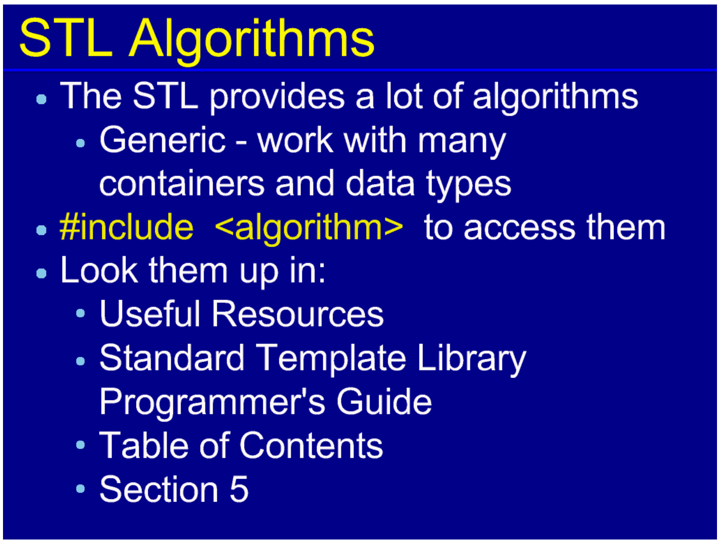
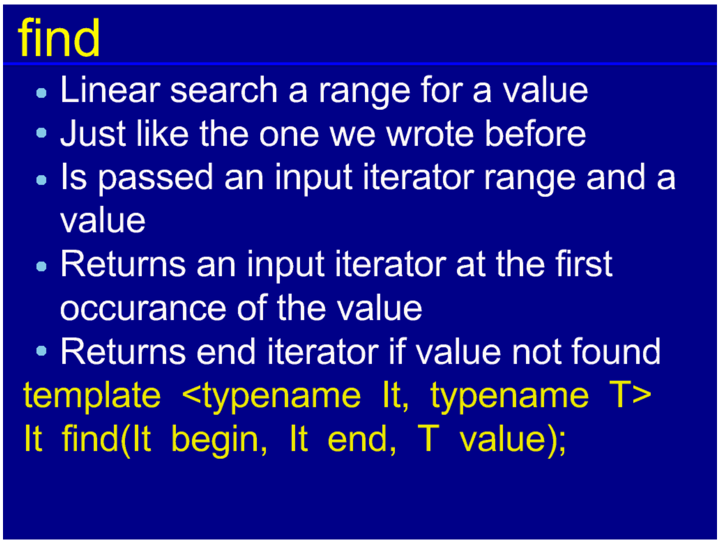
COMP2004 Programming Practice 2002 Summer School

Kevin Pulo
School of Information Technologies
University of Sydney



STL Algorithms

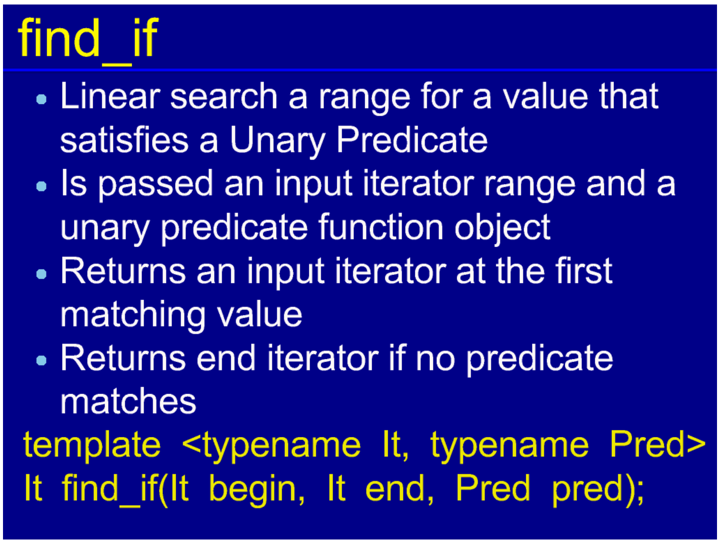
- The STL provides a lot of algorithms
  - Generic - work with many containers and data types
- #include <algorithm> to access them
- Look them up in:
  - Useful Resources
  - Standard Template Library Programmer's Guide
  - Table of Contents
  - Section 5



find

- Linear search a range for a value
- Just like the one we wrote before
- Is passed an input iterator range and a value
- Returns an input iterator at the first occurance of the value
- Returns end iterator if value not found

