Standard Template Library

Wednesday, October 10, 2007
10:09 AM

- 9.3 "Quick Peek"
  - STL history
    - 1990s
      - Alex Stepanov & Meng Lee of HP Labs
    - 1994 ANSI/IS0 standard
  - Components
    - Container class templates
    - Iterators
    - Algorithm templates

    - Iterators are interface between containers & algorithms
  - Examples
    - vector - dynamic array
    - deque - double-ended queue
    - stack
    - queue
    - list

- 9.7 deque, stack, queue
  - Deque operations
    - constructor
    - empty()
    - push_front (elem)
    - push_back (elem)
    - front()
    - back()
    - pop_front()
    - pop_back()
  - Deque example
    - deque<int> d;
    - d.push_front(3);
    - d.back() = 5;
    - d.pop_front();
  - Deque Notes
    - allows [] operator
    - allows insert & delete at any point like vectors
    - iterators act like vector iterators
    - memory organized as series of memory blocks, typically 4KB
      - Example:
        - push_front 555
        - push_back 1, 2, 3, 4
        - push_front 77

        - map
          - 0 Block 2
          - 1 Block 1
        - Block 1 contains 555, 1, 2, 3, 4

- Block 2 contains 77

- Stack operations
    - constructor - wraps around a container STL
    - empty()
    - top()
    - push(elem)
    - pop()
    - size()
    - comparison operators
- Stack example
    - stack<int, vector<int>> iStack;
    - stack<int> bStack;
        - uses deque as container
- Queue operations
    - constructor - also wraps around container
    - empty()
    - front()
    - back()
    - pop()
    - push(elem)
- Queue example
    - queue<int, vector<int>> aQueue;
    - queue<int> iQueue;
        - use deque as container

- 10.5 Standard Algorithms
    - operate on container iterators
    - sort
        - using < to compare elements
            - vecto<int> v;
            - // put stuff in v
            - sort(v.begin(), v.end());
        - using "less-then" to compare (a function)
            - bool LessThan(int a, int b)
                - { return a > b; }
            - int main() {
                - vector<int> v;
                - // add to v
                - sort(v.begin(), v.end(), LessThan);
            - }
    - Other STL algorithms
        - binary_search (begin, end, value)
        - find (begin, end, value)
        - search (begin1, end1, begin2, end2)
            - search for a sequence of values
        - copy (begin, end, container)
        - count (begin, end, value)
            - how many times value occurs
        - sort (begin, end)
        - unique (begin, end)
        - reverse (begin, end)
        - more algorithms listed in book pp 570-2
- 11.3 STL list
    - variation on doubly linked list

- comparison to other STL Containers
  - does not allow [] operator like deque
    - cannot use STL algs like sort()
  - good at inserting & deleting at any point
  - good for sequential iteration
  - higher overhead then deque
- iterator is bidirectional only (no random access)
  - supports following operators:
    - ++ to go to next node
    - -- to go to previous node
    - * to access data in current node
    - = to assign one iterator to another
    - == and != to compare two iterators
  - declaring iterator:
    - list<int>::iterator i;
    - list<int>::const_iterator ic; // read only
      - // access to list elements
    - list<int> ld;
    - //add data to ld
    - i= ld.begin(); // head node
    - i++; // second node
    - i--; // back to head node
- operations
  - constructors
    - default creates empty list
    - list(int n) creates list w/ n slots
    - list(int n, T value) creates list w/ n slots that all have passed value
    - list(startPtr, endPtr) creates list w/ contents of startPtr up to (not including) endPtr
  - copy constructor
  - destructor
  - empty()
  - size()
  - push_back(T elem) tail insert
  - push_front(T elem) head insert
  - insert(position, T elem) position is an iterator
    - returns iterator to new node
  - insert(position, int n, T elem) put n copies of element at given position
  - insert (position, startPtr, endPtr)
    - does not include endPtr, like constructor
  - pop_back() tail delete
  - pop_front() head delete
  - erase(position) delete node at position
  - erase(position1, position2) delete from position1 to position 2
  - remove(T elem) delete all nodes containing elem
  - unique() collapse repeating sequences
  - front() retrieve head's value
  - back() retrieve tail's value
  - begin() return iterator to head
  - end() return iterator to 1 past tail
  - rbegin() return reverse iterator to tail

- rbegin() return reverse iterator to tail
- rend() return reverse iterator to 1 before head
- sort() sort using < operator
- reverse() reverse order of elements
- merge(list2)
    - place elements from list2 into this list in < sorted order, remove all elements from list2. both lists must first be sorted.
- splice(position, list2)
    - place elements from list2 into this list in list2 order at the given position. remove elements from list2
- splice(to-pos, list2, from-pos)
    - start in list2 at iterator from-pos instead of whole list2
- splice (position, list2, start, end)
    - take elements from start to end (not including end) from list2
- swap(list2) swap this list w/ list2
- list1= list2
- list1 == list2 elements must be in same order for both lists
- list1 < list2 lexicographical less than

Demo of list code

```
// must define output operator
template <class T>
ostream & operator << (ostream & o,
    const list<T> & l)
{
    list<T>::const_iterator i;
    for(i = l.begin(); i != l.end(); i++)
        o << *i << " ";
    return o;
}
int main()
{
    list<int> la;  // default constructor
    list<int> lb(3); // set slots
    list<int> lc(5, 11); // set slots & default val
    int array[] = {2, 22, 222, 2222};
    list<int> ld(array, array+4);
    list<int>::iterator i;

    i = lc.begin();
    lc.insert(i, 65);
    lc.insert(i, 3, 78); // 3 copies
    lc.insert(i, array, array + 4);
    cout << lc << endl;

    i= find(lc.begin(), lc.end(), 65);
    if (i == lc.end())
        cout << "value 65 not found in list\n";
    else
        cout << "Value 65 found\n";
    lc.remove(22);
    i = lc.end();
    i--; i--;
```

```
lc.erase(lc. begin(), i);
```

- more examples in book pp 608-616

How list works

- uses a doubly linked circular list w/ a dummy (never used for data) head node
- keeps stack of free nodes instead of using new & delete all of the time
    - -only allocates when stack is empty
    - -one stack for each datatype
    - -allocates a chunk of memory & breaks into nodes for a free stack
    - -deallocates stack for datatype T only when all lists for T have been deleted
- iterators
    - begin() points to 1st actual node, skips dummy head node
    - end() points to dummy head node
    - rbegin() points to tail
    - rend() points to dummy head node