Back to Basics: static, inline, const, and constexpr

Presentation Material

CppCon, Aurora, CO, 2025-09-18

Andreas Fertig

Write unique code!

Version: v1.0

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`.

https://AndreasFertig.com
post@AndreasFertig.com

Back to Basics: static, inline, const, and constexpr

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

About me

- Creator of C++ Insights
  - https://cppinsights.io
- Host of the C++ Insights YouTube channel
  - https://youtube.com/@andreas_fertig
- Author
  - https://andreasfertig.com/books/
  - or checkout CppCache here at CppCon

Learn C++ with me!

- I'm available for inhouse classes, onsite or remote.
- You want to learn at your pace? Check out my self-study courses:

https://fertig.to/slcpp20

fertig

adjective /ˈfɛrtɪç/

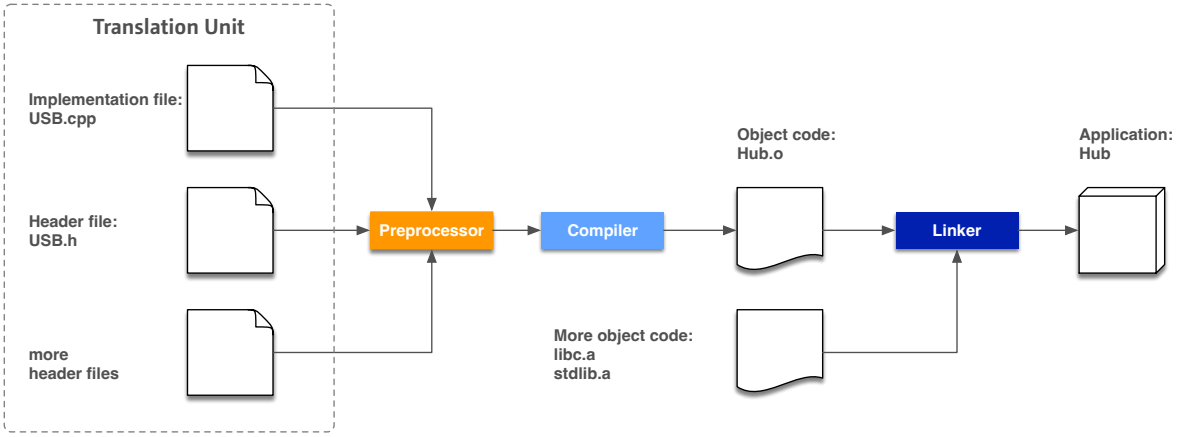
finished
ready
complete
completed

Keywords in C++

static

```
1  static void Fun()   (A) Translation unit local function
2  {
3    static int val{};   (B) static block variable
4  }
5
6  class Emotions {
7    static int mSmileDuration;   (C) static data member, out-of-class definition required
8
9  public:
10   static void Smile();   (D) static member function
11 };
12
13 int Emotions::mSmileDuration{4};   (E) out-of-class definition for static data member
14
15 void Emotions::Smile()   (F)
16 {
17   puts(":-)");
18 }
```


What's a translation unit



Translation unit local function

```
1 static void Fun()   A Translation unit local function
2 {
3   static int val{};
4 }
```

A has *internal linkage* due to **static**.

Block local **static** variables

```
1 static void Fun()
2 {
3   static int val{};   B static block variable
4 }
```

B has *static storage duration* due to **static**.

`static` data-members

Header file:

```
1 class Emotions {
2   static int mSmileDuration;   C static data member, out-of-class definition required
3
4 public:
5   static void Smile();
6 };
```

Implementation file:

```
1 int Emotions::mSmileDuration{4};   E out-of-class definition for static data member
2
3 void Emotions::Smile()
4 {
5   puts(":-)");
6 }
```

C declares a **`static`** data member.

E defines the **`static`** data member and initializes it.

`static` member-functions

```
1 class Emotions {
2   static int mSmileDuration;
3
4 public:
5   static void Smile();   D static member function
6 };
7
8 int Emotions::mSmileDuration{4};
9
10 void Emotions::Smile()   F
11 {
12   puts(":-)");
13 }
```

D declares a **`static`** member-function.

F defines the **`static`** member-function.

As for non-static member-functions, you decide whether to implement them inline or, as here, out-of-line.

https://AndreasFertig.com
post@AndreasFertig.com

inline

inline

```
1  inline void Fun()   A inline function
2  {
3    static int val{};   B static block variable
4  }
5
6  class Emotions {
7    inline static int mSmileDuration;   C declare and define
8
9  public:
10   static void Smile();                                  D
11   void        Laught();                                 E Out-of-line definition
12   void        Cheer() { puts("Go, go, go!!!"); }        F in-class definition
13 };
14
15 void Emotions::Smile()   G
16 {
17   puts(":-)");
18 }
```

`inline` function

```
1 inline void Fun()   A inline function
2 {
3   static int val{};
4 }
```

A as long as the implementation is the same, multiple definitions in multiple translations units are allowed without violating the one definition rule (ODR).

