

Fast and small C++ - When efficiency matters

Presentation Material



C++Online, online, 2025-02-27



© 2025 Andreas Fertig
AndreasFertig.com
All rights reserved

All programs, procedures and electronic circuits contained in this book have been created to the best of our knowledge and belief and have been tested with care. Nevertheless, errors cannot be completely ruled out. For this reason, the program material contained in this book is not associated with any obligation or guarantee of any kind. The author therefore assumes no responsibility and will not accept any liability, consequential or otherwise, arising in any way from the use of this program material or parts thereof.

Version: v1.0

The work including all its parts is protected by copyright. Any use beyond the limits of copyright law requires the prior consent of the author. This applies in particular to duplication, processing, translation and storage and processing in electronic systems.

The reproduction of common names, trade names, product designations, etc. in this work does not justify the assumption that such names are to be regarded as free in the sense of trademark and brand protection legislation and can therefore be used by anyone, even without special identification.

Planning, typesetting and cover design: Andreas Fertig
Cover art and illustrations: Franziska Panter <https://franziskapanter.com>
Production and publishing: Andreas Fertig

Style and conventions

The following shows the execution of a program. I used the Linux way here and skipped supplying the desired output name, resulting in `a.out` as the program name.

```
$ ./a.out  
Hello, C++!
```

- `<string>` stands for a header file with the name `string`
- `[[xyz]]` marks a C++ attribute with the name `xyz`.



Fast and small C++ - When efficiency matters



Andreas Fertig
<https://AndreasFertig.com>
post@AndreasFertig.com
bsky.app/profile/andreasfertig.com

About me

- I offer in-house training, on-site or remote.
 - Feel free to contact me!
- Would you like to learn at your own pace? Check out my self-study courses:



<https://fertig.to/ssc>



Andreas Fertig

You?

fertig
adjective /'fɛrtɪç/

finished
ready
complete
completed



Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

3



Compressed pair

```
1 auto f =  
2   std::unique_ptr<FILE, decltype(&fclose)>{fopen("SomeFile.txt", "r"), &fclose};
```



Andreas Fertig

v1.0

Fast and small C++ - When efficiency matters

4

Compressed pair

```
1 auto f =  
2   std::unique_ptr<FILE, decltype(&fclose)>{fopen("SomeFile.txt", "r"), &fclose};  
3  
4   static_assert(sizeof(f) > sizeof(void*));
```



Andreas Fertig

v1.0

Fast and small C++ - When efficiency matters

5



Compressed pair

```

1 class Base {
2 public:
3     void Fun() { puts("Hello, EB0!"); }
4 };
5
6 class Derived : public Base {
7     uint32_t mData{};
8
9 public:
10 };
11
12 void Use()
13 {
14     Derived d{};
15     static_assert(sizeof(d) == sizeof(uint32_t)); A
16
17     d.Fun();
18 }

```



Andreas Fertig

Fast and small C++ - When efficiency matters

6

Compressed pair

```

1 template<class T>
2 struct default_delete {
3     constexpr void operator()(T* ptr) const noexcept
4     {
5         static_assert(0 < sizeof(T), "can't delete an incomplete type");
6         delete ptr;
7     }
8 };

```



Andreas Fertig

Fast and small C++ - When efficiency matters

7



Compressed pair

```

1 template<class T, class Del = default_delete<T>>
2 class unique_ptr {
3     CompressedPair<Del, T*> mPair;  A
4
5 public:
6     unique_ptr(T* ptr) : mPair{ptr} {}
7     unique_ptr(T* ptr, Del deleter) : mPair{deleter, ptr} {}  B C
8
9     unique_ptr(const unique_ptr&) = delete;
10    unique_ptr operator=(const unique_ptr&) = delete;
11
12    unique_ptr(unique_ptr&& src) : mPair{std::exchange(src.mPair.ptr, nullptr)} {}
13    unique_ptr& operator=(unique_ptr&& src)
14    {
15        mPair.ptr = std::exchange(src.mPair.ptr, mPair.ptr);
16        mPair.deleter = std::exchange(src.mPair.deleter, mPair.deleter);
17        return *this;
18    }
19
20    ~unique_ptr()
21    {
22        if(mPair.ptr) { mPair.deleter(mPair.ptr); }  D
23    }
24
25    T* operator->() { return mPair.ptr; }
26 };