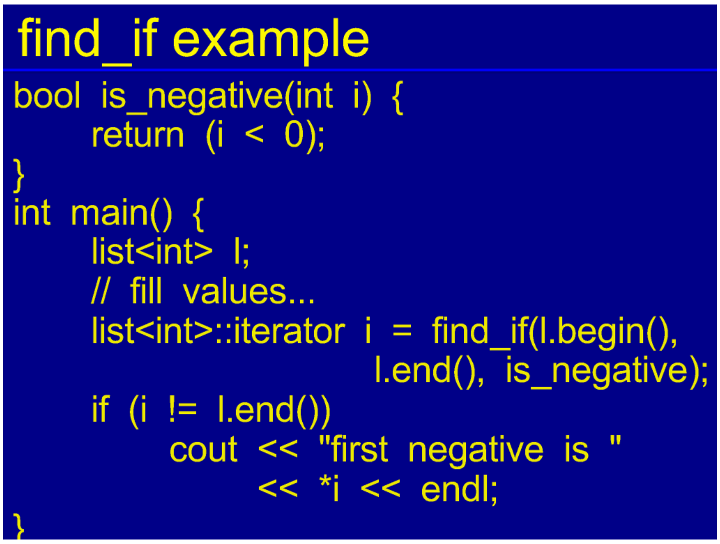
```
template <typename It, typename T>
It find(It begin, It end, T value);
```



find_if

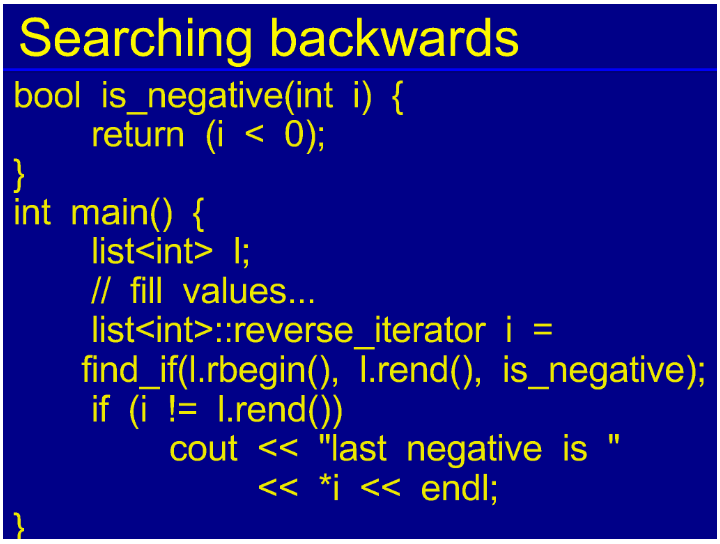
- Linear search a range for a value that satisfies a Unary Predicate
- Is passed an input iterator range and a unary predicate function object
- Returns an input iterator at the first matching value
- Returns end iterator if no predicate matches

```
template <typename It, typename Pred>
It find_if(It begin, It end, Pred pred);
```



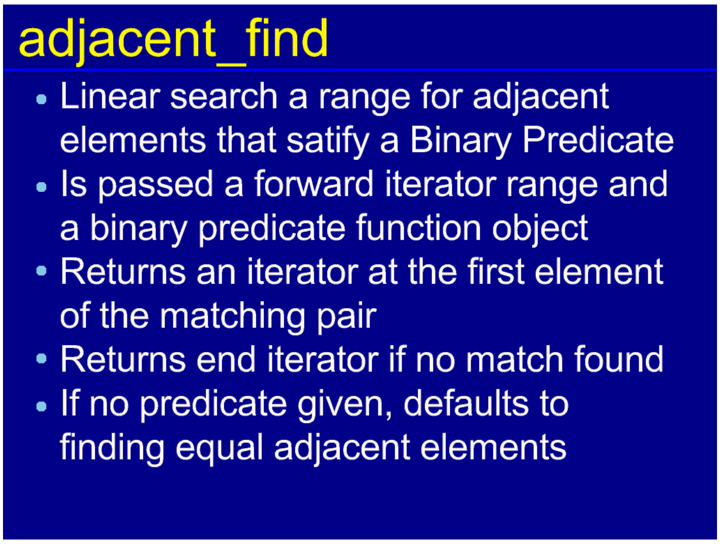
find_if example

```
bool is_negative(int i) {
    return (i < 0);
}
int main() {
    list<int> l;
    // fill values...
    list<int>::iterator i = find_if(l.begin(),
                            l.end(), is_negative);
    if (i != l.end())
        cout << "first negative is "
             << *i << endl;
}
```



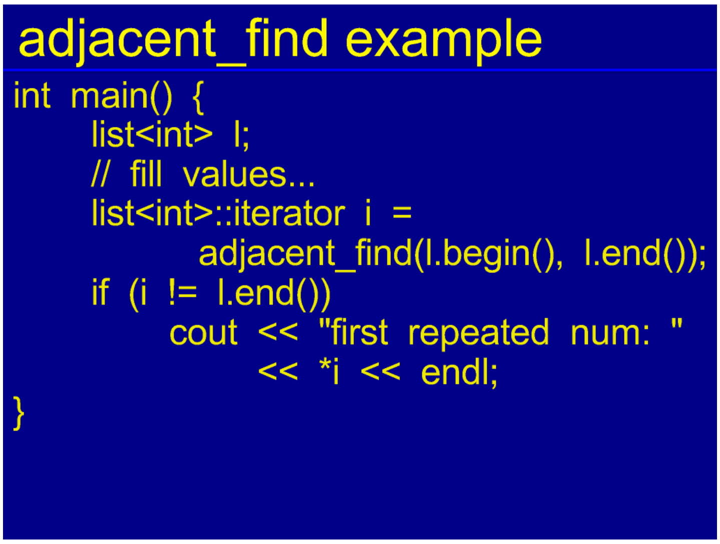
Searching backwards

```
bool is_negative(int i) {
    return (i < 0);
}
int main() {
    list<int> l;
    // fill values...
    list<int>::reverse_iterator i =
   find_if(l.rbegin(), l.rend(), is_negative);
    if (i != l.rend())
        cout << "last negative is "
             << *i << endl;
}
```



