Assignment 5

Sunday, May 14, 2023 9:24 PM

COEN 79

Object-Oriented Programming and Advanced Data Structures

Assignment #5 – Template and Iterators

Name: Yen-Jung Lu **Date:** 05/15/2023

1. Write a template function that takes two bags (maybe holding different types of elements) and returns the size of the bigger bag, i.e., if bag1 has 5 elements and bag2 has 7 elements, the function returns 7.

1.)
```
template < typename T1, typename T2 >
typename T1 :: size_type get_size (const T1& bag1, const T2& bag2)
{
    typename T1 :: size_type size1 = bag.size();
    typename T2 :: size_type size2 = bag.size();
    return ( size1 > size2) ? size1 : size2;
}
```

2. What are the *iterator invalidation rules* for a data structure that stores items in a *linked list*?

2.)
- Insertion: insert an element in any position in the linked-list does not invalidate any exist
- Deletion: Delete a node that from the linked-list invalidate iterator points to the deleted node,
- Resizing: Increase or decrease the size of linked-list may invalidate all iterators,

3. What are *the iterator invalidation rules* for STL's `vector` class?

3.)
- Insertion: insert an element at the end of the vector invalidates any iterator that point to the elem
- Deletion: delete an element from a vector invalidate any iterators that point to the deleted element
- Resizing: increase or decrease the size of the vector may invalidate the iterators,

4. What are the features of a *random access iterator*? Present the name of two STL data structures that offer random access iterators.

4.)
- Arithmetic operations: Random access iterators support +, −, +=
- And can also do comparison: like ==, <, >

STL data structure:

⇒ std::vector

std::array

5. The bag class is defined as follows:

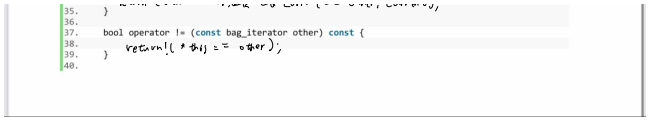
```
template < class Item >
class bag {
public:
    // TYPEDEFS and MEMBER CONSTANTS
    typedef Item value_type;
    typedef std::size_t size_type;

    static const size_type DEFAULT_CAPACITY = 30;

    typedef bag_iterator < Item > iterator;

    bag(size_type initial_capacity = DEFAULT_CAPACITY);
    bag(const bag& source);
    ~bag();

    // MODIFICATION MEMBER FUNCTIONS
    // ...

    iterator begin();
    iterator end();

private:
    Item* data;          // Pointer to partially filled dynamic array
    size_type used;      // How much of array is being used
    size_type capacity;  // Current capacity of the bag
};
```

- This class implements the following functions to create iterators:

```
template <class Item>
typename bag <Item>::iterator bag<Item>::begin() {
    return iterator(capacity, used, 0, data);
}

template <class Item>
typename bag<Item>::iterator bag<Item>::end() {
    return iterator(capacity, used, used, data);
}
```

- Please complete the implementation of the following iterator:

```
template < class Item >
class bag_iterator: public std::iterator < std::forward_iterator_tag, Item >
{
private:
    size_type capacity;
    size_type used;
    size_type current;
    Item* data;

Public:
    typedef std::size_t size_type;
    bag_iterator(size_type capacity, size_type used, size_type current, Item* data) {
        this -> capacity = capacity;
        this -> used = used;
        this -> current = current;
        this -> data = data;
    }

    Item& operator* () const {
        return data[current];
    }

    bag_iterator& operator++()   // Prefix ++
    {
        ++current;
        return *this;
    }

    bag_iterator operator++(int) // Postfix ++
    {   bag_iterator temp = *this;
        ++(*this);
        return temp;
    }

    bool operator == (const bag_iterator other) const {
        return ( data == other.data && current == other.current);
```



```
35.     }
36.
37.     bool operator != (const bag_iterator other) const {
38.         return !( *this == other);
39.     }
40.
```