Modern C++ - Why and What

JLM Charedi Tech Meetup

Yehezkel Bernat - YehezkelShB@gmail.com

Thunderbolt^{TM} SW team, Intel

Agenda

- Modern C++ - Meaning

Agenda

- Modern C++ - Meaning
    - To make sure we use the same terminology

Agenda

- Modern C++ - Meaning
  - To make sure we use the same terminology
- Modern C++ - Why?

Agenda

- Modern C++ - Meaning
  - To make sure we use the same terminology
- Modern C++ - Why?
  - To make you interested in C++

Agenda

- Modern C++ - Meaning
  - To make sure we use the same terminology
- Modern C++ - Why?
  - To make you interested in C++
- Modern C++ - What?

Agenda

- Modern C++ - Meaning
  - To make sure we use the same terminology
- Modern C++ - Why?
  - To make you interested in C++
- Modern C++ - What?
  - To give you a taste of what is possible

Agenda

- Modern C++ - Meaning
    - To make sure we use the same terminology
- Modern C++ - Why?
    - To make you interested in C++
- Modern C++ - What?
    - To give you a taste of what is possible
    - “Not your father’s C++”

Agenda

- Modern C++ - Meaning
  - To make sure we use the same terminology
- Modern C++ - Why?
  - To make you interested in C++
- Modern C++ - What?
  - To give you a taste of what is possible
  - “Not your father’s C++”
    - (Herb Sutter’s title for a presentation in Lang.NEXT 2012)

Outline

Part I

Modern C++ - Meaning

Outline

1. What it means "Modern C++"?
   - New standards
   - Not just the standards

Slow start. . .

- The language started in ’79

Slow start. . .

- The language started in '79
- Other important points in the language history:

Slow start. . .

- The language started in '79
- Other important points in the language history:
  - '85 - 1st edition of TCPL

Slow start. . .

- The language started in ’79
- Other important points in the language history:
  - ’85 - 1st edition of TCPL
  - ’87 - Support in GCC

Slow start. . .

- The language started in ’79
- Other important points in the language history:
  - ’85 - 1st edition of TCPL
  - ’87 - Support in GCC
  - ’90 - ARM - The Annotated C++ Reference Manual (de-facto standard)

Slow start. . .

- The language started in '79
- Other important points in the language history:
  - '85 - 1st edition of TCPL
  - '87 - Support in GCC
  - '90 - ARM - The Annotated C++ Reference Manual (de-facto standard)
  - '91 - ISO C++ Commitee founded

Slow start. . .

- The language started in '79
- Other important points in the language history:
  - '85 - 1st edition of TCPL
  - '87 - Support in GCC
  - '90 - ARM - The Annotated C++ Reference Manual (de-facto standard)
  - '91 - ISO C++ Commitee founded
  - '92 - STL implemented

Slow start. . .

- The language started in ’79
- Other important points in the language history:
  - ’85 - 1st edition of TCPL
  - ’87 - Support in GCC
  - ’90 - ARM - The Annotated C++ Reference Manual (de-facto standard)
  - ’91 - ISO C++ Commitee founded
  - ’92 - STL implemented
  - **’98 - C++98**

Slow start...

- The language started in '79
- Other important points in the language history:
  - '85 - 1st edition of TCPL
  - '87 - Support in GCC
  - '90 - ARM - The Annotated C++ Reference Manual (de-facto standard)
  - '91 - ISO C++ Commitee founded
  - '92 - STL implemented
  - **'98 - C++98**
  - '98 - 3rd edition of TCPL

Slow start. . .

- The language started in ’79
- Other important points in the language history:
  - ’85 - 1st edition of TCPL
  - ’87 - Support in GCC
  - ’90 - ARM - The Annotated C++ Reference Manual (de-facto standard)
  - ’91 - ISO C++ Commitee founded
  - ’92 - STL implemented
  - **’98 - C++98**
  - ’98 - 3rd edition of TCPL
  - ’99 - Boost started

Slow start. . .

