Templates –C++

Dr.R.Priyadarshini, VIT SCOPE

Topics

- Class Templates
- Function Templates
- Standard Library Templates

Types of Templates

- Templates are powerful features of C++ which allows us to write generic programs. There are two ways we can implement templates:
- Function Templates
- Class Templates

Function Templates

```
template <class T>
class className {
  private:
    T var;
    ... .. ...
  public:
    T functionName(T arg);
    ... .. ...
};
```

Class Template Declaration

- A class template starts with the keyword template followed by template parameter(s) inside <> which is followed by the class declaration.

STL

- **The C++ Standard Template Library (STL)**
- The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc.
- It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized.

STL

- **TL has four components**
- Algorithms
- Containers
- Functions
- Iterators

STL- Algorithms

- The header algorithm defines a collection of functions especially designed to be used on ranges of elements.
- They act on containers and provide means for various operations for the contents of the containers.
- STL has an ocean of algorithms, for all < algorithm > library functions : Refer here.
- Some of the most used algorithms on vectors and most useful one’s in Competitive Programming are mentioned as follows :

https://www.geeksforgeeks.org/algorithms-library-c-stl/

Non-Manipulating Algorithms

- **sort(first_iterator, last_iterator)** – To sort the given vector.
- **reverse(first_iterator, last_iterator)** – To reverse a vector.
- ***max_element (first_iterator, last_iterator)** – To find the maximum element of a vector.
- ***min_element (first_iterator, last_iterator)** – To find the minimum element of a vector.
- **accumulate(first_iterator, last_iterator, initial value of sum)** – Does the summation of vector elements

Non-Manipulating Algorithms

- count(first_iterator, last_iterator,x) – To count the occurrences of x in vector.

- find(first_iterator, last_iterator, x) – Returns an iterator to the first occurence of x in vector and points to last address of vector ((name_of_vector).end()) if element is not present in vector.

Vectors and Lists in C++

STL

C++ STL

- The C++ STL (Standard Template Library) is a powerful set of C++ template classes to provide general-purpose classes and functions with templates that implement many popular and commonly used algorithms and data structures like vectors, lists, queues, and stacks.

