Lecture 14b
The Standard Template Library: Algorithms & Memory Management
April 29th, 2016

Announcements
• PA 10 is due Wednesday
  - Distinguish between
    - Poses within a video
    - Coordinates within a pose
  - Sort, swap & left operate on poses
  - Rotate operates on coordinates
  - Transform operates on both
  - Other questions?
• Recitation
  - This week: installing/using google test (mandatory)
  - Next week: extra help (optional / ungraded)
• Final: same style as midterms
  - Cumulative, but with emphasis on templates
  - Code will be handed out one week from today
• ASCSU surveys today
  - I need a volunteer...

STL
1. Containers
   - Abstractions of data structures
   - Hold a single, arbitrary data type
2. Iterators
   - Abstractions of pointers
   - Allow you to iterate through containers
   - (Begin() and end()) are pointers to first element and first non-element
3. Algorithms
   - Universal algorithms (e.g. sort)
     - Without reference to object type being acted on
     - Without reference to data structure holding the objects
4. Other
   - Multi-threading
   - Numerics
   - Smart pointers

STL Algorithms
• Goals:
  - Provide useful basic algorithms (think 'sort')
  - Without regard to the data type operated on
  - Without regard to the data structure it is stored in
  - Over a whole container or a fragment thereof
• Example
  Template<typename ITER, typename VALUE>
  VALUE accumulate(ITER start, ITER end,
  VALUE init_value)
  {
    for(ITER iter=start; iter!=end; iter++)
    { init_value += *iter; }
    return init_value;
  }

STL Function Calls
• Many STL functions operate on a single container
  - Sort, find, count, reverse, rotate, random_shuffle, ...
  - These take two iterators as arguments
    - Start: where in the container to begin
    - End: where in the container to end
  - Some allow a functor predicate as an optional 3rd argument
    - Example: sort
    - Others have a _if version that takes a functor predicate
      - Example: count_if

Functor Examples
class Evenp {
public:
  bool operator() (int n) {return (n == 2*(n/2));}
};

class Mod_less {
public:
  mod_less(int mod_base): base(mod_base) {} 
  bool operator< (int a, int b) 
  { return ((a % base) < (b % base)); }
protected:
  int base;
}
STL Function Call Examples

// initialize data for example
vector<int> vec = initialize();

// count even elements in vec
int n = count_if(vec.begin(), vec.end(), Evenp());

// sort the elements in vec
sort(vec.begin(), vec.end());

// stable sort elements by mod 111
stable_sort(vec.begin(), vec.end(), Mod_less(111));

// why did I use stable_sort above?

STL Function Calls (II)

• Other STL functions map from one container to another
  – Example: copy
  – Iterate (start, end, iter dest)
  – Return value is end of destination after the copy

• Warning: STL routines do not allocate memory
  – Copy uses assignment (=)
  – Copy increments dest iterator (++)
  – If dest doesn’t have enough elements, bad things happen...

Copy

// possible implementation of copy
// notice that no memory is allocated
template <typename ITER>
ITER copy(ITER start, ITER end, ITER dest)
{
    for(ITER iter = start; iter != end; iter++)
    {
        *dest++ = *iter
    }
}

Buggy Code

vector<int> vec = initialize();
vector<int> second_vec;

// this crashes
copy(vec.begin(), vec.end(),
     second_vec.begin());

Fixed Code

vector<int> vec = initialize();
vector<int> second_vec(vec.length());

// this doesn't crash
copy(vec.begin(), vec.end(),
     second_vec.begin());

STL Function Calls (III)

• Still other STL functions combine two containers into a 3rd:
  – Example: merge
  – Iterate (iter start1, iter end1, iter start2, iter end2, iter dest)
  – Optional 6th argument is a functor to replace ==

• Again, STL functions do not allocate memory...
Why so many functors?

- Imagine your goal is to sort Quaggas by weight. You have two options:
  - Overload < for Quaggas
  - Write a functor, pass it to sort

- Imagine your goal is to sort Quaggas by weight in one part of your program, and height in another
  - You need to write functors
  - You cannot re-define < mid program

Eliminating Loops

- STL algorithms can eliminate most loops
  - transform
    - `Iter transform(Iter start, Iter end, Iter dest, functor fn)`
    - `Fn` can be any functor such that:
      - It is unary on *start
      - Its return type can be stored in *dest
  - `Iter transform(Iter start1, Iter end1, Iter start2, Iter end2, Iter dest)`
    - `Fn` can be any functor such that:
      - It is binary on *start1 and *start2
      - Its return type can be stored in *dest
  - `for_each`
    - `Fn for_each(Iter start, Iter end, functor fn)`
    - `Fn` can be any functor such that:
      - It is unary on *start
      - `Fn` is for side-effects; its return value is ignored