- The language started in ’79
- Other important points in the language history:
    - ’85 - 1st edition of TCPL
    - ’87 - Support in GCC
    - ’90 - ARM - The Annotated C++ Reference Manual (de-facto standard)
    - ’91 - ISO C++ Commitee founded
    - ’92 - STL implemented
    - **’98 - C++98**
    - ’98 - 3rd edition of TCPL
    - ’99 - Boost started
    - ’03 - C++03

Slow start. . .

- The language started in ’79
- Other important points in the language history:
  - ’85 - 1st edition of TCPL
  - ’87 - Support in GCC
  - ’90 - ARM - The Annotated C++ Reference Manual (de-facto standard)
  - ’91 - ISO C++ Commitee founded
  - ’92 - STL implemented
  - **’98 - C++98**
  - ’98 - 3rd edition of TCPL
  - ’99 - Boost started
  - ’03 - C++03
- `http://en.cppreference.com/w/cpp/language/history`

. . . and then (almost full) stop

- Compiler support wasn’t great

. . . and then (almost full) stop

- Compiler support wasn’t great
  - nor full

. . . and then (almost full) stop

- Compiler support wasn’t great
  - nor full
- Other languages gained momentum (Java - started ’95, C# - started ’02)

. . . and then (almost full) stop

- Compiler support wasn’t great
    - nor full
- Other languages gained momentum (Java - started ’95, C# - started ’02)
- Vague promises for having C++0x

. . . and then (almost full) stop

- Compiler support wasn’t great
  - nor full
- Other languages gained momentum (Java - started ’95, C# - started ’02)
- Vague promises for having C++0x
- Nothing much meanwhile

. . . and then (almost full) stop

- Compiler support wasn't great
  - nor full
- Other languages gained momentum (Java - started '95, C# - started '02)
- Vague promises for having C++0x
- Nothing much meanwhile
  - (TR1?)

But then it started again!

- Standard

But then it started again!

- Standard
  - C++11

But then it started again!

- Standard
    - C++11
        - (C++0b?)

But then it started again!

- Standard
  - C++11
    - (C++0b?)
  - C++14

But then it started again!

- Standard
  - C++11
    - (C++0b?)
  - C++14
  - C++17

But then it started again!

- Standard
  - C++11
    - (C++0b?)
  - C++14
  - C++17
  - Many TSes

But then it started again!

- Standard
  - C++11
    - (C++0b?)
  - C++14
  - C++17
  - Many TSes
- Compiler support

But then it started again!

- Standard
  - C++11
    - (C++0b?)
  - C++14
  - C++17
  - Many TSes
- Compiler support
  - Even a completly new and shiny compiler, Clang (LLVM based)

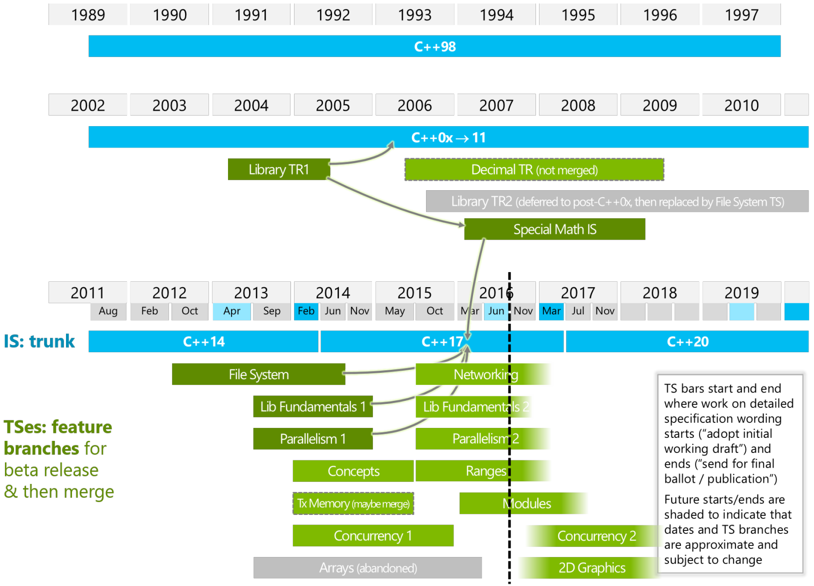
But then it started again!