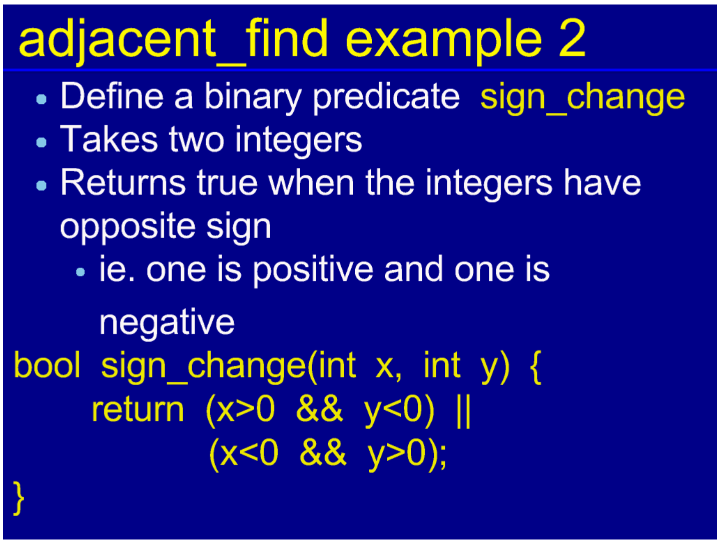
adjacent_find

- Linear search a range for adjacent elements that satify a Binary Predicate
- Is passed a forward iterator range and a binary predicate function object
- Returns an iterator at the first element of the matching pair
- Returns end iterator if no match found
- If no predicate given, defaults to finding equal adjacent elements



adjacent_find example

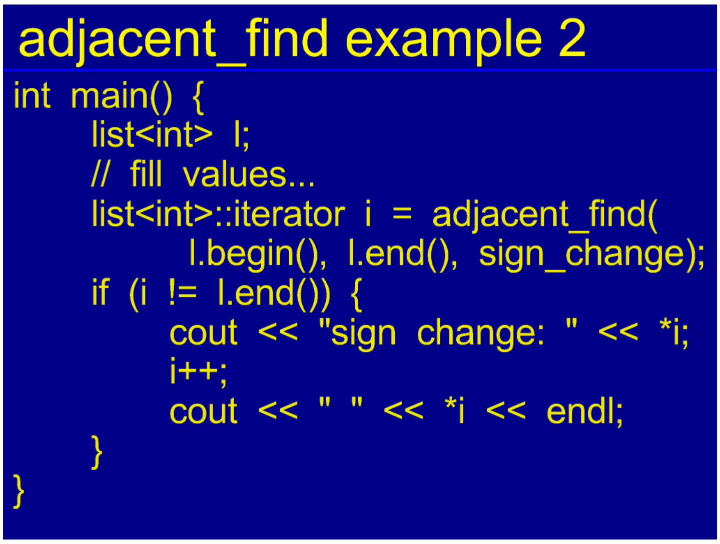
```
int main() {
    list<int> l;
    // fill values...
    list<int>::iterator i =
             adjacent_find(l.begin(), l.end());
    if (i != l.end())
        cout << "first repeated num: "
             << *i << endl;
}
```



adjacent_find example 2

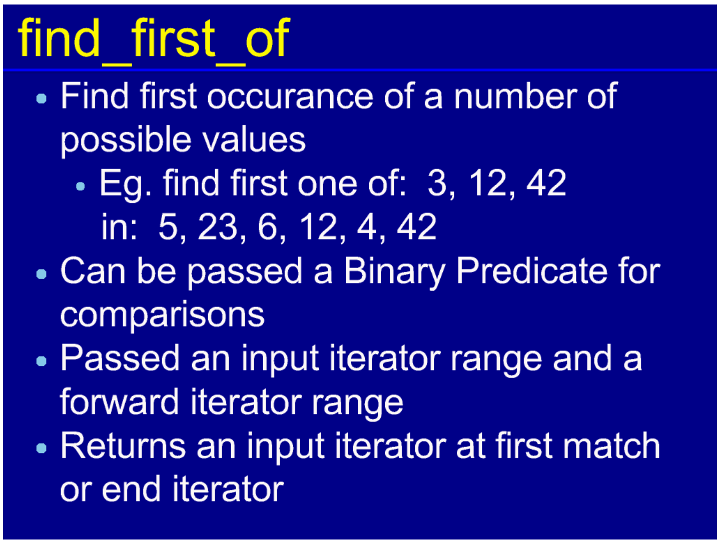
- Define a binary predicate sign_change
- Takes two integers
- Returns true when the integers have opposite sign
  - ie. one is positive and one is negative

```
bool sign_change(int x, int y) {
    return (x>0 && y<0) ||
           (x<0 && y>0);
}
```