A if there are different implementations you're looking at undefined behavior (UB).

`inline` function

```
1 inline void Fun()
2 {
3   static int val{};   B static block variable
4 }
```

B exists only once, even if the function is inlined.

https://AndreasFertig.com
post@AndreasFertig.com

`inline static` data-members

C++17

```
 1 class Emotions {
 2   inline static int mSmileDuration{4};   C declare and define
 3 public :
 4
 5   static void
 6        Smile();
 7   void Laught();
 8   void Cheer() { puts("Go, go, go!!!"); }
 9 };
10
11 void Emotions::Smile()
12 {
13   puts(":-)");
14 }
```

C declares a data member that is **`static`** and **`inline`**. Drops the needs for a dedicated definition.

`inline` and member functions

C++17

```
 1 class Emotions {
 2   inline static int mSmileDuration{4};
 3 public:
 4   static void Smile();                          D Out-of-line definition
 5   void        Laught();                         E Out-of-line definition
 6   void        Cheer() { puts("Go, go, go!!!"); } F in-class definition
 7 };
 8
 9 void Emotions::Smile()  G
10 {
11   puts(":-)");
12 }
```

D and E are out-of-line definitions.

F is a in-class definitions and implicitly **`inline`**.

`const`

Q: What's a cv-qualifier?

A: *The cv stands for* ***`const`*** *and* ***`volatile`****.*

[1] or as changing without the compilers able to see for `volatile`.

Q: What's a **top-level** cv-qualifier?

A: *When we as developers tell the compiler to treat a variable or parameter as read-only[1] by making it **const**.*

[1] or as changing without the compilers able to see for `volatile`.

The different shades of const: Variables

```
1 char                robert{};    A
2 const char          amy{};       B top-level qualifier
3 char*               bill{};      C
4 const char*         alfred{};    D pointing to constant memory
5 char* const         linda{};     E top-level qualifier
6 const char* const   melissa{};   F pointing to constant memory plus top-level qualifier
```


https://AndreasFertig.com
post@AndreasFertig.com

The different shades of const: Variables

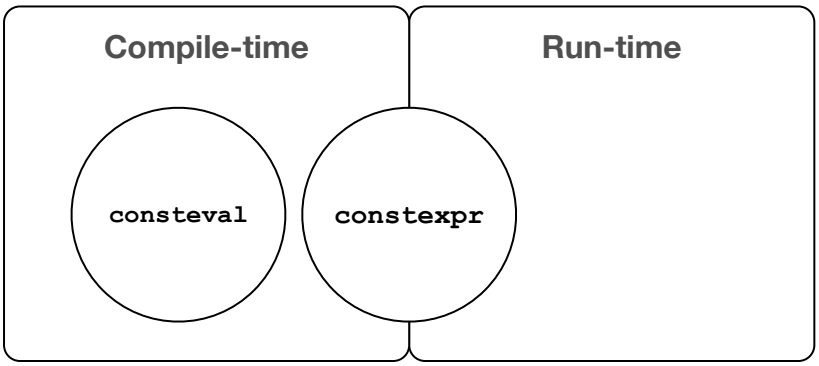
```
1 char                                          robert{};    A
2 const_by_user_request char                    amy{};       B
3 char*                                         bill{};      C
4 read_only_data char*                          alfred{};    D
5 char* const_by_user_request                   linda{};     E
6 read_only_data char* const_by_user_request    melissa{};   F
```

The different shades of const: Function parameters

```
1 void Fun(char);                 A
2 void Fun(const char);           B Duplicate of A
3 void Fun(const char*);          C
4 void Fun(char* const);          D
5 void Fun(const char* const);    E Duplicate of C
```

https://AndreasFertig.com
post@AndreasFertig.com

The `const`-world



The purpose of a `constexpr` function is to produce a constant at compile time.

constexpr

```
1  constexpr int Pow2(unsigned n)
2  {
3    static constexpr std::array table{1, 2, 4, 8, 16};   B static block variable
4    return (n < table.size()) ? table.at(n) : 0;
5  }
6
7  class Emotions {
8    static constexpr int mSmileDuration{4};   C declare and define
9
10 public:
11   constexpr static void Smile();    D
12   constexpr void        Laught();   E Out-of-line definition
13   constexpr void        Cheer() { /*puts("Go, go, go!!!");*/ }   F
14 };
15
16 constexpr void Emotions::Smile()   G
17 {
18   /*puts(":-)");*/
19 }
```

constexpr function

```
1  constexpr int Pow2(unsigned n)   A constexpr function
2  {
3    static constexpr std::array table{1, 2, 4, 8, 16};
4    return (n < table.size()) ? table.at(n) : 0;
5  }
```

