(R)evolution of C++

aka The Hitchhiker's Guide to C++

Łukasz Ziobroń

lukasz@ziobron.net

http://ziobron.net

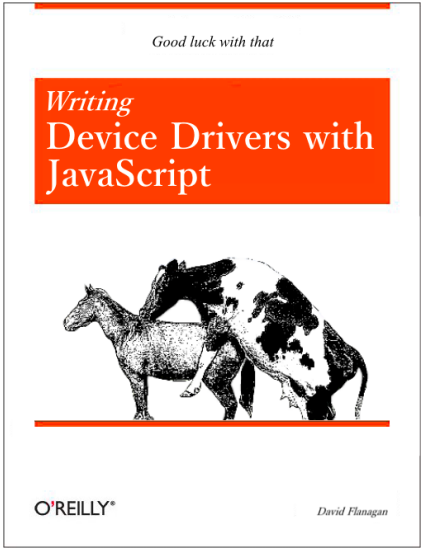
code::dive, 2016-11-15

About the author

About the author

About the author

(R)evolution of JS



About the author

About the author

C/C++

About the author

About the author

About the author

About the author

Interests:

- Archery
- Digital photography
- Machine learning
- Image processing
- High tech
- Starcraft
- Blogging at `ziobron.net`

Key messages

1. C++ had a **clear aim**, which made it popular: to **organize code better without the loss of efficiency**

Key messages

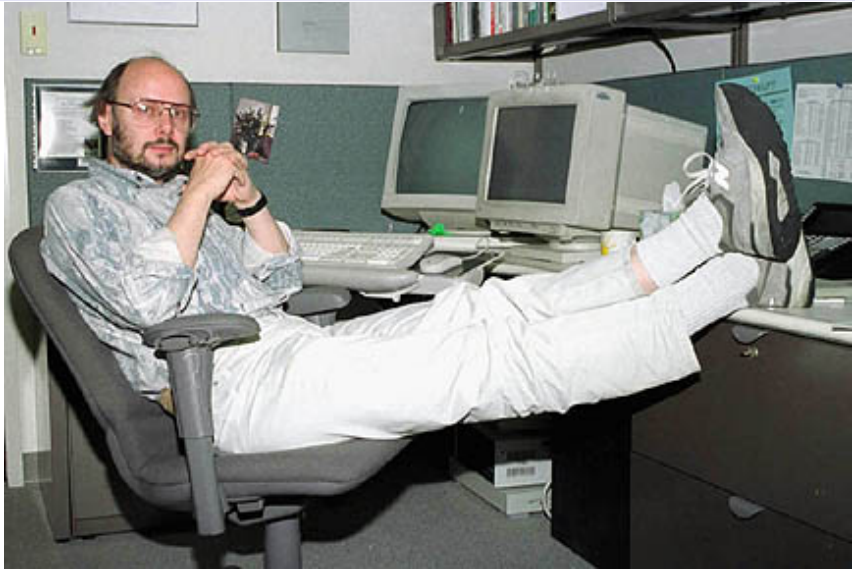
1. C++ had a **clear aim**, which made it popular: to **organize code better without the loss of efficiency**
2. C++ is even more popular now, because of new standards: **C++11** and **C++14**

Key messages

1. C++ had a **clear aim**, which made it popular: to **organize code better without the loss of efficiency**
2. C++ is even more popular now, because of new standards: **C++11** and **C++14**
3. In future C++ will be **one of the most popular programming languages** so it's worth learning

Agenda

C with Classes



Roots of C++

Domain-specific abstraction

Fortran

Cobol

General-purpose abstraction

Simula

Java

Direct mapping to hardware

Assembler

BCPL

C

C++

C++11

C++14

C#

Roots of C++

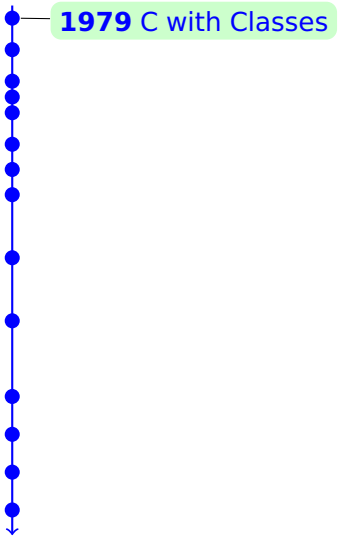
Languages that were considered as a base of C++:

- Modula2
- Ada
- Smalltalk
- Mesa
- Clu
- **C**

C with Classes

Additions to C language:

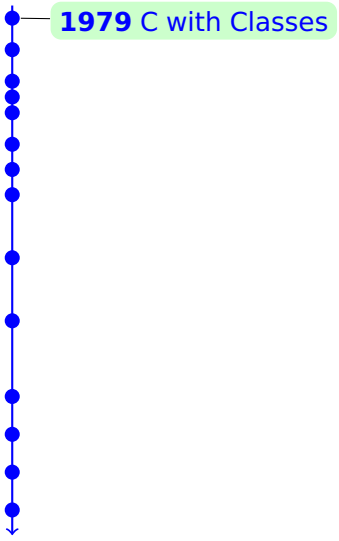
- classes



C with Classes

Additions to C language:

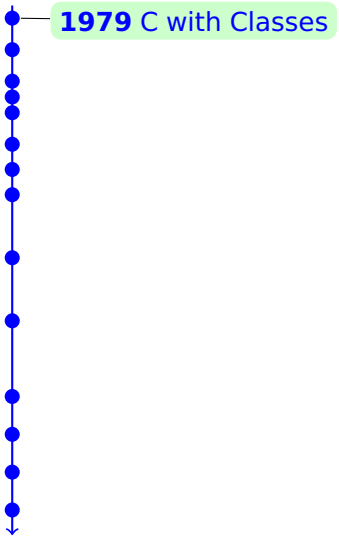
- classes
- derived classes



C with Classes

Additions to C language:

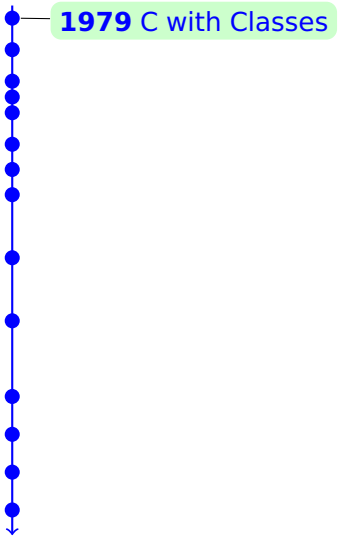
- classes
- derived classes
- public and private access control



C with Classes

Additions to C language:

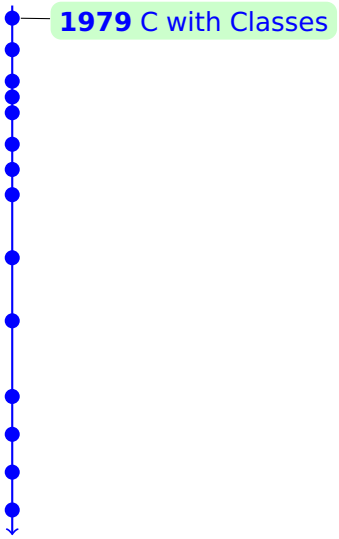
- classes
- derived classes
- public and private access control
- constructors and destructors



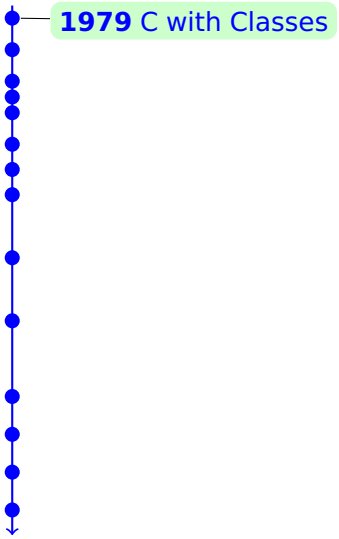
C with Classes

Additions to C language:

- classes
- derived classes
- public and private access control
- constructors and destructors
- call and return functions (removed later)



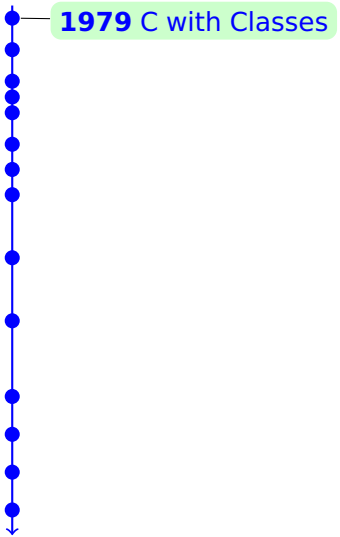
C with Classes



Additions to C language:

- classes
- derived classes
- public and private access control
- constructors and destructors
- call and return functions (removed later)
- friend classes

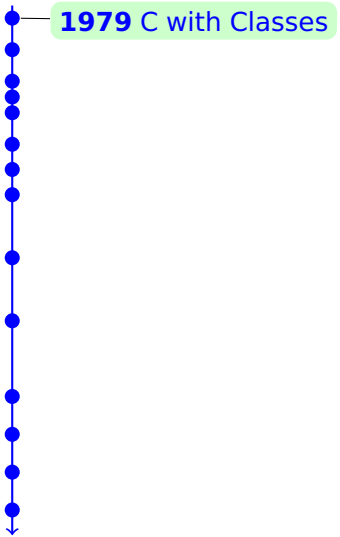
C with Classes



Additions to C language:

- classes
- derived classes
- public and private access control
- constructors and destructors
- call and return functions (removed later)
- friend classes
- type checking and conversion of function arguments

C with Classes



Additions to C language:

- classes
- derived classes
- public and private access control
- constructors and destructors
- **call and return functions (removed later)**
- friend classes
- type checking and conversion of function arguments

Example code in C with Classes

```
1  class stack {
2      char s[SIZE]; /* array of characters */
3      char * min;   /* pointer to bottom of stack */
4      char * top;   /* pointer to top of stack */
5      char * max;   /* pointer to top of allocated space */
6      void new();   /* initialization function (constructor) */
7      public:
8      void push(char);
9      char pop();
10 };
11 char stack.pop()    // member functions always defined "elsewhere", not in class definition
12 {                   // dot operator instead of double colon
13     if (top <= min) error("stack underflow");
14     return *(--top);
15 }
16 class stack s1, s2; /* two variables of class stack, class was mandatory for declaration */
17 class stack * p1 = &s2; /* p1 points to s2 */
18 class stack * p2 = new stack; /* p2 points to stack object allocated on free store */
19 s1.push('h'); /* use object directly */
20 p1->push('s'); /* use object through pointer */
```

Example code in C with Classes

```
1  class stack {
2      char s[SIZE]; /* array of characters */
3      char * min;   /* pointer to bottom of stack */
4      char * top;   /* pointer to top of stack */
5      char * max;   /* pointer to top of allocated space */
6      void new();   /* initialization function (constructor) */
7      public:
8      void push(char);
9      char pop();
10 };
11 char stack.pop()    // member functions always defined "elsewhere", not in class definition
12 {                   // dot operator instead of double colon
13     if (top <= min) error("stack␣underflow");
14     return *(--top);
15 }
16 class stack s1, s2; /* two variables of class stack, class was mandatory for declaration */
17 class stack * p1 = &s2; /* p1 points to s2 */
18 class stack * p2 = new stack; /* p2 points to stack object allocated on free store */
19 s1.push('h'); /* use object directly */
20 p1->push('s'); /* use object through pointer */
```

Example code in C with Classes

```
1  class stack {
2      char s[SIZE]; /* array of characters */
3      char * min;   /* pointer to bottom of stack */
4      char * top;   /* pointer to top of stack */
5      char * max;   /* pointer to top of allocated space */
6      void new();   /* initialization function (constructor) */
7      public:
8      void push(char);
9      char pop();
10 };
11 char stack.pop()    // member functions always defined "elsewhere", not in class definition
12 {                   // dot operator instead of double colon
13     if (top <= min) error("stack underflow");
14     return *(--top);
15 }
16 class stack s1, s2; /* two variables of class stack, class was mandatory for declaration */
17 class stack * p1 = &s2; /* p1 points to s2 */
18 class stack * p2 = new stack; /* p2 points to stack object allocated on free store */
19 s1.push('h'); /* use object directly */
20 p1->push('s'); /* use object through pointer */
```