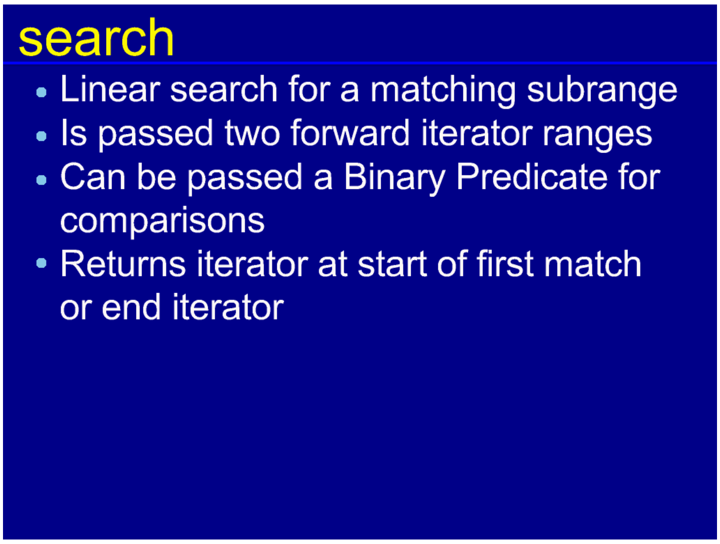
adjacent_find example 2

```
int main() {
    list<int> l;
    // fill values...
    list<int>::iterator i = adjacent_find(
            l.begin(), l.end(), sign_change);
    if (i != l.end()) {
        cout << "sign change: " << *i;
        i++;
        cout << " " << *i << endl;
    }
}
```



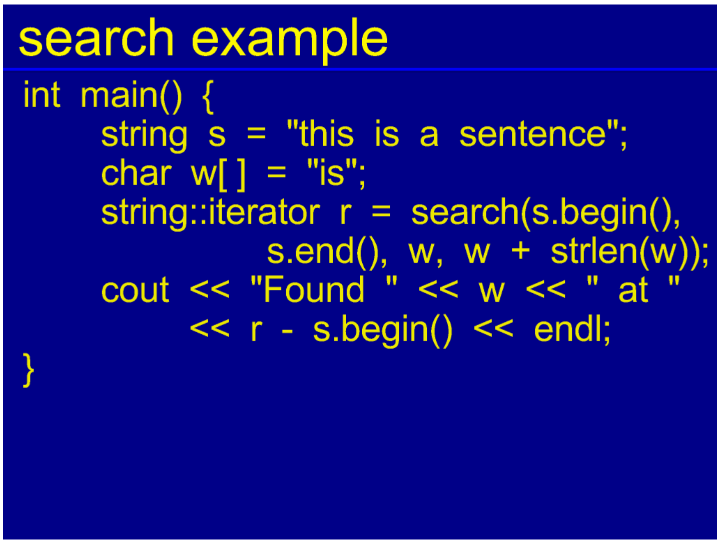
find_first_of

- Find first occurance of a number of possible values
  - Eg. find first one of: 3, 12, 42
    in: 5, 23, 6, 12, 4, 42
- Can be passed a Binary Predicate for comparisons
- Passed an input iterator range and a forward iterator range
- Returns an input iterator at first match or end iterator



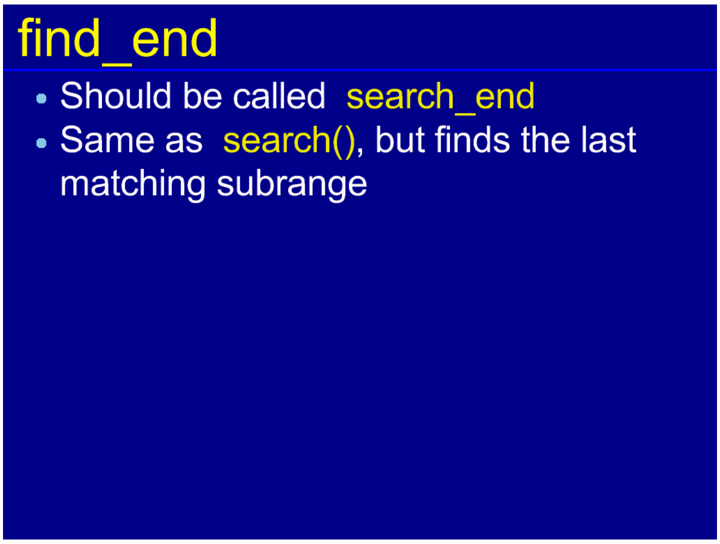
search

- Linear search for a matching subrange
- Is passed two forward iterator ranges
- Can be passed a Binary Predicate for comparisons
- Returns iterator at start of first match or end iterator



