C++11 for Java developers

Lars Gullik Bjønnes & Olve Maudal

This is a quick introduction to C++11 for Java developers. We will mostly focus on the new stuff in the 2011 standard. In this presentation we will give you some hints about how to get started with a C++ project, but quickly move into discussions and comparison of language details.

Our goal is to make a presentation where the best Java programmers can learn just enough about the new version of C++ so that they can make an informed decision about learning more about C++11 or not.

A 60 minute presentation (incl QA)
JavaZone, September 12-13, 2012

Lynkurs i C++11 for Java utviklere

Lars Gullik Bjønnes & Olve Maudal

Vi tilbyr en kjapp innføring i C++ for Java utviklere. Det er særlig de nye tingene i 2011 standarden av C++ som vil få fokus. I denne presentasjonen vil vi demonstrere hvordan du kommer i gang med et C++ prosjekt, men vi vil fort gå over til å diskutere og sammenligne sære språkdetaljer.

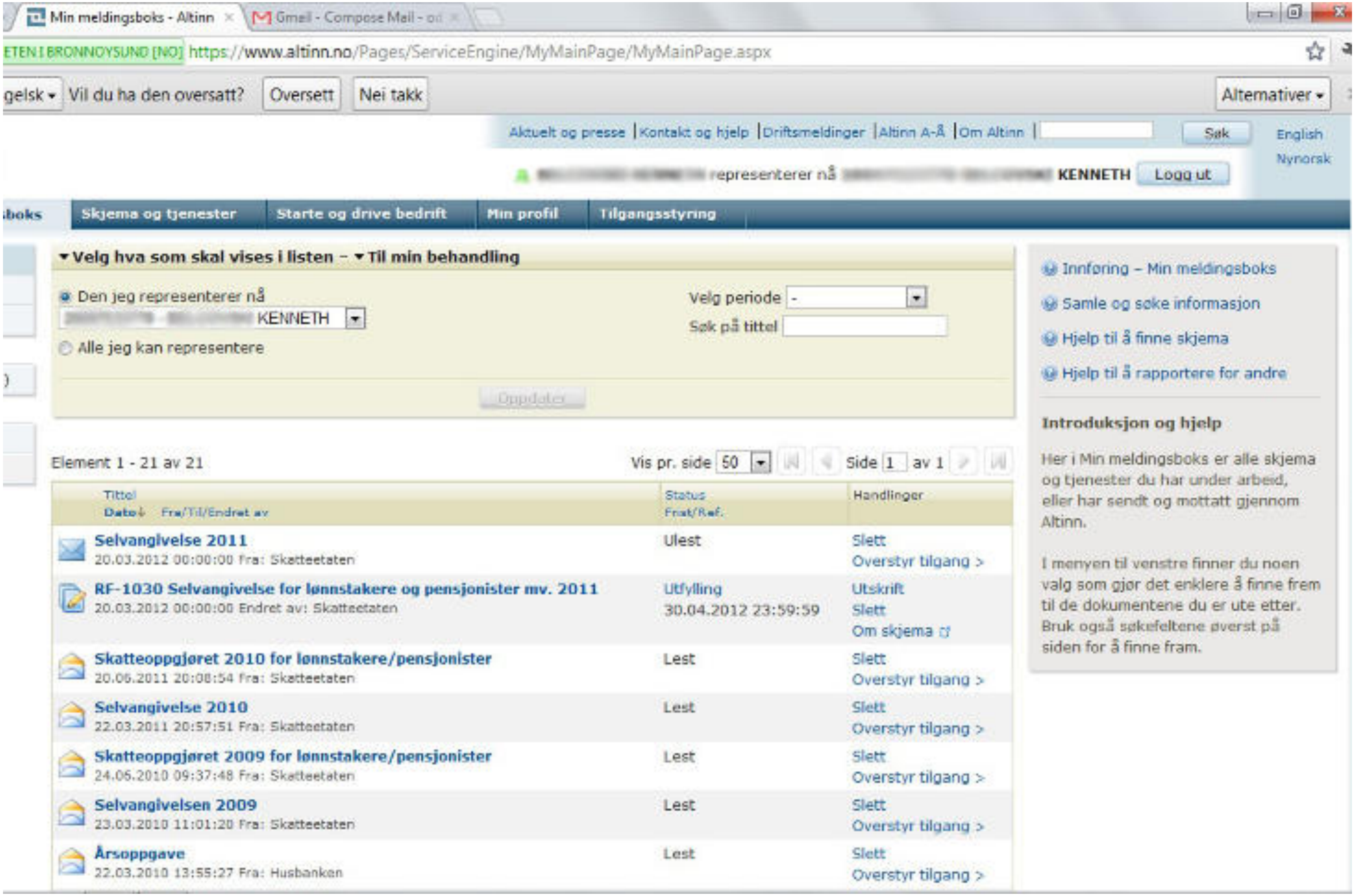
Målet vårt er å lage en presentasjon hvor de flinkeste Java programmere kan lære akkurat nok om den nye versjonen av C++ til å gjøre et informert valg om de vil lære mer om C++11 eller ikke.

60 minutter presentasjon (inkl QA)
JavaZone, September 12-13, 2012

Why C++11 ?

Where not to use C++ ?

Web portals



Min meldingsboks - Altinn × | Gmail - Compose Mail - o ×

ETEN I BRONNOYSUND [NO] https://www.altinn.no/Pages/ServiceEngine/MyMainPage/MyMainPage.aspx

gelsk ▾ Vil du ha den oversatt? | Oversett | Nei takk | Alternativer ▾

Aktuelt og presse | Kontakt og hjelp | Driftsmeldinger | Altinn A-Å | Om Altinn | Søk | English | Nynorsk

[redacted] representerer nå [redacted] KENNETH | Logg ut

boks | Skjema og tjenester | Starte og drive bedrift | Min profil | Tilgangsstyring

▾ Velg hva som skal vises i listen – ▾ Til min behandling

● Den jeg representerer nå
[redacted] KENNETH ▾
○ Alle jeg kan representere

Velg periode - ▾
Søk på tittel

Oppdater

Element 1 - 21 av 21

Vis pr. side 50 ▾ Side 1 av 1

Tittel Dato↓ Fra/Til/Endret av	Status Frist/Ref.	Handlinger
Selvangivelse 2011 20.03.2012 00:00:00 Fra: Skatteetaten	Ulest	Slett Overstyr tilgang >
RF-1030 Selvangivelse for lønnstakere og pensjonister mv. 2011 20.03.2012 00:00:00 Endret av: Skatteetaten	Utfylling 30.04.2012 23:59:59	Utskrift Slett Om skjema
Skatteoppgjøret 2010 for lønnstakere/pensjonister 20.06.2011 20:08:54 Fra: Skatteetaten	Lest	Slett Overstyr tilgang >
Selvangivelse 2010 22.03.2011 20:57:51 Fra: Skatteetaten	Lest	Slett Overstyr tilgang >
Skatteoppgjøret 2009 for lønnstakere/pensjonister 24.06.2010 09:37:48 Fra: Skatteetaten	Lest	Slett Overstyr tilgang >
Selvangivelsen 2009 23.03.2010 11:01:20 Fra: Skatteetaten	Lest	Slett Overstyr tilgang >
Årsoppgave 22.03.2010 13:55:27 Fra: Husbanken	Lest	Slett Overstyr tilgang >

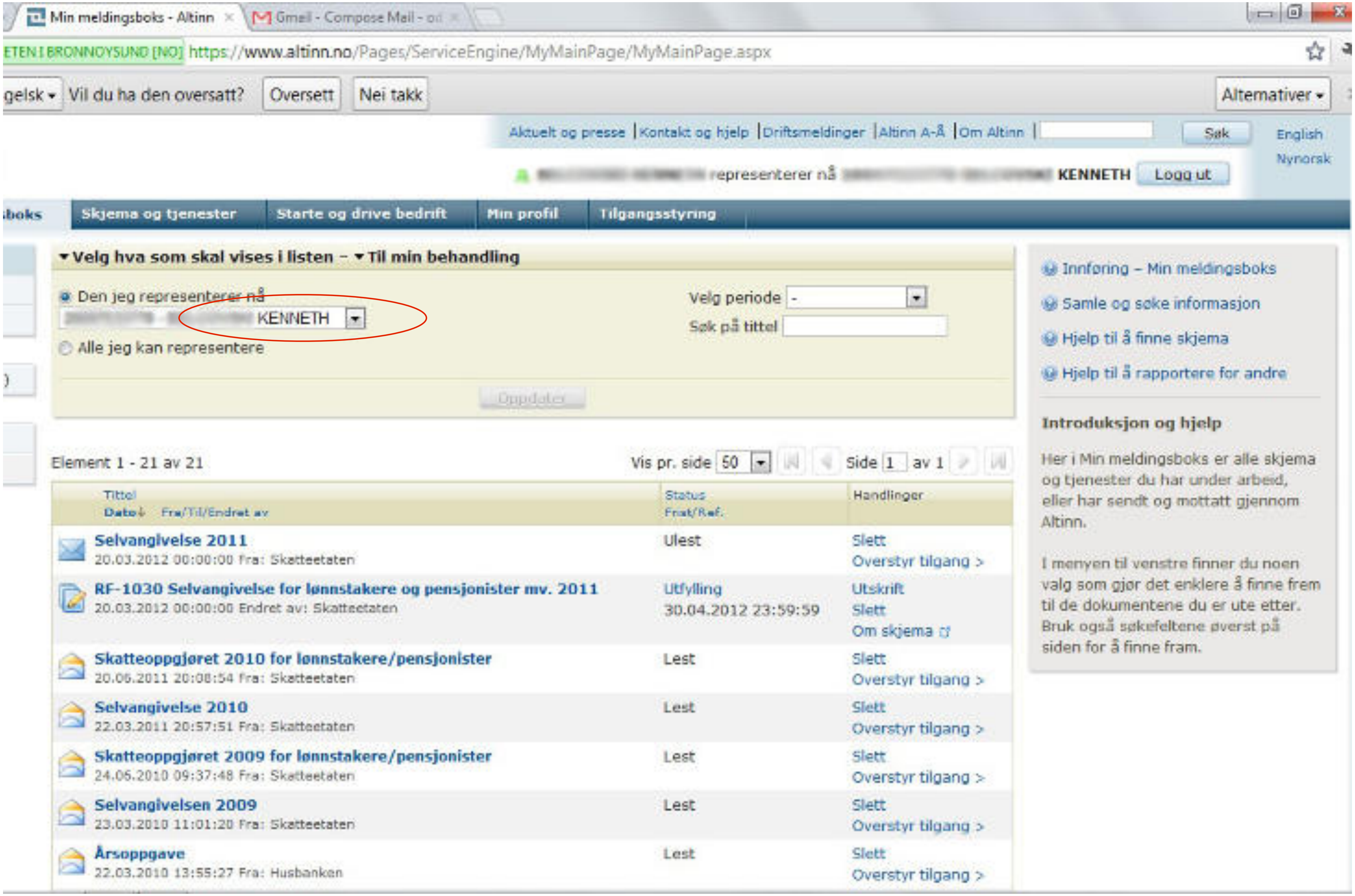
Innføring – Min meldingsboks
Samle og søke informasjon
Hjelp til å finne skjema
Hjelp til å rapportere for andre

Introduksjon og hjelp

Her i Min meldingsboks er alle skjema og tjenester du har under arbeid, eller har sendt og mottatt gjennom Altinn.