```



Andreas Fertig

Fast and small C++ - When efficiency matters

8

Compressed pair

```

1 template<typename Del, typename T>
2 struct CompressedPair {
3     [[no_unique_address]] Del deleter;
4     [[no_unique_address]] T ptr;
5
6     CompressedPair(T s) : ptr{s} {}
7     CompressedPair(Del f, T s) : deleter{f}, ptr{s} {}
8 };

```



Andreas Fertig

Fast and small C++ - When efficiency matters

9



Compressed pair

```

1 template<typename T, auto DeleteFn>
2 using unique_ptr_deleter =
3     std::unique_ptr<T, decltype([](T* obj) { DeleteFn(obj); })>;
4
5 auto f = unique_ptr_deleter<FILE, fclose>{fopen("SomeFile.txt", "r")};
6
7 static_assert(sizeof(f) == sizeof(void*));

```



Andreas Fertig

Fast and small C++ - When efficiency matters

10

Compressed pair

```

1 template<typename T, auto DeleteFn>
2 using unique_ptr_deleter =
3     std::unique_ptr<T, decltype([](T* obj) static { DeleteFn(obj); })>;
4
5 auto f = unique_ptr_deleter<FILE, fclose>{fopen("SomeFile.txt", "r")};
6
7 static_assert(sizeof(f) == sizeof(void*));

```

C++23



Andreas Fertig

Fast and small C++ - When efficiency matters

11



Implementing the Small String Optimization

```

1 struct string {
2     size_t mSize{};
3     size_t mCapacity{};
4     char* mData{};
5     char mSsoBuffer[16]{};
6     bool mIsSso{true};
7 };
8
9 static_assert(sizeof(string) == 48);

```



Andreas Fertig

Fast and small C++ - When efficiency matters

12

Implementing the Small String Optimization - libstdc++

```

1 struct string {
2     char* mPtr;
3     size_t mSize{};
4     union {
5         size_t mCapacity;
6         char mBuf[8];
7     };
8
9     static_assert((sizeof(mBuf) + sizeof(mPtr) + sizeof(mSize)) == 24);
10
11     constexpr static size_t max_cap() { return std::numeric_limits<size_t>::max(); }
12     constexpr static size_t sso_cap() { return sizeof(mBuf) - 1; /* -1 for '\0' */ }
13     constexpr static bool fits_into_sso(size_t len) { return len <= sso_cap(); }
14     constexpr bool is_long() const { return mPtr != mBuf; }
15
16     constexpr string() : mPtr{mBuf}, mBuf{} {}
17     constexpr string(const char* data, size_t len)
18     : mPtr{fits_into_sso(len) ? mBuf : new char[len]}, mSize{len}, mBuf{}
19     {
20         if(is_long()) { mCapacity = len; } // next: copy _data to data()
21     }
22
23     constexpr size_t size() const { return mSize; }
24     constexpr const char* data() const { return mPtr; }
25     constexpr size_t capacity() const { return is_long() ? mCapacity : sso_cap(); }
26 };

```



Andreas Fertig

Fast and small C++ - When efficiency matters

13



Implementing the Small String Optimization - MS STL

```

1 struct string {
2     union {
3         char* mPtr;
4         char mBuf[8];
5     };
6     size_t mSize{};
7     size_t mCapacity{};
8
9     static_assert((sizeof(mBuf) + sizeof(mCapacity) + sizeof(mSize)) == 24);
10
11     constexpr static size_t max_cap() { return std::numeric_limits<size_t>::max(); }
12     constexpr static size_t sso_cap() { return sizeof(mBuf) - 1; /* -1 for '\0' */ }
13     constexpr static bool fits_into_sso(size_t len) { return len <= sso_cap(); }
14     constexpr bool is_long() const { return mCapacity > sso_cap(); }
15
16     constexpr string() : mBuf{} {}
17     constexpr string(const char* _data, size_t len)
18     : mBuf{}, mSize{len}, mCapacity{fits_into_sso(len) ? sso_cap() : len}
19     {
20         if(is_long()) { mPtr = new char[len]; }
21         // copy _data to data()
22     }
23
24     constexpr size_t size() const { return mSize; }
25     constexpr const char* data() const { return is_long() ? mPtr : mBuf; }
26     constexpr size_t capacity() const { return mCapacity; }
27 };