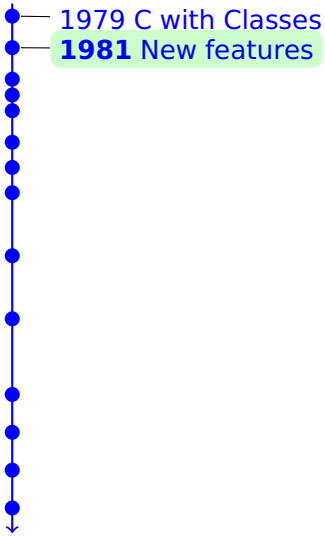
Example code in C with Classes

```
1  class stack {
2      char s[SIZE]; /* array of characters */
3      char * min;   /* pointer to bottom of stack */
4      char * top;   /* pointer to top of stack */
5      char * max;   /* pointer to top of allocated space */
6      void new();   /* initialization function (constructor) */
7      public:
8      void push(char);
9      char pop();
10 };
11 char stack.pop()    // member functions always defined "elsewhere", not in class definition
12 {                   // dot operator instead of double colon
13     if (top <= min) error("stack underflow");
14     return *(--top);
15 }
16 class stack s1, s2; /* two variables of class stack, class was mandatory for declaration */
17 class stack * p1 = &s2; /* p1 points to s2 */
18 class stack * p2 = new stack; /* p2 points to stack object allocated on free store */
19 s1.push('h'); /* use object directly */
20 p1->push('s'); /* use object through pointer */
```

C with Classes

New features added in 1981:

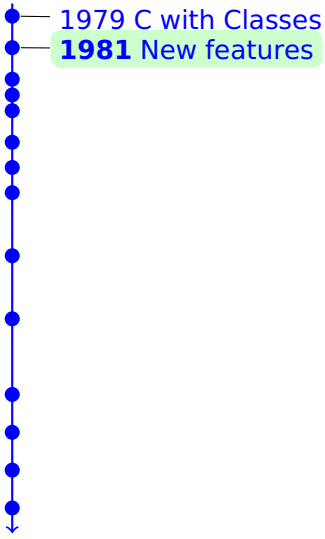
- inline functions



C with Classes

New features added in 1981:

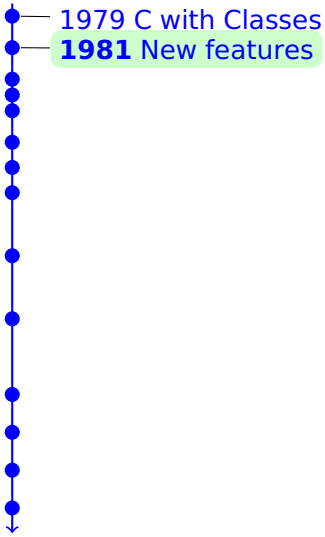
- inline functions
- default arguments



C with Classes

New features added in 1981:

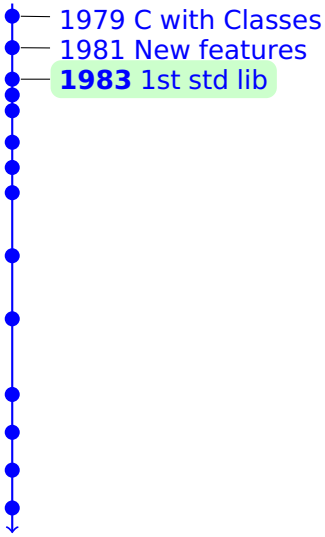
- inline functions
- default arguments
- overloading of the assignment operator



First standard library

First elements in std lib:

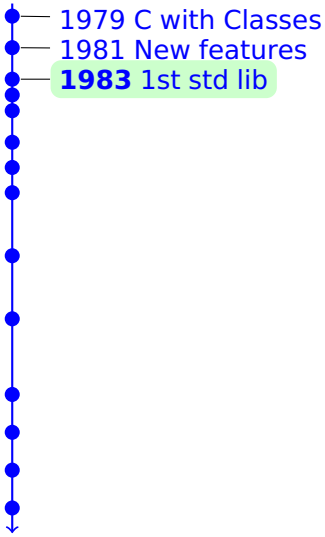
- complex numbers



First standard library

First elements in std lib:

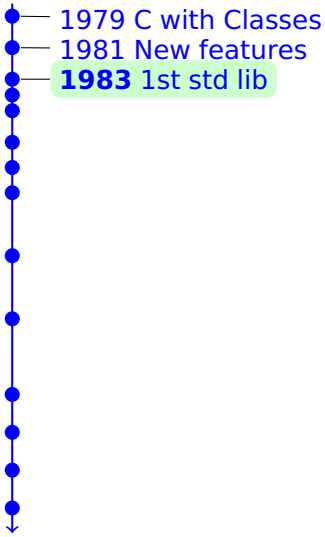
- complex numbers
- string



First standard library

First elements in std lib:

- complex numbers
- string
- later: iostreams



C with Classes - summary

- Years of development: 1979-1983

C with Classes - summary

- Years of development: 1979-1983
- The idea was great

C with Classes - summary

- Years of development: 1979-1983
- The idea was great
- The aim was clear:

C with Classes - summary

- Years of development: 1979-1983
- The idea was great
- The aim was clear:
  - help programmers to organize code with classes

C with Classes - summary

- Years of development: 1979-1983
- The idea was great
- The aim was clear:
  - help programmers to organize code with classes
  - without the loss of efficiency

C with Classes - summary

- Years of development: 1979-1983
- The idea was great
- The aim was clear:
  - help programmers to organize code with classes
  - without the loss of efficiency
  - and without requiring from users learning something completely new

C with Classes - summary

- Years of development: 1979-1983
- The idea was great
- The aim was clear:
  - help programmers to organize code with classes
  - without the loss of efficiency
  - and without requiring from users learning something completely new
- C with Classes didn’t have many users :(

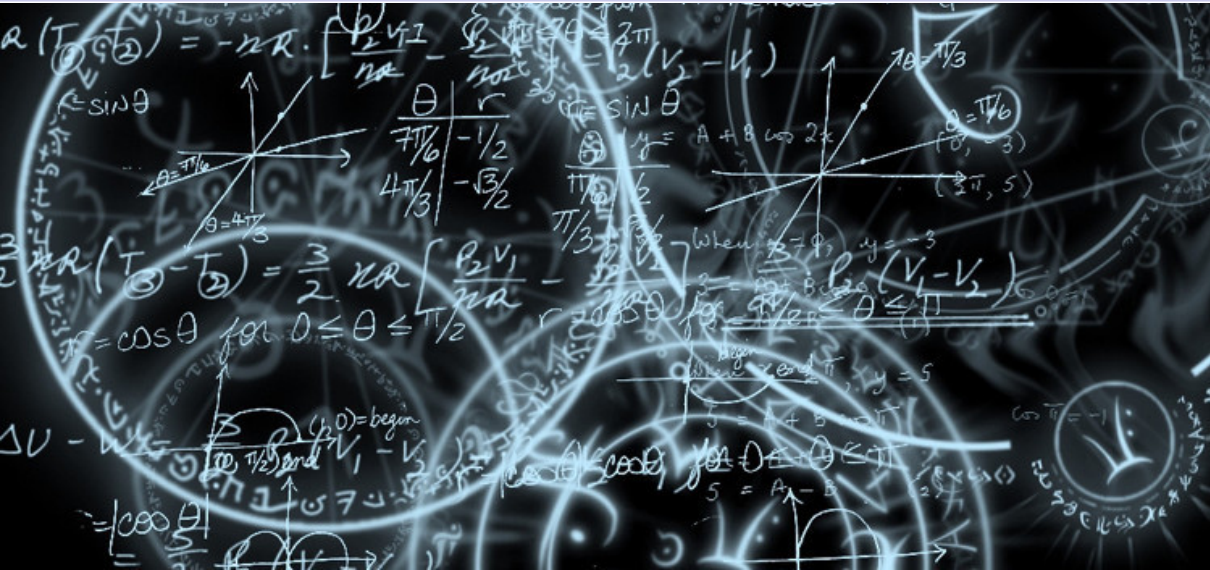
C with Classes - summary

- Years of development: 1979-1983
- The idea was great
- The aim was clear:
  - help programmers to organize code with classes
  - without the loss of efficiency
  - and without requiring from users learning something completely new
- C with Classes didn’t have many users :(
- It wouldn’t pay to support this language in the form it was

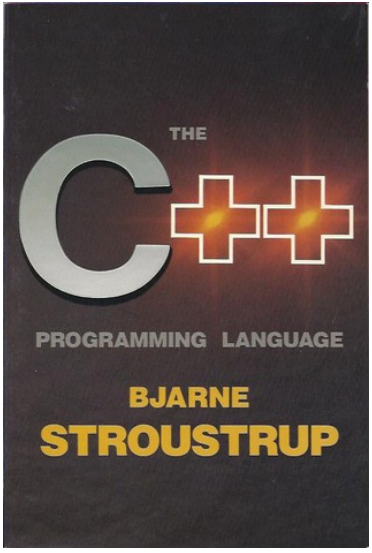
C with Classes - summary

- Years of development: 1979-1983
- The idea was great
- The aim was clear:
  - help programmers to organize code with classes
  - without the loss of efficiency
  - and without requiring from users learning something completely new
- C with Classes didn’t have many users :(
- It wouldn’t pay to support this language in the form it was
- C with Classes was a "medium success"

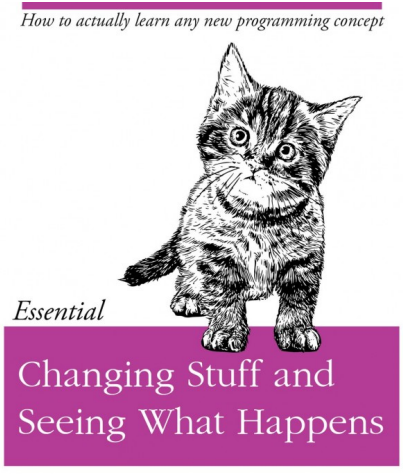
Cfront era



The C++ Programming language - 1st edition (1983)



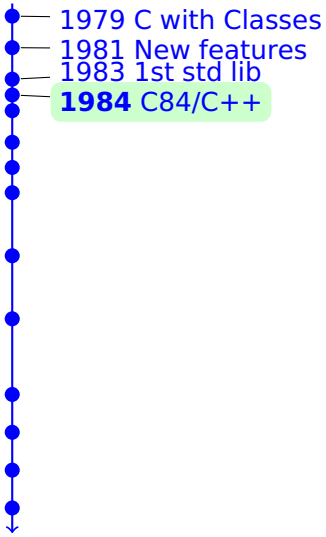
The C++ Programming language - 1st edition (1983)



The C++ Programming language - 1st edition (1983)

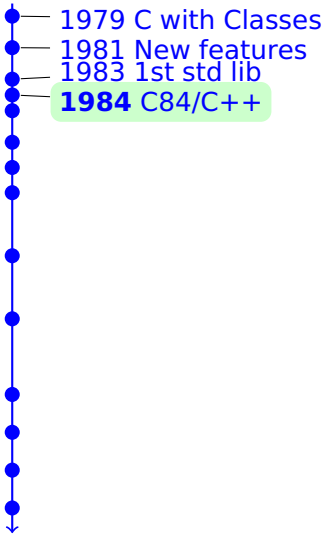
C84 and Cfront

- C with Classes got new name - C84



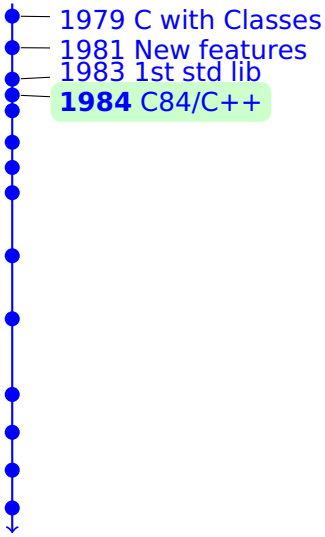
C84 and Cfront

- C with Classes got new name - C84
- A few months later C84 got a new name - C++