I menyen til venstre finner du noen valg som gjør det enklere å finne frem til de dokumentene du er ute etter. Bruk også søkefeltene øverst på siden for å finne fram.

Web portals



[redacted] representerer nå [redacted] KENNETH

▾ Velg hva som skal vises i listen – ▾ Til min behandling

● Den jeg representerer nå

[redacted] KENNETH ▾

○ Alle jeg kan representere

Velg periode - ▾

Søk på tittel

Element 1 - 21 av 21

Vis pr. side 50 ▾ Side 1 av 1

Tittel Dato↓ Fra/Til/Endret av	Status Frist/Ref.	Handlinger
Selvangivelse 2011 20.03.2012 00:00:00 Fra: Skatteetaten	Ulest	Slett Overstyr tilgang >
RF-1030 Selvangivelse for lønnstakere og pensjonister mv. 2011 20.03.2012 00:00:00 Endret av: Skatteetaten	Utfylling 30.04.2012 23:59:59	Utskrift Slett Om skjema
Skatteoppgjøret 2010 for lønnstakere/pensjonister 20.06.2011 20:08:54 Fra: Skatteetaten	Lest	Slett Overstyr tilgang >
Selvangivelse 2010 22.03.2011 20:57:51 Fra: Skatteetaten	Lest	Slett Overstyr tilgang >
Skatteoppgjøret 2009 for lønnstakere/pensjonister 24.06.2010 09:37:48 Fra: Skatteetaten	Lest	Slett Overstyr tilgang >
Selvangivelsen 2009 23.03.2010 11:01:20 Fra: Skatteetaten	Lest	Slett Overstyr tilgang >
Årsoppgave 22.03.2010 13:55:27 Fra: Husbanken	Lest	Slett Overstyr tilgang >

Introduksjon og hjelp

Her i Min meldingsboks er alle skjema og tjenester du har under arbeid, eller har sendt og mottatt gjennom Altinn.

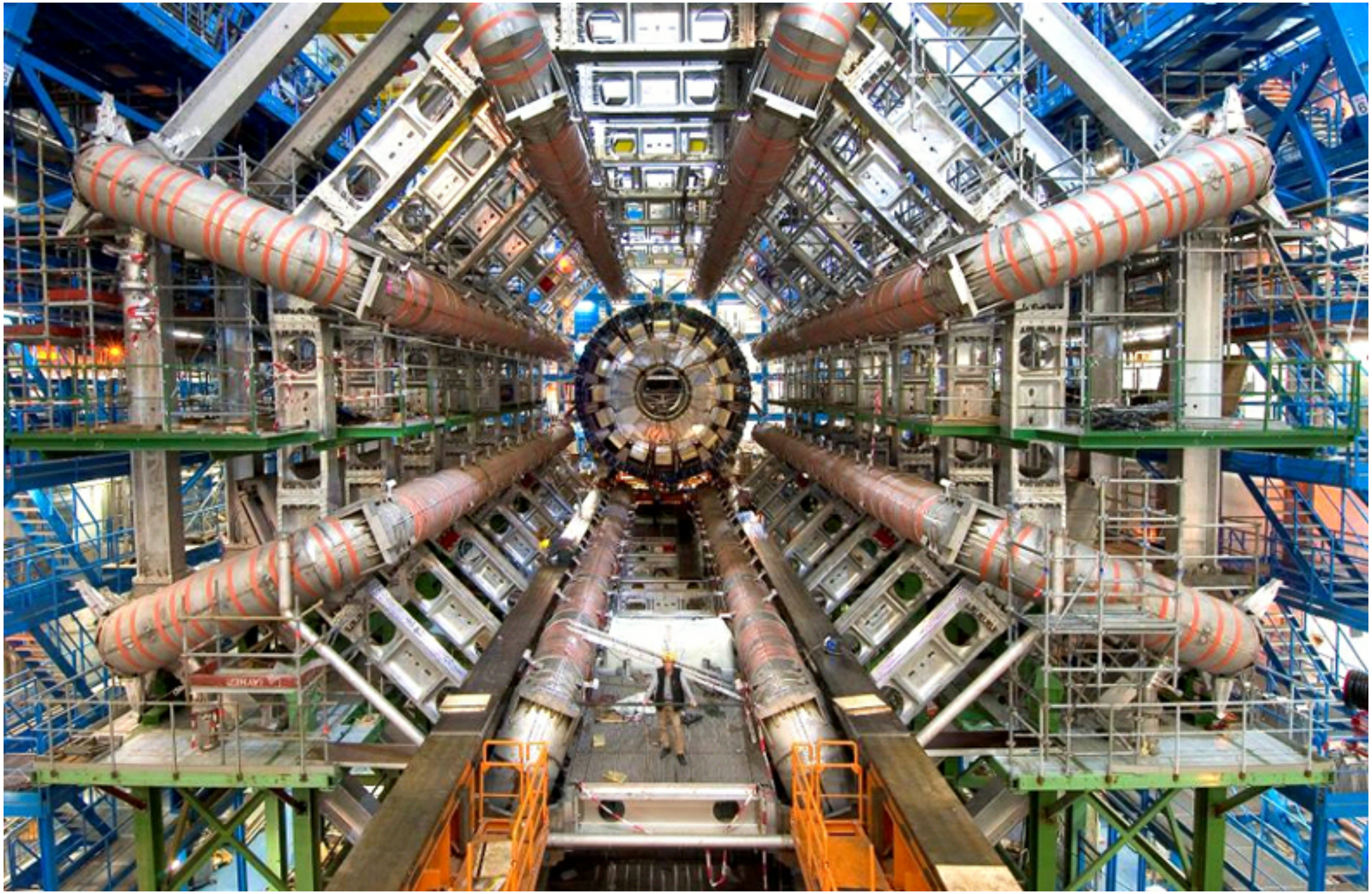
I menyen til venstre finner du noen valg som gjør det enklere å finne frem til de dokumentene du er ute etter. Bruk også søkefeltene øverst på siden for å finne fram.

Very simple stuff

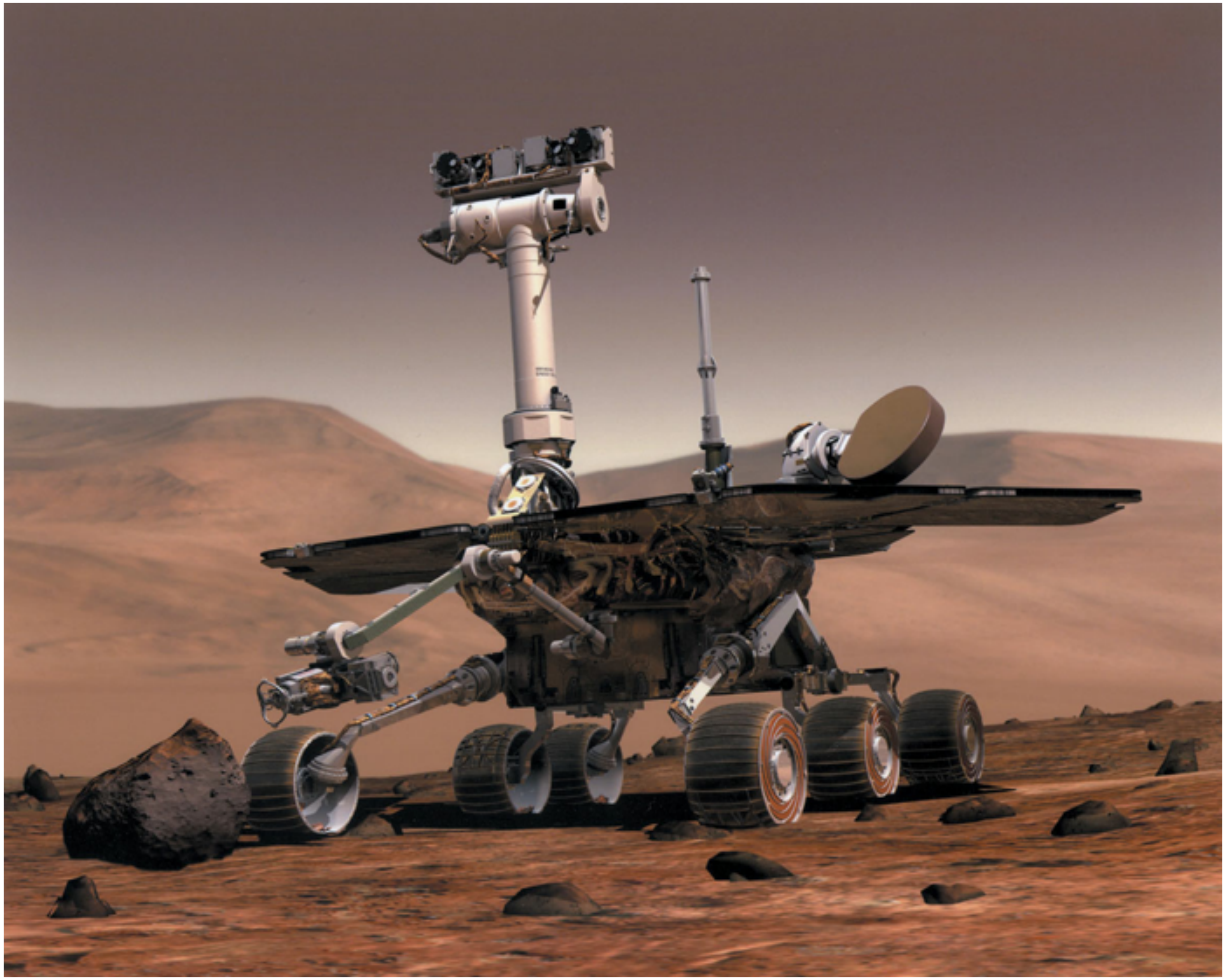
Where is C++ relevant?