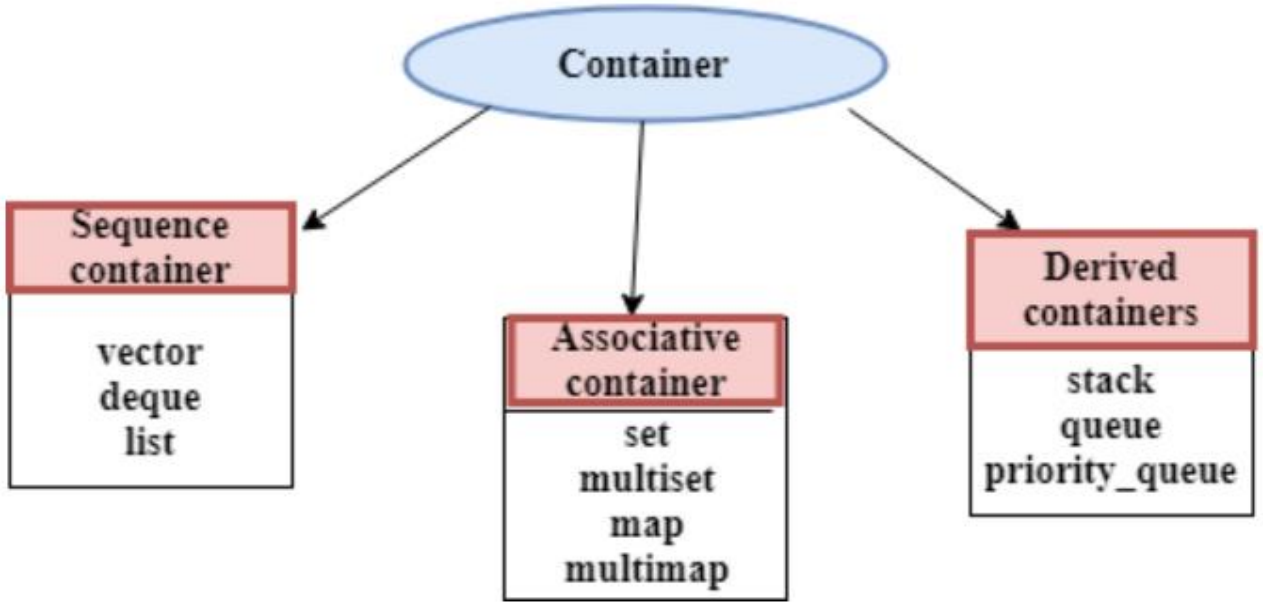
Components

Sr.No	Component & Description
1	**Containers** Containers are used to manage collections of objects of a certain kind. There are several different types of containers like deque, list, vector, map etc.
2	**Algorithms** Algorithms act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.
3	**Iterators** Iterators are used to step through the elements of collections of objects. These collections may be containers or subsets of containers.

Container types

Container	Description	Header file	iterator
vector	vector is a class that creates a dynamic array allowing insertions and deletions at the back.	<vector>	Random access
list	list is the sequence containers that allow the insertions and deletions from anywhere.	<list>	Bidirectional
deque	deque is the double ended queue that allows the insertion and deletion from both the ends.	<deque>	Random access
set	set is an associate container for storing unique sets.	<set>	Bidirectional
multiset	Multiset is an associate container for storing non- unique sets.	<set>	Bidirectional
map	Map is an associate container for storing unique key-value pairs, i.e. each key is associated with only one value(one to one mapping).	<map>	Bidirectional
multimap	multimap is an associate container for storing key- value pair, and each key can be associated with more than one value.	<map>	Bidirectional
stack	It follows last in first out(LIFO).	<stack>	No iterator
queue	It follows first in first out(FIFO).	<queue>	No iterator
Priority-queue	First element out is always the highest priority element.	<queue>	No iterator

Containers


Container
Sequence container
vector
deque
list
Associative container
set
multiset
map
multimap
Derived containers
stack
queue
priority_queue

C++ Vector (STL)

- In C++, vectors are used to store elements of similar data types. However, unlike arrays, the size of a vector can grow dynamically.

- Vectors are part of the C++ Standard Template Library. To use vectors, we need to include the vector header file

- #include<vector>

Vector Declaration

std::vector<T> vector_name;

Vector<int> num;

vector<int> vector1 = {1, 2, 3, 4, 5};
Vector<int> vector2{1,2,3,4,5,6}

vector<int> vector3(5, 12);

Lists

- List is a contiguous container while vector is a non-contiguous container i.e list stores the elements on a contiguous memory and vector stores on a non-contiguous memory.
- Insertion and deletion in the middle of the vector is very costly as it takes lot of time in shifting all the elements. Linklist overcome this problem and it is implemented using list container.
- List supports a bidirectional and provides an efficient way for insertion and deletion operations.
- Traversal is slow in list as list elements are accessed sequentially while vector supports a random access.

List with Templates

```
#include<iostream>
#include<list>
using namespace std;
int main()
{
   list<int> l;
}
```

List – Intialization

```
#include<iostream>
#include<list>
using namespace std;
int main()
{
   list<int> l{1,2,3,4};
}
```

C++ List insert()

- C++ List insert() function inserts a new element just before the specified position.
- It increases the size of the list container by the number of elements added in the list.
- In this example, iterator points to the first element of the list. Therefore, 5 is inserted before the first element of the list using insert() function.

```
#include<list>
using namespace std;
int main()
{
  list<int> li={1,2,3,4};
  list<int>::iterator itr=li.begin();
  li.insert(itr,5);
  for(itr=li.begin();itr!=li.end();++itr)
  cout<<*itr;
  return 0;
}
```

