
UNIVERSITY OF LIVERPOOL

# Application Development with C++ (ELEC362)

## Lecture 10: Standard Template Library (STL)

mihasan@liverpool.ac.uk

# Previous lecture

- The concept of class Inheritance was discussed.
- Different types of class inherence were discussed.
- An introduction to C++ libraries was given, and the different types of linking were discussed.

# This lecture

- What is covered in this lecture?

  1. Standard Template Library (STL)

- Why it is covered?

  STL is the most widely used library in all of C++.

- How are topics covered in this lecture:

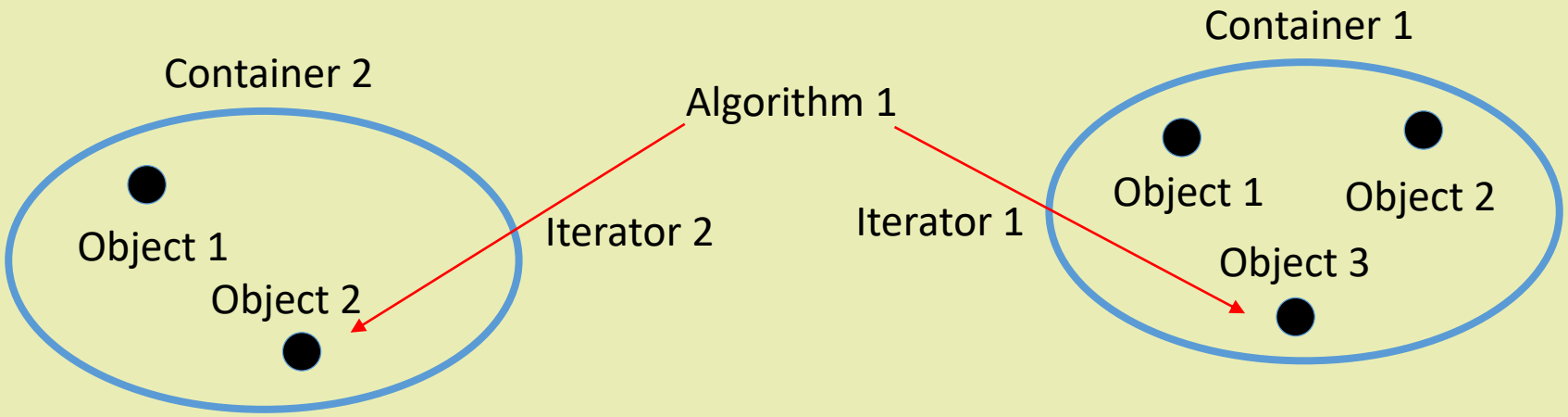
  4 source codes, and a live demonstration on documentation.

# Standard Template Library

- Data structure management is a common aspect in almost every programme.
- The Standard Template Library (STL) is a built-in library in C++, primarily used for data structures.
- All templates of STL are defined in namespace std.
- Components of STL can be classified into three major categories:

1. Containers: used to manage and store collections of objects of a certain kind.
2. Algorithms: provide the means by which one performs initialisation, sorting, etc.
3. Iterators: used to step through the collection of objects (i.e. containers).

# Standard Template Library




- Some iterators belong to a specific type of containers, while others are stand-alone objects.
- Almost all objects in STL are allocated on the heap.

# Containers

- A container is a holder object that stores a collection of other objects (its elements).
- There are two categories of containers:

1. Sequential containers: maintains the orders of the inserted elements

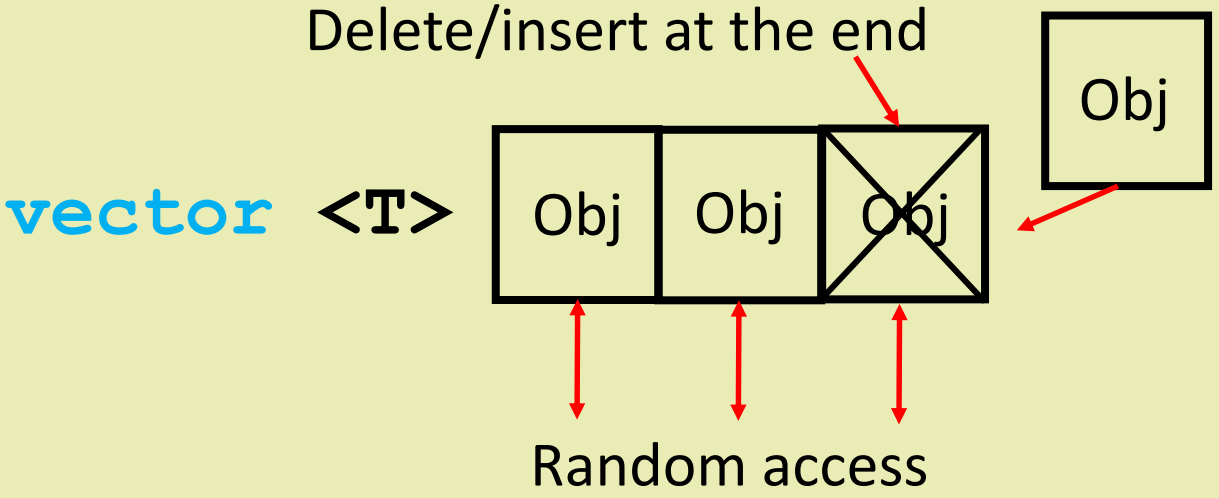
   Examples: `vector <T>, array <T,n>, deque <T>, list <T>`

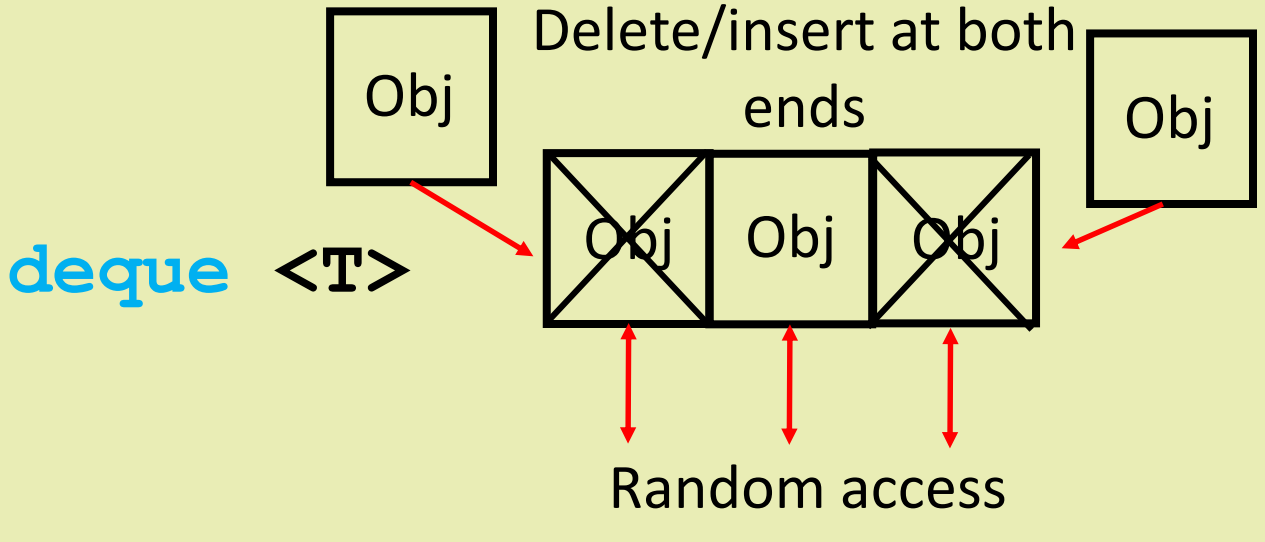
2. Associative containers: elements are ordered based on a criteria set by a “key”.

