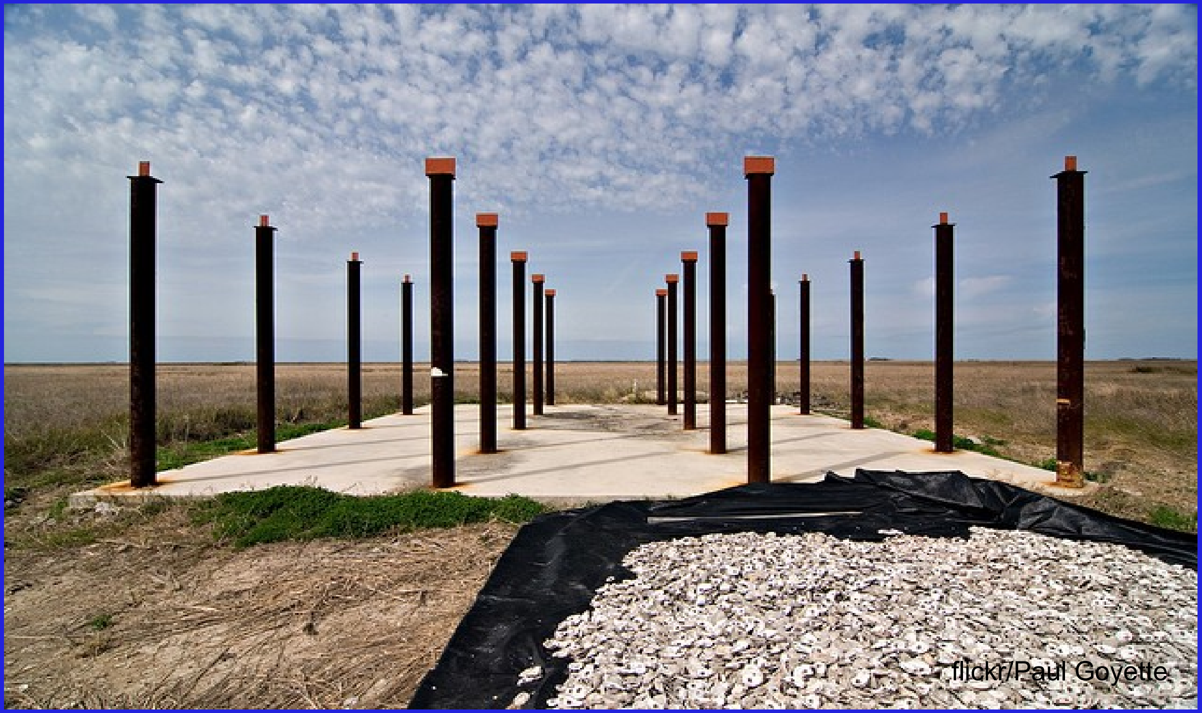
C++ Foundation



Standard Libraries

Standard Libraries

- containers and iterators
- algorithms
- string
- iostream, stringstream
- pair
- functional
- the C library
- C++ in the future: boost, tr1, C++0x

Containers

- Sequential: vector, list, deque, queue, stack
- Associative: map, multimap, set, multiset

```
template<typename Type>
class list
{
public:
    bool empty() const;
    size_t size() const;

    void push_front(const Type &);
    void push_back(const Type &);
    void clear();
    ...
};
```

Iterators

- Modelled on pointers

```
template<typename Type>
class list<Type>
{
public:
    class iterator
    {
    public:
        Type & operator*() const;
        Type * operator->() const;
        iterator operator++();
        ...
    };
    bool operator==(iterator, iterator);
    bool operator!=(iterator, iterator);
};
```

Iterators

- A pair of iterators [begin, end) specifies a range

```
template<typename Type>
class list
{
public:
    ...
    template<typename It>
    iterator(It begin, It end);

    iterator begin();
    iterator end()
    ...
    void insert(iterator, const Type &);
    void erase(iterator);
    ...
};
```

