Towards a small-system alternative for std::string

Input for discussion in SG14
Wouter van Ooijen; 2017-05-12; version 1.0

Context

The C++ library has the std::string that can store a flexible length string. Implementations use the heap, reallocating the stored content when needed, or at their whim. An implementation can use SSO (Small String Optimization) to avoid using the heap when the length of the stored string doesn't exceed a certain (small) value.

A small real-time embedded system (a microcontroller with kilobytes of RAM) generally doesn't use a heap, because the RAM is small, the timing of heap operations is not predictable, and (link-time) certainty about resource use is generally more important than run-time flexibility. Hence such a system needs a different string datatype.

Inputs

To be usable on a wide range of small real-time embedded systems a library element must not use the heap. It is not enough to avoid heap use at it a run-time: it must be linkable without heap support. For various reasons it must in the same way avoid using exceptions. Larger timing-critical systems (gaming, fast trading) might likewise want to avoid run-time use of the heap (and probably exceptions), but the requirement to be able to link without these features will be less of an issue for such systems.

It is very desirable that it allows for (easy) link-time RAM usage analysis. This boils down to not using code indirection. In C++ context this means no virtual functions.

The use of a string-alternative is in a small embedded system is typically expected to be in user-interface parts, which tend to take large amounts of (source) code, but tend to be outside the timing-critical parts. Hence ease of use is more important than run-time performance.

The current state-of-the-art in micro-controllers is that code runs from ROM (Flash), and RAM is much more expensive and scarce than ROM. Hence limiting RAM use is more important than limiting code size. But it might be questionable to base critical design decisions on such volatile facts.

Interoperability with std::string is not needed for the main use (in the final embedded product), but might still be an advantage during development and testing, and might make a library that uses the embedded string type easier to use in a context where std::string is used. Such interworking features are not meant for constrained embedded use, hence they don't need to adhere to the restrictions (avoid heap, exceptions, and indirection).

Use a base class?

An embedded string can't use the heap, hence it must store its content in-place. One size won't fit all, hence it must take its size (maximum length) as a template parameter. To be able to pass such string objects as a parameters its class must either derive from a base class, or the accepting function must be a template function.

The no-base-class approach could lead to code bloat, and is quite a burden om the author.

```
string< 20 > a = ...
string< 30 > b = ...

// pass each string as a potentially different type
template< typename a_type, typename b_type >
... some_function( a_type & a, b_type & b )...

// pass the base class
... some_function( string_base & a, string_base & b ) ...
```

When a base class is used, it is essentially a view/range of the stored data (which is in the derived class). This makes some things easier, but some RAM overhead (pointers in the base class) seems unavoidable.

Expressions

Strings of different size (maximum lengths) are different types. Hence functions like operator+ that return a new object have a problem: what should the type of that object be? The integer approach (take the larger of the two input sizes) obviously won't work well. For operator+ a conservative but potentially RAM-wasting approach would be to take the sum of the two sizes, but that won't work when only the base types are known.

One approach is to offer only modifying operators like +=, and functions that return a (readonly) range.

Other issues

- Find a good name for the use case (small embedded real-time system is a bit long)
- Find a good name for this type of storage (fixed maximum size, flexible current size). Is 'inplace' an established term?
- (avoiding?) undefined behaviour
- std::string compatibility
- .c_str() and '/0' termination
- Base character type; inter-operability between strings of different base types

Existing work

Mateusz Pusz has an inplace-string at https://github.com/mpusz/inplace_string. It doesn't use a base class. It appears to be skeleton (start) of an implementation. It has no known users.

Martin Broers made a fixed string implementation for me as a research project. It can be found at https://github.com/FMBroers/fixed_string. It uses a base class. It is more complete than Mateusz' work, but uses older (obsolete?) techniques. It has no known users.