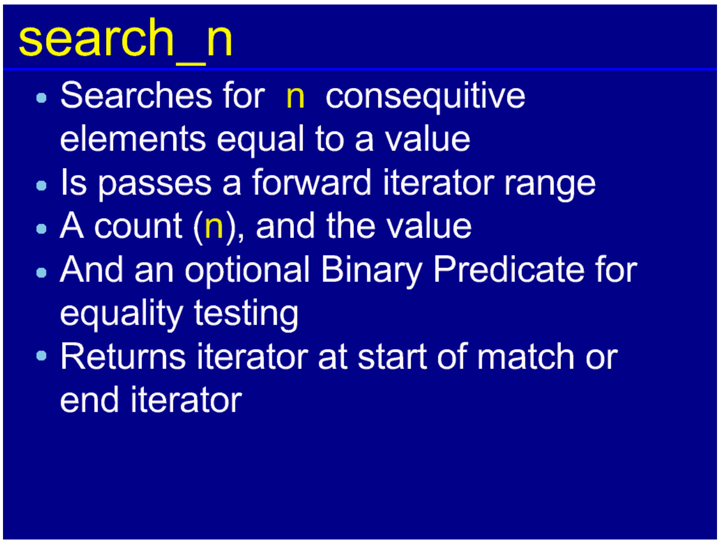
search example

```
int main() {
      string s = "this is a sentence";
      char w[ ] = "is";
      string::iterator r = search(s.begin(),
                  s.end(), w, w + strlen(w));
      cout << "Found " << w << " at "
            << r - s.begin() << endl;
}
```



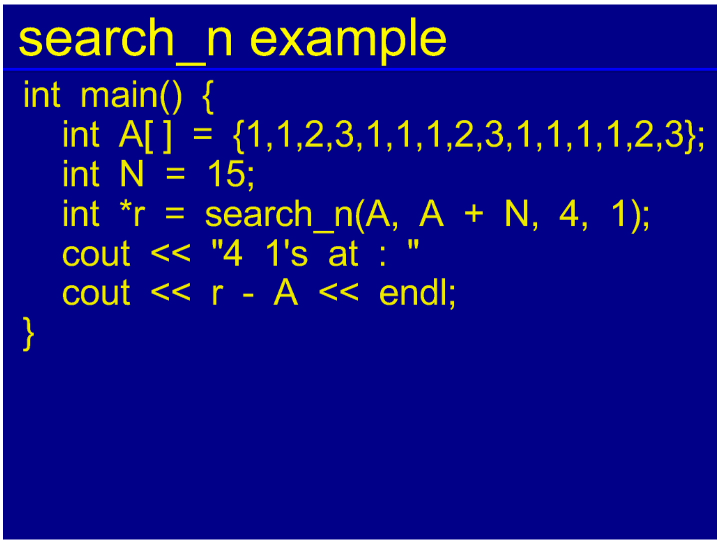
find_end

- Should be called search_end
- Same as search(), but finds the last matching subrange



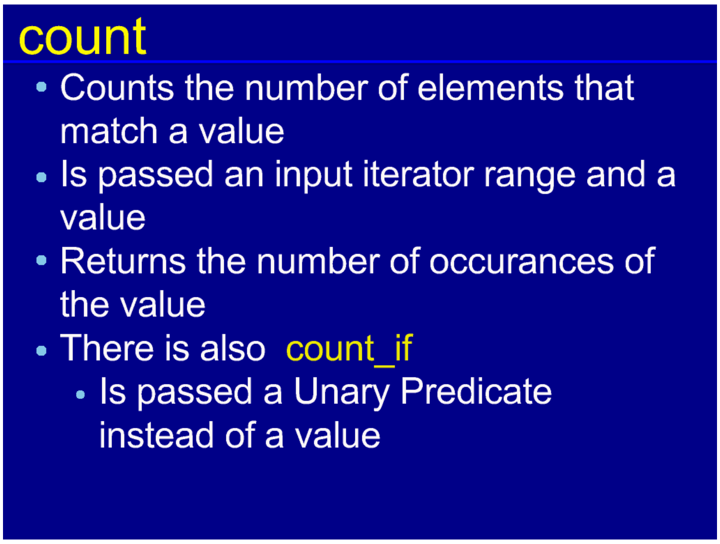
search_n

- Searches for n consequitive elements equal to a value
- Is passes a forward iterator range
- A count (n), and the value
- And an optional Binary Predicate for equality testing
- Returns iterator at start of match or end iterator