Lots of <algorithm>s

sequence: non-modifying

adjacent_find, count, count_if, equal,
for_each, *find*, *find_if*, find_end, find_first_of,
mismatch, search, seach_n

sequence: modifying

copy, copy_backward, generate, generate_n, fill,
fill_n, iter_swap, partition, replace, replace_if,
replace_copy, replace_copy_if, *remove*, remove_if,
remove_copy, remove_copy_if, reverse, reverse_copy,
rotate, rotate_copy, random_shuffle,
stable_partition, swap, swap_ranges, *transform*,
unique, unique_copy

sorting

nth_element, partial_sort, partial_sort_copy,
sort, stable_sort

Lots of <algorithm>s

binary search

```
binary_search, equal_range,
lower_bound, upper_bound
```

merge

```
inplace_merge, includes, merge, set_union,
set_intersection, set_difference,
set_symmetric_difference
```

heaps

```
make_heap, push_heap, pop_heap, sort_heap
```

min-max

```
lexicographic_compare,
min, max, min_element, max_element,
next_permutation, prev_permutation
```

Algorithms

- Function templates - iterator pairs

```
template<typename Iter, typename Type>
Iter find(Iter at, Iter end, const Type & value)
{
    for (; at != end; ++at)
        if (*at == value)
            break;
    return at;
}
```

```
template<typename Iter, typename Pred>
Iter find_if(Iter at, Iter end, Pred pred)
{
    for (; at != end; ++at)
        if (pred(*at))
            break;
    return at;
}
```

Algorithms

- Function templates - iterator pairs

```
template<typename InputIter,
         typename OutputIter,
         typename UnaryOp>
OutputIter transform(InputIter at, InputIter end,
                     OutputIter result,
                     UnaryFunc f)
{
    while(at != end)
    {
        *result = f(*at);
        ++result;
        ++at;
    }
    return result;
}
```

Writing loops?

- Many loops can be refactored to an algorithm

```
typedef std::list<int>::iterator iterator;
for (iterator at = values.begin();
     at != values.end();
     ++at)
{
    std::cout << *at << ',';
}
```

```
void couter(int value)
{
    std::cout << value << '.';
}
std::for_each(values.begin(), values.end(),
              couter);
```

string

- Goodbye char * horribleness

```
class string
{
public:
    string();
    string(const char *);

    size_t size() const;
    bool empty() const;
    void clear();
    char & operator[](size_t);
    const char & operator[](size_t) const;
    ...
};
```

simplified

string

- Retrofitted to STL container model

```
class string
{
public:
    class iterator;
    class const_iterator;

    template<typename It>
    string(It, It);

    iterator begin();
    iterator end();
    const_iterator begin() const;
    const_iterator end() const;
    ...
};
```

simplified

Streaming << or >>

- Write the stream object first, then the operator
  - >> to indicate data flowing out of the stream
  - << to indicate data flowing into the stream

```
void in(istream & is)
{
    int value;
    is >> value;
    ...
}
```

```
void out(ostream & os)
{
    int value = 42;
    os << value;
    ...
}
```

Streaming

- Providing operator<< allows you to write to files

```
ostream & operator<<(ostream &, const date &);
```

```
#include <fstream>

void eg()
{
   date xmas(2011,12,25);
   std::ofstream ofs(“date.txt”);
   ofs << xmas;
}
```

date.txt →

```
2011/12/25
```

Streaming

- Providing operator<< also allows you to write to strings - very handy for test diagnostics