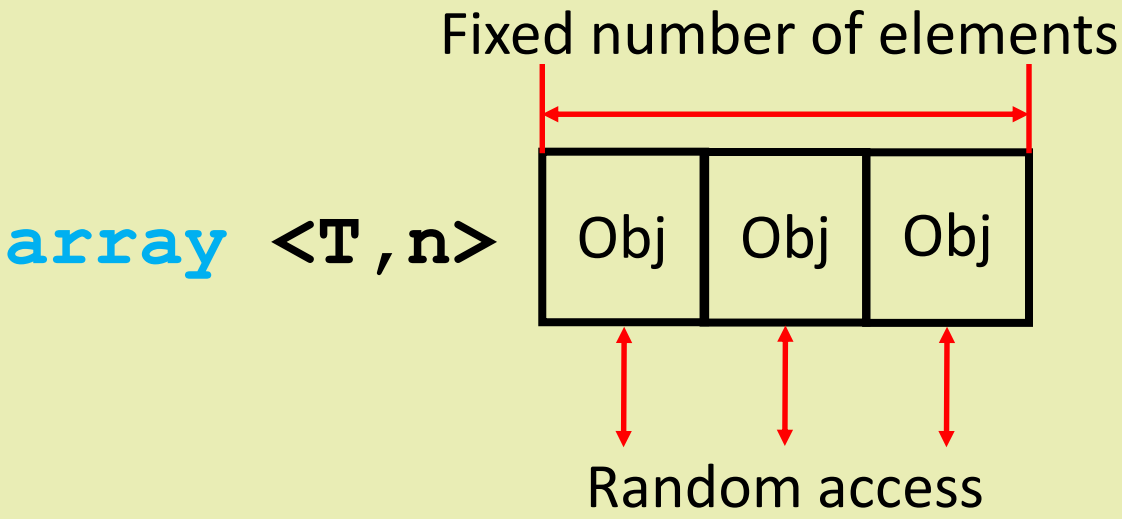
   Examples: `set <T>, map <k,T>, multiset <T>, bitset <n>`

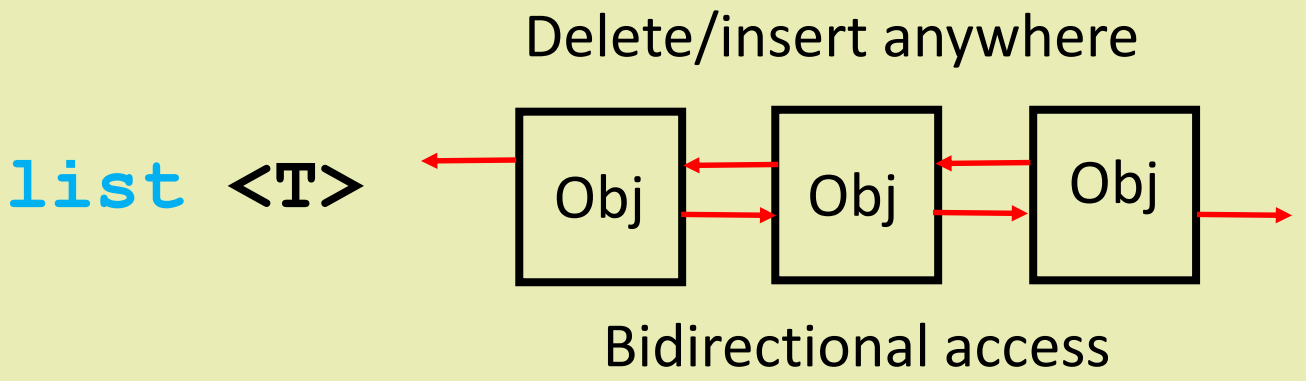
- Reference: http://www.cplusplus.com/reference/stl/ (demonstration)

