Sets

CSE 2020 Computer Science II

Learning Objectives

- Define Set ADT
- Design and implement Set ADT
- Apply set class defined in STL

Set ADT

- A set is a container that stores a collection of unique elements following a specific order.
  - The value of an element identifies the element itself and each element value must be unique.
  - The elements’ values cannot be modified but they can be inserted or removed from the set.
  - The elements are always sorted following a specific *ordering* criterion indicated by its internal comparable objects.

Operations of Sets

- bool **isEmpty**() const: returns true if the set is empty
- int **getSize**() const: returns the number of elements in set
- bool **find**(const C& x) const: returns true if x is in set
- void **insert**(const C& x): inserts x to the set
- void **remove**(const C& x): removes x from the set
- void **makeEmpty**(): makes the set to empty state

Implementation

- Using sorted array to implement Set
- Using sorted linked structure to implement Set
- Using binary search tree implement Set

Sorted Array Impl.

- Attributes: dynamic array, size, capacity
- find(x)
  - binary search, O(logN)
- insert(x)
  - find the right position for x in array, if x is in array, do noting; else insert x at the right place of array, O(logN) + O(N) = O(N)
- remove(x)
  - find x in array, if x is not in array, do noting; else remove x from the array, O(logN) + O(N) = O(N)

Sorted Linked Structure Impl.

- Attributes: pointer head, size
- find(x)
  - linear search, O(N)
- insert(x)
  - find the right position for x in linked structure, if x is in linked structure, do noting; else insert x at the right place, O(N) + O(1) = O(N)
- remove(x)
  - find x in linked structure, if x is not in linked structure, do noting; else remove x from the linked structure, O(N) + O(1) = O(N)

Binary Search Tree Impl

- Attributes: pointer root, size. root points to the root node of the balanced binary search tree
- find(x)
  - find x in bst, O(logN)
- insert(x)
  - find the right position for x in bst, if x is in bst, do noting; else insert x at the right place of bst, O(logN)
- remove(x)
  - find x in bst, if x is not in bst, do noting; else remove x from the bst, O(logN)

Iterator in Set

- Why iterator? Access the nodes in the set
- iterator is the nested class
  - private attribute pointer *current* points to the current node
  - stack *antes* is for non-recursive inorder traversal
  - operations
    - dereference * returns the element of current node
    - prefix ++ returns the next node in inorder traversal
    - ==, != return true if the address passed is same (different) to the address of current node
- in Set class
  - iterator begin() returns the iterator representing the 1st node
  - iterator end() returns position after the last node
- Set.cpp in Set.txt on Canvas

Use Iterator

- print the elements in a set

```
Set<int> myset;
....
for (Set<int>::iterator itr = myset.begin(); itr !=
myset.end(); ++itr)
  cout << *itr << “, ”;
```

- print()

```
template <typename C>
void print(const Set<C> & s){
    for (typename Set<C>::iterator itr = s.begin(); itr !=
          s.end(); ++itr)
          cout << *itr << ",";
}
```

- TestSet.cpp in Set.txt on Canvas

Set Union A + B

- overload operator+

```
template <typename C>
Set<C> operator+(const Set<C> & s1, const Set<C> & s2)
{
    Set<C> result;
    for (typename Set<C>::iterator itr = s1.begin(); itr != s1.end(); ++itr)
       result.insert(*itr);

    for (typename Set<C>::iterator itr = s2.begin(); itr != s2.end(); ++itr)
      result.insert(*itr);
    return result;
 }
```

Set Subtraction A - B

- overload operator-

```
template <typename C>
Set<C> operator-(const Set<C> & s1, const Set<C> & s2)
{
   Set<C> result;
   for (typename Set<C>::iterator itr = s1.begin(); itr !=
   s1.end(); ++itr)
     result.insert(*itr);

   for (typename Set<C>::iterator itr = s2.begin(); itr !=
   s2.end(); ++itr)
     result.remove(*itr);
   return result;
}
```

Set Intersection A * B

- function intersection

```
template <typename C>
Set<C> operator*(const Set<C> & s1, const Set<C> & s2)
{
   Set<C> result;
   for (typename Set<C>::iterator itr = s1.begin(); itr
!= s1.end(); ++itr)
      if (s2.contains(*itr))
         result.insert(*itr);
   return result;
}
```

set in STL

- In STL, C++ implements set class template using binary search tree.

```
#include <set>
#include <iterator>
set<int> intset;
intset.insert(10);
intset.insert(5);
intset.erase(5);
set<int>::iterator itr;
for (itr = intset.begin(); itr != intset.end(); itr++)
    cout << *itr << " ";
```