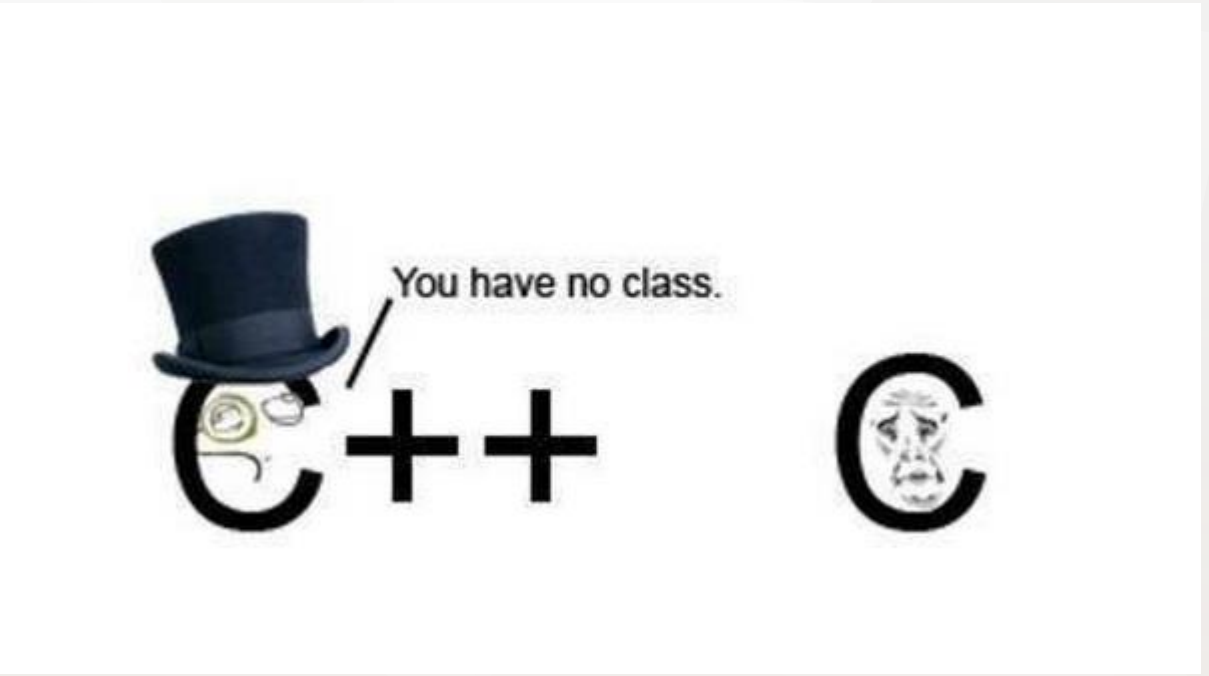
History of C++

1979 – 1991

As part of the HOPL course, meeting 3

Mattias Nordahl



Bjarne Stroustrup

Ph.D thesis (late 1970s), Cambridge University in England

Simulator for running programs on distributed systems

Simula helpful, especially classes and inheritance

But could not scale – Slow compile and linking, 80% garbage collection despite resource management and no garbage

Never again(!) without suitable tools

Bjarne Stroustrup

What constituts a ”suitable tool” for large projects?

- Simula features: classes and hierarchies, concurrency, static type checking
- BCPL features: Fast programs, easy to link separately compiled units, linkage convention to combine units written in C, ALGOL 68, FORTRAN, BCPL, …
- Highly portable

C with classes

April 1979, Bell Laboratories, distributed systems

- Close to hardware (memory management, network drivers, …)
- But also higher order design

Set aside developing a suitable tool

- → Added Simula's class concept to C

October 1979, Cpre

March 1980, 1 real usage and multiple experiments

Why C?

Alternatives considered included:

Modula-2, Smalltalk, Ada, …

C because it was:

- Flexible (apply to most areas, use most techniques)
- Efficient (low level, efficiently utilize hardware resources)
- Available (on both small and large computers)
- Portable (relatively easy to port to other hardware or OS)

1980 implementation

- Classes and class hierarchies (base and derived classes)
- public/private access control
- Constructors and destructors
- *call* and *return* functions (later removed)
- *friend* classes
- Type checking and conversion of function argument

(and later, 1981)

- Inline functions
- Default arguments
- Overloading of the assignment operator

call and return

Automatically called before and after every call of class member functions.