Insert in the front

- In this example, insert() function inserts the string "java" 2 times before the first element of the list.

```
#include <iostream>
#include<list>
using namespace std;
int main()
{
   list<string> li={"C is a language"};
   list<string>::iterator itr=li.begin();
   li.insert(itr,2,"java ");
   for(itr=li.begin();itr!=li.end();++itr)
   cout<<*itr;
    return 0;
}
```

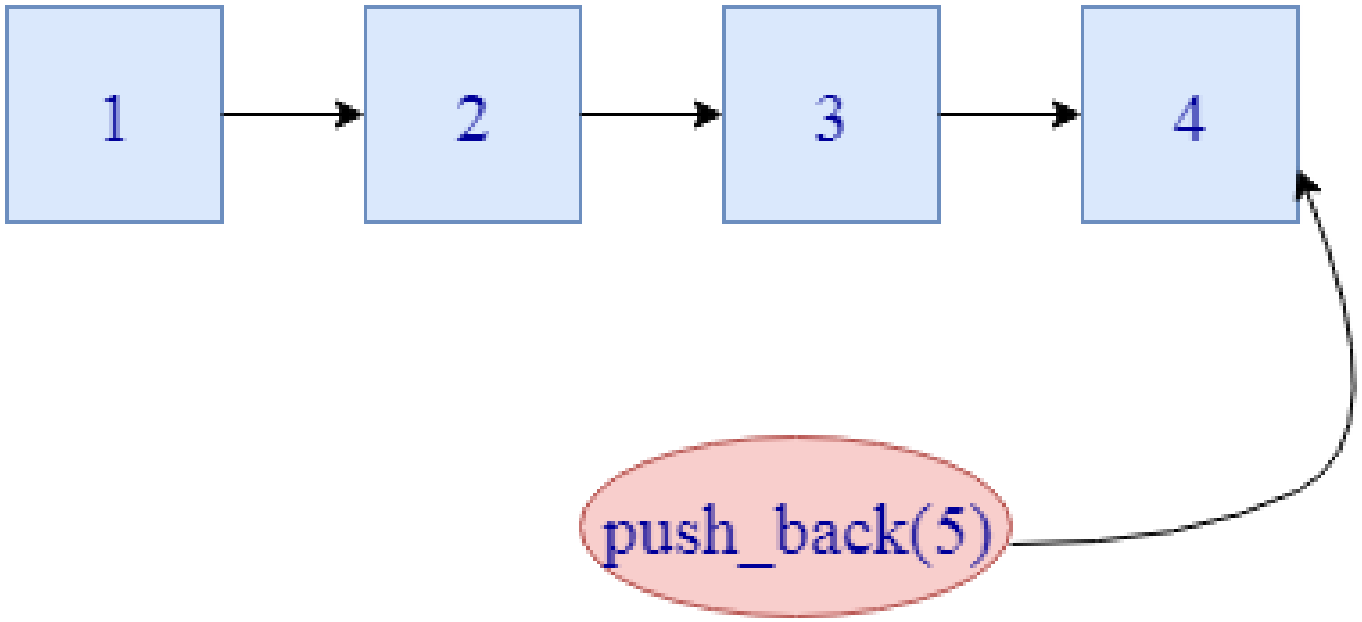
Insert in the middle

- In this example, range(first, last) of list li1 is given. Therefore, the insert() function inserts the elements between this range in the list li.