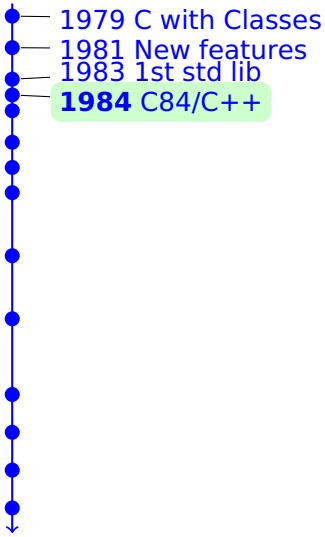
C84 and Cfront

- C with Classes got new name - C84
- A few months later C84 got a new name - C++
- First C++ compiler - Cfront



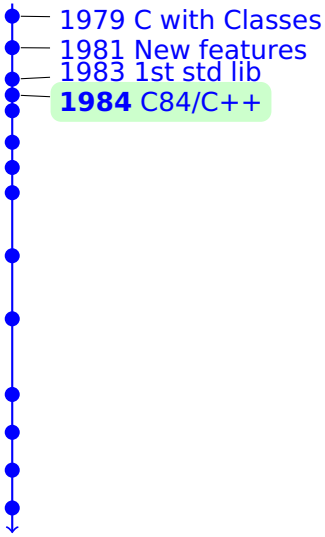
C84 and Cfront

- C with Classes got new name - C84
- A few months later C84 got a new name - C++
- First C++ compiler - Cfront
  - Originally written in... C with Classes



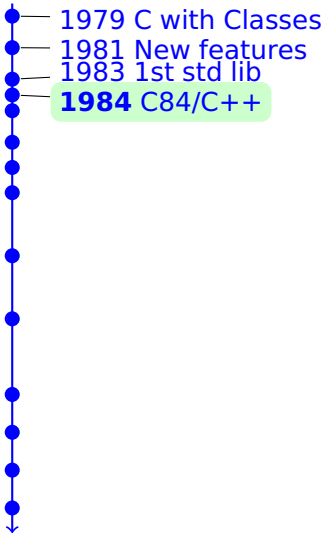
C84 and Cfront

- C with Classes got new name - C84
- A few months later C84 got a new name - C++
- First C++ compiler - Cfront
  - Originally written in... C with Classes
  - Transpiler to C code



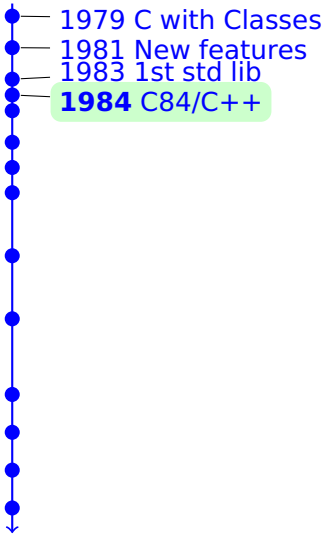
C84 and Cfront

- C with Classes got new name - C84
- A few months later C84 got a new name - C++
- First C++ compiler - Cfront
  - Originally written in... C with Classes
  - Transpiler to C code
  - Portability matters



C84 and Cfront

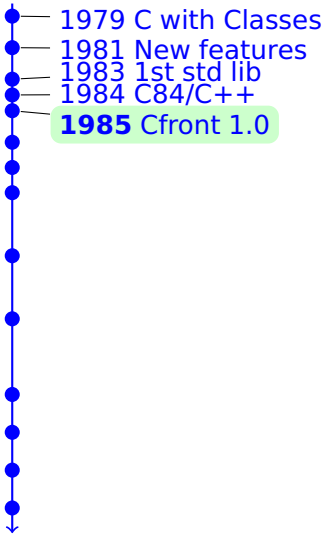
- C with Classes got new name - C84
- A few months later C84 got a new name - C++
- First C++ compiler - Cfront
  - Originally written in... C with Classes
  - Transpiler to C code
  - Portability matters
  - C++ versions were named after Cfront releases



Cfront 1.0

New features:

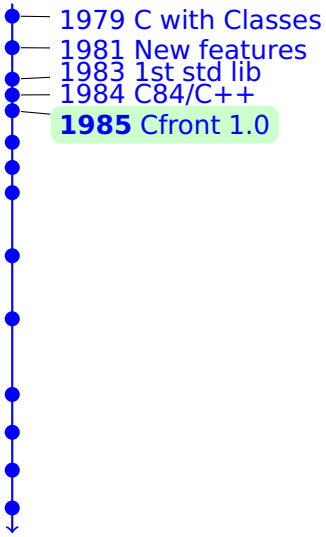
1. virtual functions



Cfront 1.0

New features:

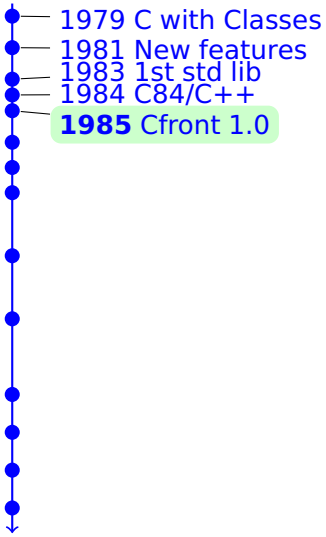
1. virtual functions
2. function name and operator overloading



Cfront 1.0

New features:

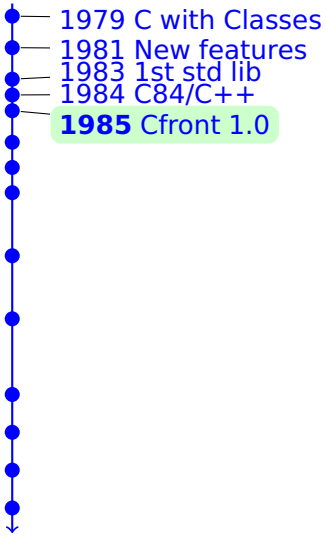
1. virtual functions
2. function name and operator overloading
3. references



Cfront 1.0

New features:

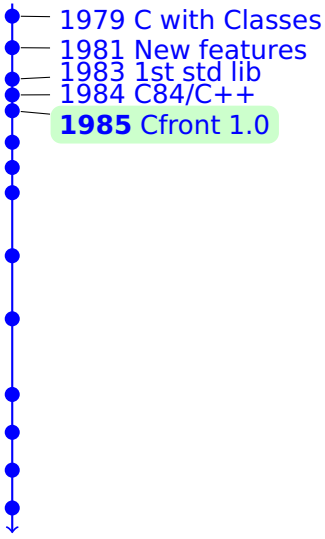
1. virtual functions
2. function name and operator overloading
3. references
4. constants (const)



Cfront 1.0

New features:

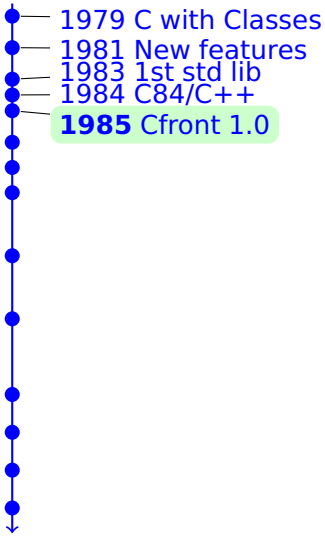
1. virtual functions
2. function name and operator overloading
3. references
4. constants (const)
5. new and delete operators



Cfront 1.0

New features:

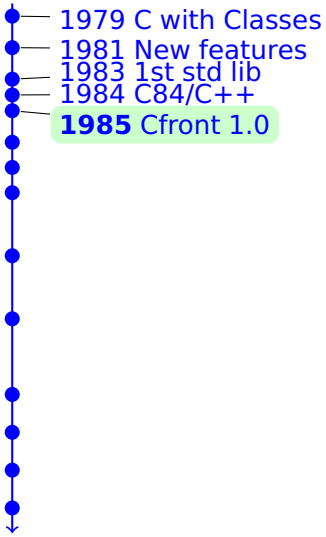
1. virtual functions
2. function name and operator overloading
3. references
4. constants (const)
5. new and delete operators
6. improved type checking



Cfront 1.0

New features:

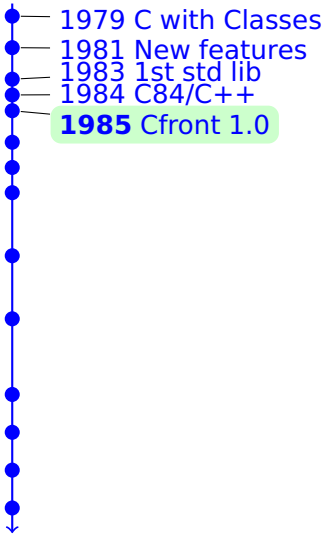
1. virtual functions
2. function name and operator overloading
3. references
4. constants (const)
5. new and delete operators
6. improved type checking
7. scope resolution operator (::)



Cfront 1.0

New features:

1. virtual functions
2. function name and operator overloading
3. references
4. constants (const)
5. new and delete operators
6. improved type checking
7. scope resolution operator (::)
8. BCPL-style comment terminated by end-of-line



Example code in C++ 1.0

Virtual functions:

```
1  class shape {
2      point center;
3      color col;
4  public:
5      point where() { return center; }
6      void move(point to) { center = to; draw(); }
7      virtual void draw();
8      virtual void rotate(int);
9  };
10 class circle : public shape {
11     int radius;
12 public:
13     void draw() { /* ... */ };
14     void rotate(int) {} // yes, the null function
15 };
```

Example code in C++ 1.0

Virtual functions:

```
1  class shape {
2      point center;
3      color col;
4  public:
5      point where() { return center; }
6      void move(point to) { center = to; draw(); }
7      virtual void draw();
8      virtual void rotate(int);
9  };
10 class circle : public shape {
11     int radius;
12 public:
13     void draw() { /* ... */ };
14     void rotate(int) {} // yes, the null function
15 };
```

Example code in C++ 1.0

Overloaded functions:

```
1 overload print;
2 void print(int);
3 void print(const char*);
```

Example code in C++ 1.0

New and delete operators: Because you want to write that:

```
1  X* p = new X(2);
```

Example code in C++ 1.0

New and delete operators: Because you want to write that:

```
1 X* p = new X(2);
```

Instead of that:

```
1 struct X * p = (struct X *) malloc(sizeof(struct X));
2 if (p == 0) error("memory exhausted");
3 p->init(2);
```

Example code in C++ 1.0

Improved type checking:

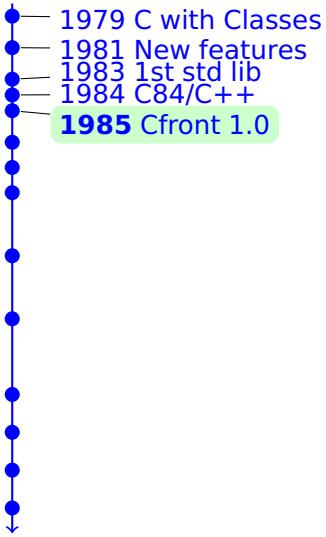
```
1 int printf(const char* ...); // accept any argument after
2                              // the initial character string
3 printf("date: %s %d 19%d\n", month, day, year); // maybe right
```

Cfront 1.1 (1986) & Cfront 1.2 (1987)

New features:

1. pointers to members
2. protected members

+ bug fixes



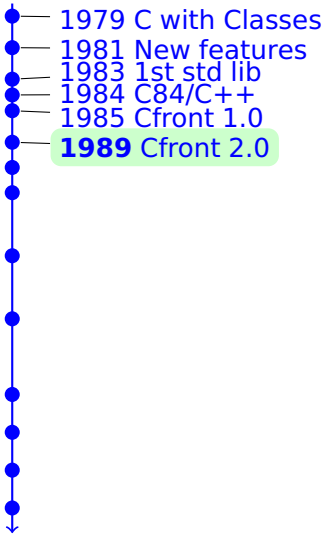
Cfront 2.0

New features:

Cfront 2.0

New features:

1. multiple inheritance
2. type-safe linkage (name mangling)
3. recursive definition of assignment and initialization
4. abstract classes
5. static member functions
6. const member functions
7. overloading of operator ->



