```
        while (d_size < size) new (rawElementPtr(d_size++)) TYPE;
        while (d_size > size) rawElementPtr(--d_size)->~TYPE();
    }

    // ...
};
```

Without the use of **alignas**, `d_buffer` (in the code snippet above), which is an array of characters, would itself have an alignment requirement of 1. The compiler would therefore be free to place it on any address boundary, which ~~is problematic~~ for any `TYPE` argument with an alignment requirement stricter than 1.

Ensuring proper alignment for architecture-specific instructions

Architecture-specific instructions or compiler intrinsics might require the data they act on to have a specific alignment. One example of such intrinsics is the Streaming SIMD Extensions (SSE)[3] instruction set available on the x86 architecture. SSE instructions operate on groups of four 32-bit single-precision floating-point numbers at a time, which are required to be 16-byte aligned.[4] The **alignas** specifier can be used to create a type satisfying this requirement:

```
struct SSEVector
{
    alignas(16) float d_data[4];
};
```

Each object of the `SSEVector` type above is guaranteed always to be aligned to a 16-byte boundary and can therefore be safely and conveniently used with SSE intrinsics:

```
#include <cassert>     // standard C assert macro
#include <xmmintrin.h> // __m128 and _mm_XXX functions

void f()
{
    const SSEVector v0 = {0.0f, 1.0f, 2.0f, 3.0f};
    const SSEVector v1 = {10.0f, 10.0f, 10.0f, 10.0f};

    __m128 sseV0 = _mm_load_ps(v0.d_data);
    __m128 sseV1 = _mm_load_ps(v1.d_data);
        // _mm_load_ps requires the given float array to be 16-byte aligned.
        // The data is loaded into a dedicated 128-bit CPU register.

    __m128 sseResult = _mm_add_ps(sseV0, sseV1);
        // sum two 128-bit registers; typically generates an addps instruction
```

[3]**inteliig**, "Technologies: SSE"

[4]"Data must be 16-byte aligned when loading to and storing from the 128-bit XMM registers used by SSE/SSE2/SSE3/SSSE3": see **intel16**, section 4.4.4, "Data Alignment for 128-Bit Data," pp. 4-19–4-20.