# Sequence containers



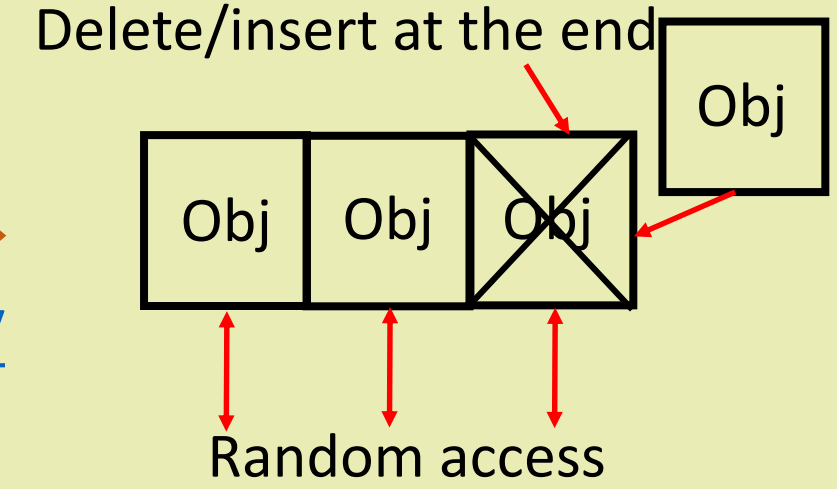



# Vector

- The most widely used container for dynamic allocation.
- It allows editing at the end only.
- Can be included in any code using `#include <vector>`
- Ref: http://www.cplusplus.com/reference/vector/vector/
- The functions **`size()`** and **`capacity()`** are important in codes where the size of the vector changes rapidly.




- The function **`size()`** returns the number of elements in the vector while **`capacity()`** returns the free memory set aside for growth of the vector.
- Go to L10D1.cpp and L10D2.cpp

Practical note: Whenever possible use vectors instead of basic dynamic memory allocation

# Extending Container's functionality

- Memory pre-allocation significantly increases the speed of the programme.
- It is possible to control memory pre-allocation of containers by using allocators.

  Example: `vector <T, allocator <T>>`

- It is also possible to modify the editing properties of a container by using adaptors.

  Example: `queue <T, vector <T>>`

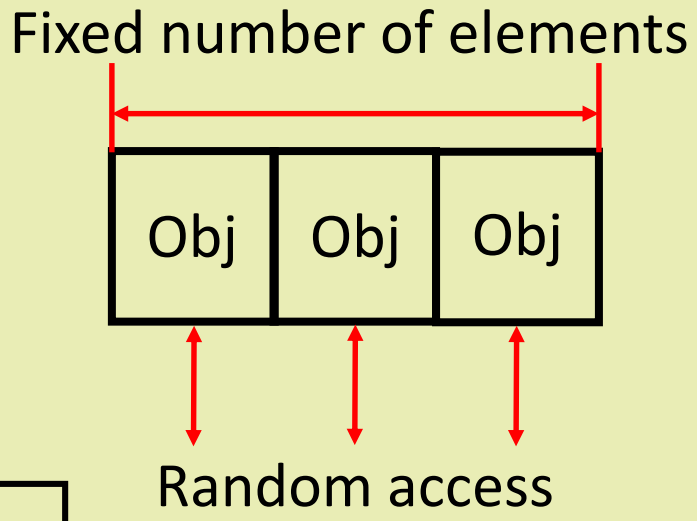
  In this example, the vector now adds elements at one end and delete elements at the other end (First-In-First-Out).

- Reference: http://www.cplusplus.com/reference/queue/queue/?kw=queue

# Array container

- Similar to standard arrays.
- It has a fixed size.
- Can be included in any code using `#include <array>`
- Ref: http://www.cplusplus.com/reference/array/array/
- Syntax:

```
array<data_type,size> array_name;
```




- Example:

```
array<int, 5> someInts = {123,234,345,456,567};
someInts.size(); // 5 elements
```

- In comparison to raw arrays, the STL array has a lot of additional features.

