Places, People, and Code

Bjarne Stroustrup
Texas A&M University
April 28, 2012
Places

Texas A&M University
College Station

Bell Labs
Murray Hill, NJ

Cambridge

Aarhus
Aarhus
Aarhus University
Microprogramming

• Instruction set design
• Machine architecture
• Machine Oriented Languages
• “High-level” instruction sets and interfaces
  – Inspirational talk by Bob Barton
• The OS/hardware interface
• Concurrent execution
Simula

- Object-oriented programming
- Object-oriented design
- Concurrent programming
Cambridge
Cambridge

“Keep a high external profile”
Cambridge

• *Systems should be build to do useful things for real people* – Roger Needham

• Martin Richards
Cambridge
AT&T Bell Labs, Murray Hill, NJ
127 aka CSRC
C with Classes

• C + Simula

Code Organization

Systems programming and performance

• A strong and flexible static type system was/is the ideal/aim
• “Elegant and efficient code”
Constructors and destructors

• A constructor established the run-time environment for member functions
  – incl. acquiring needed resources

• The destructor reverses the actions of the constructor
  – incl. releasing owned resources

“The idea was to allow the programmer to establish guarantees, sometimes called “invariants,” that other member function could rely on.”
Constructors and destructors

- **Constructor**: make an object from memory
  - An object holds a value
  - Has a type
  - Has an interface
  - Has meaning
- **Destructor**: make an object (back) into memory
  - Memory is just interpreted bits
A resource handle

- **Examples**
  - Containers
    - vector, list, map, …
  - Smart pointers
    - unique_ptr, shared_ptr, delayed_value, remote_object, …
  - Locks, thread handles, sockets, iostreams, file handles
  - …
Vector: the archetypical resource handle

- Slight simplification of `std::vector`

```cpp
template<typename T> // T is the element type
class Vector {
public:
  Vector(); // default constructor: make empty vector
  Vector(int n); // constructor: initialize to n elements
  Vector(initializer_list<T>); // constructor: initialize with element list
  ~Vector(); // destructor: deallocate elements
  int size(); // number of elements
  T& operator[](int i); // access the ith element
  void push_back(const T& x); // add x as a new element at the end
  T* begin(); // fist element
  T* end(); // one-beyond-last element
private:
  int sz; // number of elements
  T* elem; // pointer to sz elements of type T
};
```
Some liked C++, some didn’t
TC++PL
TC++PL2
D&E
I have always wished for my computer to be as easy to use as my telephone; my wish has come true because I can no longer figure out how to use my telephone.
J16/WG21 Meetings 1979-2012-

- Austin
- Batavia
- Bellevue
- Berlin
- Bloomington
- Boston
- Curacao
- Copenhagen
- Dublin
- Frankfurt
- Kona
- Lillehammer
- London
- Lund
- Madrid
- Morristown
- Monterey
- Mt. Tremblant
- Nashua
- Oxford
- Portland
- Pittsburg
- Portland
- Raperswil
- Redmond
- Santa Cruz
- San Diego
- San Francisco
- Sidney
- Seattle
- Somerset
- Sophia Antipolis
- Stockholm
- Summit
- Tokyo
- Toronto
- Valley Forge
- Washington, D.C.
C++ standardization – why bother?

- The ISO standards process is central
  - C++ has no rich owner
    • who can dictate changes, pay for design, implementation, marketing, etc.
  - The C++ standards committee is the central forum of the C++ community
    • Endless discussions among people who would never meet otherwise
  - The committee receives massive feedback from a broad section of the community
    • Much of it industrial
  - The committee is somewhat proactive
    • Adds features not previously available in the C++ world
  - Standard support needed for mainstream use
    • Huge potential for improvement of application code
    • For (far too) many “if it isn’t in the standard it doesn’t exist”
  - Significant defense against vendor lock-in
Interlocking themes

• Stability and Compatibility
  – “make the language much better but don’t break my code”
• Scale
  – Million-line projects became common
  – Specification – precise and complete
  – Portability
• Resource management
  – Invariants
• Type safety
  – Containers
• Performance
  – Compactness
• Equal support for user-defined and built-in types
  – Value types, scoped objects
• User skills required
C++98 example: Resource management

- Standard library containers
  - with exception-safety guarantees (e.g., vector)
  - the techniques can be used by every user (e.g., File_handle)
- No resources are leaked
  - Based on a simple and systematic view of resource management
    - Exception safety guarantees
    - RAII
  - Destructors do cleanup
    - guaranteed, implicitly

```cpp
void f(string s)
{
    vector<int> v;
    File_handle h(s,"r");
    // …
    int x;
    while (cin>>x) {
        if (x<=0) throw Bad_value(x);
        v.push(x);
    }
    // …
}
```
The STL