- Standard
  - C++11
    - (C++0b?)
  - C++14
  - C++17
  - Many TSes
- Compiler support
  - Even a completly new and shiny compiler, Clang (LLVM based)
- Tool support

But then it started again!

- Standard
  - C++11
    - (C++0b?)
  - C++14
  - C++17
  - Many TSes
- Compiler support
  - Even a completly new and shiny compiler, Clang (LLVM based)
- Tool support
  - IDEs, static and dynamic analyzers, etc.

Perspective



TS bars start and end where work on detailed specification wording starts ("adopt initial working draft") and ends ("send for final ballot / publication")

Future starts/ends are shaded to indicate that dates and TS branches are approximate and subject to change

https://herbsutter.com/2016/06/30/trip-report-summer-iso-c-standards-meeting-oulu/

Outline

Not only 'what', it's also about the 'how'

- C++ Core Guidelines

Not only 'what', it's also about the 'how'

- C++ Core Guidelines
  - `http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines`

Not only 'what', it's also about the 'how'

- C++ Core Guidelines
  - http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
- Scott Meyers' “Effective C++” series

Not only 'what', it's also about the 'how'

- C++ Core Guidelines
  - `http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines`
- Scott Meyers' “Effective C++” series
  - Effective C++ (3rd ed.), More Effective C++, Effective STL, Effective Modern C++

Not only 'what', it's also about the 'how'

- C++ Core Guidelines
  - `http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines`
- Scott Meyers' “Effective C++” series
  - Effective C++ (3rd ed.), More Effective C++, Effective STL, Effective Modern C++
- More...

Not only 'what', it's also about the 'how'

- C++ Core Guidelines
  - `http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines`
- Scott Meyers' "Effective C++" series
  - Effective C++ (3rd ed.), More Effective C++, Effective STL, Effective Modern C++
- More...
- Generally: the common wisdom the C++ developer community collected over the years

Outline

Part II

Modern C++ - Why?

Outline

2 Wasn't C++ dead years ago?

3 Aren't managed languages all what we need?

The Free Lunch Is Over

- In '05, Herb Sutter wrote an article titled "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software"

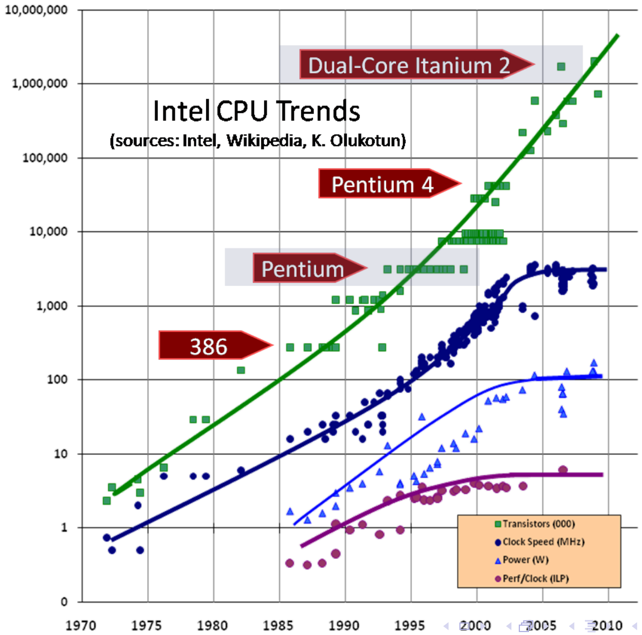
The Free Lunch Is Over

- In '05, Herb Sutter wrote an article titled "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software"
  - `http://www.gotw.ca/publications/concurrency-ddj.htm`