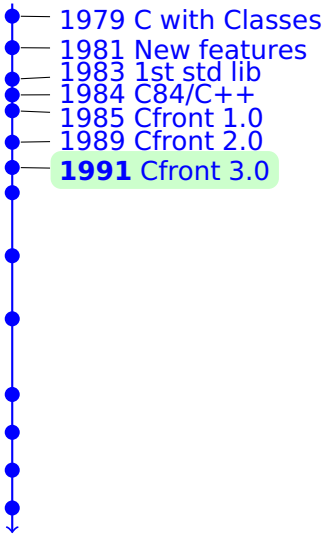
Cfront 3.0

New features:

Cfront 3.0

New features:

1. namespaces
2. templates
3. nested classes
4. exceptions ("code approach")



Exceptions - code approach

- root of "bad exceptions" myth

[1] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - code approach

- root of "bad exceptions" myth
- runtime overhead

[1] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - code approach

- root of "bad exceptions" myth
- runtime overhead
- memory overhead

[1] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - code approach

- root of "bad exceptions" myth
- runtime overhead
- memory overhead
- legends about resource consumption

[1] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - code approach

- root of "bad exceptions" myth
- runtime overhead
- memory overhead
- legends about resource consumption
- see: Bartosz ’BaSz’ Szurgot - *Hello Huston* [1]

[1] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - table approach

- no runtime overhead (if no exception)

[2] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - table approach

- no runtime overhead (if no exception)
- no memory overhead

[2] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - table approach

- no runtime overhead (if no exception)
- no memory overhead
- increased binary size

[2] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - table approach

- no runtime overhead (if no exception)
- no memory overhead
- increased binary size
- unknown exception handling time

[2] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - table approach

- no runtime overhead (if no exception)
- no memory overhead
- increased binary size
- unknown exception handling time
- the fastest solution (if no exception)

[2] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - table approach

- no runtime overhead (if no exception)
- no memory overhead
- increased binary size
- unknown exception handling time
- the fastest solution (if no exception)
- impossible without compiler support

[2] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - table approach

- no runtime overhead (if no exception)
- no memory overhead
- increased binary size
- unknown exception handling time
- the fastest solution (if no exception)
- impossible without compiler support
- exceptions vs Cfront

[2] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Exceptions - table approach

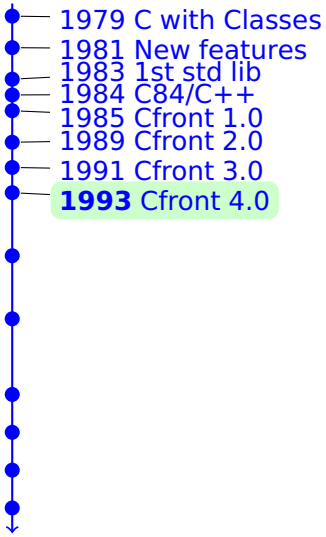
- no runtime overhead (if no exception)
- no memory overhead
- increased binary size
- unknown exception handling time
- the fastest solution (if no exception)
- impossible without compiler support
- exceptions vs Cfront
- see: Bartosz ’BaSz’ Szurgot - *Hello Huston* [2]

[2] http://www.baszerr.eu/lib/exe/fetch.php/docs/hello_houston.pdf

Cfront 4.0

New features:

1. exceptions ("table approach")



Cfront RIP

Cfront - summary

- Years of development: 1985-1993

Cfront - summary

- Years of development: 1985-1993
- Compiler’s frontend

Cfront - summary

- Years of development: 1985-1993
- Compiler’s frontend
- Translated C++ to C (free portability)

Cfront - summary

- Years of development: 1985-1993
- Compiler’s frontend
- Translated C++ to C (free portability)
- Explosion of interest:

Cfront - summary

- Years of development: 1985-1993
- Compiler's frontend
- Translated C++ to C (free portability)
- Explosion of interest:
  - commercial release in 1985

Cfront - summary

- Years of development: 1985-1993
- Compiler's frontend
- Translated C++ to C (free portability)
- Explosion of interest:
  - commercial release in 1985
  - from **500 users in 1985**...

Cfront - summary

- Years of development: 1985-1993
- Compiler's frontend
- Translated C++ to C (free portability)
- Explosion of interest:
  - commercial release in 1985
  - from **500 users in 1985**...
  - to **over 1 500 000 users in 1993**!

Cfront - summary

- Years of development: 1985-1993
- Compiler's frontend
- Translated C++ to C (free portability)
- Explosion of interest:
  - commercial release in 1985
  - from **500 users in 1985**...
  - to **over 1 500 000 users in 1993**!
  - exponential growth!

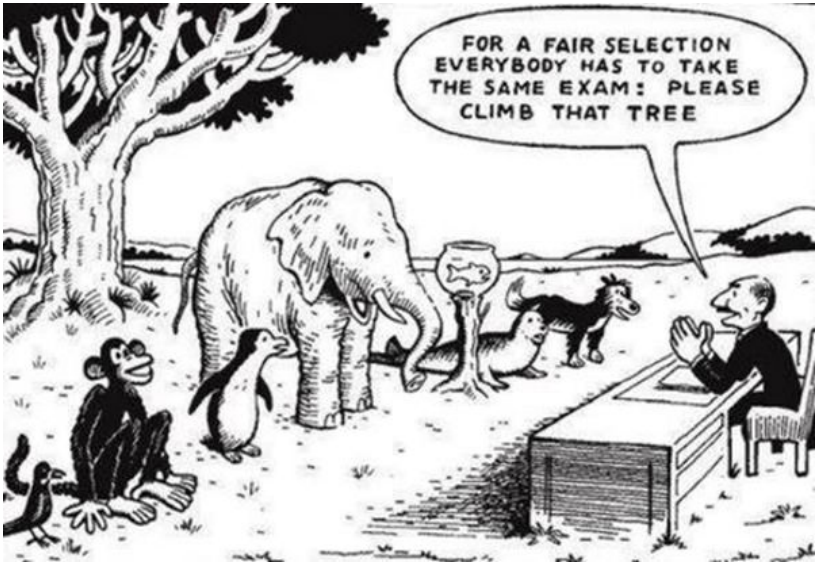
Cfront - summary

- Years of development: 1985-1993
- Compiler's frontend
- Translated C++ to C (free portability)
- Explosion of interest:
  - commercial release in 1985
  - from **500 users in 1985**...
  - to **over 1 500 000 users in 1993**!
  - exponential growth!
- Many new features added

Cfront - summary

- Years of development: 1985-1993
- Compiler's frontend
- Translated C++ to C (free portability)
- Explosion of interest:
  - commercial release in 1985
  - from **500 users in 1985**...
  - to **over 1 500 000 users in 1993**!
  - exponential growth!
- Many new features added
- New approach to exceptions was impossible to add

Standardization time



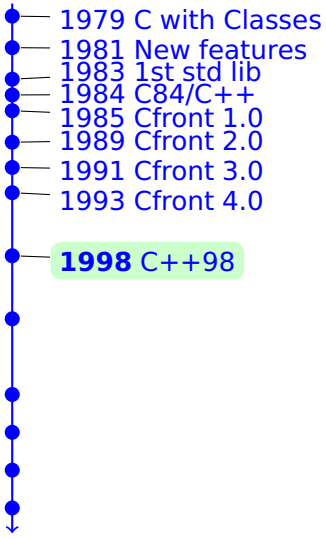
1998 - First ISO C++ standard

Features:

1998 - First ISO C++ standard

Features:

- RTTI (dynamic_cast, typeid)
- cast operators
- mutable
- bool
- declarations in conditions
- member functions templates
- export



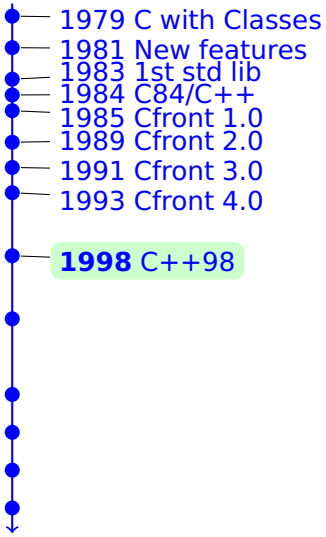
1998 - First ISO C++ standard

Library additions:

1998 - First ISO C++ standard

Library additions:

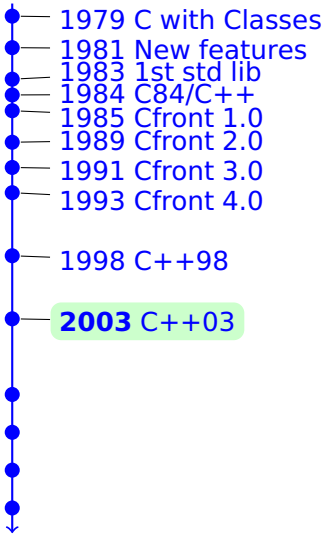
- containers
- algorithms
- iterators
- bitset
- valarray
- auto_ptr
- templatized string
- iostream
- complex



C++03 - Bugfix release

Fixes:

- 125 defects fixed
- defect 69: incontiguous std::vector



C++0x

> *“C++11 feels like a new language”*
>
> — Bjarne Stroustrup

C++0x

> *“C++11 feels like a new language”*
>
> — Bjarne Stroustrup

C++0x == C++11

C++0x

> *"C++11 feels like a new language"*
>
> — Bjarne Stroustrup

C++0x == C++11 (for x = 0xB)

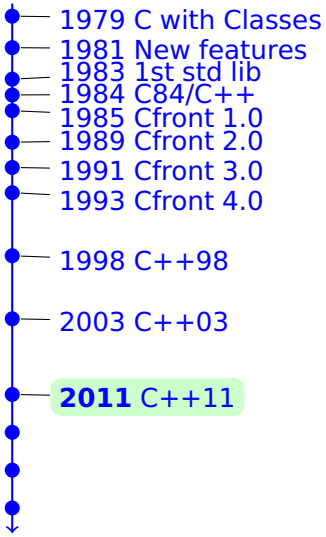
C++11

New language features:

C++11

New language features:

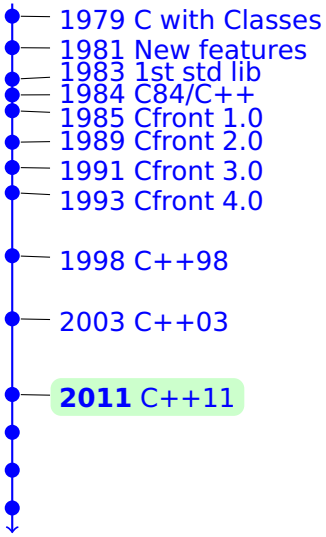
- auto and decltype
- default, delete, final, override keywords
- rvalue references
- move constructors / move assignment
- scoped enums
- constexpr
- list initialization, brace initializers
- delegating constructors
- nullptr
- type aliases
- smart pointers



C++11

New language features:

- variadic templates
- user-defined literals
- attributes
- lambda expressions
- noexcept
- alignof and alignas
- multithreaded memory model
- thread-local storage
- range based for loop
- static assertions



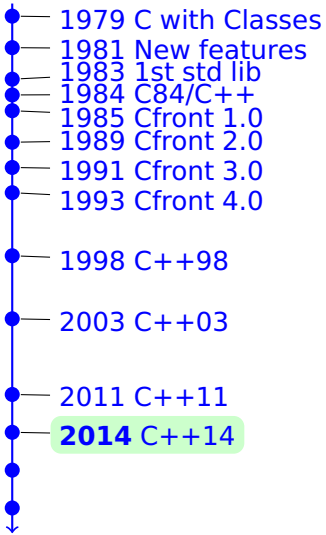
C++14

New language features:

C++14

New language features:

- generic lambdas
- lambda captures expressions
- function return type deduction
- alternate type deduction on declaration
- relaxed restrictions on constexpr functions
- variable templates
- binary literals
- digit separators
- deprecated attribute



Standardization - summary

- C++ standardization started in 1989

Standardization - summary

- C++ standardization started in 1989
- First ISO C++ standard: C++98

Standardization - summary

- C++ standardization started in 1989
- First ISO C++ standard: C++98
- Next standards: C++03, C++11, C++14

Standardization - summary

- C++ standardization started in 1989
- First ISO C++ standard: C++98
- Next standards: C++03, C++11, C++14
- Every bigger company had its own compiler

Standardization - summary

- C++ standardization started in 1989
- First ISO C++ standard: C++98
- Next standards: C++03, C++11, C++14
- Every bigger company had its own compiler
- Only 3 compilers fully support C++14:

Standardization - summary

- C++ standardization started in 1989
- First ISO C++ standard: C++98
- Next standards: C++03, C++11, C++14
- Every bigger company had its own compiler
- Only 3 compilers fully support C++14:
  - gcc

Standardization - summary

- C++ standardization started in 1989
- First ISO C++ standard: C++98
- Next standards: C++03, C++11, C++14
- Every bigger company had its own compiler
- Only 3 compilers fully support C++14:
  - gcc
  - clang

Standardization - summary

- C++ standardization started in 1989
- First ISO C++ standard: C++98
- Next standards: C++03, C++11, C++14
- Every bigger company had its own compiler
- Only 3 compilers fully support C++14:
  - gcc
  - clang
  - MSVC

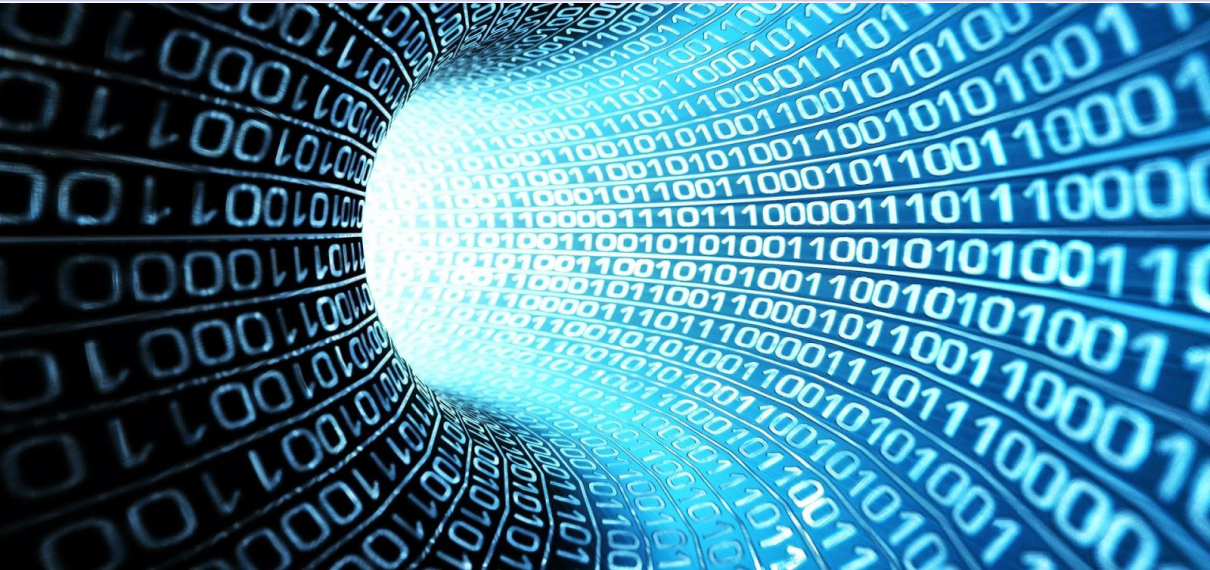
Standardization - summary

- C++ standardization started in 1989
- First ISO C++ standard: C++98
- Next standards: C++03, C++11, C++14
- Every bigger company had its own compiler
- Only 3 compilers fully support C++14:
  - gcc
  - clang
  - MSVC
  - full list: `http://en.cppreference.com/w/cpp/compiler_support`

Standardization - summary

- C++ standardization started in 1989
- First ISO C++ standard: C++98
- Next standards: C++03, C++11, C++14
- Every bigger company had its own compiler
- Only 3 compilers fully support C++14:
  - gcc
  - clang
  - MSVC
  - full list: `http://en.cppreference.com/w/cpp/compiler_support`
- Number of users: about 3 000 000

C++ future



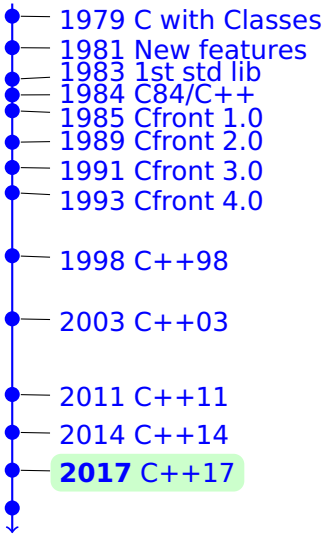
C++17

New features:

C++17

New features:

- Filesystem TS (Tech Spec)
- Paralelism TS
- Library fundamentals TS
- if constexpr(expression)
- auto in templates
- structured bindings
- if and switch with initializer
- many, many more...



C++17

- Michael Wong - C++17, Will It Be Great Or Just OK
- Many planned features are out of scope
- Backward compatibility hamper the standardization of new features
- Stackoverflow list of C++17 features [3]

[3] http://stackoverflow.com/questions/38060436/what-are-the-new-features-in-c17

C++20

- 1979 C with Classes
- 1981 New features
- 1983 1st std lib
- 1984 C84/C++
- 1985 Cfront 1.0
- 1989 Cfront 2.0
- 1991 Cfront 3.0
- 1993 Cfront 4.0
- 1998 C++98
- 2003 C++03
- 2011 C++11
- 2014 C++14
- 2017 C++17
- **2020** C++20

C++ future - summary

- C++17 is almost ready

C++ future - summary

- C++17 is almost ready
- Future versions are planned to be released every 3 years

C++ future - summary

- C++17 is almost ready
- Future versions are planned to be released every 3 years
- Next planned version: C++20 (minor)

C++ future - summary

- C++17 is almost ready
- Future versions are planned to be released every 3 years
- Next planned version: C++20 (minor)
- Backward compatibility hamper the standardization of new features

C++ future - summary

- C++17 is almost ready
- Future versions are planned to be released every 3 years
- Next planned version: C++20 (minor)
- Backward compatibility hamper the standardization of new features
- gcc and clang already support many of C++17 features

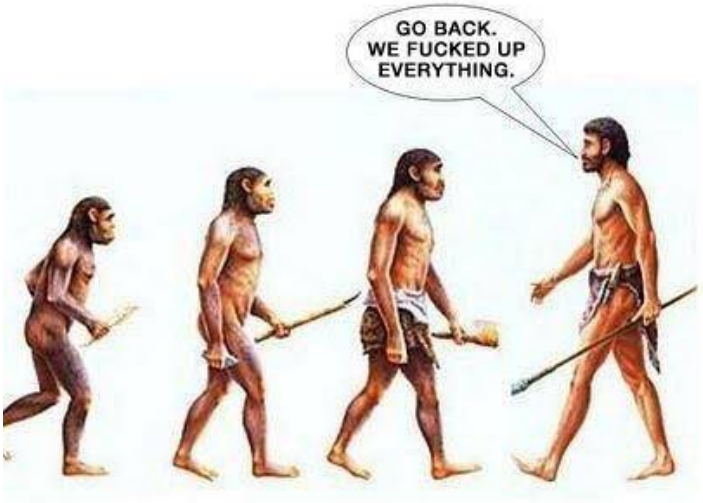
C++ future - summary

- C++17 is almost ready
- Future versions are planned to be released every 3 years
- Next planned version: C++20 (minor)
- Backward compatibility hamper the standardization of new features
- gcc and clang already support many of C++17 features
- MSVC is behind them

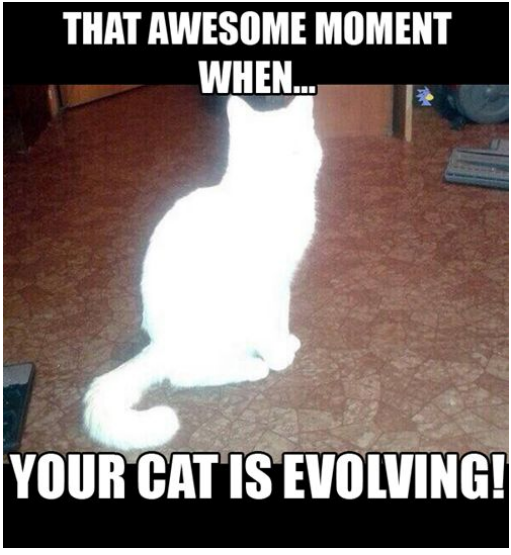
(R)evolution!

Evolution of C++ vs Revolution of C++

(R)evolution!



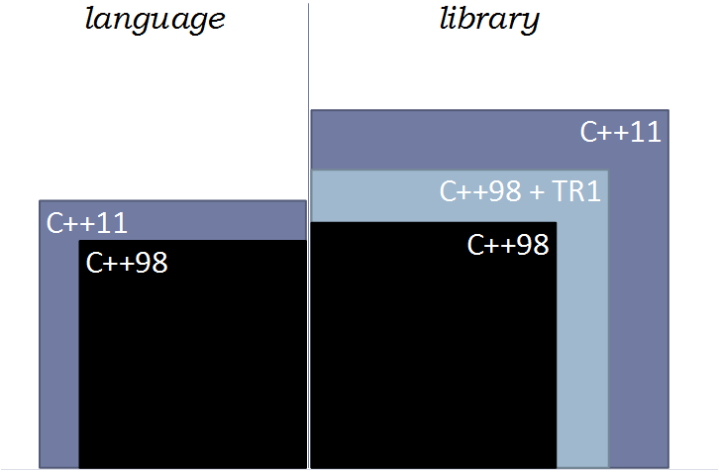
(R)evolution!



C++ is becoming more and more complicated...

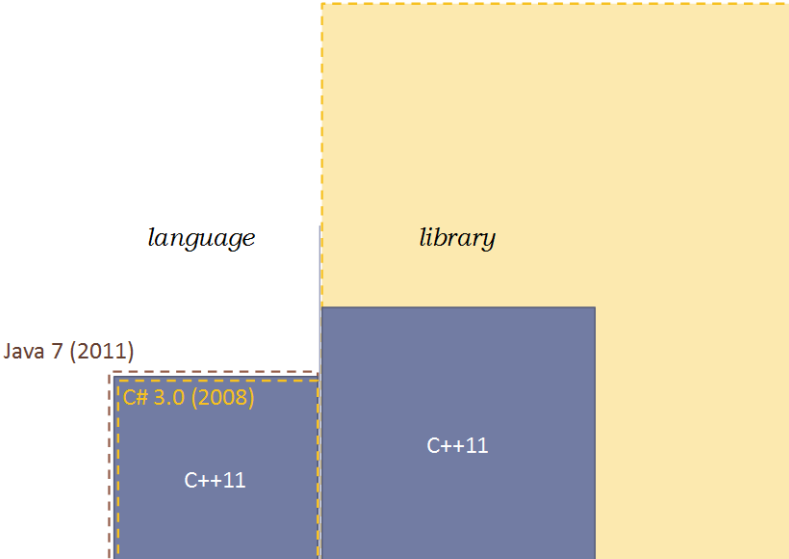
stdlib is poor

- Some people say that C++ standard library is small...