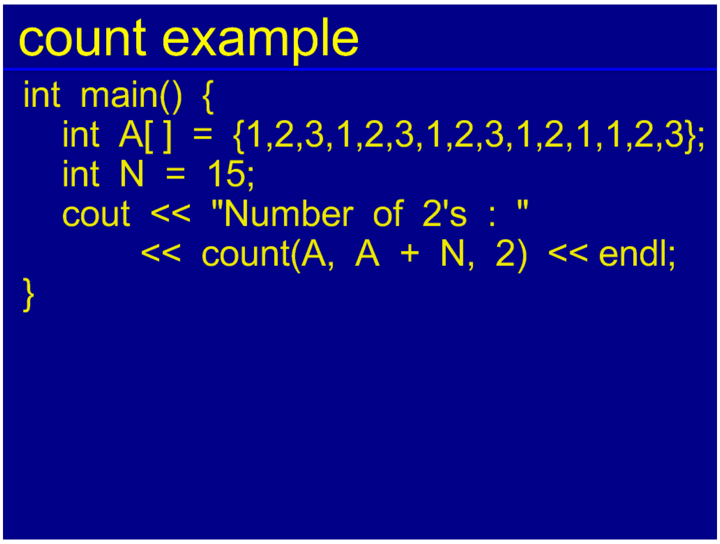
search_n example

```
int main() {
   int A[ ] = {1,1,2,3,1,1,1,2,3,1,1,1,1,2,3};
   int N = 15;
   int *r = search_n(A, A + N, 4, 1);
   cout << "4 1's at : "
   cout << r - A << endl;
}
```



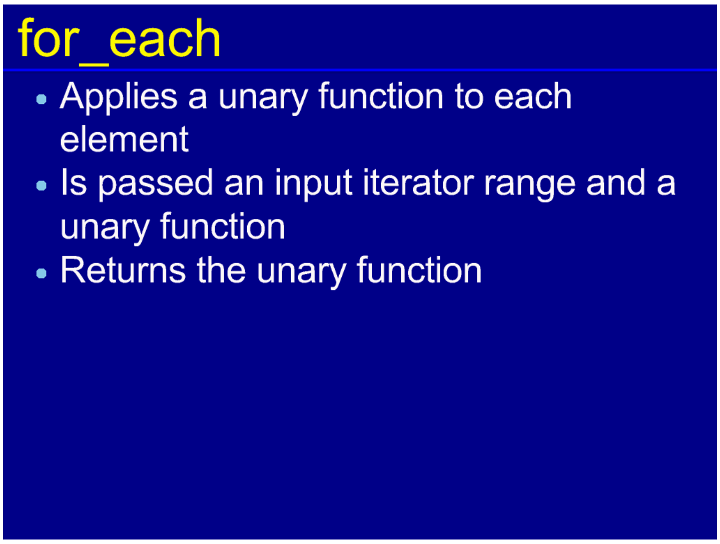
count

- Counts the number of elements that match a value
- Is passed an input iterator range and a value
- Returns the number of occurances of the value
- There is also count_if
  - Is passed a Unary Predicate instead of a value



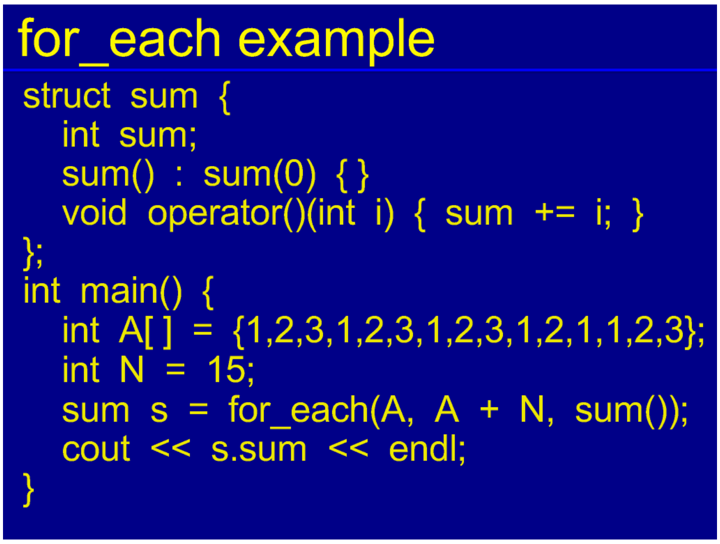
count example

```
int main() {
   int A[ ] = {1,2,3,1,2,3,1,2,3,1,2,1,1,2,3};
   int N = 15;
   cout << "Number of 2's : "
         << count(A, A + N, 2) << endl;
}
```