The Free Lunch Is Over

- In '05, Herb Sutter wrote an article titled "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software"
  - `http://www.gotw.ca/publications/concurrency-ddj.htm`
- Key observation was simply looking at the following graph:

CPU graph



Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Dual-Core Itanium 2
Pentium 4
Pentium
386
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
0
1970
1975
1980
1985
1990
1995
2000
2005
2010
■ Transistors (000)
● Clock Speed (MHz)
▲ Power (W)
● Perf/Clock (ILP)

Why C++

- No free lunch

Why C++

- No free lunch
- One conclusion is that concurrency and parallelism became very important to utilize the CPU advancement

Why C++

- No free lunch
- One conclusion is that concurrency and parallelism became very important to utilize the CPU advancement
- Another conclusion, as Sutter explained in "C++ and Beyond 2011", is that we care again about performance

Why C++

- No free lunch
- One conclusion is that concurrency and parallelism became very important to utilize the CPU advancement
- Another conclusion, as Sutter explained in “C++ and Beyond 2011”, is that we care again about performance
- Performance per Watt

Why C++

- No free lunch
- One conclusion is that concurrency and parallelism became very important to utilize the CPU advancement
- Another conclusion, as Sutter explained in "C++ and Beyond 2011", is that we care again about performance
- Performance per Watt
    - Both mobile and datacenters

Why C++

- No free lunch
- One conclusion is that concurrency and parallelism became very important to utilize the CPU advancement
- Another conclusion, as Sutter explained in "C++ and Beyond 2011", is that we care again about performance
- Performance per Watt
    - Both mobile and datacenters
- Performance per Size

Why C++

- No free lunch
- One conclusion is that concurrency and parallelism became very important to utilize the CPU advancement
- Another conclusion, as Sutter explained in “C++ and Beyond 2011”, is that we care again about performance
- Performance per Watt
  - Both mobile and datacenters
- Performance per Size
- Performace per Cycle

Why C++

- No free lunch
- One conclusion is that concurrency and parallelism became very important to utilize the CPU advancement
- Another conclusion, as Sutter explained in “C++ and Beyond 2011”, is that we care again about performance
- Performance per Watt
    - Both mobile and datacenters
- Performance per Size
- Performace per Cycle
    - Do more with the same for better experience

Outline

Aren't managed languages all what we need?

Aren't managed languages all what we need?

- Nope

More seriously

- Depends on what we are optimizing for

More seriously

- Depends on what we are optimizing for
- The usual trade-off (no, not space/time trade-off)

More seriously

- Depends on what we are optimizing for
- The usual trade-off (no, not space/time trade-off)
- Developer/development time - Managed

More seriously

- Depends on what we are optimizing for
- The usual trade-off (no, not space/time trade-off)
- Developer/development time - Managed
- Performance - Unmanaged

More seriously

- Depends on what we are optimizing for
- The usual trade-off (no, not space/time trade-off)
- Developer/development time - Managed
- Performance - Unmanaged
    - (Compatibility is also a big win for C++, but that isn't new)

More seriously

- Depends on what we are optimizing for
- The usual trade-off (no, not space/time trade-off)
- Developer/development time - Managed
- Performance - Unmanaged
  - (Compatibility is also a big win for C++, but that isn't new)
- What it means Managed / Unmanaged?

Outline

Part III

Modern C++ - What?

Outline

What are the main pain points you have about C++?