```

Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

14

Implementing the Small String Optimization - libc++

```

1 struct string {
2     static constexpr unsigned BIT_FOR_CAP{sizeof(size_t) * 8 - 1};
3
4     struct normal {
5         size_t large : 1;
6         size_t capacity : BIT_FOR_CAP; // A MSB for large bit
7         size_t size;
8         char* data;
9     };
10
11     struct sso {
12         uint8_t large : 1;
13         uint8_t size : (sizeof(uint8_t) * 8) - 1; // B large+size == sizeof(uint8_t)
14         uint8_t padding[sizeof(size_t) - sizeof(uint8_t)]; // C Padding large+size+padding == sizeof(size_t)
15         char data[sizeof(normal) - sizeof(size_t)];
16     };
17
18     union {
19         sso small;
20         normal large;
21     } packed;
22
23     static_assert((sizeof(normal) == sizeof(sso)) and (sizeof(normal) == 24));
24     // to be continued

```

Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

15



Implementing the Small String Optimization - libc++

```

1 // continue
2 static size_t max_cap() { return std::pow(2, BIT_FOR_CAP); }
3 constexpr static size_t sso_cap() { return sizeof(sso::data) - 1; /* -1 for '\0' */ }
4 constexpr static bool fits_into_sso(size_t len) { return len <= sso_cap(); }
5 constexpr bool is_long() const { return packed.small.large; }
6
7 constexpr string() : packed{} {}
8 constexpr string(const char* _data, size_t len) : packed{}
9 {
10     if(fits_into_sso(len)) {
11         packed.small.size = len;
12     } else {
13         packed.large.large = true;
14         packed.large.size = len;
15     }
16
17     // copy _data to data()
18 }
19
20 constexpr size_t size() const { return is_long() ? packed.large.size : packed.small.size; }
21 constexpr const char* data() const { return is_long() ? packed.large.data : packed.small.data; }
22 constexpr size_t capacity() const { return is_long() ? packed.large.capacity : sso_cap(); }
23 };

```



Andreas Fertig

Fast and small C++ - When efficiency matters

16

Implementing the Small String Optimization - fbstring

```

1 struct string {
2     struct normal {
3         char* data;
4         size_t size;
5         size_t capacity; // virtually reduced by one byte
6     };
7
8     struct sso {
9         char data[sizeof(normal)]; // MSB for long string mode indicator
10    };
11
12    union {
13        sso small;
14        normal large;
15    } packed;
16
17    static_assert((sizeof(normal) == sizeof(sso)) and (sizeof(normal) == 24));
18
19    static size_t max_cap() { return std::pow(2, sizeof(normal::capacity) * 8 - 8); }
20    constexpr static size_t sso_cap() { return sizeof(sso) - 1; }
21    constexpr char& get_mode_byte() { return packed.small.data[sizeof(normal) - 1]; }
22    constexpr char get_mode_byte() const { return packed.small.data[sizeof(normal) - 1]; }
23    constexpr static bool fits_into_sso(size_t len) { return len <= sso_cap(); }
24    constexpr bool is_long() const { return get_mode_byte() & 0x40; /* 0b100'0000 */ }
25    // to be continued

```



Andreas Fertig

Fast and small C++ - When efficiency matters

17



Implementing the Small String Optimization - fbstring

```

1 // continue
2 constexpr string() : packed{} {}
3 constexpr string(const char* _data, size_t len) : packed{}
4 {
5     if(fits_into_sso(len)) {
6         get_mode_byte() = sso_cap() - len; A The bytes remaining!
7     } else {
8         get_mode_byte() = 0x40; B Set bit for large mode
9         packed.large.size = len;
10    }
11
12    // copy _data to data()
13 }
14
15 constexpr size_t size() const
16 { return is_long() ? packed.large.size : sso_cap() - get_mode_byte(); }
17 constexpr const char* data() const { return is_long() ? packed.large.data : packed.small.data; }
18 constexpr size_t capacity() const { return is_long() ? packed.large.capacity : sso_cap(); }
19 };