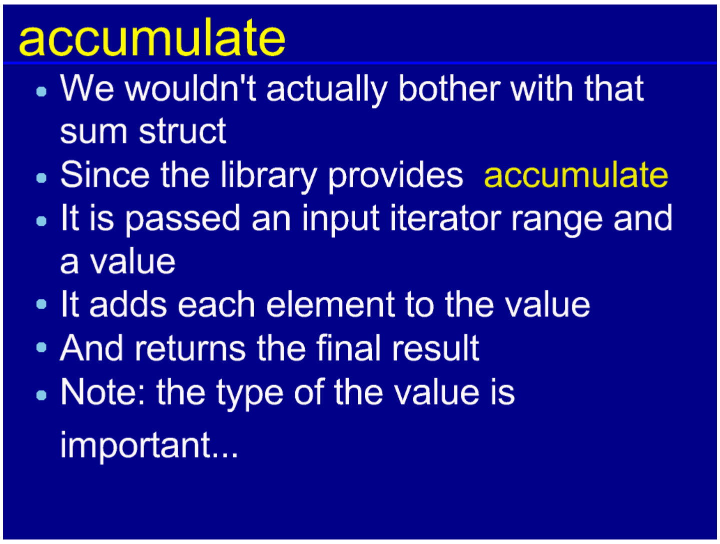
for_each

- Applies a unary function to each element
- Is passed an input iterator range and a unary function
- Returns the unary function



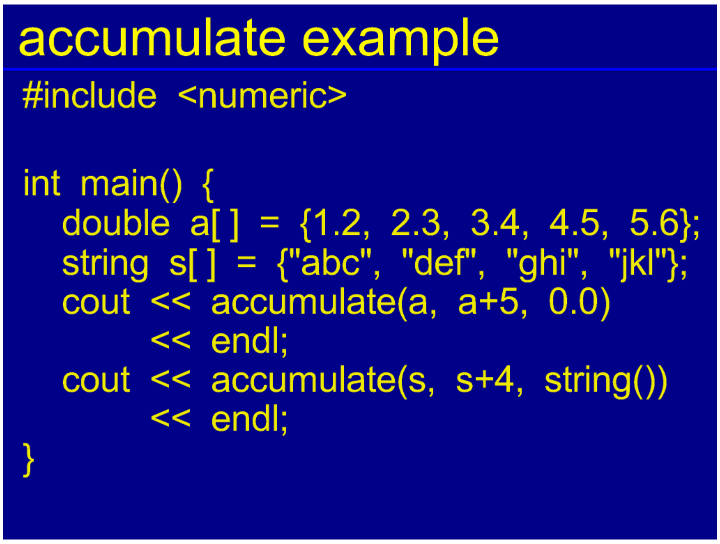
for_each example

```
struct sum {
   int sum;
   sum() : sum(0) { }
   void operator()(int i) { sum += i; }
};
int main() {
   int A[ ] = {1,2,3,1,2,3,1,2,3,1,2,1,1,2,3};
   int N = 15;
   sum s = for_each(A, A + N, sum());
   cout << s.sum << endl;
}
```



accumulate

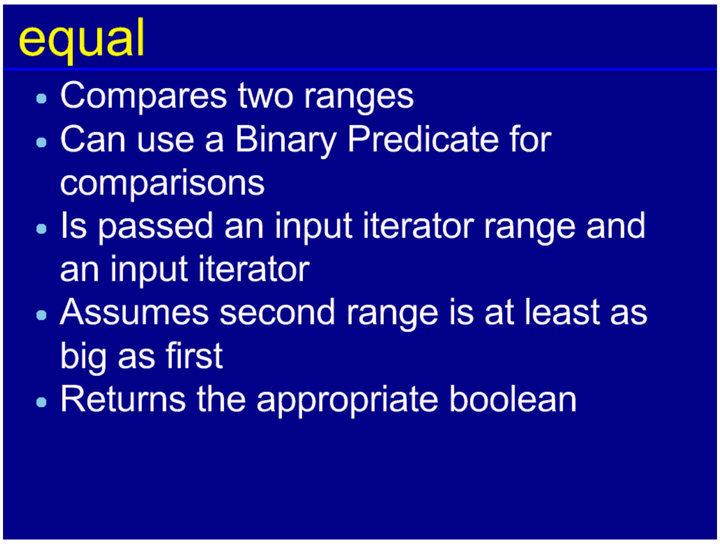
- We wouldn't actually bother with that sum struct
- Since the library provides accumulate
- It is passed an input iterator range and a value
- It adds each element to the value
- And returns the final result
- Note: the type of the value is important...



accumulate example

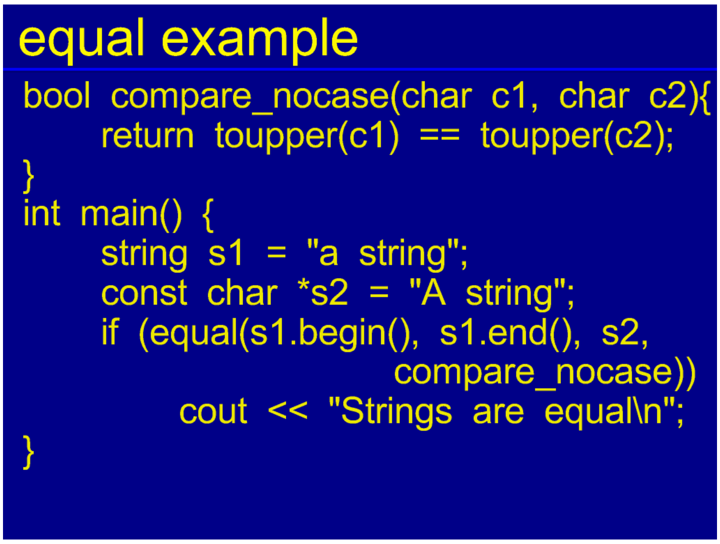
```
#include <numeric>

int main() {
   double a[ ] = {1.2, 2.3, 3.4, 4.5, 5.6};
   string s[ ] = {"abc", "def", "ghi", "jkl"};
   cout << accumulate(a, a+5, 0.0)
          << endl;
   cout << accumulate(s, s+4, string())
          << endl;
}
```