embedded

supercomputing



realtime



datacenters

green computing

mobile computing

Motion
Patient info #245498
Monday, February 28, 3:35 PM
CLINICAL PATHOLOGY LABORATORY REQUEST FORM

multimedia systems

competition programming

Where is C++ relevant?

embedded
supercomputing
realtime
datacenters
green computing
mobile computing
multimedia systems
competition programming

We are working on a new project...
Which language to use?

can a script language do the job?

is there support for a vm based language?

is it convenient to use just C?

if no to all previous questions,
then you may want to consider C++

C++

History of C++

- PhD, Simula, BCPL (Cambridge)
- C with Classes (Cpre, 1979)
- First external paper (1981)
- C++ named (1983)
- CFront 1.0 (1985)
- TC++PL, Ed1 (1985)
- ANSI X3J16 meeting (1989)
- The Annotated C++ Reference Manual (1990)
- First WG21 meeting (1991)
- The Design and Evolution of C++ (1994)
- ISO/IEC 14882:1998 (C++98)
- ISO/IEC 14882:2003 (C++03)
- ISO/IEC TR 19768:2007 (C++TR1)
- ISO/IEC 14882:2011 (C++11)

(About C++ vs Java standardization)

“Must be interesting working with a language where they actually release their features rather than pushing them back every release”

Chris Searle, Java guru, private conversation, September 2012

Compilers with decent C++11 support

- Windows (Visual Studio, mingw/gcc, clang)
- Linux (gcc, clang)
- Mac (Xcode, clang, gcc)

Getting started with C++

- “hello, world”
- print arguments
- File structure
- OOP

```
$ alias c++=g++ -std=c++11
```

“hello, world”

“hello, world”

Hello.java

```
class Hello
{
    public static void main(String args[]) {
        System.out.println("hello, world");
    }
}
```

```
$ javac Hello.java
$ java Hello
hello, world
```

“hello, world”

Hello.java

```java
class Hello
{
    public static void main(String args[]) {
        System.out.println("hello, world");
    }
}
```

```
$ javac Hello.java
$ java Hello
hello, world
```

hello.cpp

```cpp
#include <iostream>

int main()
{
    std::cout << "hello, world" << std::endl;
}
```

```
$ c++ -o hello hello.cpp
$ ./hello
hello, world
```

print arguments

print arguments

PrintArgs.java

```
class PrintArgs
{
    public static void main(String args[]) {
        for (String arg : args)
            System.out.println(arg);
    }
}
```

```
$ javac PrintArgs.java
$ java PrintArgs 1 2 3
1
2
3
```

print arguments

PrintArgs.java

```java
class PrintArgs
{
    public static void main(String args[]) {
        for (String arg : args)
            System.out.println(arg);
    }
}
```

```
$ javac PrintArgs.java
$ java PrintArgs 1 2 3
1
2
3
```

printargs.cpp

```cpp
#include <iostream>
#include <vector>

int main(int argc, char * argv[])
{
    std::vector<std::string> args(argv + 1, argv + argc);
    for (std::string arg : args)
        std::cout << arg << std::endl;
}
```

```
$ c++ -o printargs printargs.cpp
$ ./printargs 1 2 3
1
2
3
```

print arguments

PrintArgs.java

```
class PrintArgs
{
    public static void main(String args[]) {
        for (String arg : args)
            System.out.println(arg);
    }
}
```

```
$ javac PrintArgs.java
$ java PrintArgs 1 2 3
1
2
3
```

printargs.cpp

```
#include <iostream>
#include <vector>

int main(int argc, char * argv[])
{
    std::vector<std::string> args(argv + 1, argv + argc);
    for (std::string arg : args)
        std::cout << arg << std::endl;
}
```

range-based for loops

```
$ c++ -o printargs printargs.cpp
$ ./printargs 1 2 3
1
2
3
```

example of file structure (C++11)

mymath/values.hpp

```
#include <vector>

namespace mymath {
    int sum(const std::vector<int> & values);
    int max(const std::vector<int> & values);
}
```

mymath/values.cpp

```
#include "mymath.hpp"

int mymath::sum(const std::vector<int> & values)
{
    int result = 0;
    for (int v : values)
        result += v;
    return result;
}

int mymath::max(const std::vector<int> & values)
{
    int result = values[0];
    for (auto v : values)
        if (v > result)
            result = v;
    return result;
}
```

mymath_demo.cpp

```
#include "mymath/values.hpp"

#include <iostream>
#include <vector>

int main()
{
    std::vector<int> values = {1,5,9,4};
    int sum = mymath::sum(values);
    int max = mymath::max(values);
    std::cout << sum << std::endl;
    std::cout << max << std::endl;
}
```

```
$ cd mymath
$ c++ -c values.cpp
$ ar -r mymath.a values.o
$ cd ..
$ c++ mymath_demo.cpp mymath/mymath.a
19
9
```

example of file structure (C++11)

mymath/values.hpp

```
#include <vector>

namespace mymath {
    int sum(const std::vector<int> & values);
    int max(const std::vector<int> & values);
}
```

mymath/values.cpp

```
#include "mymath.hpp"

int mymath::sum(const std::vector<int> & values)
{
    int result = 0;
    for (int v : values)
        result += v;
    return result;
}

int mymath::max(const std::vector<int> & values)
{
    int result = values[0];
    for (auto v : values)
        if (v > result)
            result = v;
    return result;
}
```

- keep declarations and definitions separate
- namespace corresponds to directory
- “importing” namespaces less common in C++

mymath_demo.cpp

```
#include "mymath/values.hpp"

#include <iostream>
#include <vector>

int main()
{
    std::vector<int> values = {1,5,9,4};
    int sum = mymath::sum(values);
    int max = mymath::max(values);
    std::cout << sum << std::endl;
    std::cout << max << std::endl;
}
```

```
$ cd mymath
$ c++ -c values.cpp
$ ar -r mymath.a values.o
$ cd ..
$ c++ mymath_demo.cpp mymath/mymath.a
19
9
```

example of file structure (C++11)

mymath/values.hpp

```
#include <vector>

namespace mymath {
    int sum(const std::vector<int> & values);
    int max(const std::vector<int> & values);
}
```

mymath/values.cpp

```
#include "mymath.hpp"

int mymath::sum(const std::vector<int> & values)
{
    int result = 0;
    for (int v : values)
        result += v;
    return result;
}

int mymath::max(const std::vector<int> & values)
{
    int result = values[0];
    for (auto v : values)
        if (v > result)
            result = v;
    return result;
}
```

auto deduction of type

- keep declarations and definitions separate
- namespace corresponds to directory
- “importing” namespaces less common in C++

mymath_demo.cpp

```
#include "mymath/values.hpp"

#include <iostream>
#include <vector>

int main()
{
    std::vector<int> values = {1,5,9,4};
    int sum = mymath::sum(values);
    int max = mymath::max(values);
    std::cout << sum << std::endl;
    std::cout << max << std::endl;
}
```

```
$ cd mymath
$ c++ -c values.cpp
$ ar -r mymath.a values.o
$ cd ..
$ c++ mymath_demo.cpp mymath/mymath.a
19
9
```

Example of OPP in C++11

```
#include <iostream>

class shape {
public:
    virtual ~shape() {}
    virtual double area() const = 0;
};

class square : public shape {
public:
    explicit square(double width) : width_(width) {}
    double area() const override { return width_ * width_; };
private:
    double width_;
};

void print_area(const shape & s)
{
    std::cout << "Area = " << s.area() << std::endl;
}

int main()
{
    square s(3);
    print_area(s);
}
```

Example of OPP in C++11

```
#include <iostream>

class shape {
public:
    virtual ~shape() {}
    virtual double area() const = 0;
};

class square : public shape {
public:
    explicit square(double width) : width_(width) {}
    double area() const override { return width_ * width_; };
private:
    double width_;
};

void print_area(const shape & s)
{
    std::cout << "Area = " << s.area() << std::endl;
}

int main()
{
    square s(3);
    print_area(s);
}
```