- Here are (some of) mine:

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
    - Hard-to-spell long template instantiation types

```
 const std::set<std::map<std::string, std::pair<
int, std::vector<double>>>::iterator>::const_iterator
```

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types

```
const std::set<std::map<std::string, std::pair<
int, std::vector<double>>>::iterator>::const_iterator
```

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
    - Hard-to-spell long template instantiation types

```
const std::set<std::map<std::string, std::pair<
int, std::vector<double>>>::iterator>::const_iterator
```

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types

```
    const std::set<std::map<std::string, std::pair<
int, std::vector<double>>>::iterator>::const_iterator
```

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types

```
    const std::set<std::map<std::string, std::pair<
int, std::vector<double>>>::iterator>::const_iterator
```

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types

```
    const std::set<std::map<std::string, std::pair<
int, std::vector<double>>>::iterator>::const_iterator
```

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
    - Hard-to-spell long template instantiation types
    - Writing explicit loops

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types
  - Writing explicit loops
    - Example: 01-ExplicitLoops.cpp

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types
  - Writing explicit loops
    - Example: 01-ExplicitLoops.cpp
  - Inconsistency (e.g. initializing struct vs. class vs. array vs. container)

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
    - Hard-to-spell long template instantiation types
    - Writing explicit loops
        - Example: 01-ExplicitLoops.cpp
    - Inconsistency (e.g. initializing struct vs. class vs. array vs. container)
        - Example: 02-InitDifferences.cpp (first part)

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types
  - Writing explicit loops
    - Example: 01-ExplicitLoops.cpp
  - Inconsistency (e.g. initializing struct vs. class vs. array vs. container)
    - Example: 02-InitDifferences.cpp (first part)
- Safety issues:

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types
  - Writing explicit loops
    - Example: 01-ExplicitLoops.cpp
  - Inconsistency (e.g. initializing struct vs. class vs. array vs. container)
    - Example: 02-InitDifferences.cpp (first part)
- Safety issues:
  - Pointers are dangerous

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types
  - Writing explicit loops
    - Example: 01-ExplicitLoops.cpp
  - Inconsistency (e.g. initializing struct vs. class vs. array vs. container)
    - Example: 02-InitDifferences.cpp (first part)
- Safety issues:
  - Pointers are dangerous
  - Explicit memory handling (no garbage collector)

What are the main pain points you have about C++?

- Here are (some of) mine:
- Usability issues:
  - Hard-to-spell long template instantiation types
  - Writing explicit loops
    - Example: 01-ExplicitLoops.cpp
  - Inconsistency (e.g. initializing struct vs. class vs. array vs. container)
    - Example: 02-InitDifferences.cpp (first part)
- Safety issues:
  - Pointers are dangerous
  - Explicit memory handling (no garbage collector)
  - Implicit conversions

Outline

Easier-to-use added features

- We'll present here a few features of modern C++

Easier-to-use added features

- We'll present here a few features of modern C++
- Most of them made the language easier to use

Easier-to-use added features

- We'll present here a few features of modern C++
- Most of them made the language easier to use
- Usually they improve safety too

Easier-to-use added features

- We'll present here a few features of modern C++
- Most of them made the language easier to use
- Usually they improve safety too
- Sometimes they also improve performance (but in no case hurt performance)

auto

- auto

auto

- auto
- AAA - Almost Always Auto

auto

- auto
- AAA - Almost Always Auto
  - (Improved in C++17 with copy elision guarantee and not deducing initializer-list)

auto

- auto
- AAA - Almost Always Auto
  - (Improved in C++17 with copy elision guarantee and not deducing initializer-list)
- It's also about performance

auto

- auto
- AAA - Almost Always Auto
  - (Improved in C++17 with copy elision guarantee and not deducing initializer-list)
- It's also about performance
  - Prevents hidden implicit copy (Effective Modern C++, Scott Meyers)

auto

- auto
- AAA - Almost Always Auto
  - (Improved in C++17 with copy elision guarantee and not deducing initializer-list)
- It's also about performance
  - Prevents hidden implicit copy (Effective Modern C++, Scott Meyers)

```
std::map<std::string, int> m;

// Fill with data ...

for (auto i = m.begin(); i != m.end(); ++i)

{

    const std::pair<std::string, int>& entry = *i;

    // Work with entry

}
```