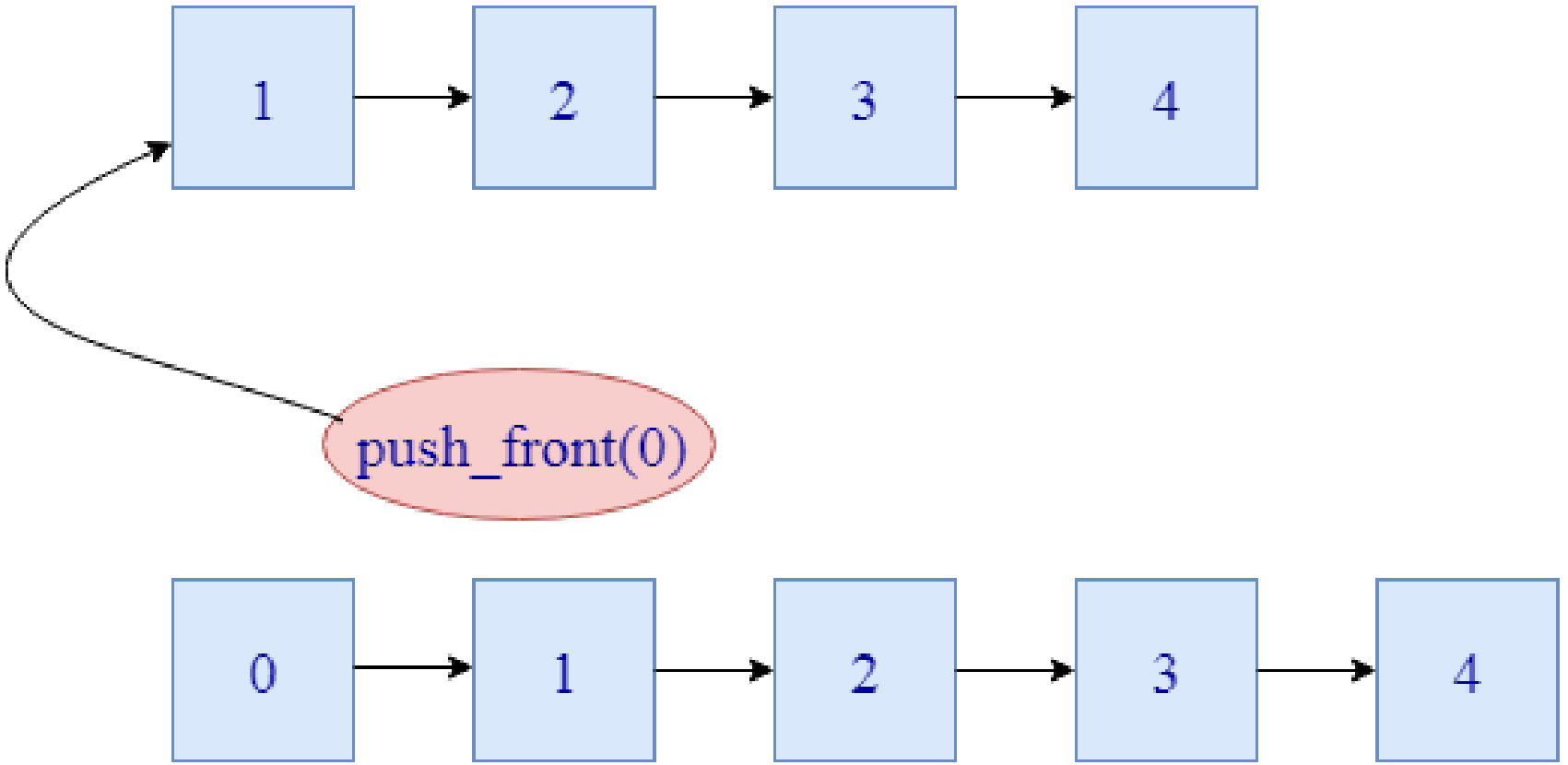
```
#include <iostream>
#include<list>
using namespace std;

int main()
{
  list<int> li={1,2,3,4,5};
  list<int> li1={6,7,8,9};
  list<int>::iterator itr=li.begin();
  li.insert(itr,li1.begin(),li1.end());
  for(itr=li.begin();itr!=li.end();++itr)
{
  cout<<*itr;
  cout<<? ?;
          }
        return 0;
}
```

