

C++ Foundation



Introduction to C++

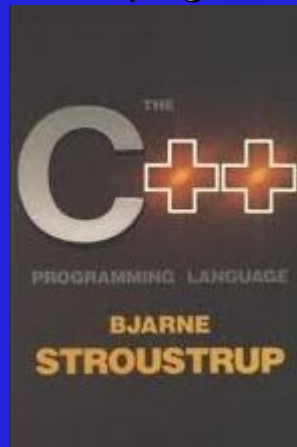
Introduction to C++

- history and use
- sequence points
- evaluation order
- undefined behaviour
- where and why to use C++
- spirit of C++
- hello C++

History

- The 3 ages of C++ correspond to editions of The C++ Programming Language book
 - 1st age, 1985
 - 2nd age, 1991
 - 3rd age, 1997
 - 4th age, ????
- C++0x (sic)
 - promises many new language features and libraries

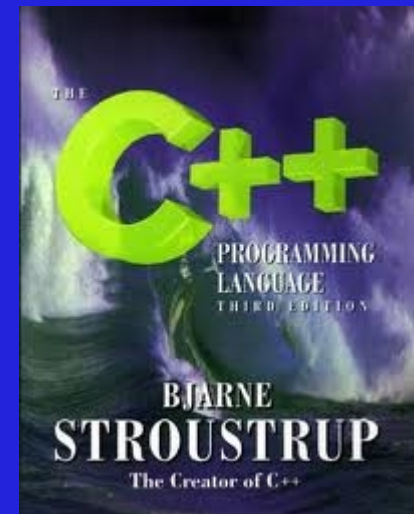
331 pages



720 pages



1040 pages



C/C++ Use

- C and C++ continue to be popular languages
- Tiobe index, language ranking
 - <http://www.tiobe.com>

	1986	1996	2006	2011
C++	8	3	3	3
C	1	1	2	2

Undefined Behaviour

- Conformance
 - If a “shall” or “shall not” requirement... is violated the behavior is undefined. (4)
- Undefined behavior
 - Behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirement. (3.4.3)

Evaluation Order

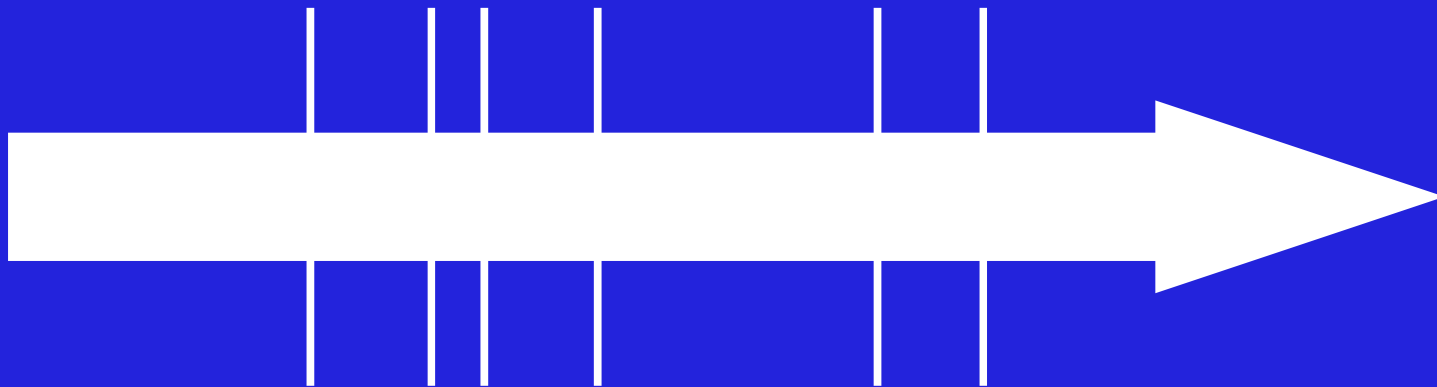
- Evaluation order is not always determined by the layout order of the code
 - statements, initialisers, operands of the short-circuiting and comma operators are evaluated in strict sequence... and that's pretty much it
 - can affect expression certainty and correctness

```
a = f() * g() + h();  
a = f() * (g() + h());
```

there is no required difference in evaluation order between these two assignments

Sequence Points

- At certain specified points in the execution sequence called sequence points, all side effects of previous evaluations shall be complete and no side effects of subsequent evaluations shall have taken place (5.1.2.3)



Sequence Points

- sequence points occur...
 - at the end of a full expression
 - after the first operand of these operators
 - && logical and
 - || logical or
 - ?: ternary
 - , comma
 - after evaluation of all arguments and function expressions in a function call
 - at the end of an initializer
 - at the end of a full declarator
 - that's it!

