

# CSCI-1200 Data Structures — Spring 2023

## Lecture 17 Preview – STL Sets

### 17.1 Standard Library Sets

- STL sets are *ordered* containers storing unique “keys”. An ordering relation on the keys, which defaults to `operator<`, is necessary. Because STL sets are ordered, they are technically not traditional mathematical sets.
- Sets are like maps except they have only keys, there are no associated values. Like maps, the keys are **constant**. This means you can’t change a key while it is in the set. You must remove it, change it, and then reinsert it.
- Access to items in sets is extremely fast!  $O(\log n)$ , just like maps.
- Like other containers, sets have the usual constructors as well as the `size` member function.

### 17.2 Set iterators

- Set iterators, similar to map iterators, are bidirectional: they allow you to step forward (`++`) and backward (`--`) through the set. Sets provide `begin()` and `end()` iterators to delimit the bounds of the set.
- Set iterators refer to const keys (as opposed to the pairs referred to by map iterators). For example, the following code outputs all strings in the set `words`:

```
for (set<string>::iterator p = words.begin(); p!= words.end(); ++p)
    cout << *p << endl;
```

### 17.3 Set insert

- There are two different versions of the `insert` member function. The first version inserts the entry into the set and returns a pair. The first component of the returned pair refers to the location in the set containing the entry. The second component is true if the entry wasn’t already in the set and therefore was inserted. It is false otherwise. The second version also inserts the key if it is not already there. The iterator `pos` is a “hint” as to where to put it. This makes the insert faster if the hint is good.

```
pair<iterator,bool> set<Key>::insert(const Key& entry);
iterator set<Key>::insert(iterator pos, const Key& entry);
```

### 17.4 Set erase

- There are three versions of `erase`. The first `erase` returns the number of entries removed (either 0 or 1). The second and third erase functions are just like the corresponding erase functions for maps. Note that the `erase` functions do not return iterators. This is different from the `vector` and `list` erase functions.

```
size_type set<Key>::erase(const Key& x);
void set<Key>::erase(iterator p);
void set<Key>::erase(iterator first, iterator last);
```

### 17.5 Set find

- The `find` function returns the `end` iterator if the key is not in the set:

```
const_iterator set<Key>::find(const Key& x) const;
```