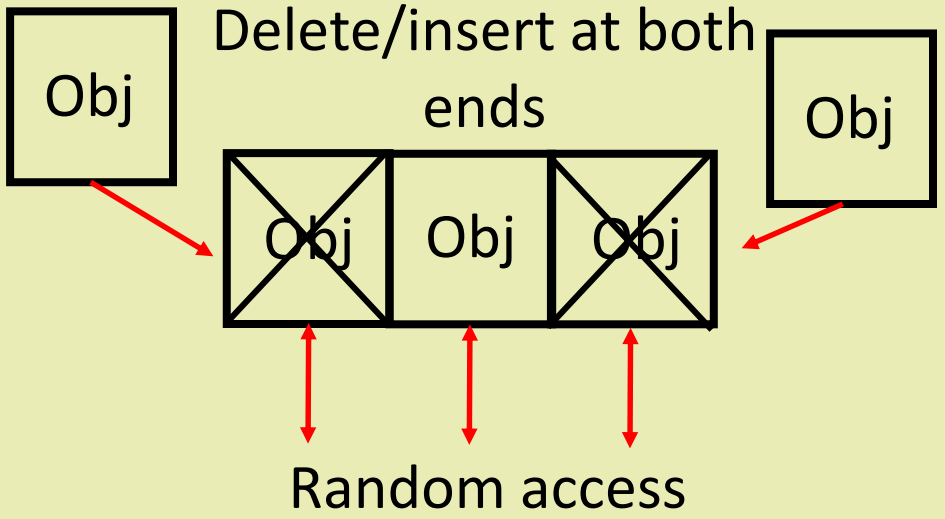
# Deque container

- Stands for: Double Ended Qeue container.
- Header: `#include <deque>`
- Ref: http://www.cplusplus.com/reference/deque/
- The functions “ `push_front` ” and “`pop_front` ” add or delete the elements at the beginning.
- Syntax:

```
deque<data_type> deque_name;
```

- Example:

```
deque<int> d5{1,2,3,4,5};
d5.push_front(2); // Adds “2” at beginning
```

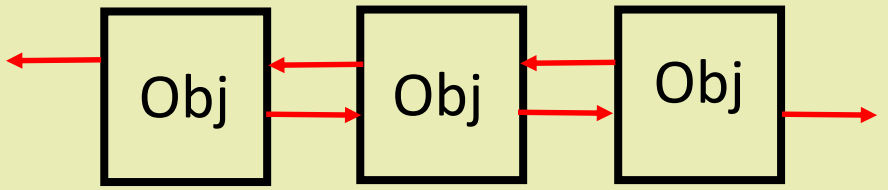


# List container

- Allow insertion and deletion anywhere with bidirectional access.
- Header: `#include <list>`
- Ref: http://www.cplusplus.com/reference/list/list/
- Syntax:

```
list<data_type> list_name;
```

Delete/insert anywhere




Bidirectional access

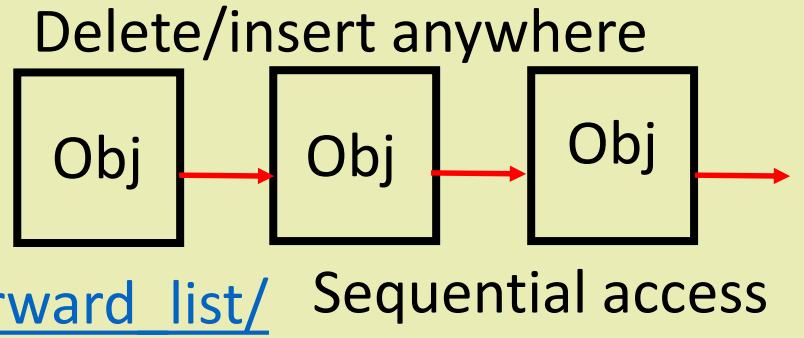
- Example:

```
list<string> names; // creates a list

list<string> surnames(20); //with 20 elements

list<string> firstname(20, "Bruce"); //all initiated to "Bruce"
```

- Go to L10D3.cpp

# Forward list container

- Allow insertion and deletion anywhere with forward access only.
- Header: `#include <forward_list>`
- Ref: http://www.cplusplus.com/reference/forward_list/forward_list/
- Syntax:




```
Forward_list<data_type> fwrd_list_name;
```

- Example:

```
forward_list<int> f5{1,2,3,4,5};

cout <<*(--f5.end())<<endl; // Error!!
```

# Map container

- Maps are associative containers, which use a key to access an object.
- Example: a phonebook map uses a “name” key to access a “phone number object”.
- Header: `#include <map>`
- Ref: http://www.cplusplus.com/reference/map/map/
- Syntax:

```
Map <key_data_type,data_type> map_name;
// To create a pair
using Entry = pair <key_data_type,data_type>;
```

- Go to L10D4.cpp

# Summary

- The Standard Template Library (STL) was introduced and its components were discussed.
- Different types of containers were discussed.
- Vectors, deques, and lists were discussed.
- Maps were discussed.