Sequence Point Rules

- Rule 1
 - Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression (6.5)
- Rule 2
 - Between the previous and next sequence point... the prior value shall be read only to determine the value to be stored (6.5)

```
n = n++
```

Violates rule 1 - undefined

```
n + n++
```

Violates rule 2 - undefined

Multi Paradigm?

- C++ supports many many different programming styles
 - traditional C style
 - data abstraction
 - operator overloading
 - object oriented
 - generic programming (templates)
 - multiple inheritance
 - exceptional control flow
- Very very easy to make a big bad mess
 - keep it simple, focus on a few styles only

Comparing Languages

- Where to use C++?
 - only C and C++ compiler is available
- Why use C++?
 - you need a higher level of abstraction than C
- What to compare C++ with?
 - Compare with Java, C#, etc
 - Don't compare C++ with C

Spirit of C++

- trust the programmer
- speed trumps portability
- stay close to the hardware
- be as close as possible to C, but no closer
- don't pay for what you don't use
- catch errors at compile time
- avoid the preprocessor
- you can write the library in the language



Hello C++

- traditional first program

The diagram shows a C++ program snippet on a yellow background. Annotations with arrows point to specific parts of the code:

- preprocessor #include (note no .h)**: Points to the `#include <iostream>` line.
- << streaming operator**: Points to the `<<` operator in the `std::cout << "Hello, world\n";` line.
- "string literal"**: Points to the `"Hello, world\n"` string in the `std::cout << "Hello, world\n";` line.
- std namespace**: Points to the `std` part of `std::cout` in the `std::cout << "Hello, world\n";` line.
- :: scope resolution operator**: Points to the `::` in `std::cout` in the `std::cout << "Hello, world\n";` line.

```
#include <iostream>

int main()
{
    std::cout << "Hello, world\n";
}
```

Key



compiles



compiles but questionable



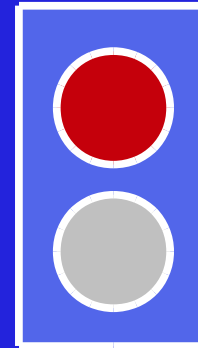
doesn't compile



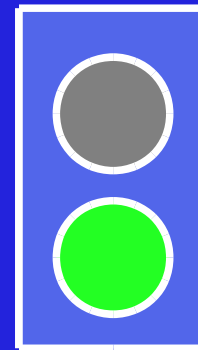
a bug



a note



tests fail



tests pass

Click to add title

- Click to add an outline

C++ Foundation



Introduction to C++

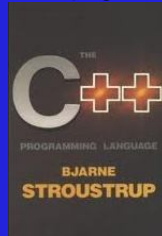
Introduction to C++

- history and use
- sequence points
- evaluation order
- undefined behaviour
- where and why to use C++
- spirit of C++
- hello C++

History

- The 3 ages of C++ correspond to editions of The C++ Programming Language book
- 1st age, 1985
- 2nd age, 1991
- 3rd age, 1997
- 4th age, ????
 - C++0x (sic)
 - promises many new language features and libraries

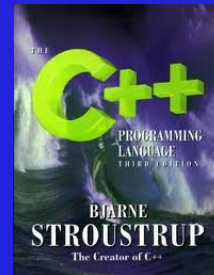
331 pages



720 pages



1040 pages



C/C++ Use

- C and C++ continue to be popular languages
- Tiobe index, language ranking
 - <http://www.tiobe.com>

	1986	1996	2006	2011
C++	8	3	3	3
C	1	1	2	2

Undefined Behaviour

- Conformance
 - If a “shall” or “shall not” requirement... is violated the behavior is undefined. (4)
- Undefined behavior
 - Behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirement. (3.4.3)