auto

- auto
- AAA - Almost Always Auto
  - (Improved in C++17 with copy elision guarantee and not deducing initializer-list)
- It's also about performance
  - Prevents hidden implicit copy (Effective Modern C++, Scott Meyers)
- It's actually also about safety

auto

- auto
- AAA - Almost Always Auto
  - (Improved in C++17 with copy elision guarantee and not deducing initializer-list)
- It's also about performance
  - Prevents hidden implicit copy (Effective Modern C++, Scott Meyers)
- It's actually also about safety
  - Better maintainability, esp. with refactoring; better generic code

auto

- auto
- AAA - Almost Always Auto
  - (Improved in C++17 with copy elision guarantee and not deducing initializer-list)
- It's also about performance
  - Prevents hidden implicit copy (Effective Modern C++, Scott Meyers)
- It's actually also about safety
  - Better maintainability, esp. with refactoring; better generic code
- (We'll see it in use in the following code examples)

Better loops

- Range-based for loops

Better loops

- Range-based for loops
  - Example: 01-ExplicitLoops-RangeBased.cpp

Better loops

- Range-based for loops
  - Example: 01-ExplicitLoops-RangeBased.cpp
- Algorithms are better (loops with name and explicit intention) - but harder to use!

Better loops

- Range-based for loops
  - Example: 01-ExplicitLoops-RangeBased.cpp
- Algorithms are better (loops with name and explicit intention) - but harder to use!
  - Example: 01-ExplicitLoops-Algorithms.cpp

Better loops

- Range-based for loops
  - Example: 01-ExplicitLoops-RangeBased.cpp
- Algorithms are better (loops with name and explicit intention) - but harder to use!
  - Example: 01-ExplicitLoops-Algorithms.cpp
- Lambda functions - make algorithms usable

Better loops

- Range-based for loops
  - Example: 01-ExplicitLoops-RangeBased.cpp
- Algorithms are better (loops with name and explicit intention) - but harder to use!
  - Example: 01-ExplicitLoops-Algorithms.cpp
- Lambda functions - make algorithms usable
  - Example: 01-ExplicitLoops-Lambdas.cpp

Better initialization

- Brace initializers (e.g. no discrimination against init of member structs and arrays; init of containers)

Better initialization

- Brace initializers (e.g. no discrimination against init of member structs and arrays; init of containers)
- In-class initializers

Better initialization

- Brace initializers (e.g. no discrimination against init of member structs and arrays; init of containers)
- In-class initializers
- Example: 02-InitDifferences.cpp (second part)

Better information passing

- std::tuple

Better information passing

- std::tuple
    - (C# is catching up with C#7)

Better information passing

- std::tuple
    - (C# is catching up with C#7)
- Further expanded by structured binding in C++17

Outline

One safety mechanism to rule them all

- RAII

One safety mechanism to rule them all

- RAII
- Solves both resource safety issues and exception safety issues (e.g. auto reset a flag)

One safety mechanism to rule them all

- RAII
- Solves both resource safety issues and exception safety issues (e.g. auto reset a flag)
- Smart pointers

One safety mechanism to rule them all

- RAII
- Solves both resource safety issues and exception safety issues (e.g. auto reset a flag)
- Smart pointers
  - Example: SmartPointers.sln

One safety mechanism to rule them all

- RAII
- Solves both resource safety issues and exception safety issues (e.g. auto reset a flag)
- Smart pointers
    - Example: SmartPointers.sln
    - Example: RAII

One safety mechanism to rule them all

- RAII
- Solves both resource safety issues and exception safety issues (e.g. auto reset a flag)
- Smart pointers
    - Example: SmartPointers.sln
    - Example: RAII
- More RAII tools (STL, fstream, lock_guard)

One safety mechanism to rule them all

- RAII
- Solves both resource safety issues and exception safety issues (e.g. auto reset a flag)
- Smart pointers
    - Example: SmartPointers.sln
    - Example: RAII
- More RAII tools (STL, fstream, lock_guard)
- Scope guard

