
    Q_GADGET

    int i = `42`;
    int y = @(42);
};
```

Other Programming language

- `$` can appear notably in perl, javascript, php, D, rust programs
- `@` can appear in rust, python, java, D
- `` ` `` can appear in Javascript, Raku, Swift

This list is not exhaustive but shows that other languages have been able to adopt these punctuators even when they often use the same code generators as C++, notably SWIG; Or use the same documentation tools (notably markdown) as C++ developers.

Recommandation

From this analysis, we conclude that:

- Using a lone `$` as punctuator would have a non-null but limited impact on existing code, and very little impact on adjacent tools.
- Using a `$` followed by another symbol such as `{`, `[`, `(` would have close to no impact on existing code.
- It must remain possible to spell a dollar as part of an identifier, This is always the case as `$` can be spelled `\u0024`
- It must remain possible for implementers to support `$` in the non-leading position in identifiers.
- Using `@` would impact objective c++ unless implementers can parse C++ headers included in objective C++ files differently, and we provide alternative spelling (assuming WG21 cares about interoperability between upcoming C++ versions and C++ features and Objective C++)
- Backticks can be used in C++
- Both `$` and `@` will require to be escaped or macro-expanded if there is a need for them to appear in code produced by code generators.

Should any of these be allowed in identifiers?

No, this would go against the direction voted in P1949R6 [1]. We should however not preclude implementation to permit it, at least in the middle of identifiers. `\u0024` should continue to work.

Should any of these be allowed in the basic character set?

The basic character set puts requirements on literal, not source files. This is made more clear by P2314R0 [2]. Therefore the question of whether something can be a punctuator and whether it must be encodable in literals are separate concerns. Note that the basic character set is still used to describe a few grammar elements notably escape sequences and raw-string delimiters. So for example adding `$` to the basic character set would make `'\$'` a conditionally supported escape sequence, and `R"$(Hello)$"` a valid raw string, which may or may not be sensible but are in any case separate concerns.

We note that there is a C proposal [1] that does that.

References

[1] Steve Downey, Zach Laine, Tom Honermann, Peter Bindels, and Jens Maurer. P1949R6: C++ identifier syntax using unicode standard annex 31. `https://wg21.link/p1949r6`, 9 2020.

[2] Jens Maurer. P2314R0: Character sets and encodings. `https://wg21.link/p2314r0`, 2 2021.

[3] Richard Smith. N4830: Committee draft, standard for programming language c++. `https://wg21.link/n4830`, 8 2019.

[4] Andrew Sutton, Wyatt Childers, and Daveed Vandevoorde. P2320R0: The syntax of static reflection. `https://wg21.link/p2320r0`, 2 2021.

[1] Philipp Klaus Krause - N2639: @ in basic source character set `http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2639.htm`

[2] ECMAScript® 2022 Language Specification `https://tc39.es/ecma262/#prod-NoSubstitutionTemplate`

[3] The Rust Reference `https://doc.rust-lang.org/reference/tokens.html#punctuation`

[4] SWIG-4.0 Documentation `http://www.swig.org/Doc4.0/SWIGDocumentation.html`

[5] Bison Documentation `https://www.gnu.org/software/bison/manual/html_node/index.html`

[6] Clang Documentation `https://clang.llvm.org/docs/ObjectiveCLiterals.html`