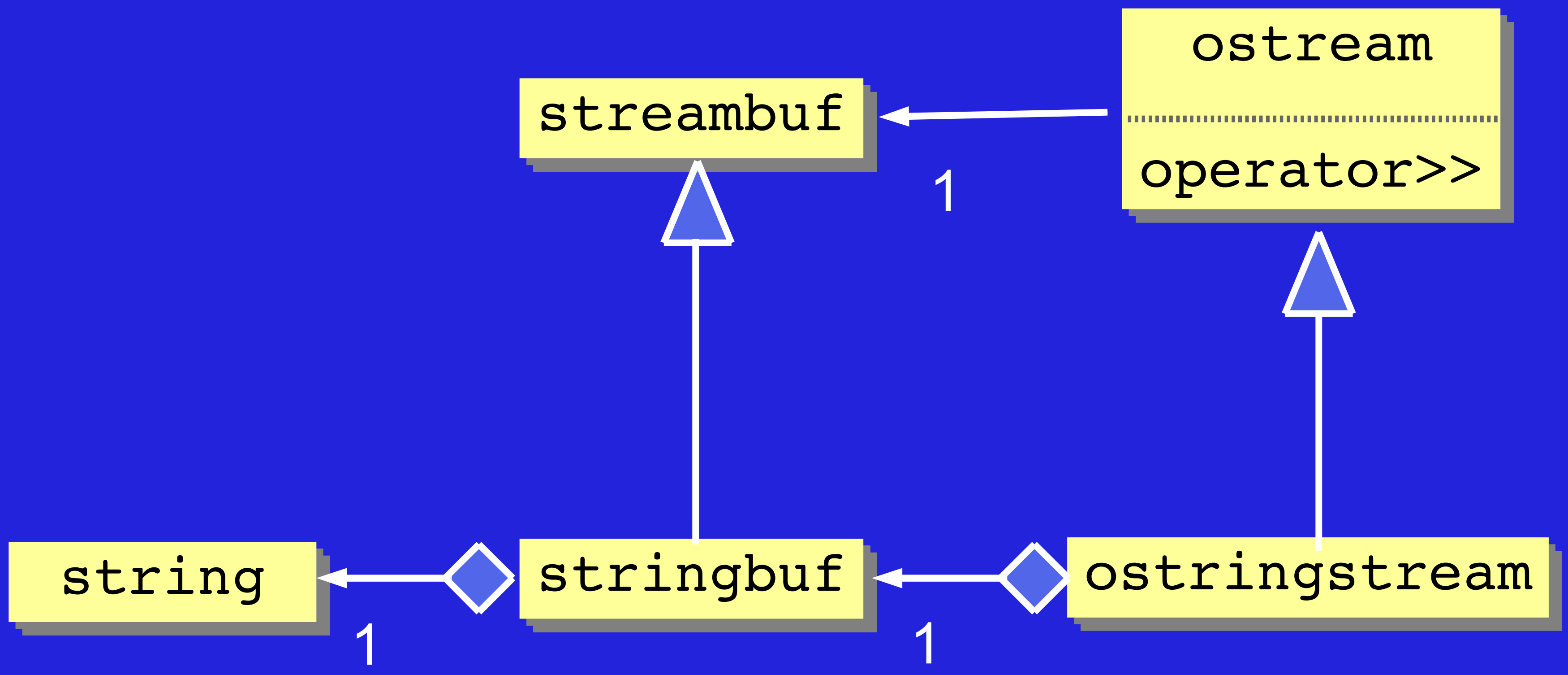
```
ostream & operator<<(ostream &, const date &);
```

```
#include <sstream>

void eg()
{
   date xmas(2011,12,25);
   std::ostringstream oss;
   oss << xmas;
   assert(oss.str() == “2011/12/25”);
}
```