- Ideal: The most general and most efficient expression of an algorithm
  - Focus on algorithms
  - Separate algorithms from data
  - Go from the concrete to the abstract
    - Not the other way
  - Use compile-time resolution to eliminate overheads
    - Inlining and overloading
  - Where needed, parameterize with policies
    - E.g. sorting criteria
STL example: `find_if()`

- **Definition**

  ```cpp
  template<class Iter, class Pred>
  Iter find_if(Iter first, Iter last, Pred p)
  {
      while (first!=last && !p(*first)) // while not at end and predicate not met
          ++first; // advance to next element
      return first; // return the element reached
  }
  ```
Design

• By committee?
C/C++ compatibility

• A constant sore point
  – Separate standards committees
    • A tragedy
  – Constant borrowing
    • Both ways
    • Often incompatibly
  – Widely demanded by users
    • Rightfully so
  – Widely despised by users
    • “Against OO”
    • “Against the spirit of C”
C++11

• And now – just a year later – large parts of C++ is shipping!
  – GCC4.7
  – Microsoft 11 beta
  – …
  – “Everybody” ships the libraries
- John Lakos: Bloomberg
- Stepanus Du Toit: Intel, Canada
- Loic Joly: INRIA, France
• Herb Sutter: Microsoft
• Steve Clamage: Oracle (Sun)
• Bill Plauger: Dinkumware
• Howard Hinnant: Apple
• Howard Hinnant: Apple
• Richard Corden: Programming Research
• Clark Nelson: Intel
• Steve Clamage: Oracle (Sun)
• Jean-Paul Rigoux: U. de Provence, France
• J.C. Van Vinkel: Netherlands and Google
• Beeman Dawes: Boost
• Detleff Vollman: Switzerland
• Loic Joly: INRIA, France
• Mike Miller: EDG
• Doug Gregor: Apple
• Roger Orr: UK
• Alisdair Meredith: Bloomberg
• Martin Sebor: Rogue Wave
• Pete Becker: Roundhouse Consulting
• Jaakko Jarvi: TAMU
• Daniel Garcia: U. Charles III
• Hans Boehm: HP
• Beeman Dawes: Boost
• Mike Wong: IBM
• Lawrence Crowl: Google
• Herb Sutter: Microsoft
• Daniel Krugler: ???
• John Lakos: Bloomberg
Vector<int*> find_all(Vector<int>& v, int val) // find all occurrences of val in v
{
    Vector<int*> res;
    for (int& x : v)
        if (x==val)
            res.push_back(&x); // add the address of the element to res
    return res;
}

void test()
{
    Vector<int> lst {1,2,3,1,2,3,4,1,2,3,4,5};
    for (int* p : find_all(lst,3))
        cout << "address: " << p << "", value: " << *p << "\n";
        // ...
}
Copy and move

- **Move constructor**

  ```cpp
template<typename T>
Vector<T>::Vector(const Vector&& v)   // move constructor
    : sz{v.sz}, elem{v.elem}   // grab v’s elements
{
    v.elem = nullptr;  // make v empty
    v.sz = 0;
}
```

*this:  

v:
Copy and move

- Move assignment

```cpp
template<typename T>
Vector<T>& Vector<T>::operator=(Vector<T>&& v)    // move assignment
{
    destroy<T>(elem,sz); // delete old elements
    elem = v.elem;       // grab v's elements
    sz = v.sz;
    v.elem = nullptr;    // make v empty
    v.sz = 0;
    return *this;
}
```

*bthis:* v:
Vector

• Copy and move declarations added to Vector

```cpp
template<typename T> // T is the element type
class Vector {
public:
    // ...
    Vector(const Vector&); // copy constructor
    Vector(Vector&&); // move constructor
    Vector& operator=(const Vector&); // copy assignment
    Vector& operator=(Vector&&); // move assignment
    // ...
};
```
The real problems

• Help people to write better programs
  – Easier to write
  – Easier to maintain
  – Easier to achieve acceptable resource usage

"...And that, in simple terms, is what’s wrong with your software design."
College Station
Texas A&M
Texas A&M
C++ is shockingly misused

• “If that’s C++, then I don’t like it either!”
  – Bjarne Stroustrup, 2004
  • (after surveying a couple of dozen C++ textbooks)
PPP
Programming languages

• A programming language exists to help people express ideas

• Programming language features exist to serve design and programming techniques

• The primary value of a programming language is in the applications written in it

The quest for better languages has been long and must continue
Programming Languages

Domain-specific abstraction:
- Fortran
- Cobol

General-purpose abstraction:
- Simula
- Java
- C++
- C++11
- C#

Direct mapping to hardware:
- Assembler
- BCPL
C++

Key areas of strength:
  • Software infrastructure
  • Resource-constrained applications

A light-weight abstraction programming language
C++ applications
C++ Applications

- www.research.att.com/~bs/applications.html
C++ Applications

www.lextrait.com/vincent/implementations.html
Thanks!