Used in monitor class to provide synchronization

```
class monitor : object {
        /* ... */
        call() { /* grab lock */ }
        return() { /* release lock */ }
};
```

C++

By 1982, C with Classes had become a “medium success”

- Useful enough in and by itself to be worth it
- Not attractive enough for support and development organizations

So

- Drop it, or
- Create new, better language

Still called C with Classes

new C or *C* vs *old C or plain C*

C84 → C++

Cfront

C++ features

- Virtual functions (Simula)
- Function name and operator overloading (ALGOL 68)
- References (ALGOL 68)
- Constants (*const*)
- User-controlled free-store memory control (*new* and *delete*)
- Improved type casting
- Other minor changes
  - BCPL style comments // and /* */
  - Variable declarations anywhere in block (ALGOL 68)

What is-paper

What is Object-Oriented Programming? (1986)

What C++ **could** do vs what is **ought** to be able to do

Lists three deficiencies of C++

- No parameterized types (*)
- No exception handling
- No multiple inheritance (*)

* = Considered for 1982

C++ 2.0 (1989)

Improved generality

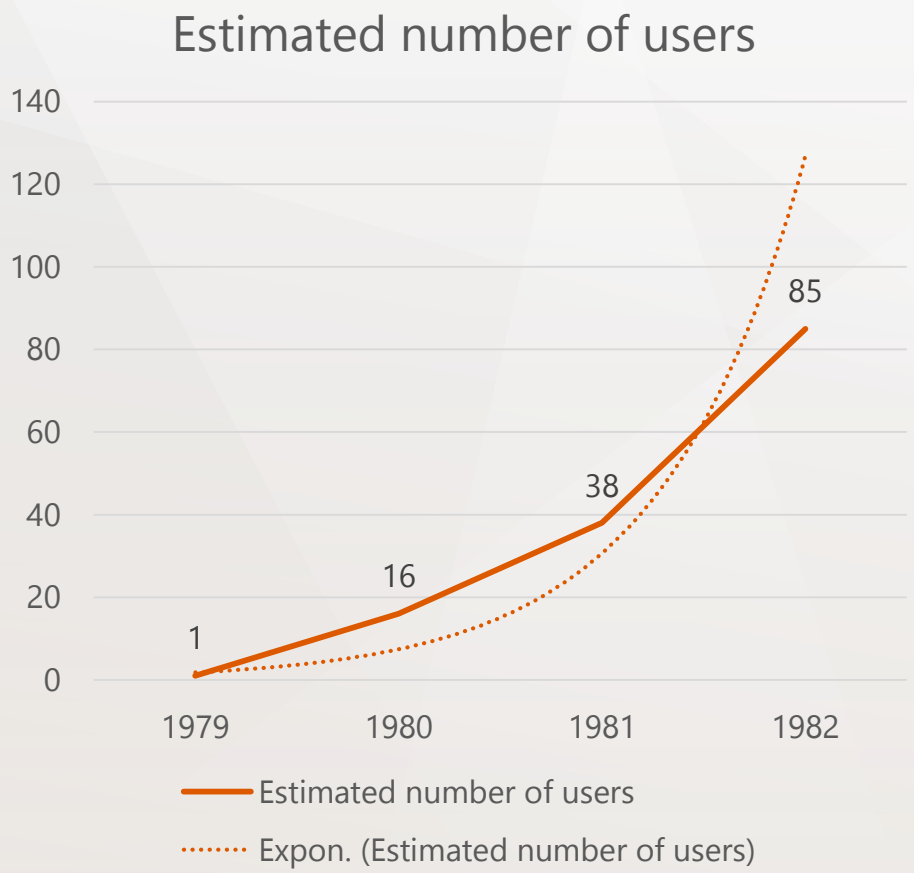
Cleaned up

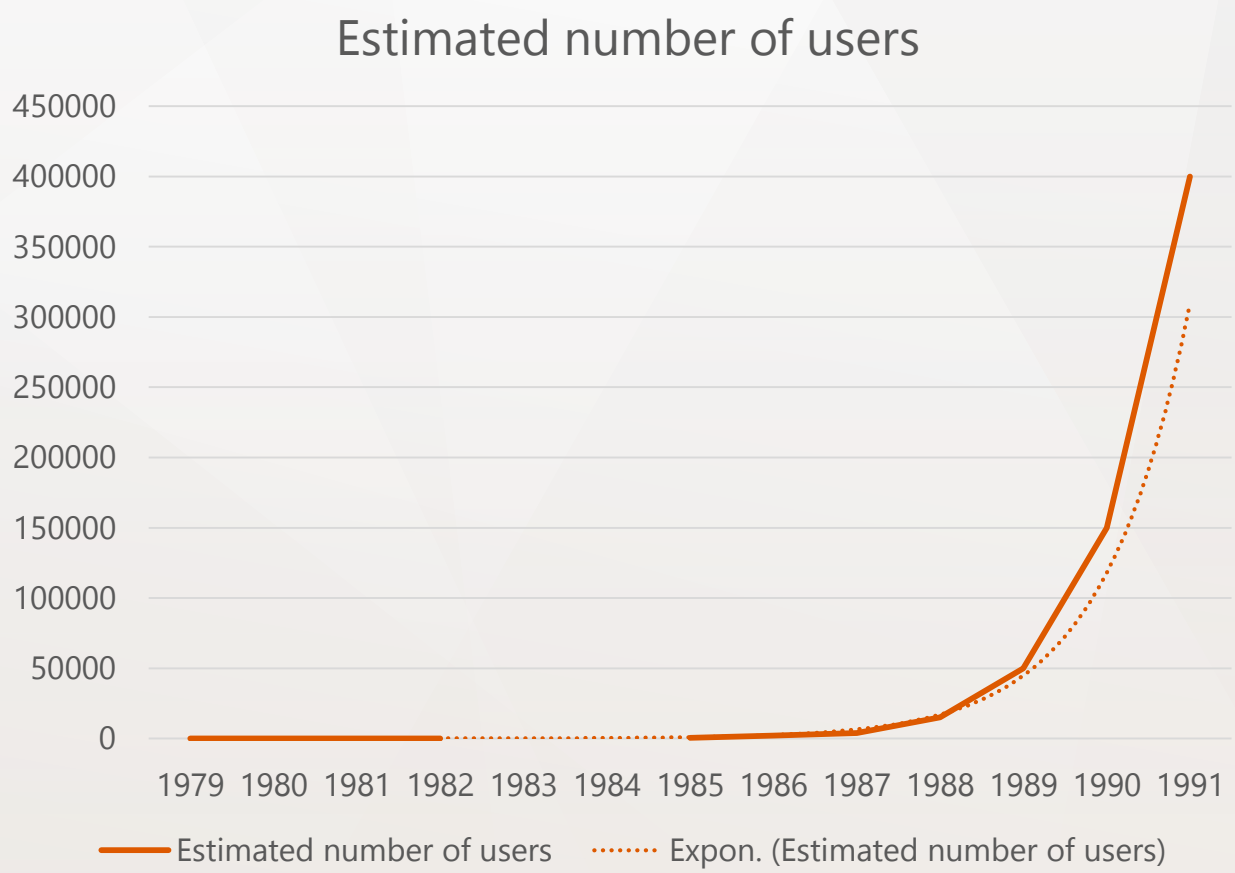
Modified to fit together better

Included

- Multiple inheritance
- Abstract classes
- …

Increased usage





Standardization

Annotated C++ Reference Manual (ARM) (1990)
by Bjarne Stroustrup and Margaret Ellis

ANSI C++ committee (1989)
ISO C++ committee (1991)

New features included templates (parameterized types) and exception handling

Expected standard in 1995-1996
Actually 1998 (C++98)
Then C++03, C++07, C++11, C++14, C++17

Multiple inheritance

```
class A {
    public: void f();
    /* ... */
};
class B {
    public: void f();
    /* ... */
};
class C : public A, public B {
    /* no f() */
    /* ... */
};

void g( ) {
        C* p;
