generally be controlled through compiler flags. The only portable way of embedding the cocktail emoji is to use its corresponding Unicode code point escape sequence (`u8"\U0001F378"`).

Lack of library support for Unicode

Essential **vocabulary types**, such as `std::string`, are completely unaware of encoding. They treat any stored string as a sequence of bytes. Even when correctly using Unicode string literals, programmers unfamiliar with Unicode might be surprised by seemingly innocent operations, such as asking for the size of a string representing the cocktail emoji:

```
#include <cassert>  // standard C assert macro
#include <string>   // std::string

void f()
{
    std::string cocktail(u8"\U0001F378"); // big character
    assert(cocktail.size() == 1);         // assertion failure
}
```

Even though the cocktail emoji is a *single* code point, `std::string::size` returns the number of code units (bytes) required to encode it. The lack of Unicode-aware vocabulary types and utilities in the Standard Library can be a source of defects and misunderstandings, especially in the context of international program localization.

Problematic treatment of UTF-8 in the type system

UTF-8 string literals use **`char`** as their ~~underlying~~ **`type`**. Such a choice is inconsistent with UTF-16 and UTF-32 literals, which provide their own distinct character types, **`char16_t`** and **`char32_t`**, respectively. Lack of a UTF-8-specific character type precludes providing distinct behavior for UTF-8 encoded strings using function overloading or template specialization because they are indistinguishable from strings having the encoding of the execution character set. Furthermore, whether the underlying type of **`char`** is a **`signed`** or **`unsigned`** type is itself implementation defined. Note that **`char`** is distinct from both **`signed char`** and **`unsigned char`**, but its behavior is guaranteed to be the same as one of those.

C++20 fundamentally changes how UTF-8 string literals work, by introducing a new non-aliasing **`char8_t`** character type whose representation is guaranteed to match **`unsigned char`**. The new character type provides several benefits.

- Ensures an **`unsigned`** and distinct type for UTF-8 character data
- Enables overloading for regular string literals versus UTF-8 string literals
- Potentially achieves better performance due to the lack of special aliasing rules