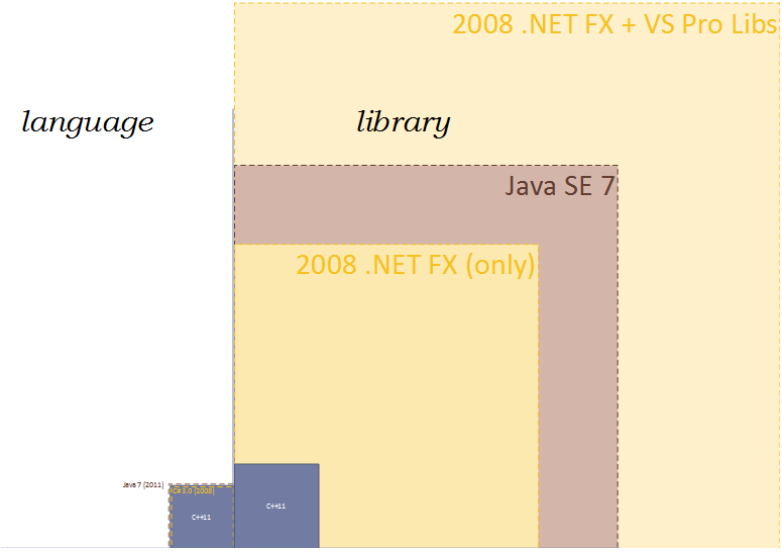
stdlib is poor

- Some people say that C++ standard library is small...
- In comparison to other languages



stdlib is poor

- Some people say that C++ standard library is small...
- In comparison to other languages
- Examples from presentation "*One C++*" by Herb Sutter



stdlib is poor

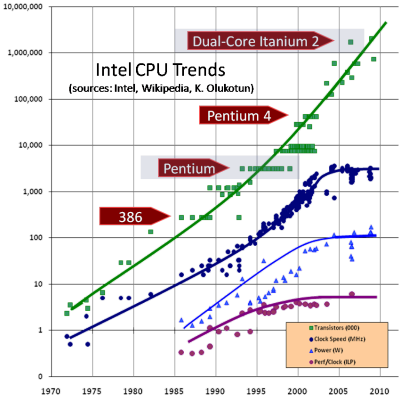
- Some people say that C++ standard library is small...
- In comparison to other languages
- Examples from presentation "*One C++*" by Herb Sutter
- But it will grow in next standards

C++ standard implementation delays

Version	Standard	First implementation
C84	"the ARM" - 1989	-
C++98	IX 1998	2003 (EDG + Dinkumware)
C++03	X 2003	?
C++11	IX 2011	IV 2013 (clang3.3)
C++14	III 2014	XI 2013 (clang 3.4)
C++17	?	?

Free lunch is over

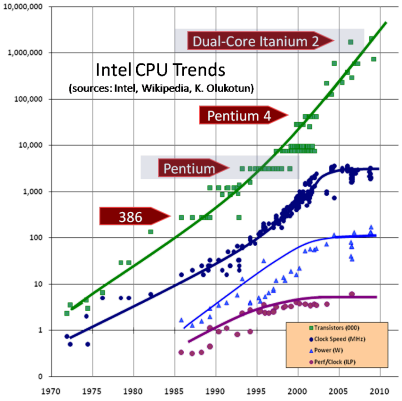
- Moore's law is no applicable anymore



http://www.gotw.ca/publications/concurrency-ddj.htm

Free lunch is over

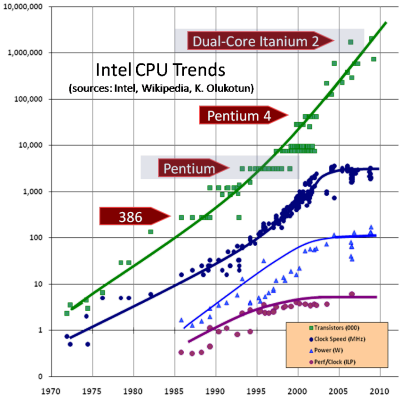
- Moore's law is no applicable anymore
- Processor speed doesn't rise



http://www.gotw.ca/publications/concurrency-ddj.htm

Free lunch is over

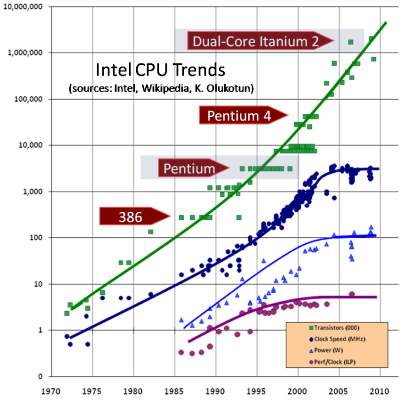
- Moore's law is no applicable anymore
- Processor speed doesn't rise
- We must go into concurrency



http://www.gotw.ca/publications/concurrency-ddj.htm

Free lunch is over

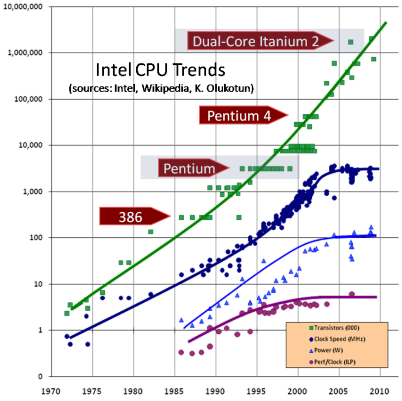
- Moore’s law is no applicable anymore
- Processor speed doesn’t rise
- We must go into concurrency
- We must write multithreaded apps



http://www.gotw.ca/publications/concurrency-ddj.htm

Free lunch is over

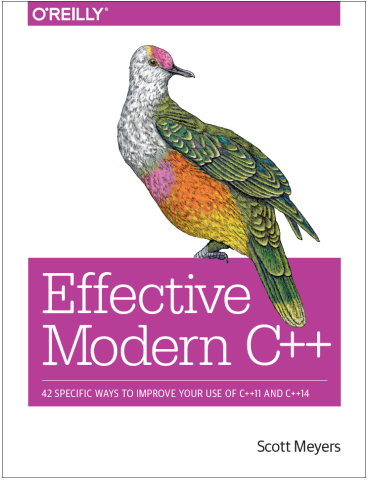
- Moore’s law is no applicable anymore
- Processor speed doesn’t rise
- We must go into concurrency
- We must write multithreaded apps
- We must write efficient and effective code



http://www.gotw.ca/publications/concurrency-ddj.htm

Free lunch is over

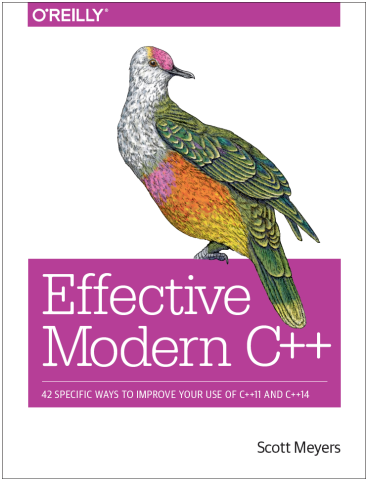
- Moore’s law is no applicable anymore
- Processor speed doesn’t rise
- We must go into concurrency
- We must write multithreaded apps
- We must write efficient and effective code
- Modern C++ facilitate above needs



(This book cover is real)

Free lunch is over

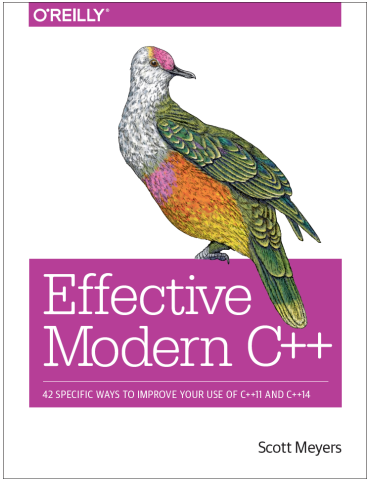
- Moore’s law is no applicable anymore
- Processor speed doesn’t rise
- We must go into concurrency
- We must write multithreaded apps
- We must write efficient and effective code
- Modern C++ facilitate above needs
- VM languages will not be faster than C++



(This book cover is real)

Free lunch is over

- Moore's law is no applicable anymore
- Processor speed doesn't rise
- We must go into concurrency
- We must write multithreaded apps
- We must write efficient and effective code
- Modern C++ facilitate above needs
- VM languages will not be faster than C++
- More and more mobile apps are written in C++



(This book cover is real)

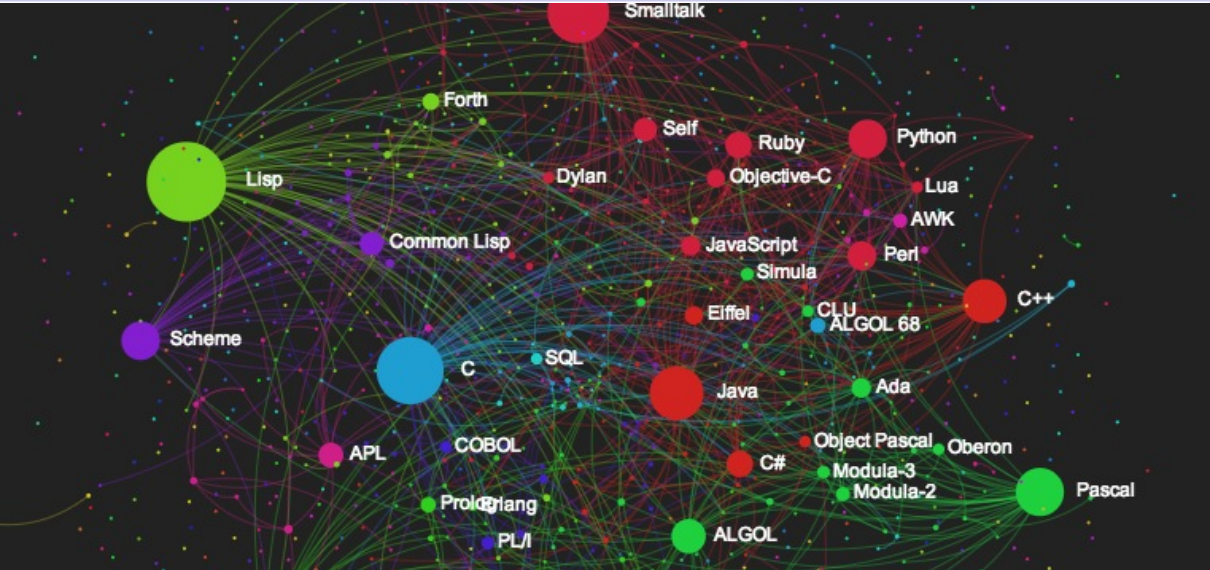
Free lunch is over

- Moore’s law is no applicable anymore
- Processor speed doesn’t rise
- We must go into concurrency
- We must write multithreaded apps
- We must write efficient and effective code
- Modern C++ facilitate above needs
- VM languages will not be faster than C++
- More and more mobile apps are written in C++
- Can C perform better?

Bartosz ’BaSz’ Szurgot
C++ vs. C: The embedded perspective

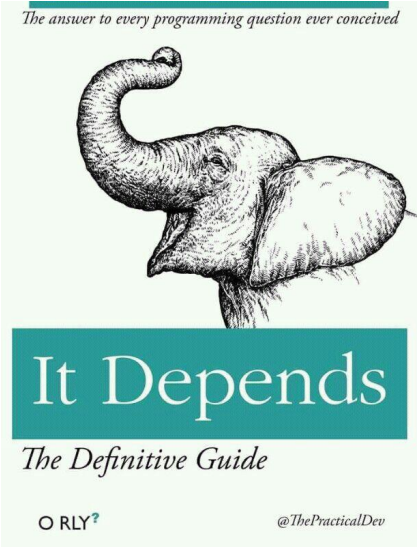
code::dive 2015

Language popularity



What is the most popular programming language?

What is the most popular programming language?