equal

- Compares two ranges
- Can use a Binary Predicate for comparisons
- Is passed an input iterator range and an input iterator
- Assumes second range is at least as big as first
- Returns the appropriate boolean



equal example

```
bool compare_nocase(char c1, char c2){
      return toupper(c1) == toupper(c2);
}
int main() {
      string s1 = "a string";
      const char *s2 = "A string";
      if (equal(s1.begin(), s1.end(), s2,
                              compare_nocase))
            cout << "Strings are equal\n";
}
```

mismatch

- Returns the first positions where two ranges differ
- Parameters are the same as equal()
- Returns a pair of iterators
  - first is iterator in first range
  - second is iterator in second range
  - at end if ranges are the same

mismatch example

```
bool compare_nocase(char c1, char c2){
        return toupper(c1) == toupper(c2);
}
int main() {
        string s1 = "a string";
        const char *s2 = "A string";
        if (mismatch(s1.begin(), s1.end(), s2,
          compare_nocase).first == s1.end())
                cout << "Strings are equal\n";
}
```

lexicographical_compare

- Returns true if first range lexicographically less than second
- Ranges do not have to be the same length
- A binary predicate can be used

Example

```
int main() {
        int A1[ ] = {3,1,4,1,4,5,9,3};
        int A2[ ] = {3,1,4,1,5,0,8,2};
        const int N1 = 8;
        const int N2 = 8;
        if (lexicographical_compare(
                A1, A1 + N1, A2, A2 + N2))
                cout << "A1 < A2" << endl;
        else
                cout << "A1 >= A2" << endl;
}
```

max_element

- Finds the largest element in a range
- Is passed a forward iterator range
- Returns iterator at maximum element
- Can use a Strict Weak Ordering function
- There is also min_element

max_element example

```
int main() {
    list<int> l;
    for(int i = 0; i < 100; ++i)
        l.push_back(rand());
    list<int>::const_iterator min, max;
    min = min_element(l.begin(), l.end());
    max = max_element(l.begin(), l.end());
    cout << "Min : " << *min << endl;
    cout << "Max : " << *max << endl;
}
```

Adapters

- Adapters transform one interface into another
- The STL provides function object adapters
- In fact it provides a lot of them

binder1st and binder2nd

- Transforms a Binary Function into a Unary Function
- binder1st
  - Binds the first argument to a specific value
  - Use helper function bind1st()
- binder2nd
  - Binds the second argument to a specific value
  - Use helper function bind2nd()

binder2nd example

```
int main() {
      vector<int> v;
      for(int i = 0; i < 10; ++i)
            v.push_back(i);
      vector<int>::iterator vi = find_if(
                  v.begin(), v.end(),
                  bind2nd(greater<int>(), 5));
      cout << "Match found at position: "
             << vi - v.begin() << '\n';
}
```

unary_negate

- unary_negate
  - Negates a unary predicate
  - Use helper function not1()
- binary_negate
  - Negates a binary predicate
  - Use helper function not2()

unary_negate example

```
int main() {
      vector<int> v;
      for(int i = 0; i < 10; ++i)
            v.push_back(i);
      vector<int>::iterator vi = find_if(
            v.begin(), v.end(),
            not1(bind2nd(greater<int>(), 5)));
      cout << "Match found at position: "
             << vi - v.begin() << '\n';
}
```

unary_compose

- unary_compose
  - Creates composition of two unary functions
  - compose1() helper function
  - compose1(f, g)(x) is f(g(x))
- binary_compose
  - Composition of three functions
  - compose2() helper function
  - compose2(f, g1, g2)(x1, x2) is f(g1(x1), g2(x2))

mem_fun_ref_t

- Allows member functions to be used as function objects
- mem_fun_ref_t
  - Turns a member function into a function object
  - mem_fun_ref() helper function
- mem_fun_t
  - Similar but uses pointer to object
  - mem_fun() helper function

Example

```
int main() {
   vector<string> v;
   string s;
   while (cin >> s) v.push_back(s);
   vector<string>::iterator i = find_if(
         v.begin(), v.end(), compose1(
            bind2nd(greater<size_t>(), 7),
            mem_fun_ref(&string::length)));
   cout << *i << endl;
}
```