override specifier

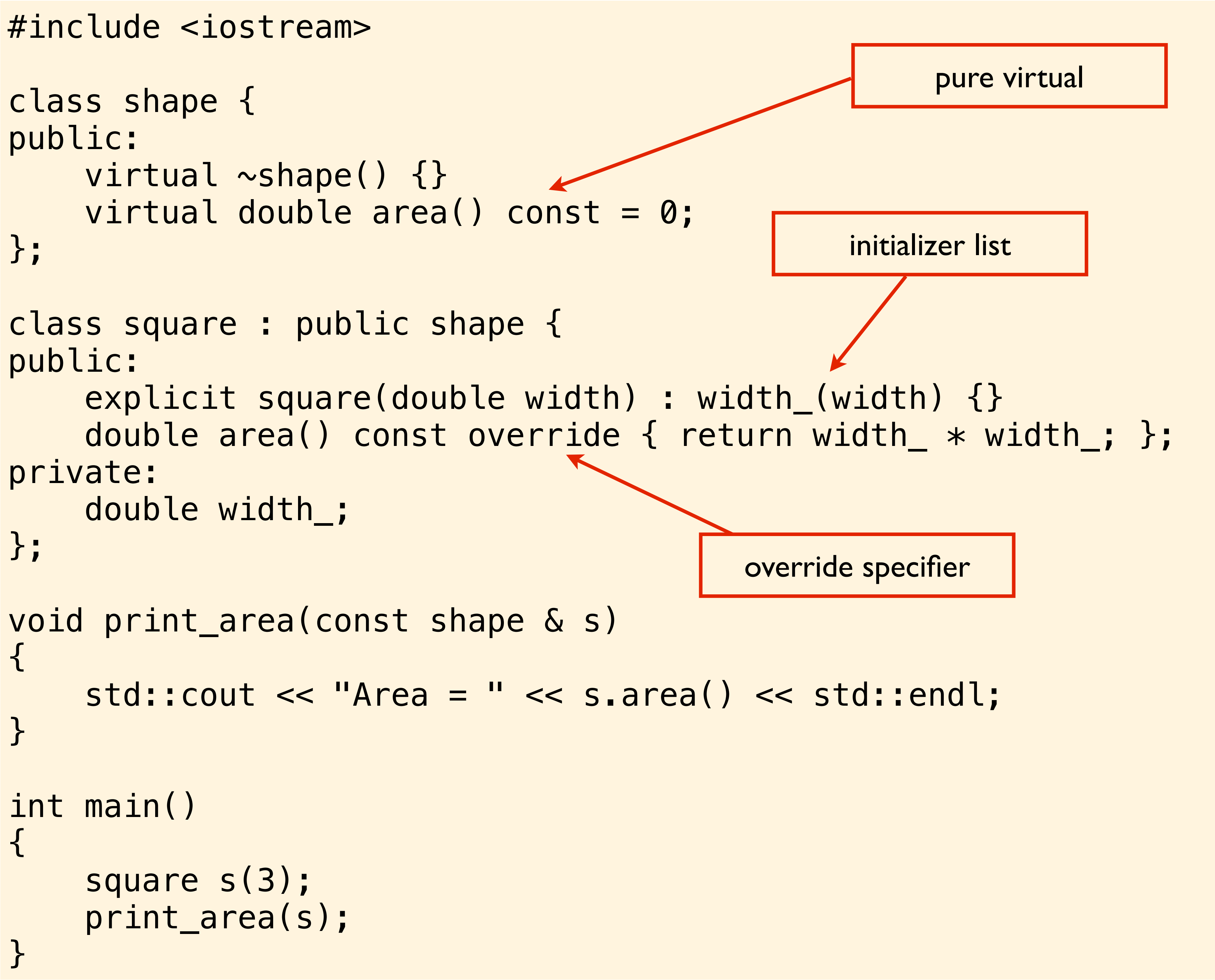
Example of OPP in C++11

```cpp
#include <iostream>

class shape {
public:
    virtual ~shape() {}
    virtual double area() const = 0;
};

class square : public shape {
public:
    explicit square(double width) : width_(width) {}
    double area() const override { return width_ * width_; };
private:
    double width_;
};

void print_area(const shape & s)
{
    std::cout << "Area = " << s.area() << std::endl;
}

int main()
{
    square s(3);
    print_area(s);
}
```

initializer list

override specifier

Example of OPP in C++11



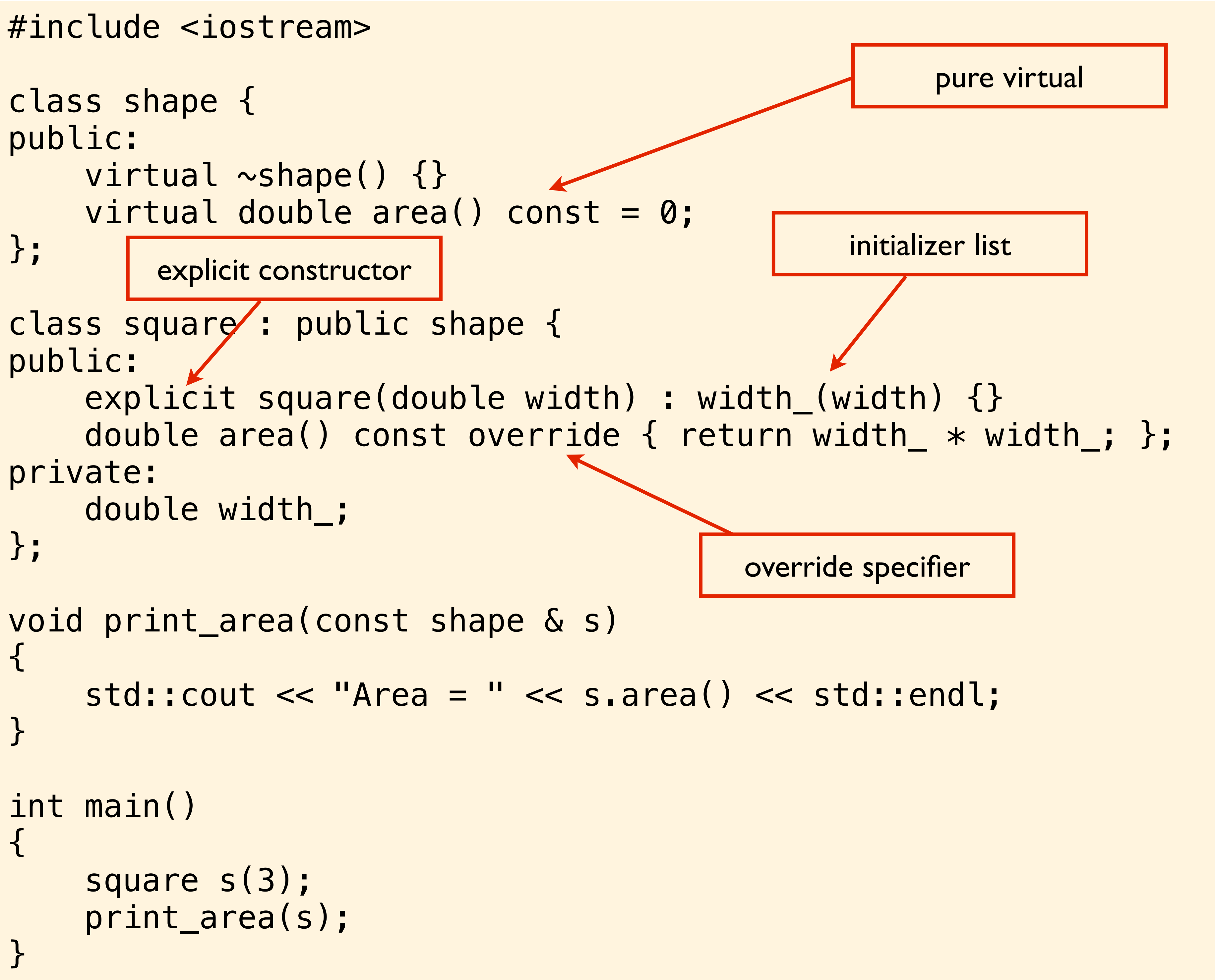
```
#include <iostream>

class shape {
public:
    virtual ~shape() {}
    virtual double area() const = 0;
};

class square : public shape {
public:
    explicit square(double width) : width_(width) {}
    double area() const override { return width_ * width_; };
private:
    double width_;
};

void print_area(const shape & s)
{
    std::cout << "Area = " << s.area() << std::endl;
}

int main()
{
    square s(3);
    print_area(s);
}
```

pure virtual

initializer list

override specifier

Example of OPP in C++11



```
#include <iostream>

class shape {
public:
    virtual ~shape() {}
    virtual double area() const = 0;
};

class square : public shape {
public:
    explicit square(double width) : width_(width) {}
    double area() const override { return width_ * width_; };
private:
    double width_;
};

void print_area(const shape & s)
{
    std::cout << "Area = " << s.area() << std::endl;
}

int main()
{
    square s(3);
    print_area(s);
}
```

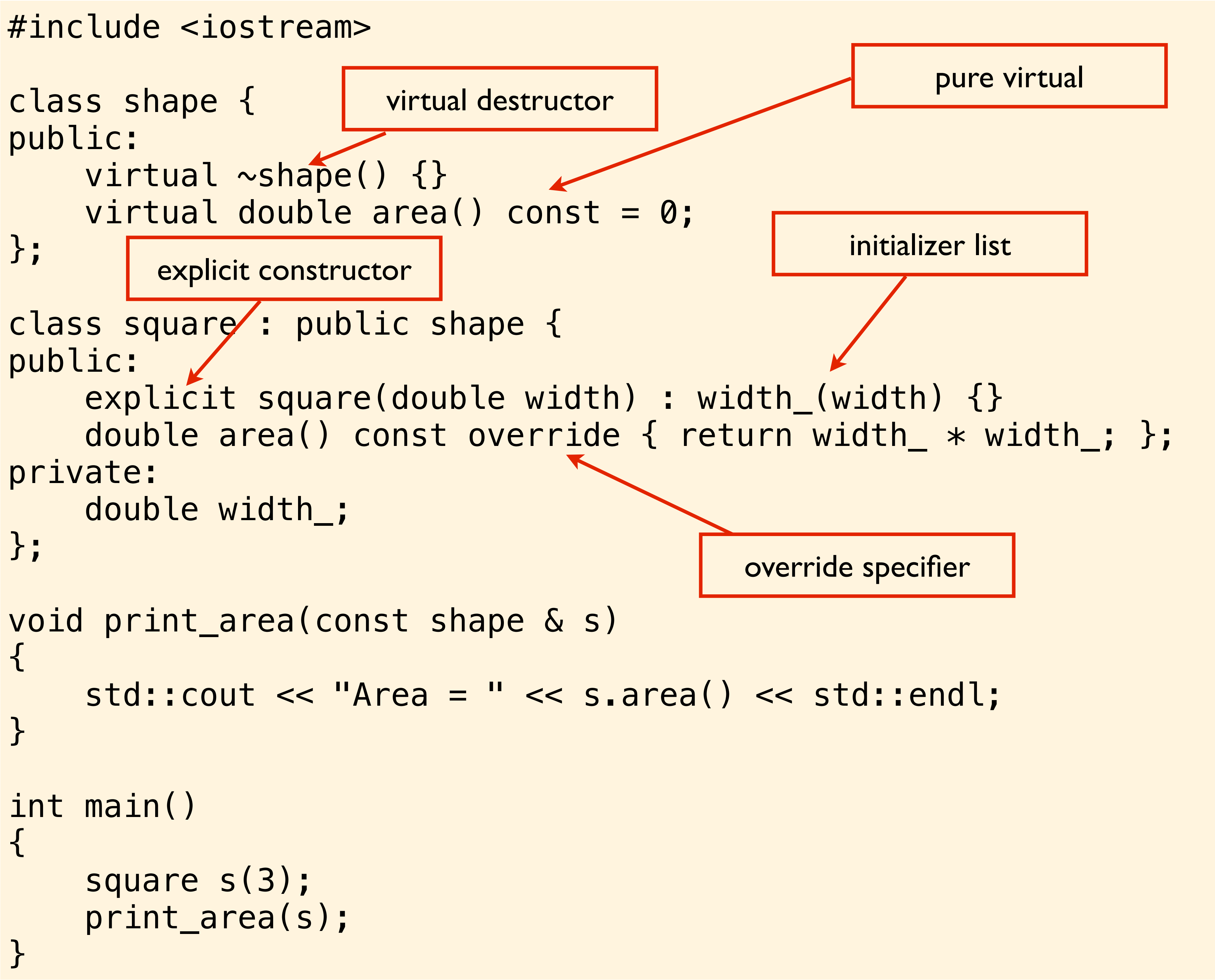
pure virtual

explicit constructor

initializer list

override specifier

Example of OPP in C++11



```
#include <iostream>

class shape {
public:
    virtual ~shape() {}
    virtual double area() const = 0;
};

class square : public shape {
public:
    explicit square(double width) : width_(width) {}
    double area() const override { return width_ * width_; };
private:
    double width_;
};

void print_area(const shape & s)
{
    std::cout << "Area = " << s.area() << std::endl;
}

int main()
{
    square s(3);
    print_area(s);
}
```

virtual destructor

pure virtual

initializer list

explicit constructor

override specifier

A glimpse into C++11

A glimpse into C++11

- variadic templates (~ “generics”)

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization (~ Arrays.asList)

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization (~ Arrays.asList)
- lambdas (~ “closures” / “on the fly functions”)

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization (~ Arrays.asList)
- lambdas (~ “closures” / “on the fly functions”)
- tuple (~ “on the fly compound data types”)

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization (~ Arrays.asList)
- lambdas (~ “closures” / “on the fly functions”)
- tuple (~ “on the fly compound data types”)
- rvalue refs and move semantics (~ JIT?)