Evaluation Order

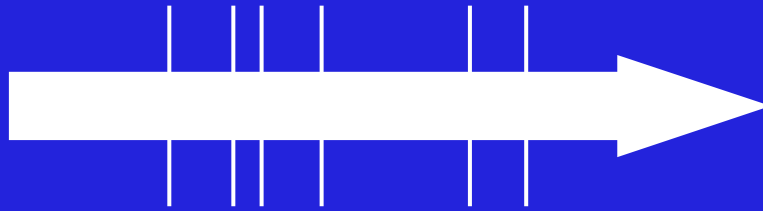
- Evaluation order is not always determined by the layout order of the code
 - statements, initialisers, operands of the short-circuiting and comma operators are evaluated in strict sequence... and that's pretty much it
 - can affect expression certainty and correctness

```
a = f() * g() + h();  
a = f() * (g() + h());
```

there is no required difference in evaluation order between these two assignments

Sequence Points

- At certain specified points in the execution sequence called sequence points, all side effects of previous evaluations shall be complete and no side effects of subsequent evaluations shall have taken place (5.1.2.3)



Sequence Points

- sequence points occur...
 - at the end of a full expression
 - after the first operand of these operators
 - && logical and
 - || logical or
 - ?: ternary
 - , comma
 - after evaluation of all arguments and function expressions in a function call
 - at the end of an initializer
 - at the end of a full declarator
 - that's it!

Sequence Point Rules

- Rule 1
 - Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression (6.5)
- Rule 2
 - Between the previous and next sequence point... the prior value shall be read only to determine the value to be stored (6.5)

```
n = n++
```

Violates rule 1 - undefined

```
n + n++
```

Violates rule 2 - undefined

Multi Paradigm?

- C++ supports many many different programming styles
 - traditional C style
 - data abstraction
 - operator overloading
 - object oriented
 - generic programming (templates)
 - multiple inheritance
 - exceptional control flow
- Very very easy to make a big bad mess
 - keep it simple, focus on a few styles only

Comparing Languages

- Where to use C++?
 - only C and C++ compiler is available
- Why use C++?
 - you need a higher level of abstraction than C
- What to compare C++ with?
 - Compare with Java, C#, etc
 - Don't compare C++ with C

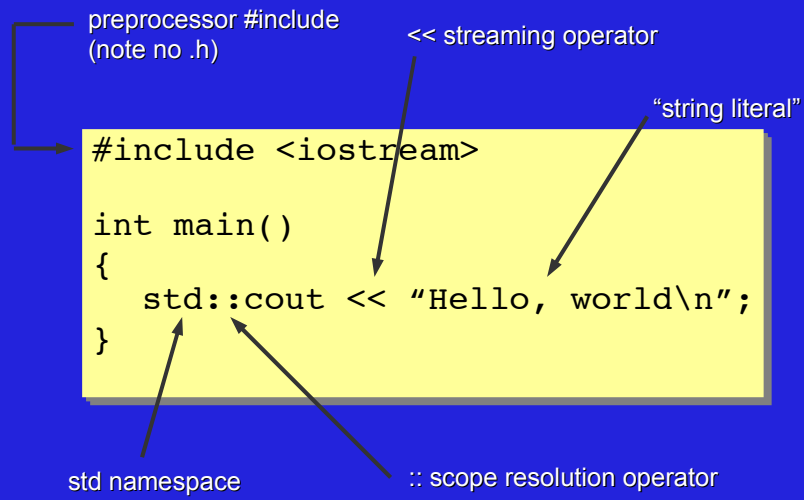
Spirit of C++

- trust the programmer
- speed trumps portability
- stay close to the hardware
- be as close as possible to C, but no closer
- don't pay for what you don't use
- catch errors at compile time
- avoid the preprocessor
- you can write the library in the language



Hello C++

- traditional first program



The diagram shows a C++ program snippet on a yellow background. Five arrows point from text labels to specific parts of the code: 'preprocessor #include (note no .h)' points to '#include <iostream>'; '<< streaming operator' points to '<<'; 'string literal' points to '"Hello, world\\n"'; 'std namespace' points to 'std'; and ':: scope resolution operator' points to '::'. The code is as follows:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world\\n";
}
```

