Five STL containers

1. `std::vector<T>` - Implemented as a dynamic array of **T** elements.
   Increasing or decreasing its size implies copy and move operations.
   Good candidate when a container required for storing large number of elements *without heavy search load.*
   Direct indexing is possible.

2. `std::list<T>` - Implemented as a doubly-linked list of **T** nodes,
   where a node of $n \in (1, ..., m)$ knows about its *previous* and *next* nodes, while n_1 undefined for previous and n_m undefined for next.
   Good candidate when a container required for storing large number of elements *with heavy element creation / deletion load.*
   Increasing or decreasing its size doesn't imply any reallocation and copying operations on existing elements.
   Direct indexing is not possible.

3. `std::map<K,V>` - Implemented as a balanced binary search tree of **T** nodes. Nodes are characterised by a *key* and a *value*.
   Good candidate when a container required for storing large number of elements *with heavy search load.*
   Direct indexing is possible with keys, where the retrieval cost is $O(log(n))$.

4. `std::stack<T>` - Implemented by following the *FILO* mechanism
   $\Rightarrow$ First In Last Out.

5. `std::set<T>` - Represents the mathematical term $\Rightarrow$ stores *unique* elements of **T** in a *sorted* way.