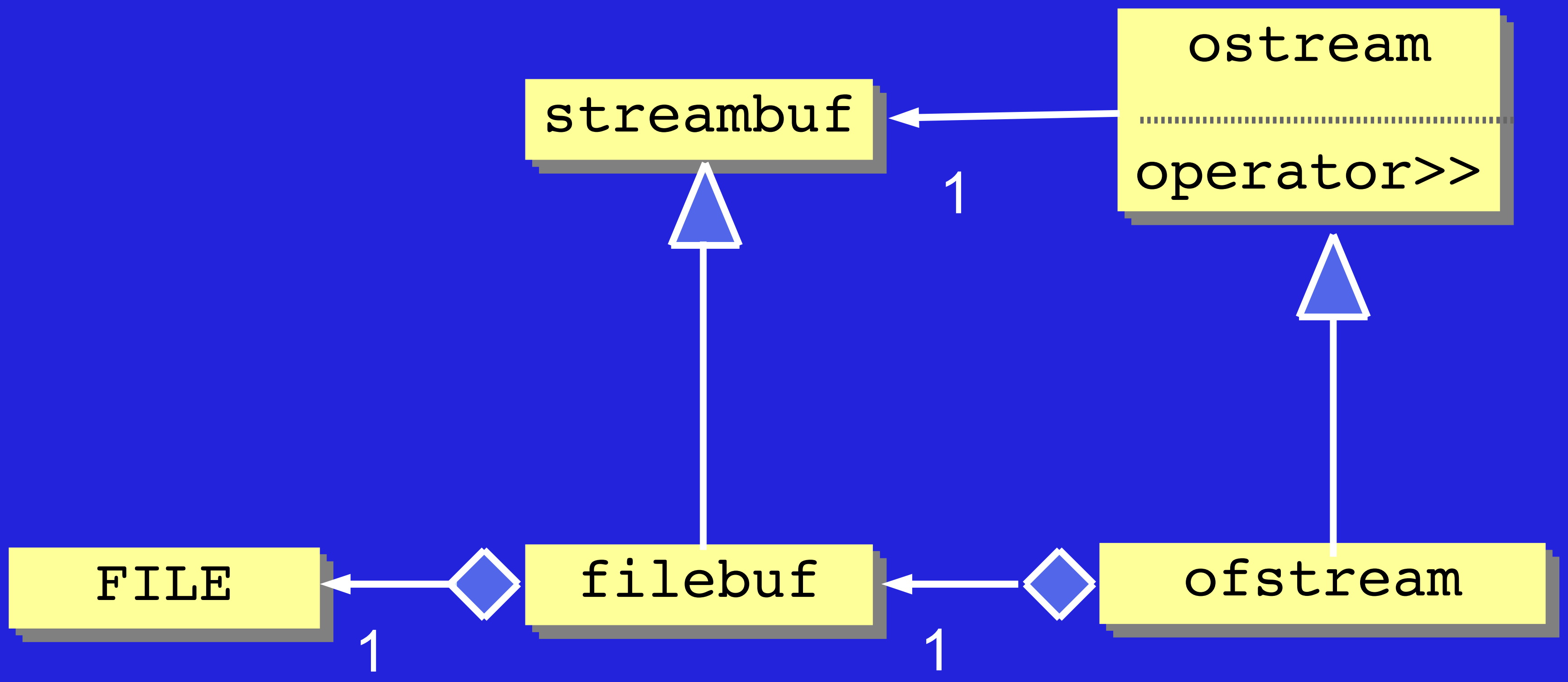
streambuf

- Buffers characters manipulated by a stream
- Subclassed in parallel with the stream



streambuf

- Buffers characters manipulated by a stream
- Subclassed in parallel with the stream



pair<T1,T2>

- A simple two-tuple in <utility>

```
template<typename T1, typename T2>
struct pair
{
    typedef T1 first_type;
    typedef T2 second_type;

    T1 first;
    T2 second;

    pair()
        : first(T1()), second(T2()) {}
    pair(const T1 & f, const T2 & s)
        : first(x), second(s) {}
    template<typename U, typename V>
    pair(const pair<U,V> & p)
        : first(p.first), second(p,second) {}
};
```

Often usable instead of a small struct

make_pair

- A simple helper function template

```
template<typename T1, typename T2>
pair<T1,T2> make_pair(T1 f, T2 s)
{
    return pair<T1,T2>(f, s);
}
```

```
std::pair(42, answer);
```
✗

```
std::pair<int,std::string>(42, answer);
```
✓

```
std::make_pair(42, answer);
```
✓✓

<functional>

- Provides a framework and classes usable as predicates for algorithms and containers

```
void eg()
{
    int values[] = { 2,5,8,3,7 };

    std::sort(values, values + 5);
    // [2,3,5,7,8]

    std::sort(values, values + 5,
        std::greater<int>());
    // [8,7,5,3,2]
}
```

<functional>

- Provides a framework and classes usable as predicates for algorithms and containers

```
template<typename T> struct equal_to;      // ==
template<typename T> struct not_equal_to;  // !=
template<typename T> struct less;          // <
template<typename T> struct less_equal;    // >=
template<typename T> struct greater;       // >
template<typename T> struct greater_equal; // >=
```

```
template<typename T>
struct greater : ...
{
    bool operator()(const T & x, const T & y) const
    {
        return x > y;
    }
};
```

The C Library

- Most C <*header*.h>'s have a corresponding std namespace wrapping C++ <c*header*>

```
#include <string.h>

struct c_str_less
{
    bool operator()(const char * lhs, const char * rhs) const
    {
        return strcmp(lhs, rhs) < 0;
    }
};
```

```
#include <cstring>

struct c_str_less
{
    bool operator()(const char * lhs, const char * rhs) const
    {
        return std::strcmp(lhs, rhs) < 0;
    }
};
```

✓

http://www.boost.org

- Where future C++ libraries are born and grow
  - Aims to establish reference implementations of existing practice
  - High quality
  - Peer reviewed
  - Proving ground for TR1 and TR2
  - Any, Threading, Date and Time, Lambda, FileSystem, Parsing, Serialization, Tokenization, Graphs, Hashing

Technical Report 1 (tr1)

- Library components slated for C++0x (sic)
  - <memory> shared_ptr<T>, weak_ptr<T>
  - <functional> function<T> - polymorphic function call
  - <type_traits> meta-programming utilities
  - <random> number generators
  - <tuple>
  - <array> fixed size array
  - <unordered_set> - hash based set
  - <unordered_map> - hash based map
  - <regex>

www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1745.pdf