L2: Linked List with dynamic allocation

Implement in C++ the given **container** (ADT) using a given representation and a **linked list with dynamic allocation** as a data structure. You are not allowed to use the *list* from STL or from any other library.

Note: SLL means singly-linked list while DLL means doubly-linked list

1. **ADT Matrix** – represented as a sparse matrix, using a SLL with <line, column, value> triples (value ≠ 0), ordered lexicographically considering the line and column of every element.
2. **ADT Matrix** – represented as a sparse matrix, using a DLL with <line, column, value> triples (value ≠ 0), ordered lexicographically considering the line and column of every element.
3. **ADT Bag** – using a SLL with (element, frequency) pairs.
4. **ADT Bag** – using a DLL with (element, frequency) pairs.
5. **ADT SortedBag** – using a SLL with (element, frequency) pairs. Pairs are ordered based on a relation between the elements.
6. **ADT SortedBag** – using a DLL with (element, frequency) pairs. Pairs are ordered based on a relation between the elements.
7. **ADT SortedSet** – using a SLL where elements are ordered based on a relation between the elements.
8. **ADT SortedSet** – using a DLL where elements are ordered based on a relation between the elements.
9. **ADT Set** – using a SLL.
10. **ADT Set** – using a DLL.
11. **ADT Map** – using a SLL with (key, value) pairs.
12. **ADT Map** – using a DLL with (key, value) pairs.
13. **ADT MultiMap** – using a SLL with (key, value) pairs. A key can appear in multiple pairs. Pairs do not have to be ordered.
14. **ADT MultiMap** – using a DLL with (key, value) pairs. A key can appear in multiple pairs. Pairs do not have to be ordered.
15. **ADT MultiMap** – using a SLL with *unique* keys. Every key will be associated with a SLL of the values belonging to that key.
16. **ADT MultiMap** – using a DLL with *unique* keys. Every key will be associated with a DLL of the values belonging to that key.
17. **ADT SortedMap** – using a SLL with (key, value) pairs ordered based on a relation on the keys.
18. **ADT SortedMap** – using a DLL with (key, value) pairs ordered based on a relation on the keys.
19. **ADT SortedMultiMap** – using a SLL with *unique* keys ordered based on a relation on the keys. Every key will be associated with a SLL of the values belonging to that key.

20. **ADT SortedMultiMap** – using a DLL with *unique* keys ordered based on a relation on the keys. Every key will be associated with a DLL of the values belonging to that key.
21. **ADT SortedMultiMap** – using a SLL with (key, value) pairs ordered based on a relation on the keys. A key can appear in multiple pairs.
22. **ADT SortedMultiMap** – using a DLL with (key, value) pairs ordered based on a relation on the keys. A key can appear in multiple pairs.
23. **ADT List** (interface with **TPozition = Integer** - IndexedList) – using a SLL
24. **ADT List** (interface with **TPozition = Iterator** - IteratedList) – using a SLL
25. **ADT List** (interface with **TPozition = Integer** - IndexedList) – using a DLL
26. **ADT List** (interface with **TPozition = Iterator** - IteratedList) – using a DLL
27. **ADT SortedList** (interface with **TPozition = Integer** - IndexedList) – using a SLL where elements are ordered based on a relation.
28. **ADT SortedList** (interface with **TPozition = Iterator** - IteratedList) – using a SLL where elements are ordered based on a relation.
29. **ADT SortedList** (interface with **TPozition = Integer** - IndexedList) – using a DLL where elements are ordered based on a relation.
30. **ADT SortedList** (interface with **TPozition = Iterator** - IteratedList) – using a DLL where elements are ordered based on a relation.
31. **ADT Priority Queue** – using a SLL with (element, priority) pairs ordered based on a relation between the priorities.
32. **ADT Priority Queue** – using a DLL with (element, priority) pairs ordered based on a relation between the priorities.