Push_back

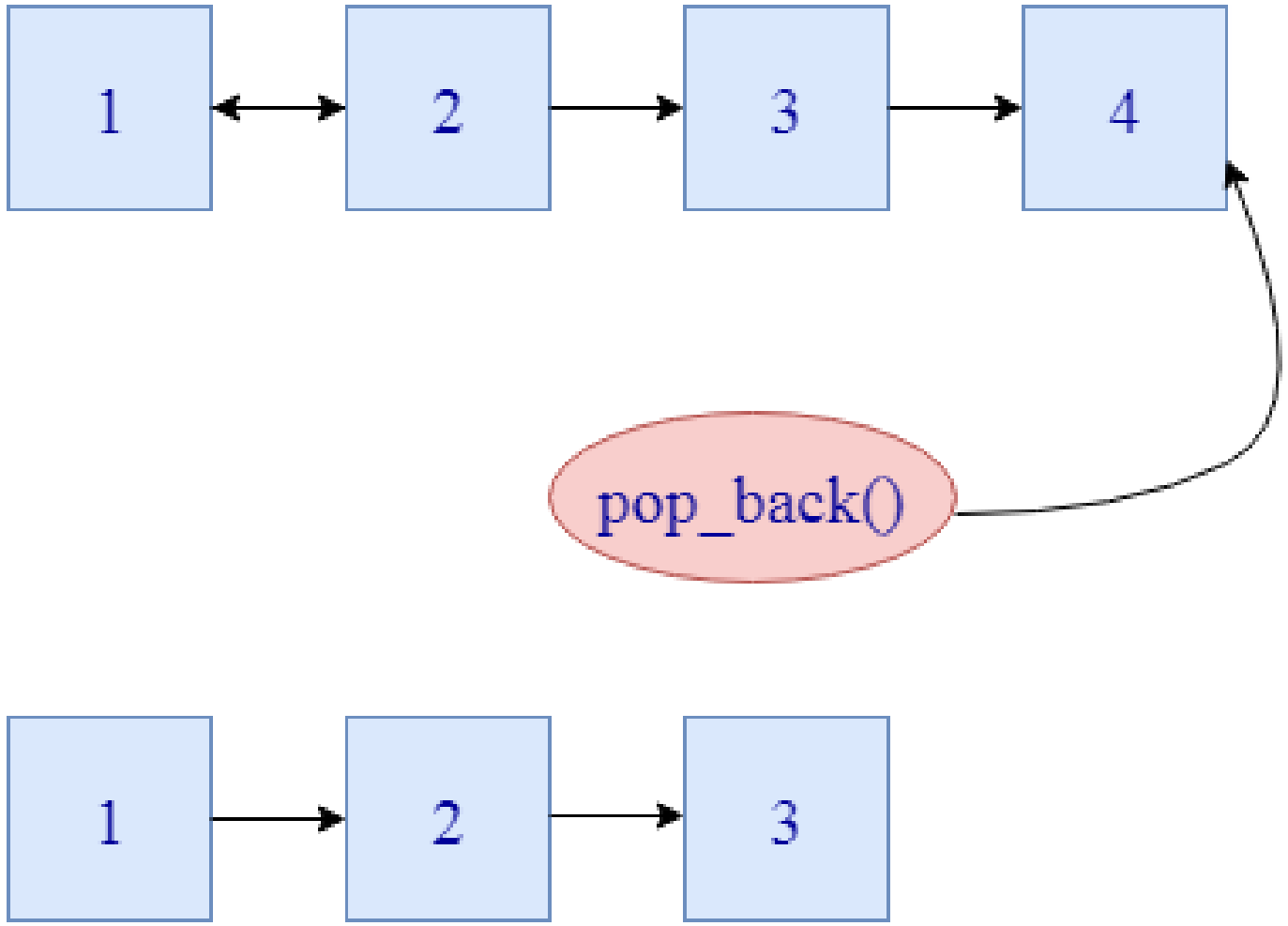
- push_back(**const** value_type& x);



Push_front



Pop_back



Sort()

C++ List sort() function arranges the elements of a given list in an increasing order. It does not involve in any construction and destruction of elements. Elements are only moved within the container.

- **void** sort();