Fast and small C++ – When efficiency matters

Presentation Material

code::dive, Wrocław, 2024-11-25

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`.

fertig

adjective /ˈfɛrtɪç/

finished
ready
complete
completed

Compressed pair

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

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*));
```

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 };
```

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} {}                    B
 7   unique_ptr(T* ptr, Del deleter) : mPair{deleter, ptr} {}   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.second, nullptr)} {}
13   unique_ptr& operator=(unique_ptr&& src)
14   {
15     mPair.second = std::exchange(src.mPair.second, mPair.second);
16     mPair.first  = std::exchange(src.mPair.first, mPair.first);
17     return *this;
18   }
19
20   ~unique_ptr()
21   {
22     if(mPair.second) { mPair.first(mPair.second); }   D
23   }
24
25   T* operator->() { return mPair.second; }
26 };
```

Compressed pair

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

Compressed pair

```cpp
template<typename T, auto DeleteFn>
using unique_ptr_deleter =
  std::unique_ptr<T, decltype([](T* obj) { DeleteFn(obj); })>;

auto f = unique_ptr_deleter<FILE, fclose>{fopen("SomeFile.txt", "r")};

static_assert(sizeof(f) == sizeof(void*));
```

Compressed pair

C++23

```cpp
template<typename T, auto DeleteFn>
using unique_ptr_deleter =
  std::unique_ptr<T, decltype([](T* obj) static { DeleteFn(obj); })>;

auto f = unique_ptr_deleter<FILE, fclose>{fopen("SomeFile.txt", "r")};

static_assert(sizeof(f) == sizeof(void*));
```

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);
```

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 };
```

Implementing the Small String Optimization - MS STL

```cpp
 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 };
```

Implementing the Small String Optimization - libc++

```cpp
 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;   Ⓐ 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;        Ⓑ large+size == sizeof(uint8_t)
14     uint8_t padding[sizeof(size_t) - sizeof(uint8_t)]; Ⓒ 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
```

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 };
```

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!")};
```

}

I am Fertig.

https://AndreasFertig.com

Available online:

https://AndreasFertig.com

Images by Franziska Panter:

https://panther-concepts.de

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/
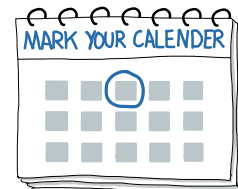
References

Images:

19: Franziska Panter

Upcoming Events

Talks

- *Effizientes C++ - Tips und Tricks aus dem Alltag*, ESE Kongress, December 04

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/.

About Andreas Fertig

Andreas Fertig, is an expert C++ trainer and consultant providing classes world wide, remote and on-site.

Andreas is involved in the C++ standardization committee, developing the new standards. At international conferences, he presents how code can be written better. He publishes specialist articles, e.g., for iX magazine, and has published several textbooks on C++.

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.

Before training and consulting, he worked for Philips Medizin Systeme GmbH for ten years as a C++ software developer and architect focusing on embedded systems.

You can find Andreas online at andreasfertig.com.

Photo: Kristijan Matic www.kristijanmatic.de