One safety mechanism to rule them all

- RAII
- Solves both resource safety issues and exception safety issues (e.g. auto reset a flag)
- Smart pointers
    - Example: SmartPointers.sln
    - Example: RAII
- More RAII tools (STL, fstream, lock_guard)
- Scope guard
    - Example: ScopeGuard

Another interesting safety mechanism

- Mars land vehicle story

Another interesting safety mechanism

- Mars land vehicle story
  - `https://en.wikipedia.org/wiki/Mars_Climate_Orbiter`

Another interesting safety mechanism

- Mars land vehicle story
  - `https://en.wikipedia.org/wiki/Mars_Climate_Orbiter`
- UDLs

Another interesting safety mechanism

- Mars land vehicle story
  - `https://en.wikipedia.org/wiki/Mars_Climate_Orbiter`
- UDLs
  - Bjarne Stroustrup's talk, slides 18-23

Another interesting safety mechanism

- Mars land vehicle story
  - `https://en.wikipedia.org/wiki/Mars_Climate_Orbiter`
- UDLs
  - Bjarne Stroustrup's talk, slides 18-23
  - Example: UDLs

Outline

Part IV

Summary

Summary

- We discussed:

Summary

- We discussed:
    - Why

Summary

- We discussed:
  - Why
  - (A bit of) What

Outline

Part V

References

References

- C++ Today: The Beast Is Back; Gašper Ažman, Jon Kalb; `http://www.oreilly.com/programming/free/c++-today.csp`

References

- C++ Today: The Beast Is Back; Gašper Ažman, Jon Kalb; `http://www.oreilly.com/programming/free/c++-today.csp`
- Elements of Modern C++ Style; Herb Sutter; `https://herbsutter.com/elements-of-modern-c-style/`

References

- C++ Today: The Beast Is Back; Gašper Ažman, Jon Kalb; http://www.oreilly.com/programming/free/c++-today.csp
- Elements of Modern C++ Style; Herb Sutter; https://herbsutter.com/elements-of-modern-c-style/
- C++ Super-FAQ; https://isocpp.org/faq

References

- C++ Today: The Beast Is Back; Gašper Ažman, Jon Kalb; `http://www.oreilly.com/programming/free/c++-today.csp`
- Elements of Modern C++ Style; Herb Sutter; `https://herbsutter.com/elements-of-modern-c-style/`
- C++ Super-FAQ; `https://isocpp.org/faq`
- C++ reference; `http://cppreference.com`

References

- C++ Today: The Beast Is Back; Gašper Ažman, Jon Kalb; http://www.oreilly.com/programming/free/c++-today.csp
- Elements of Modern C++ Style; Herb Sutter; https://herbsutter.com/elements-of-modern-c-style/
- C++ Super-FAQ; https://isocpp.org/faq
- C++ reference; http://cppreference.com
- C++ Core Guidelines; http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines

References

- C++ Today: The Beast Is Back; Gašper Ažman, Jon Kalb; `http://www.oreilly.com/programming/free/c++-today.csp`
- Elements of Modern C++ Style; Herb Sutter; `https://herbsutter.com/elements-of-modern-c-style/`
- C++ Super-FAQ; `https://isocpp.org/faq`
- C++ reference; `http://cppreference.com`
- C++ Core Guidelines; `http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines`
- Ask me for more references, articles, books and videos

References

- C++ Today: The Beast Is Back; Gašper Ažman, Jon Kalb; `http://www.oreilly.com/programming/free/c++-today.csp`
- Elements of Modern C++ Style; Herb Sutter; `https://herbsutter.com/elements-of-modern-c-style/`
- C++ Super-FAQ; `https://isocpp.org/faq`
- C++ reference; `http://cppreference.com`
- C++ Core Guidelines; `http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines`
- Ask me for more references, articles, books and videos
  - (or just google it)