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization (~ Arrays.asList)
- lambdas (~ “closures” / “on the fly functions”)
- tuple (~ “on the fly compound data types”)
- rvalue refs and move semantics (~ JIT?)
- smart pointers (~ “garbage collection”)

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization (~ Arrays.asList)
- lambdas (~ “closures” / “on the fly functions”)
- tuple (~ “on the fly compound data types”)
- rvalue refs and move semantics (~ JIT?)
- smart pointers (~ “garbage collection”)
- async, future, promise (~ “higher-order parallelism”)

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization (~ Arrays.asList)
- lambdas (~ “closures” / “on the fly functions”)
- tuple (~ “on the fly compound data types”)
- rvalue refs and move semantics (~ JIT?)
- smart pointers (~ “garbage collection”)
- async, future, promise (~ “higher-order parallelism”)
- user-defined literals (~ “DSL syntax”)

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization (~ Arrays.asList)
- lambdas (~ “closures” / “on the fly functions”)
- tuple (~ “on the fly compound data types”)
- rvalue refs and move semantics (~ JIT?)
- smart pointers (~ “garbage collection”)
- async, future, promise (~ “higher-order parallelism”)
- user-defined literals (~ “DSL syntax”)

“There are two types of people in the world: Those who can extrapolate conclusions from incomplete data.”

templates

simple templates

```
#include <iostream>

template<typename T>
void sayit(const T & value)
{
    std::cout << value << std::endl;
}

int main()
{
    sayit(42);
    sayit(3.14);
    sayit("Hello");
}
```

```
42
3.14
Hello
```

simple templates

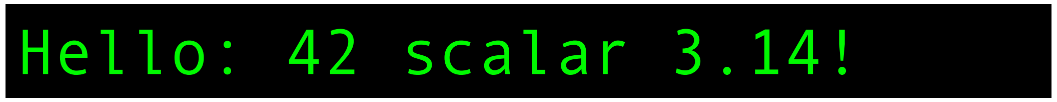
```cpp
#include <iostream>

template<typename T>
void sayit(const T & value)
{
    std::cout << value << std::endl;
}

int main()
{
    sayit(42);
    sayit(3.14);
    sayit("Hello");
}
```

```
42
3.14
Hello
```

```cpp
#include <iostream>

void sayit(const int & value)
{
    std::cout << value << std::endl;
}

void sayit(const double & value)
{
    std::cout << value << std::endl;
}

void sayit(const std::string & value)
{
    std::cout << value << std::endl;
}

int main()
{
    sayit(42);
    sayit(3.14);
    sayit("Hello");
}
```

variadic templates

```
#include <iostream>

void my_printf(const char * s)
{
    while (*s)
        std::cout << *s++;
}


template<typename T, typename... Args>
void my_printf(const char * s, const T & value, const Args & ... args)
{
    while (*s) {
        if (*s == '*') {
            std::cout << value;
            my_printf(++s, args...);
            return;
        }
        std::cout << *s++;
    }
}

int main()
{
    std::string s("Hello");
    my_printf("*: * scalar *!\n", s, 42, 3.14);
}
```



```
Hello: 42 scalar 3.14!
```

variadic templates

```
#include <iostream>

void my_printf(const char * s)
{
    while (*s)
        std::cout << *s++;
}


template<typename T, typename... Args>
void my_printf(const char * s, const T & value, const Args & ... args)
{
    while (*s) {
        if (*s == '*') {
            std::cout << value;
            my_printf(++s, args...);
            return;
        }
        std::cout << *s++;
    }
}

int main()
{
    std::string s("Hello");
    my_printf("*: * scalar *!\n", s, 42, 3.14);
}
```

```
Hello: 42 scalar 3.14!
```

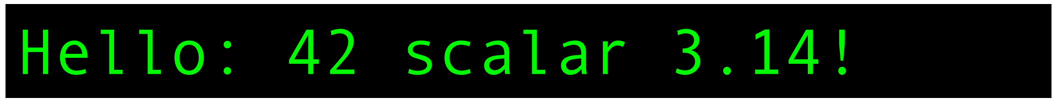
variadic templates

```
#include <iostream>

void my_printf(const char * s)
{
    while (*s)
        std::cout << *s++;
}


template<typename T, typename... Args>
void my_printf(const char * s, const T & value, const Args & ... args)
{
    while (*s) {
        if (*s == '*') {
            std::cout << value;
            my_printf(++s, args...);
            return;
        }
        std::cout << *s++;
    }
}

int main()
{
    std::string s("Hello");
    my_printf("*: * scalar *!\n", s, 42, 3.14);
}
```

```
Hello: 42 scalar 3.14!
```

variadic templates

```
#include <iostream>

void my_printf(const char * s)
{
    while (*s)
        std::cout << *s++;
}


template<typename T, typename... Args>
void my_printf(const char * s, const T & value, const Args & ... args)
{
    while (*s) {
        if (*s == '*') {
            std::cout << value;
            my_printf(++s, args...);
            return;
        }
        std::cout << *s++;
    }
}

int main()
{
    std::string s("Hello");
    my_printf("*: * scalar *!\n", s, 42, 3.14);
}
```



```
Hello: 42 scalar 3.14!
```

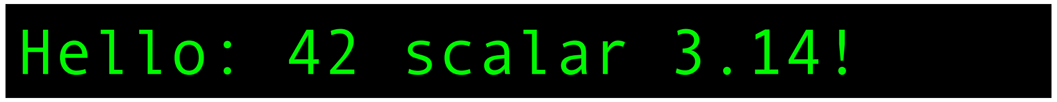
variadic templates

```cpp
#include <iostream>

void my_printf(const char * s)
{
    while (*s)
        std::cout << *s++;
}


template<typename T, typename... Args>
void my_printf(const char * s, const T & value, const Args & ... args)
{
    while (*s) {
        if (*s == '*') {
            std::cout << value;
            my_printf(++s, args...);
            return;
        }
        std::cout << *s++;
    }
}

int main()
{
    std::string s("Hello");
    my_printf("*: * scalar *!\n", s, 42, 3.14);
}
```

```
Hello: 42 scalar 3.14!
```

variadic templates

```
#include <iostream>

void my_printf(const char * s)
{
    while (*s)
        std::cout << *s++;
}


template<typename T, typename... Args>
void my_printf(const char * s, const T & value, const Args & ... args)
{
    while (*s) {
        if (*s == '*') {
            std::cout << value;
            my_printf(++s, args...);
            return;
        }
        std::cout << *s++;
    }
}

int main()
{
    std::string s("Hello");
    my_printf("*: * scalar *!\n", s, 42, 3.14);
}
```



```
Hello: 42 scalar 3.14!
```

```
#include <iostream>

void my_printf(const char * s)
{
    while (*s)
        std::cout << *s++;
}

void my_printf(const char * s, double d)
{
    while (*s) {
        if (*s == '*') {
            std::cout << d;
            my_printf(++s);
            return;
        }
        std::cout << *s++;
    }
}

void my_printf(const char * s, int i, double d)
{
    while (*s) {
        if (*s == '*') {
            std::cout << i;
            my_printf(++s, d);
            return;
        }
        std::cout << *s++;
    }
}

void my_printf(const char * s, const std::string & str,
    int i, double d)
{
    while (*s) {
        if (*s == '*') {
            std::cout << str;
            my_printf(++s, i, d);
            return;
        }
        std::cout << *s++;
    }
}

int main()
{
    std::string s("Hello");
    my_printf("*: * scalar *!\n", s, 42, 3.14);
}
```

tuple, decltype, auto

tuple

```
#include <iostream>
#include <tuple>

int main()
{
    std::tuple<std::string, int, double> person =
        std::make_tuple("Olve", 1971, 180.1);

    std::cout << std::get<0>(person)
              << " (" << std::get<1>(person) << ") : "
              << std::get<2>(person) << " cm" << std::endl;
}
```

```
Olve (1971) : 180.1 cm
```

tuple

```
#include <iostream>
#include <tuple>

int main()
{
    std::tuple<std::string, int, double> person =
        std::make_tuple("Olve", 1971, 180.1);

    std::cout << std::get<0>(person)
              << " (" << std::get<1>(person) << ") : "
              << std::get<2>(person) << " cm" << std::endl;
}
```

```
Olve (1971) : 180.1 cm
```

tuple (decltype example)

```
#include <iostream>
#include <tuple>

int main()
{
    decltype(std::make_tuple("Olve", 1971, 180.1)) person =
        std::make_tuple("Olve", 1971, 180.1);

    std::cout << std::get<0>(person)
              << " (" << std::get<1>(person) << ") : "
              << std::get<2>(person) << " cm" << std::endl;
}
```

```
Olve (1971) : 180.1 cm
```

tuple (auto example)

```
#include <iostream>
#include <tuple>

int main()
{
    auto person =
        std::make_tuple("Olve", 1971, 180.1);

    std::cout << std::get<0>(person)
              << " (" << std::get<1>(person) << ") : "
              << std::get<2>(person) << " cm" << std::endl;
}
```

```
Olve (1971) : 180.1 cm
```