```

CppCon 2016: Nicholas Ormrod "The strange details of std::string at Facebook" [1]



Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

18

The powers of constexpr

```

1 template<size_t N>
2 class FixedString {
3     size_t mSize{};
4     char mData[N]{};
5
6 public:
7     FixedString() = default;
8     FixedString(const char* str) : mSize{std::char_traits<char>::length(str)}
9     {
10         std::copy_n(str, size(), mData);
11     }
12
13     size_t size() const { return mSize; }
14
15     std::string_view data() const { return {mData, mSize}; }
16 };
17
18 template<size_t N>
19 auto make_fixed_string(const char (&str)[N])
20 {
21     return FixedString<N>{str};
22 }
23
24 const static FixedString<50> x{"Hello, embedded World!"};
25 const static auto y{make_fixed_string("Hello, some other planet!")};

```



Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

19



std::initializer_list

```
1 void Fun()  
2 {  
3     std::initializer_list<int> list{3, 4, 5, 6};  
4  
5     Receiver(list);  
6 }
```

Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

20

std::initializer_list

```
1 void Receiver(const int list[4]) noexcept;  
2  
3 void Fun() noexcept  
4 {  
5     const int list[4]{3, 4, 5, 6};  
6  
7     Receiver(list);  
8 }
```

The array size of `Receiver` is misleading! This function takes any array or pointer.

Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

21



`std::initializer_list`

```

1 void Receiver(const int list[4]) noexcept;
2
3 void Fun() noexcept
4 {
5     static const int list[4]{3, 4, 5, 6};
6
7     Receiver(list);
8 }

```

The array size of `Receiver` is misleading! This function takes any array or pointer.



Andreas Fertig

Fast and small C++ - When efficiency matters

22

`std::initializer_list`

```

1 void Receiver(std::initializer_list<int> list) noexcept;
2
3 void Fun() noexcept
4 {
5     std::initializer_list<int> list{3, 4, 5, 6};
6
7     Receiver(list);
8 }

```



Andreas Fertig

Fast and small C++ - When efficiency matters

23



}

I am Fertig.

<https://AndreasFertig.com>

Available online:



<https://AndreasFertig.com>

Images by Franziska Panter:



<https://panther-concepts.de>



Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

24

Used Compilers & Typography

Used Compilers

- **Compilers used to compile (most of) the examples.**

- GCC 14.1.0
- Clang 18.1.0

Typography

- **Main font:**

- Camingo Dos Pro by Jan Fromm (<https://janfromm.de/>)

- **Code font:**

- CamingoCode by Jan Fromm licensed under Creative Commons CC BY-ND, Version 3.0 <http://creativecommons.org/licenses/by-nd/3.0/>



Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

25



References

[1] ORMROD N., "Cppcon 2016: The strange details of std::string at facebook". <https://youtu.be/kPR8h4-qZdk>

Images:

3: fran

27: Franziska Panter



Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

26

Upcoming Events

For my upcoming talks you can check <https://andreasfertig.com/talks/>.

For my courses you can check <https://andreasfertig.com/courses/>.

Like to always be informed? Subscribe to my newsletter: <https://andreasfertig.com/newsletter/>.



Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

27



About Andreas Fertig



Photo: Kristijan Matic www.kristijanmatic.de

Andreas Fertig is an expert C++ trainer and consultant who delivers engaging and impactful training sessions, both on-site and remotely, to teams around the globe. As an active member of the C++ standardization committee, Andreas contributes directly to shaping the future of the language. He shares insights on writing cleaner, more efficient C++ code at international conferences. He publishes specialist articles, e.g., for iX magazine, and has published several C++ textbooks.

With C++ Insights (<https://cppinsights.io>), Andreas has created an internationally recognized tool that enables users to look behind the scenes of C++ and thus understand constructs even better.

Discover more about Andreas and his work at andreasfertig.com.



Andreas Fertig

v1.0

Fast and small C++ - When efficiency matters

28

