Basically SETS are kind of RB Trees, and hence elements must be distinct and also are ordered, so they need a comparator. When the data type is primitive we can use user defined comparators like < (default) or > etc but when we make structures/classes as our data types, we need to specifically define how to COMPARE them.  
Multisets can store more than one element with equivalent values. Two elements are equal if the comparator returns false **Reflexively** when the two elements are sent as parameters.

Declaration :- **multiset** < *data type , comparator* > *var\_name* ; ( set <int> ms1 ) , ( set <int , greater(int) > ms1 )  
 1D-Array => **(** vector < multiset<int> > v1 **)**

ms1.begin () - Returns an iterator pointing to the first element in the multiset  
ms1.end () - Returns an iterator pointing to the theoretical element that follows last element in the multiset

ms1.rbegin () –Returns a reverse iterator pointing to the last element in the multiset (reverse beginning).   
 It moves from last to first element  
ms1.rend () – Returns a reverse iterator pointing to the theoretical element preceding the first element in the multiset  
 (considered as reverse end)

ms1.size( ) - Returns the number of elements in the multiset  
ms1.empty() - Check for emptiness of the multiset or not  
ms1.count( val ) - Returns the number of a particular element , *here val* in the Set ( either 0 or 1 )  
 Returns the number of a particular element , *here val* present in the Mutliset   
ms1.find(val ) - Returns the iterator to *val* if present, else returns s1.end( )  
 Returns the iterator to any instance *val* if present, else returns s1.end( )

ms1.insert (20) - Returns a Pair :< iterator to inserted 20/already present 20, bool value *depending on insertion>*  
ms1.insert (it, 20) - Returns an iterator pointing to 20, new or old one. Hint iterator *it* not have much role as such.  
ms1.insert (itL, itR) - Inserts elements in range [itL , itR) , which are iterators of some other container.  
 We Can Use Assignment Operator To Assign Values Of One Set To Another

ms1.emplace( 20 ) - Returns a Pair: <iterator to inserted 20/already present 20 , bool value *depend on insertion>*ms1.emplace\_hint( it , 20 ) - Returns an iterator pointing to val. Hint iterator *it* not have much role as such  
  
*The difference between insert and emplace is , emplace mein the element data is created at the place of insertion , whereas insert mein what happens is there is a lot of moving around / copying of already created stuff .*

ms1.erase (it) - Erases element referenced by the position ‘it’   
ms1.erase (val) - Erases with value *val* (returns number of element deleted i.e. **0** or **1**)  
 Erases with value *val* (returns number of element deleted)  
ms1.erase(itL , itR) - Erases elements in range [itL , itR)

ms1.swap(ms2 ) - Swaps elements in multiset s1 with multiset s2   
ms1.clear( ) - Clears multiset s1   
  
ms1.equal\_range(val ) - returns a pair < ms1.lower\_bound (val ) , ms1.upper\_bound (val ) >   
ms1.lower\_bound (val ) - returns an iterator pointing to the first element equivalent or after ‘val’ *(if it was there)*.   
ms1.upper\_bound (val ) - returns an iterator pointing to the first element after ‘val’ *(if it would have been there)* .