tuple (auto example)

```
#include <iostream>
#include <tuple>

int main()
{
    auto person =
        std::make_tuple("Olve", 1971, 180.1);

    std::cout << std::get<0>(person)
              << " (" << std::get<1>(person) << ") : "
              << std::get<2>(person) << " cm" << std::endl;
}
```

```
Olve (1971) : 180.1 cm
```

lambda

lambda

```
#include <iostream>

int main()
{
    auto func = [](int a, int b) { return a * b; };

    std::cout << "The answer is " << func(6,7) << std::endl;
}
```

```
The answer is 42
```

lambda

```cpp
#include <iostream>

int main()
{
    auto func = [](int a, int b) { return a * b; };

    std::cout << "The answer is " << func(6,7) << std::endl;
}
```

```
The answer is 42
```

lambda

```
#include <iostream>

int main()
{
    auto func = [](int a, int b) { return a * b; };

    std::cout << "The answer is " << func(6,7) << std::endl;
}
```

```
The answer is 42
```

lambda

```
#include <iostream>

int main()
{
    auto func = [](int a, int b) { return a * b; };

    std::cout << "The answer is " << func(6,7) << std::endl;
}
```

```
The answer is 42
```

lambda

```
#include <iostream>

int main()
{
    auto func = [](int a, int b) { return a * b; };

    std::cout << "The answer is " << func(6,7) << std::endl;
}
```

```
The answer is 42
```

lambda

```
#include <iostream>

int main()
{
    int answer = 0;
    int b = 7;
    auto func = [&answer,b](int a) { answer = a * b; };

    func(6);

    std::cout << "The answer is " << answer << std::endl;
}
```

```
The answer is 42
```

lambda

```
#include <iostream>

int main()
{
    int answer = 0;
    int b = 7;
    auto func = [&answer,b](int a) { answer = a * b; };

    func(6);

    std::cout << "The answer is " << answer << std::endl;
}
```

```
The answer is 42
```

lambda

```
#include <iostream>

int main()
{
    int answer = 0;
    int b = 7;
    auto func = [&answer,b](int a) { answer = a * b; };

    func(6);

    std::cout << "The answer is " << answer << std::endl;
}
```

```
The answer is 42
```

lambda

```
#include <iostream>

int main()
{
    int answer = 0;
    int b = 7;
    auto func = [&answer,b](int a) { answer = a * b; };

    func(6);

    std::cout << "The answer is " << answer << std::endl;
}
```

```
The answer is 42
```

standard library and move semantics

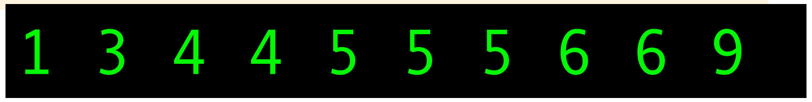
algorithm

```
#include <iostream>
#include <algorithm>
#include <vector>

bool compare_int(int a, int b)
{
    return a < b;
}

void print_int(int i)
{
    std::cout << i << " ";
}

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), compare_int);
    std::for_each(v.begin(), v.end(), print_int);
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm

```
#include <iostream>
#include <algorithm>
#include <vector>

bool compare_int(int a, int b)
{
    return a < b;
}

void print_int(int i)
{
    std::cout << i << " ";
}

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), compare_int);
    std::for_each(v.begin(), v.end(), print_int);
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```



```
1 3 4 4 5 5 5 6 6 9
```

algorithm

```
#include <iostream>
#include <algorithm>
#include <vector>

bool compare_int(int a, int b)
{
    return a < b;
}

void print_int(int i)
{
    std::cout << i << " ";
}

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), compare_int);
    std::for_each(v.begin(), v.end(), print_int);
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm

```
#include <iostream>
#include <algorithm>
#include <vector>

bool compare_int(int a, int b)
{
    return a < b;
}

void print_int(int i)
{
    std::cout << i << " ";
}

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), compare_int);
    std::for_each(v.begin(), v.end(), print_int);
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm (with lambda)

```
#include <iostream>
#include <algorithm>
#include <vector>

bool compare_int(int a, int b)
{
    return a < b;
}

void print_int(int i)
{
    std::cout << i << " ";
}

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm (with lambda)

```
#include <iostream>
#include <algorithm>
#include <vector>

~~bool compare_int(int a, int b)~~
~~{~~
~~    return a < b;~~
~~}~~

~~void print_int(int i)~~
~~{~~
~~    std::cout << i << " ";~~
~~}~~

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm (with lambda)

```
#include <iostream>
#include <algorithm>
#include <vector>











void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm (with lambda)

```
#include <iostream>
#include <algorithm>
#include <vector>







void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```

```
1 3 4 4 5 5 5 6 6 9
```

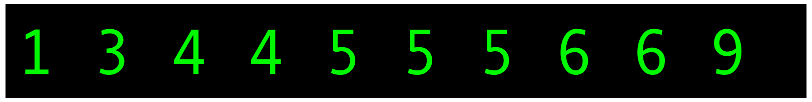
algorithm (with lambda)

```
#include <iostream>
#include <algorithm>
#include <vector>




void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm (with lambda)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```



```
1 3 4 4 5 5 5 6 6 9
```

algorithm (with lambda)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    std::vector<int> v = create_list();
    sort_and_print(v);
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm (passing reference to tmp object)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```

algorithm (passing reference to tmp object)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```

```
error: invalid initialization of non-const
reference of type 'std::vector<int>&' from an
rvalue of type 'std::vector<int>'
```

algorithm (passing reference to tmp object)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```

?

```
error: invalid initialization of non-const
reference of type 'std::vector<int>&' from an
rvalue of type 'std::vector<int>'
```

algorithm (passing reference to tmp object)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int> & v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```

?

```
error: invalid initialization of non-const
reference of type 'std::vector<int>&' from an
rvalue of type 'std::vector<int>'
```

algorithm (passing reference to tmp object)

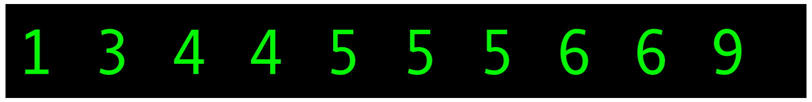
```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int>   v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm (passing reference to tmp object)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int>   v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```

?



```
1 3 4 4 5 5 5 6 6 9
```

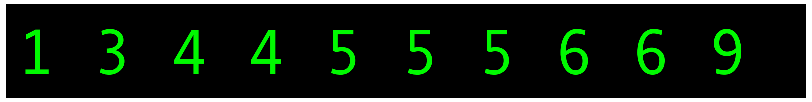
algorithm (passing reference to tmp object)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int> && v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm (rvalue ref and move semantics)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int> && v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```



```
1 3 4 4 5 5 5 6 6 9
```

algorithm (rvalue ref and move semantics)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int>    v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```

```
1 3 4 4 5 5 5 6 6 9
```

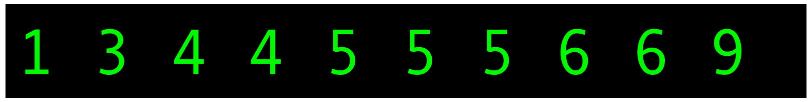
algorithm (rvalue ref and move semantics)

```
#include <iostream>
#include <algorithm>
#include <vector>

void sort_and_print(std::vector<int>    v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    sort_and_print(create_list());
}
```

```
1 3 4 4 5 5 5 6 6 9
```

algorithm (rvalue ref and move semantics)

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

void print_sorted(std::vector<int> v)
{
    std::sort(v.begin(), v.end(), [](int a, int b) { return a < b; });
    std::for_each(v.begin(), v.end(), [](int i) { std::cout << i << " "; });
}

std::vector<int> create_list()
{
    return {1,5,6,4,5,6,4,5,9,3};
}

int main()
{
    print_sorted(create_list());
}
```



```
1 3 4 4 5 5 5 6 6 9
```

combined example
(using, tuple, sorting, tie)

tuples, algorithms and lambda (using example)

```cpp
#include <iostream>
#include <tuple>
#include <vector>
#include <algorithm>

int main()
{
    using People = std::vector<std::tuple<std::string, int, double>>;

    People people;
    people.push_back(std::make_tuple("Olve", 1971, 180.1));
    people.push_back(std::make_tuple("Lars Gullik", 1972, 185.7));

    std::sort(people.begin(), people.end(),
              [](const People::value_type & p1, const People::value_type & p2) {
                  return std::get<2>(p1) > std::get<2>(p2);
              });

    for (auto p : people) {
        std::string name;
        int year;
        double height;
        std::tie(name, year, height) = p;
        std::cout << name << " (" << year << ") : "
                  << height << " cm" << std::endl;
    }
}
```

```
Lars Gullik (1972) : 185.7 cm
Olve (1971) : 180.1 cm
```

higher-order parallelism

async, futures and promises

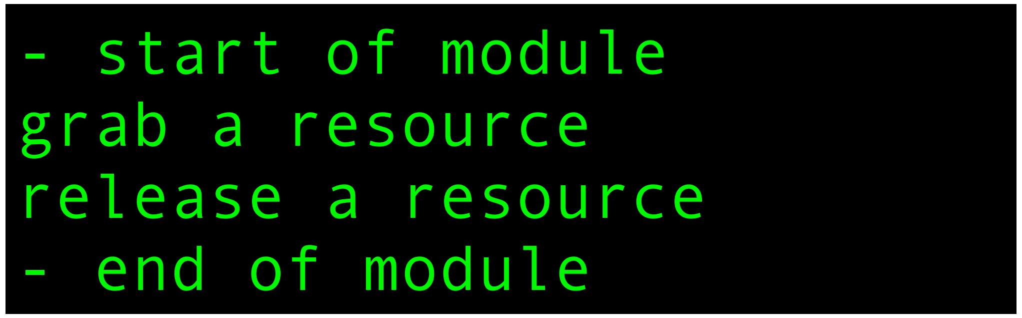
```
#include <iostream>
#include <string>
#include <future>

int print(const std::string & s)
{
    for (char c : s)
        std::cout.put(c);
    std::cout << std::endl;
    return 21;
}

int main()
{
    auto f1 = std::async (print, "First thread");
    auto f2 = std::async (std::launch::async, print, "Second thread");
    print("Main thread");
    std::cout << "The answer is " << (f1.get() + f2.get()) << std::endl;
}
```

```
MSaeicno ntdh rtehardead

First thread
The answer is 42
```

resource management

new/delete vs smart pointers

