Evolution of C++

Torsten Sehy
Seminar on Languages for Scientific Computing, 15.11.2012
Overview

- C with Classes, C++ and C++89
- C++98
  - Standard Template Library
- Boost Libraries
  - C++ TR1
- C++11
- Future
- Summary
C with Classes, C++ and C++ 2.0
Motivation for C++

- Object oriented languages existed
  - Simula
    - Usable in big software projects

- Efficient languages existed
  - Not usable for big projects

- C
  - Fast and portable
C with Classes (1979)

- Developed by Bjarne Stroustrup
- Extension of C
- Classes like in Simula-67

Additions:
- Stronger typesystem
- Derived classes
- Inline functions
- Default arguments
C++ (1983)

- Renaming from C with Classes into C++
  - Increment operator
- Extended with some features
  - Function- and operator-overloading
  - References
  - Comments like in BCPL: // at line end
  - ...

- „The C++ Programming Language“ (1985)
C++ 2.0 (1989)

- Again extensions to the language
  - Multiple inheritance
  - Abstract classes
  - Protected members
  - Static member functions
  - ...

C++98 (1998)

- Many additions from 1989 on
  - Templates
  - Exceptions
  - Namespaces
  - Boolean type
  - ...

- Standard Library
  - Based on C Standard Library
  - Streams (I/O)
  - Numeric
  - STL
Standard Template Library

- Origin in the 80s by Hewlett-Packard
- Meeting with C++ standard committee in 1993
  - Big parts included in the standard library
- HP made STL available in 1994 (via Internet)

- Focus on generic data structures and algorithms
  - Collections
  - Iterators
  - Algorithms
  - ...
ISO/IEC Standard

- Final version of C++98 released in 1998

- Compilers released years later
  - Implementation is very hard

- New ISO standard in 2003
  - Named C++03
  - Bug-fix for C++98
  - No new features
Boost Libraries

- Boost project founded in 2000
  - By members of the standard committee

- Over eighty different libraries
  - Linear algebra
  - Image processing
  - Networking
  -...

- Possible extensions for C++

- Extensions for the future standard
- Writing of the standard takes time
  - TRs are a way to extend the language faster
- "Only" library extension
  - Smart Pointers
  - Random number generators
  - Regex
  - ...

Boost (2000)

TR1 (2005)
C++11 (2011)

- Includes TR1 partially
- Extends the core language and libraries
  - Core language changes kept to a minimum
- Make C++ easier to teach and learn
- Performance increase
- Maintain compatibility
  - To C++98 and C

Core Language Extensions

- Performance improvements
  - E.g. Rvalue references

- Usability enhancements
  - Type inference
  - Range-based for-loop
  - Lambda functions

- Build time enhancements
- Functionality improvements
  - Multithreading memory model
Core Language Extensions

- Performance improvements
  - E.g. Rvalue references

- Usability enhancements
  - Type inference
  - Range-based for-loop
  - Lambda functions

- Build time enhancements

- Functionality improvements
  - Multithreading memory model

```
A a;
...
{
    std::vector<string> x;...
    a.y = x;
}

A a;
...
{
    std::vector<string> x;...
    a.y = std::move(x);
}
```
Core Language Extensions

- Performance improvements
  - E.g. Rvalue references

- Usability enhancements
  - Type inference
  - Range-based for-loop
  - Lambda functions

- Build time enhancements

- Functionality improvements
  - Multithreading memory model

```cpp
std::vector<int> a;
std::vector<int>::const_iterator first = a.begin();
```

```cpp
std::vector<int> a;
auto first = a.begin();
```

TR1 (2005)  
C++11 (2011)
Core Language Extensions

- Performance improvements
  - E.g. Rvalue references

- Usability enhancements
  - Type inference
  - Range-based for-loop
  - Lambda functions

- Build time enhancements

- Functionality improvements
  - Multithreading memory model

```cpp
std::vector<int> a = {1,2};
for(auto& it: a){
  std::cout << it;
}
```
Standard library changes

- Changes resulting from new core language

- Mostly changes from TR1
  - Not included were special Math functions and decimal types.

- Most important feature: Smart pointers
  - Automatic reference counting
  - No need for „new“ and „delete“
  - No memory leaks possible
Future
Technical Report 2

- Extensions for the future standards C++14/C++17
- Based partially on Boost
- C++14 minor release (bug fix)

- Library extensions
  - File system library
  - XML/HTML
  - Networking
  - Signals/Slots
  - Any library
  - ...

Boost

TR2 (20??)
Summary

- C++ started as an extension to C
- But became much more than C with Classes

- The first ISO standard was released in 1998

- Last year (2011) the third ISO standard was released
  - Almost a new programming language

- Future standards planned for 2014 and 2017
References

- History of C++ until 1994:
  - Stroustrup, Bjarne. The design and evolution of C++, 1994

- Later History:
  - http://www.cplusplus.com/info/history/

- General information about the programming language:
  - http://isocpp.org
  - Stroustrup, Bjarne. The C++ Programming Language, 2000

- C++11 Talks:
Cfront

- Converts C with Classes (C++) to C
- Written in C with Classes (C++)
- Problems bootstrapping Cfront
  - C++ compiler needed
  - Solution: Preprocessed code
- Used until 1993
Core Language Extensions
Lambda functions

```cpp
bool myComperator (int x, int y){
    return x > y && myClass::subFunctionResult == 7;
}

class myClass{
    static int subFunctionResult;

    void doStuff(){
        subFunctionResult = subFunction();
        std::vector<int> a = {3,1,5};
        std::sort(a.begin(), a.end(), myComperator);
    }
};
```

```cpp
class myClass{
    void doStuff(){
        int subFunctionResult = subFunction();
        std::vector<int> a = {3,1,5};
        std::sort(a.begin(), a.end(), [=](){
            return x > y && subFunctionResult == 7;
        });
    }
};
```
Standard library changes
Smart Pointers

class A;
A* aFactory(){
    A* result = new A();
    return result;
}

Class B{
    A* a;
    void doStuff(){
        a = aFactory();
        a->method1();
    }
    A* getA(){
        return a;
    }
    ~B(){
        delete a; // !!!
    }
};

class A;
std::shared_ptr<A> aFactory(){
    std::shared_ptr<A> result(new A());
    return result;
}

class B{
    std::shared_ptr<A> a;
    void doStuff(){
        a = aFactory();
        a->method1();
    }
    std::shared_ptr<A> getA(){
        return a;
    }
};
Riemann Zeta Function
Beta Function
Incomplete Elliptic Integral of the First / Second/Third Kind
Exponential Integral
Hermite Polynomials
...