A implictly **inline**.

https://AndreasFertig.com
post@AndreasFertig.com

constexpr variable

```
1 constexpr int Pow2(unsigned n)
2 {
3   static constexpr std::array table{1, 2, 4, 8, 16};   B static block variable
4   return (n < table.size()) ? table.at(n) : 0;
5 }
```

- B allowed since C++23.
- B is implicitly **const**.

constexpr data-members and member functions

```
1  class Emotions {
2    static constexpr int mSmileDuration{4};   C
3
4  public:
5    constexpr static void Smile();                                  D
6    constexpr void        Laught();                                 E
7    constexpr void        Cheer() { /*puts("Go, go, go!!!");*/ }    F
8  };
9
10 constexpr void Emotions::Smile()   G
11 {
12   /*puts(":-)");*/
13 }
```

- C , D, E, F, and G are implicitly **inline**.
- C is implicitly **const**.

C++17

constexpr if

```
1  template<typename T>
2  auto getValue(T t)
3  {
4    A constexpr if for compile-time dispatching
5    if constexpr(std::is_pointer_v<T>) {
6      assert(nullptr != t);
7      return *t;
8    } else {
9      return t;
10   }
11 }
12
13 void Use()
14 {
15   int  i  = 4;
16   int* ip = &i;
17
18   B All calls will result in plain int as type
19   auto iv  = getValue(i);
20   auto ipv = getValue(ip);
21   auto itv = getValue(43);
22 }
```


C++20

std::is_constant_evaluated

```
1  constexpr int Div(int dividend, int divisor)
2  {
3    if(0 == divisor) { Log("Division by zero"); }
4
5    return dividend / divisor;
6  }
```

Use:

```
1  auto           water = Div(8, 2);
2  constexpr auto stone = Div(8, 2);
```

std::is_constant_evaluated

C++20

```
1 constexpr void Log(std::string_view msg)
2 {
3   if(not std::is_constant_evaluated()) { puts(msg.data()); }
4 }
```


std::is_constant_evaluated

C++20

```
1 constexpr void Log(std::string_view msg)
2 {
3   A The constexpr is bad, it doesn't do what is intended
4   if constexpr(not std::is_constant_evaluated()) {
5       puts(msg.data());
6   }
7 }
```

if consteval

C++23

```
1 constexpr void Log(std::string_view msg)
2 {
3   if not consteval { puts(msg.data()); }
4 }
```


constinit

```
1 struct Air {
2   Air(int amount)
3   : _amount{amount}
4   {}
5
6   void Consume(int v) { _amount -= v; }
7   int  Available() const { return _amount; }
8
9 private:
10  int _amount;
11 };
12
13 struct Human {
14   Human(int breath);
15 };
16
17 Air   co2{9};    A Create global Air object
18 Human josh{5};   B Create global Human object
19
20 Human::Human(int breath)
21 {
22   co2.Consume(breath);   C Depends on co2
23 }
```

constinit

```
1  struct Air {
2    Air(int amount)
3    : _amount{amount}
4    {}
5
6    void Consume(int v) { _amount -= v; }
7    int  Available() const { return _amount; }
8
9  private:
10   int _amount;
11 };
12
13 struct Human {
14   Human(int breath);
15 };
16
17 Human josh{5};   B Create global Human object
18 Air   co2{9};    A Create global Air object
19
20 Human::Human(int breath)
21 {
22   co2.Consume(breath);   C Depends on co2
23 }
```

constinit

```
1  struct Air {
2    constexpr Air(int amount)
3    : _amount{amount}
4    {}
5
6    void Consume(int v) { _amount -= v; }
7    int  Available() const { return _amount; }
8
9  private:
10   int _amount;
11 };
```

C++20

constinit

```
1 Human          josh{5};   B Create global Human object
2 constinit Air  co2{9};    A Create global co2 object
```


}

I am Fertig.

Available online:

https://AndreasFertig.com

Images by Franziska Panter:

https://franziskapanter.com

Used Compilers & Typography

Used Compilers

- Compilers used to compile (most of) the examples.
  - GCC 14.1.0
  - Clang 20.1.0

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:

3: fran

42: Franziska Panter

Upcoming Events

Talks

- *Embedded-Friendly C++: Features That Make a Difference*, code::dive, October 23
- *Embedded-Friendly C++: Features That Make a Difference*, Meeting C++, November 07
- *static, inline, const und constexpr - Was sie bedeuten und wann man sie verwendet*, ESE Kongress, December 03

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

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.