```
#include <iostream>


class myresource {
public:
    myresource() { std::cout << "grab a resource" << std::endl; }
    ~myresource() { std::cout << "release a resource" << std::endl; }
};

void do_something()
{
    myresource * res = new myresource;
    // .. do something
    delete res;
}

int main()
{
    std::cout << "- start of module" << std::endl;
    do_something();
    std::cout << "- end of module" << std::endl;
}
```



```
- start of module
grab a resource
release a resource
- end of module
```

new/delete vs smart pointers

```
#include <iostream>
#include <memory>

class myresource {
public:
    myresource() { std::cout << "grab a resource" << std::endl; }
    ~myresource() { std::cout << "release a resource" << std::endl; }
};

void do_something()
{
    std::unique_ptr<myresource> res(new myresource);
    // .. do something

}

int main()
{
    std::cout << "- start of module" << std::endl;
    do_something();
    std::cout << "- end of module" << std::endl;
}
```

```
- start of module
grab a resource
release a resource
- end of module
```

new/delete vs smart pointers

```cpp
#include <iostream>


class myresource {
public:
    myresource() { std::cout << "grab a resource" << std::endl; }
    ~myresource() { std::cout << "release a resource" << std::endl; }
};

void do_something()
{
    myresource res;
    // .. do something

}

int main()
{
    std::cout << "- start of module" << std::endl;
    do_something();
    std::cout << "- end of module" << std::endl;
}
```

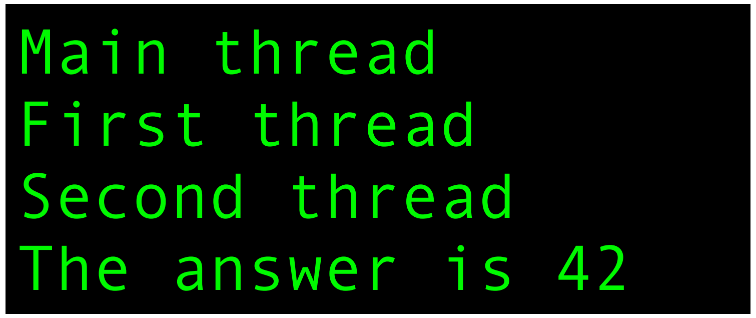
```
- start of module
grab a resource
release a resource
- end of module
```

async, futures and promises (revisited)

```
#include <iostream>
#include <string>
#include <future>
#include <thread>

std::mutex mymutex;

int print(const std::string & s)
{
    std::lock_guard<std::mutex> mylock(mymutex);
    for (char c : s)
        std::cout.put(c);
    std::cout << std::endl;
    return 21;
}

int main()
{
    auto f1 = std::async (print, "First thread");
    auto f2 = std::async (std::launch::async, print, "Second thread");
    print("Main thread");
    std::cout << "The answer is " << (f1.get() + f2.get()) << std::endl;
}
```

async, futures and promises (revisited)

```
#include <iostream>
#include <string>
#include <future>
#include <thread>

std::mutex mymutex;

int print(const std::string & s)
{
    std::lock_guard<std::mutex> mylock(mymutex);
    for (char c : s)
        std::cout.put(c);
    std::cout << std::endl;
    return 21;
}

int main()
{
    auto f1 = std::async (print, "First thread");
    auto f2 = std::async (std::launch::async, print, "Second thread");
    print("Main thread");
    std::cout << "The answer is " << (f1.get() + f2.get()) << std::endl;
}
```



```
Main thread
First thread
Second thread
The answer is 42
```

async, futures and promises (revisited)

```
#include <iostream>
#include <string>
#include <future>
#include <thread>

std::mutex mymutex;

int print(const std::string & s)
{
    std::lock_guard<std::mutex> mylock(mymutex);
    for (char c : s)
        std::cout.put(c);
    std::cout << std::endl;
    return 21;
}

int main()
{
    auto f1 = std::async (print, "First thread");
    auto f2 = std::async (std::launch::async, print, "Second thread");
    print("Main thread");
    std::cout << "The answer is " << (f1.get() + f2.get()) << std::endl;
}
```

```
Main thread
First thread
Second thread
The answer is 42
```

or

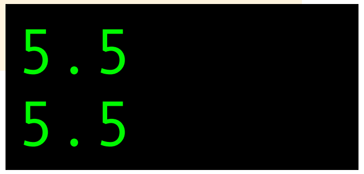
```
Second thread
Main thread
First thread
The answer is 42
```

or ...

user defined literals

user defined literals

```
#include <iostream>

class FontSize
{
public:
    explicit FontSize(double s) : size_(s) {}
    explicit operator double () const { return size_; }
    double size() const { return size_; }
private:
    double size_;
};

auto operator "" _pt (long double fs) -> decltype(FontSize(fs))
{
    return FontSize(fs);
}

int main()
{
    //FontSize fs1 = 5.0;                  // error
    FontSize fs2 = FontSize(5.2);          // ok
    FontSize fs3 = 5.5_pt;                 // ok
    //std::cout << fs2 << std::endl;       // error
    std::cout << double(fs3) << std::endl; // ok
    std::cout << fs3.size() << std::endl;  // ok
}
```