        p->f(); // error: ambiguous
}
```

"Multiple inheritance is like a parachute, you don't need it very often, but when you do it is essential."

- Grady Booth

Abstract classes

C++

```
class set {
public :
        virtual void insert(T*);
        virtual void remove(T*) ;
        virtual int is_member(T*);
        virtual T* first();
        virtual T* next();
        virtual ~set()  { }
};
```

No way to explicitly say that this is only an interface, need not be defined, instantiating it is an error.
→ Pure virtual functions (C++ 2.0)

```
class set {
public :
        virtual void insert(T*) = 0;
        virtual void remove(T*) = 0;
        // …
};
```

Java

```
interface Set {
        public void insert(T);
        public void remove(T) ;
        public boolean isMember(T);
        public T first();
        public T next();
};
```

Templates

Note: C++ Templates generate classes at compile time, kind of like advanced macros vs Java Generics operate at run-time, and are basically wrappers for object casting

C++

```
template<class T>
class vector {
    T* v;
    int sz;
public:
    vector(int);
    T& operator[] (int);
    T& elem(int i) { return v[i]; )
    // ...
};

vector<int> vl(20);
vector<complex> v2(30);

// make cvec a synonym for vector<complex>
typedef vector<complex> cvec;

cvec v3(40);  // v2 and v3 are of the same type

vl[3] = 7;
v2[3] = v3.elem(4) = complex(7,8);
```

Java

```
public class Vector<T> {
    private T[] v;
    private int sz;

    public Vector(int sz) { /* … */ }
    // No operator overloading
    public void set(int i, T val) {v[i] = val;}
    public T elem(int i) { return v[i]; )
    // ...
};

Vector<Integer> vl = new Vector<Integer>(20);
Vector<Complex> v2 = new Vector<Complex>(30);

vl.set(3,7);
Complex c = new Complex(7,8);
v2.set(3,c);
v3.set(4,c);
```

Discussions?

String I/O Library

Operator overloading