C++ users

Date	Estimated users
1979	1
1980	16
1981	38
1982	85
1983	??+2
1984	??+50
1985	500
1986	2 000
1987	4 000
1988	15 000
1989	50 000
1990	150 000
1991	400 000

C++ users

Date	Estimated users
1979	1
1980	16
1981	38
1982	85
1983	??+2
1984	??+50
1985	500
1986	2 000
1987	4 000
1988	15 000
1989	50 000
1990	150 000
1991	400 000

Main users:

- Bjarne,

C++ users

Date	Estimated users
1979	1
1980	16
1981	38
1982	85
1983	??+2
1984	??+50
1985	500
1986	2 000
1987	4 000
1988	15 000
1989	50 000
1990	150 000
1991	400 000

Main users:

- Bjarne,
- Bjarne's colleagues from AT&T Bell Labs,

C++ users

Date	Estimated users
1979	1
1980	16
1981	38
1982	85
1983	??+2
1984	??+50
1985	500
1986	2 000
1987	4 000
1988	15 000
1989	50 000
1990	150 000
1991	400 000

Main users:

- Bjarne,
- Bjarne's colleagues from AT&T Bell Labs,
- universities,

C++ users

Date	Estimated users
1979	1
1980	16
1981	38
1982	85
1983	??+2
1984	??+50
1985	500
1986	2 000
1987	4 000
1988	15 000
1989	50 000
1990	150 000
1991	400 000

Main users:

- Bjarne,
- Bjarne's colleagues from AT&T Bell Labs,
- universities,
- HP, IBM, AT&T, DEC,

C++ users

Date	Estimated users
1979	1
1980	16
1981	38
1982	85
1983	??+2
1984	??+50
1985	500
1986	2 000
1987	4 000
1988	15 000
1989	50 000
1990	150 000
1991	400 000

Main users:

- Bjarne,
- Bjarne's colleagues from AT&T Bell Labs,
- universities,
- HP, IBM, AT&T, DEC,
- Borland,

C++ users

Date	Estimated users
1979	1
1980	16
1981	38
1982	85
1983	??+2
1984	??+50
1985	500
1986	2 000
1987	4 000
1988	15 000
1989	50 000
1990	150 000
1991	400 000

Main users:

- Bjarne,
- Bjarne's colleagues from AT&T Bell Labs,
- universities,
- HP, IBM, AT&T, DEC,
- Borland,
- Later: Microsoft, Apple,

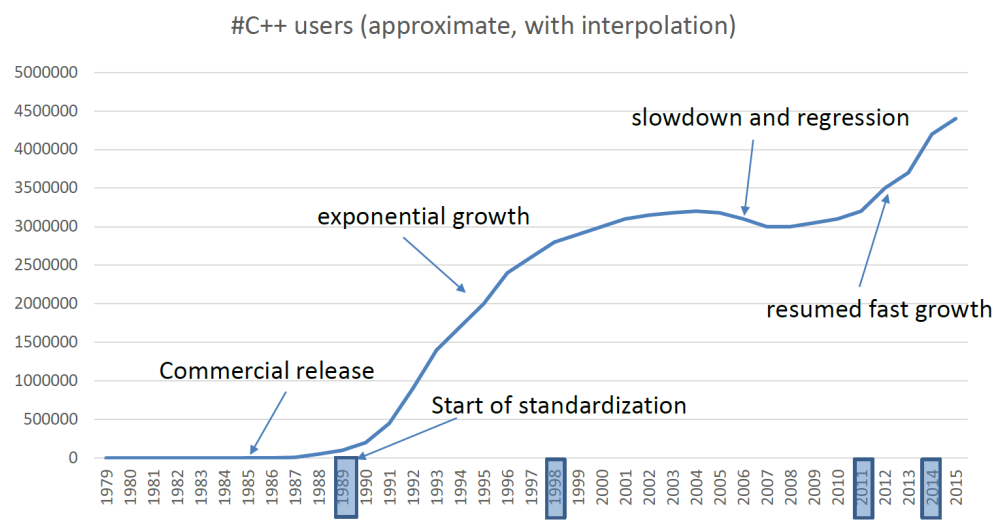
C++ users

Date	Estimated users
1979	1
1980	16
1981	38
1982	85
1983	??+2
1984	??+50
1985	500
1986	2 000
1987	4 000
1988	15 000
1989	50 000
1990	150 000
1991	400 000

Main users:

- Bjarne,
- Bjarne's colleagues from AT&T Bell Labs,
- universities,
- HP, IBM, AT&T, DEC,
- Borland,
- Later: Microsoft, Apple,
- Now: Google, Facebook

C++ users



[3] Bjarne Stroustrup - *The Evolution of C++* - https://www.youtube.com/watch?v=_wzc7a3Mc0s

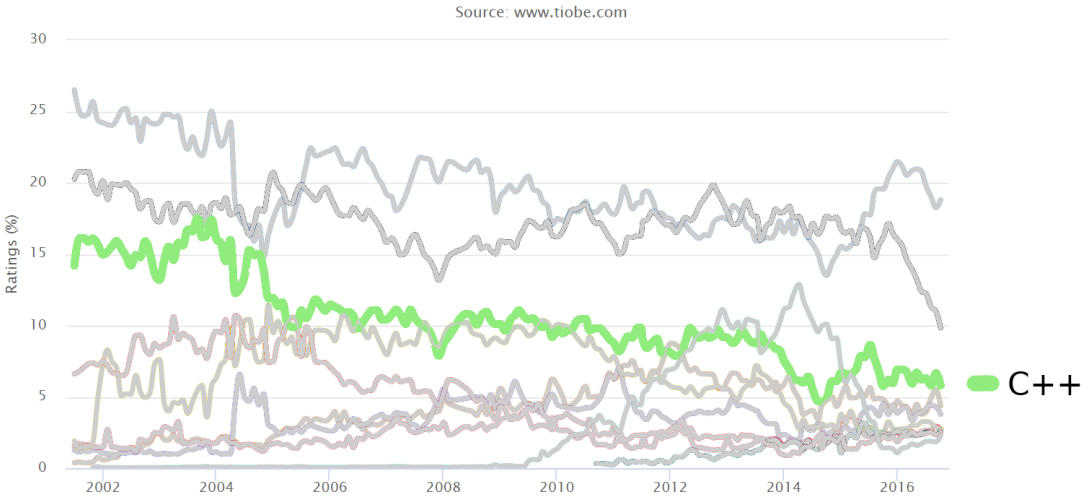
TIOBE - market share

TIOBE index

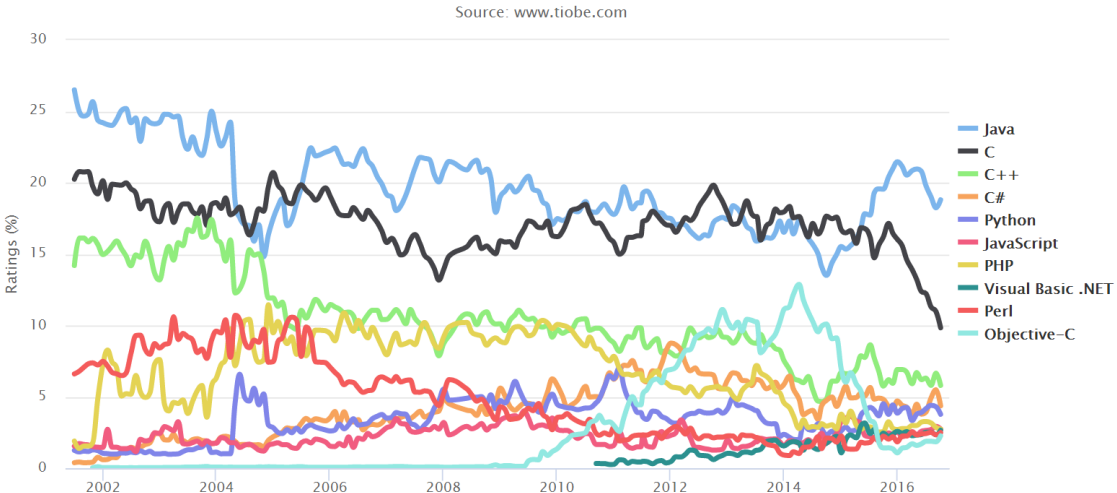
The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. Popular search engines such as **Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube** and **Baidu** are used to calculate the ratings. Search phrase is **"language programming"**

Webpage: `http://www.tiobe.com/tiobe-index//`

TIOBE - market share



TIOBE - market share



TIOBE - market share

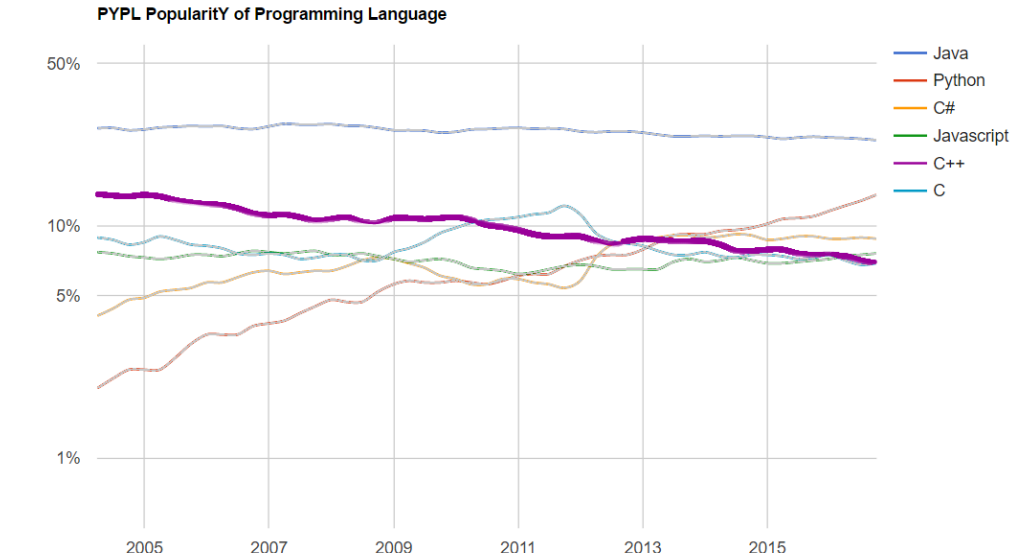
Oct 2016	Oct 2015	Change	Programming Language	Ratings	Change
1	1		Java	18.799%	-0.74%
2	2		C	9.835%	-6.35%
3	3		C++	5.797%	+0.05%
4	4		C#	4.367%	-0.46%
5	5		Python	3.775%	-0.74%
6	8	^	JavaScript	2.751%	+0.46%
7	6	v	PHP	2.741%	+0.18%
8	7	v	Visual Basic .NET	2.660%	+0.20%
9	9		Perl	2.495%	+0.25%
10	14	^^	Objective-C	2.263%	+0.84%

PYPL

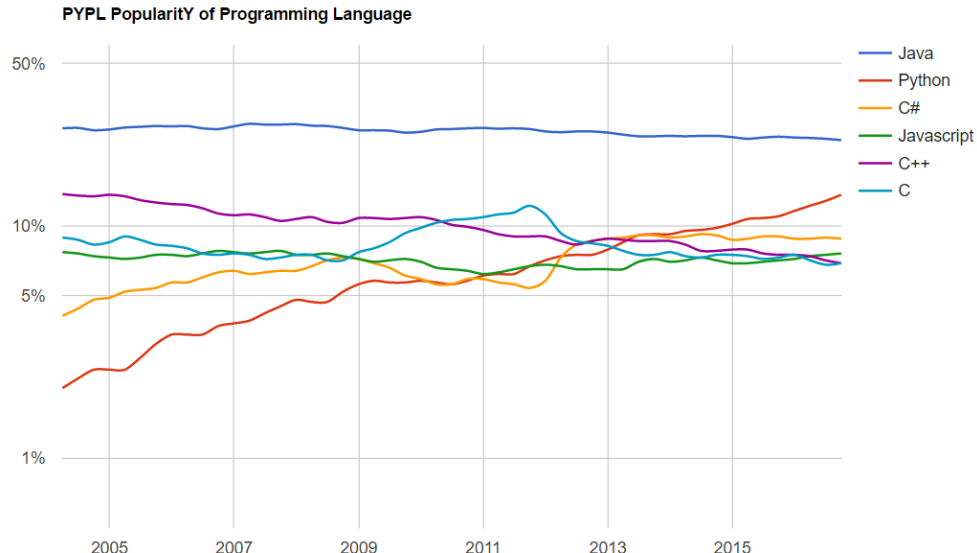
PYPL index

The PYPL PopularitY of Programming Language Index is created by analyzing how often **language tutorials** are searched on Google: the more a language tutorial is searched, the more popular the language is assumed to be. It is a leading indicator. The raw data comes from **Google Trends**.
Webpage: `http://pypl.github.io/PYPL.html`

PYPL


PYPL PopularitY of Programming Language
50%
10%
5%
1%
2005
2007
2009
2011
2013
2015
Java
Python
C#
Javascript
C++
C

PYPL


PYPL PopularitY of Programming Language
50%
10%
5%
1%
2005
2007
2009
2011
2013
2015
Java
Python
C#
Javascript
C++
C

PYPL

Worldwide, Oct 2016 compared to a year ago:

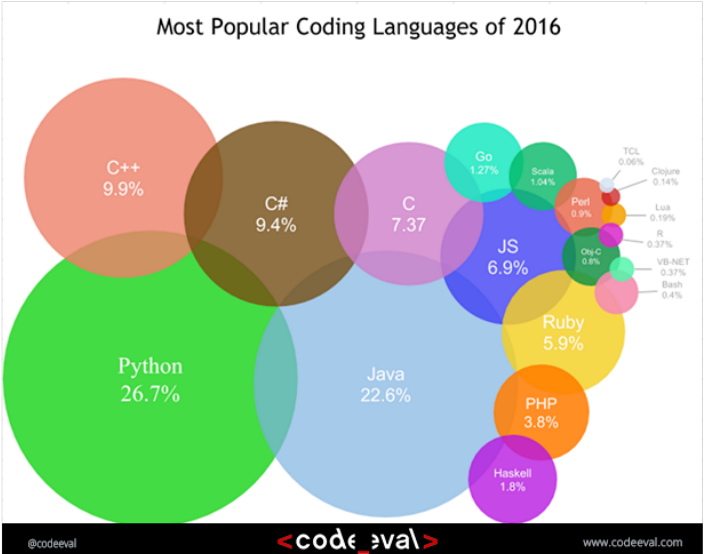
Rank	Change	Language	Share	Trend
1		Java	23.4 %	-0.8 %
2		Python	13.6 %	+2.5 %
3		PHP	9.9 %	-0.8 %
4		C#	8.8 %	-0.0 %
5	↑↑	Javascript	7.6 %	+0.6 %
6	↓	C++	6.9 %	-0.8 %
7	↓	C	6.9 %	-0.9 %
8		Objective-C	4.5 %	-0.7 %
9	↑	R	3.3 %	+0.7 %
10	↓	Swift	3.1 %	+0.3 %

codeeval

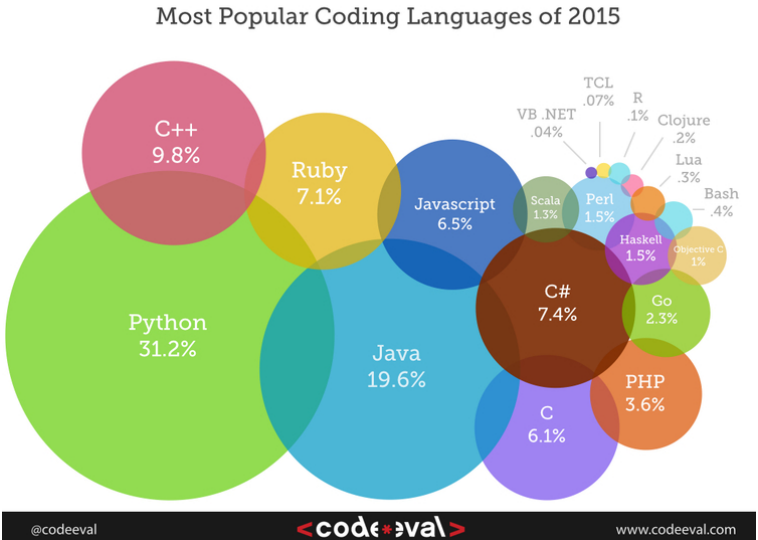
codeeval MPCL

"Most Popular Coding Languages" is based on hundreds of thousands of data points we've collected by processing over 1,200,000+ **challenge submissions on codeeval.com** in (now) 26 different programming languages. Webpage: `http://blog.codeeval.com/`

codeeval

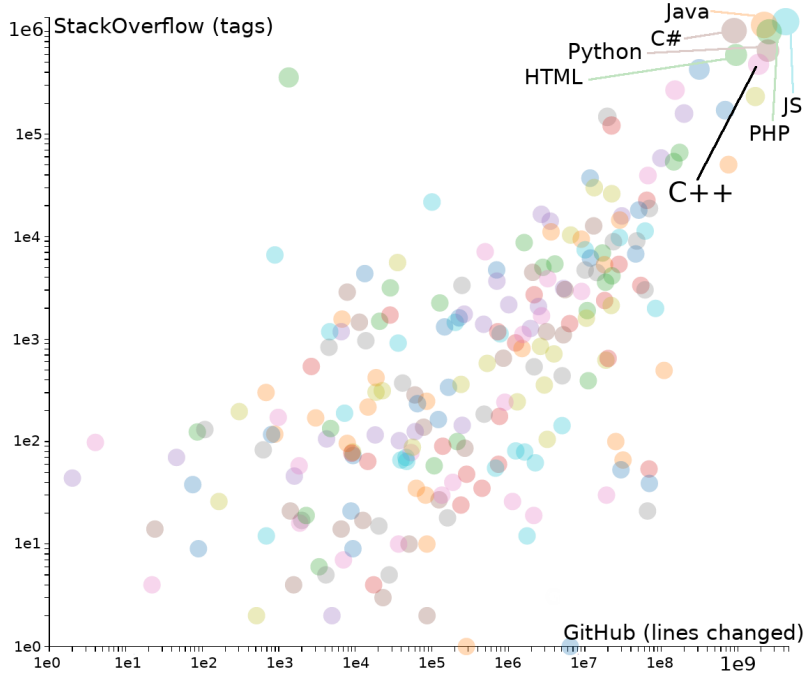

Most Popular Coding Languages of 2016
C++
9.9%
C#
9.4%
C
7.37
Go
1.27%
Scala
1.04%
TCL
0.06%
Clojure
0.14%
Perl
0.9%
Lua
0.19%
R
0.37%
Obj-C
0.8%
VB-NET
0.37%
Bash
0.4%
JS
6.9%
Ruby
5.9%
Python
26.7%
Java
22.6%
PHP
3.8%
Haskell
1.8%
@codeeval
<code_eval>
www.codeeval.com

codeeval


Most Popular Coding Languages of 2015
C++ 9.8%
Ruby 7.1%
Javascript 6.5%
Python 31.2%
Java 19.6%
C# 7.4%
C 6.1%
PHP 3.6%
Go 2.3%
Objective C 1%
Haskell 1.5%
Perl 1.5%
Scala 1.3%
VB .NET .04%
TCL .07%
R .1%
Clojure .2%
Lua .3%
Bash .4%
@codeeval
<code*eval>
www.codeeval.com

langpop.corger.nl

Webpage: `http://langpop.corger.nl/`


StackOverflow (tags)
1e6
1e5
1e4
1e3
1e2
1e1
1e0
Java
C#
Python
HTML
JS
PHP
C++
GitHub (lines changed)
1e0
1e1
1e2
1e3
1e4
1e5
1e6
1e7
1e8
1e9

Stackoverflow issues

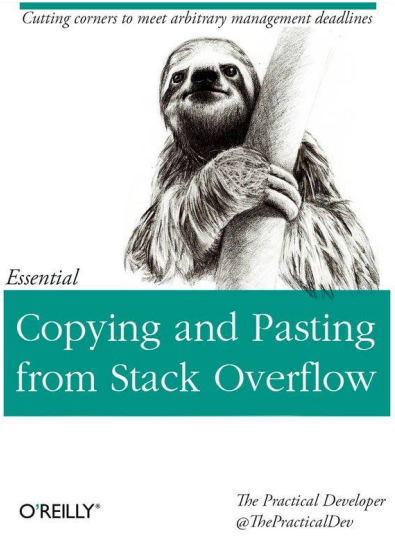
- good programmer == lazy programmer

Stackoverflow issues

- good programmer == lazy programmer
- lazy programmer != good programmer

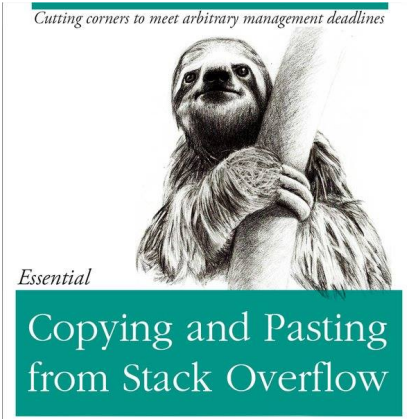
Stackoverflow issues

- good programmer == lazy programmer
- lazy programmer != good programmer
- good programmers do not reinvent the wheel



Stackoverflow issues

- good programmer == lazy programmer
- lazy programmer != good programmer
- good programmers do not reinvent the wheel
- do job once and never come back here



C++ is not dead!

- In the worst case C++ is on 7th place
- In the best case C++ is on 3rd place
- C++ is one of the most popular languages!
- Next versions of C++ will be available soon

C++ is not dead!

Future C++:

- Type and resource safe

[3] Bjarne Stroustrup - *The Evolution of C++* - https://www.youtube.com/watch?v=_wzc7a3McOs

C++ is not dead!

Future C++:

- Type and resource safe
- Significantly simpler and clearer code

[3] Bjarne Stroustrup - *The Evolution of C++* - https://www.youtube.com/watch?v=_wzc7a3McOs

C++ is not dead!

Future C++:

- Type and resource safe
- Significantly simpler and clearer code
- As fast or faster than anything else

[3] Bjarne Stroustrup - *The Evolution of C++* - https://www.youtube.com/watch?v=_wzc7a3McOs

C++ is not dead!

Future C++:

- Type and resource safe
- Significantly simpler and clearer code
- As fast or faster than anything else
- Good at using "modern hardware" (more pipelines, more concurrency)

[3] Bjarne Stroustrup - *The Evolution of C++* - https://www.youtube.com/watch?v=_wzc7a3McOs

C++ is not dead!

Future C++:

- Type and resource safe
- Significantly simpler and clearer code
- As fast or faster than anything else
- Good at using "modern hardware" (more pipelines, more concurrency)
- Significantly faster compilation catching many more errors

[3] Bjarne Stroustrup - *The Evolution of C++* - `https://www.youtube.com/watch?v=_wzc7a3McOs`

Summary

Key messages

1. C++ had a **clear aim**, which made it popular: to **organize code better without the loss of efficiency**

Key messages

1. C++ had a **clear aim**, which made it popular: to **organize code better without the loss of efficiency**
2. C++ is even more popular now, because of new standards: **C++11** and **C++14**

Key messages

1. C++ had a **clear aim**, which made it popular: to **organize code better without the loss of efficiency**
2. C++ is even more popular now, because of new standards: **C++11** and **C++14**
3. In future C++ will be **one of the most popular programming languages** so it's worth learning

Worth reading / watching

Bjarne Stroustrup – *The Evolution of C++ - Past, Present, and Future*
CppCon 2016

`https://www.youtube.com/watch?v=_wzc7a3McOs`

Herb Sutter – *One C++*
Going Native 2013

`https://channel9.msdn.com/Events/GoingNative/2013/Keynote-Herb-Sutter-One-Cpp`

Kate Gregory – *Stop teaching C*
CppCon 2015

`https://www.youtube.com/watch?v=YnWhqhNdYyk`

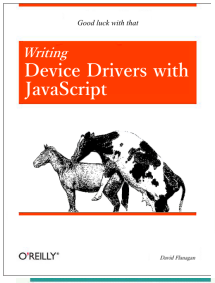
Bjarne Stroustrup – *A History of C++: 1979-1991*

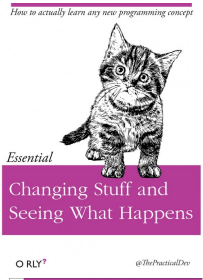
`http://www.stroustrup.com/hopl2.pdf`

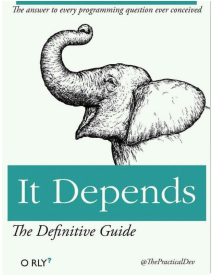
Scott Meyers – *Effective Modern C++*

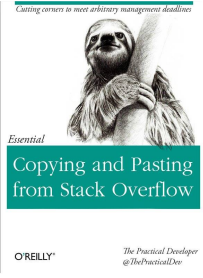
How many book covers are in my presentation?

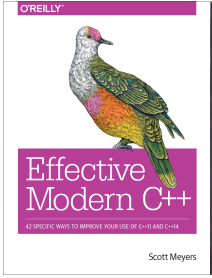
How many book covers are in my presentation?

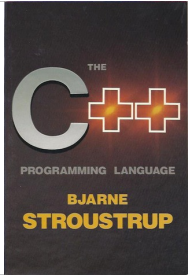












What now?

What now?

- Look forward to C++17 and learn it

What now?

What now?

- Look forward to C++17 and learn it
- Watch a lot of videos from C++ conferences

What now?

What now?

- Look forward to C++17 and learn it
- Watch a lot of videos from C++ conferences
- Visit `isocpp.org`

Famous quotation

“Learn C++. It’s an investment.”

Famous quotation

> *“Learn C++. It’s an investment.”*
>
> — Łukasz Ziobroń

KEEP
CALM
AND
~~DON'T~~ ASK
QUESTIONS