```
5.5
5.5
```

... and much more...

A glimpse into C++11

- variadic templates (~ “generics”)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization (~ Arrays.asList)
- lambdas (~ “closures” / “on the fly functions”)
- tuple (~ “on the fly compound data types”)
- rvalue refs and move semantics (~ JIT?)
- smart pointers (~ “garbage collection”)
- async, future, promise (~ “high order parallelism”)
- user-defined literals (~ “DSL syntax”)

?

Why C++11?

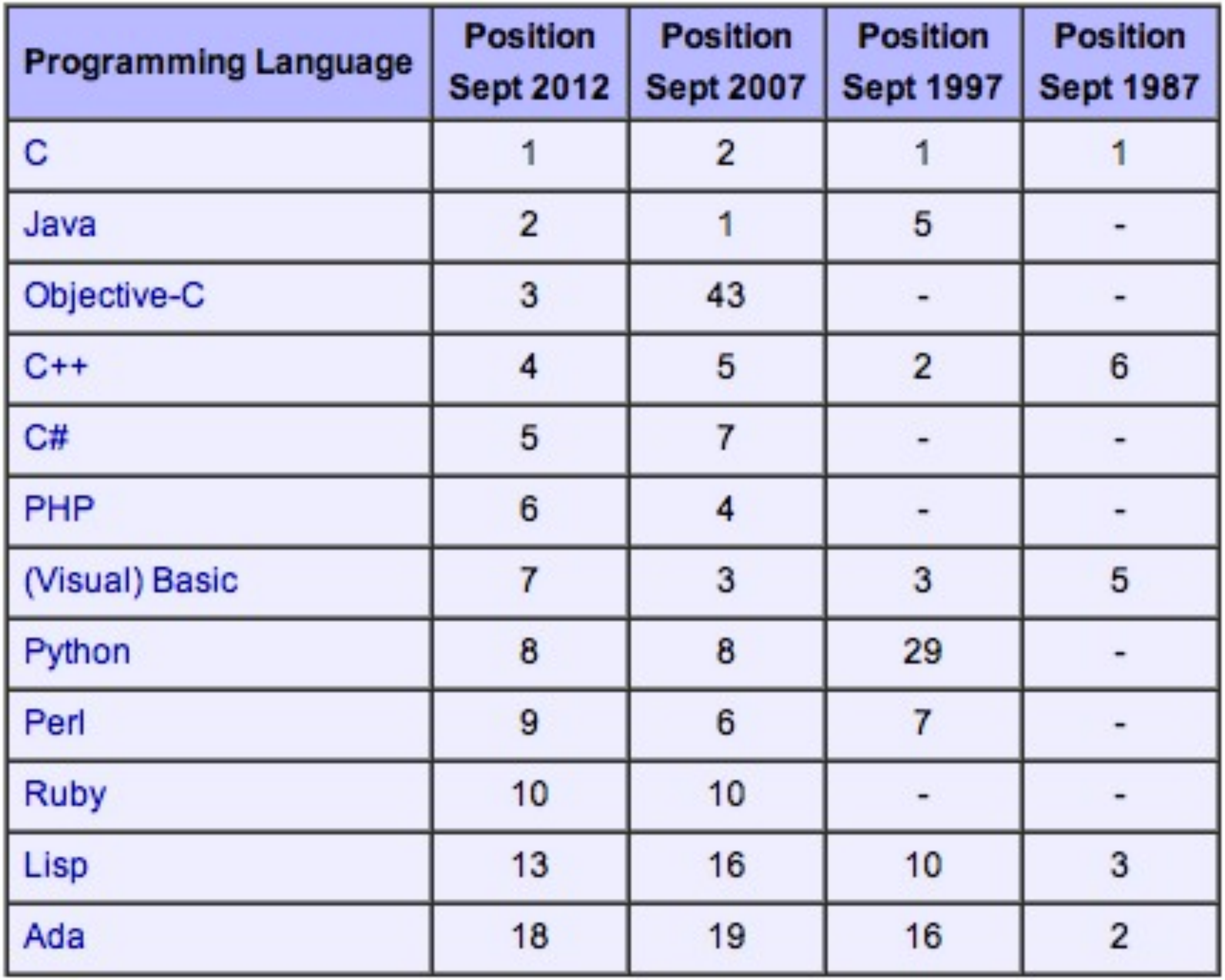
Why C++?

Why C?

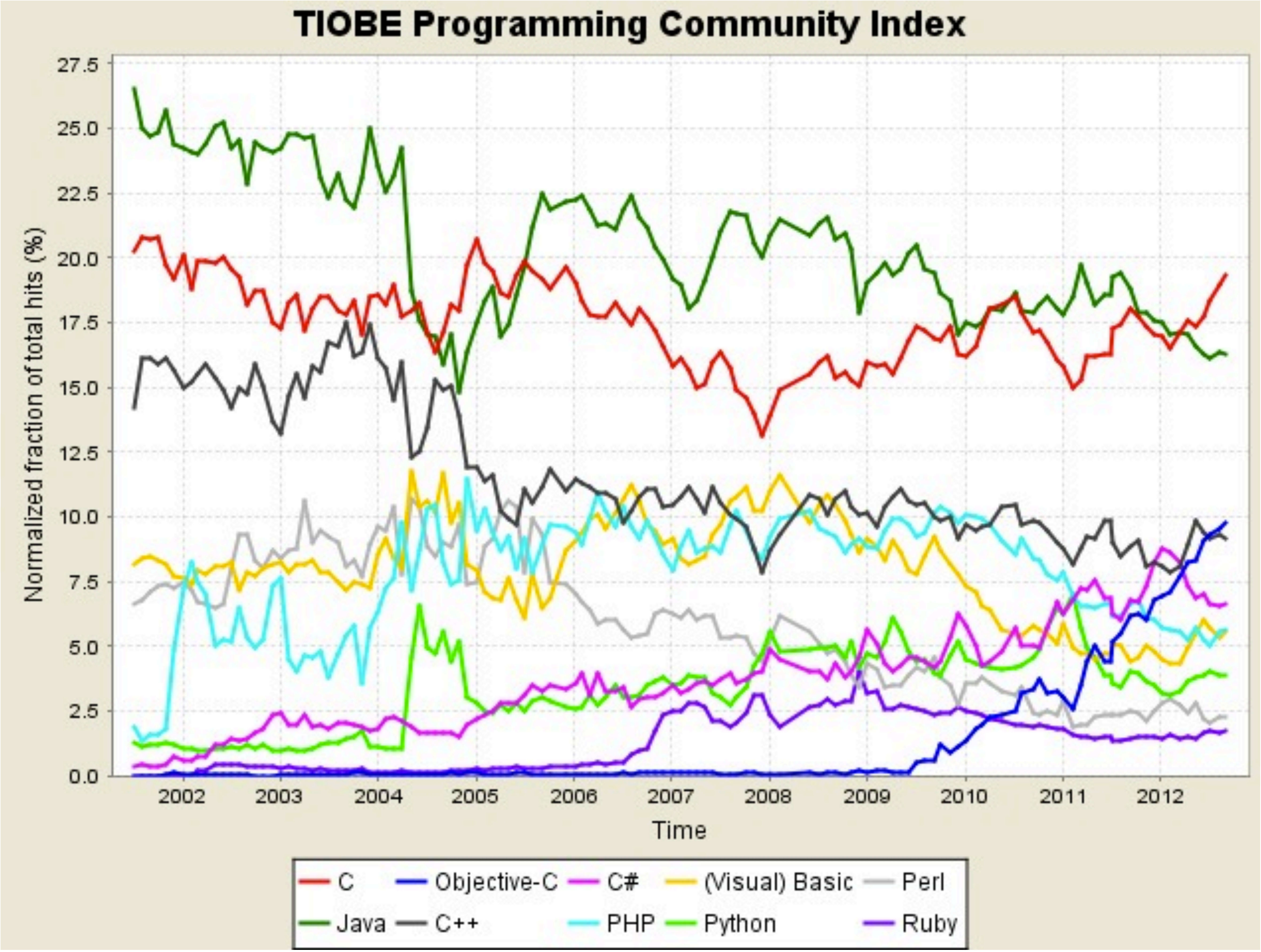
Virtual vs **Native**

Productivity vs **Performance**

Effectiveness vs **Efficiency**



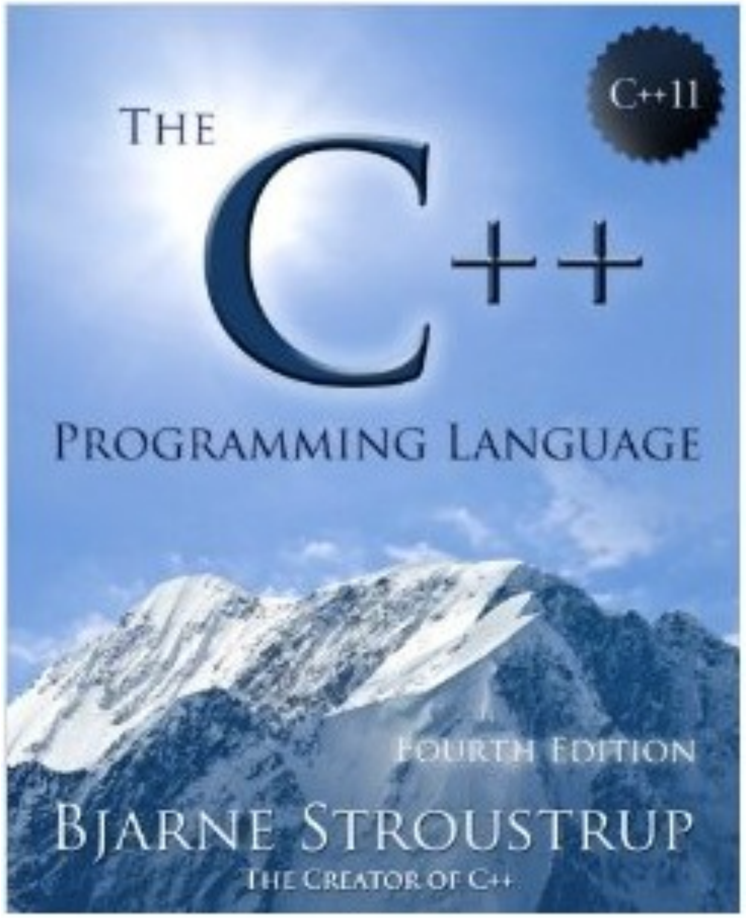
Programming Language	Position Sept 2012	Position Sept 2007	Position Sept 1997	Position Sept 1987
C	1	2	1	1
Java	2	1	5	-
Objective-C	3	43	-	-
C++	4	5	2	6
C#	5	7	-	-
PHP	6	4	-	-
(Visual) Basic	7	3	3	5
Python	8	8	29	-
Perl	9	6	7	-
Ruby	10	10	-	-
Lisp	13	16	10	3
Ada	18	19	16	2



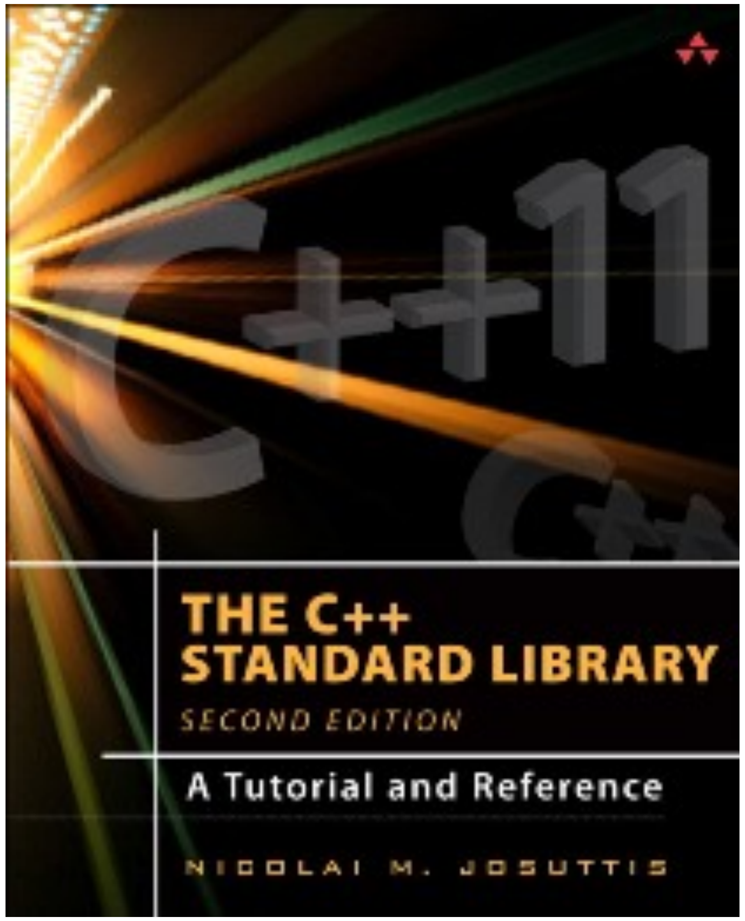
www.tiobe.com

Sep 2012

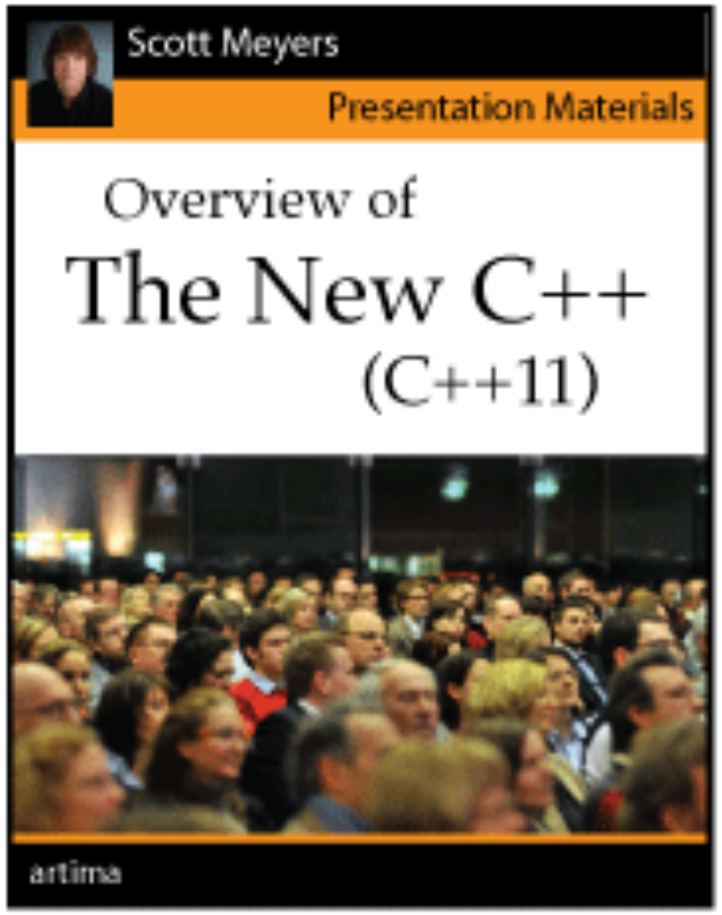
Where to find out more about C++11



(4ed, March 2013)



(2ed, April 2012)



(PDF, last update Jan 2012)

http://en.wikipedia.org/wiki/C++11

http://www.open-std.org/jtc1/sc22/wg21/

http://en.cppreference.com/w/

C++ is difficult! Takes years to learn, and a decade to master.

C++ is difficult! Takes years to learn, and a decade to master.

so you better start early!

C++ is difficult! Takes years to learn, and a decade to master.

so you better start early!

!