Introduction to Templates

- **Templates in C++ is defined as a blueprint or formula for creating a generic class or a function.**

- **We can create a single function or single class to work with different data types using templates.**

- **It is also known as generic functions or classes.**

How Templates work ?

- **Templates in C++ works in such a way that it gets expanded at compiler time, just like macros and allows a function or class to work on different data types without being rewritten.**

Macros: Macros are a piece of code in a program which is given some name. Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code. The **'#define'** directive is used to define a macro.

For Example :

```
#define LIMIT 5
```

How Templates work ?

```
template <typename T>
T myMax(T x, T y)
{
    return (x > y)? x: y;
}

int main()
{
   cout << myMax<int>(3, 7) << endl;
   cout << myMax<char>('g', 'e') << endl;
   return 0;
}
```

Compiler internally generates and adds below code

```
int myMax(int x, int y)
{
    return (x > y)? x: y;
}
```

Compiler internally generates and adds below code.

```
char myMax(char x, char y)
{
    return (x > y)? x: y;
}
```

Types of Templates

There are two types of templates in C++

- ✓ **Class templates**
- ✓ **Function template**

Class Templates

Class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, Binary Tree, Stack, Queue, Array, etc.

Syntax of Class Template:-

```
template<class Type>
class class_name
{
  //class body;
}
```

- **Here Type is a placeholder type name, which will be specified when a class instantiated.**

Function Templates

Function template in C++ is a single function template that works with multiple data types simultaneously, but a standard function works only with one set of data types.

Syntax of Functions Template:-

```
template<class type>
type functionName(parameter list)
{
  //body of the function
}
```

- **Here Type is a placeholder type name, which will be specified when a class instantiated.**

Multiple and Default Templates

Syntax of Functions Template:-

```
template<class T=float,class F=int>
```

STL (Standard Template Library)

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 OF STL :-

- ✓ Containers
- ✓ Iterators
- ✓ Algorithms
- ✓ Function Operator

Containers

Containers can be described as the objects that hold the data of the same type. Containers are used to implement different data structures for example arrays, list, trees, etc.

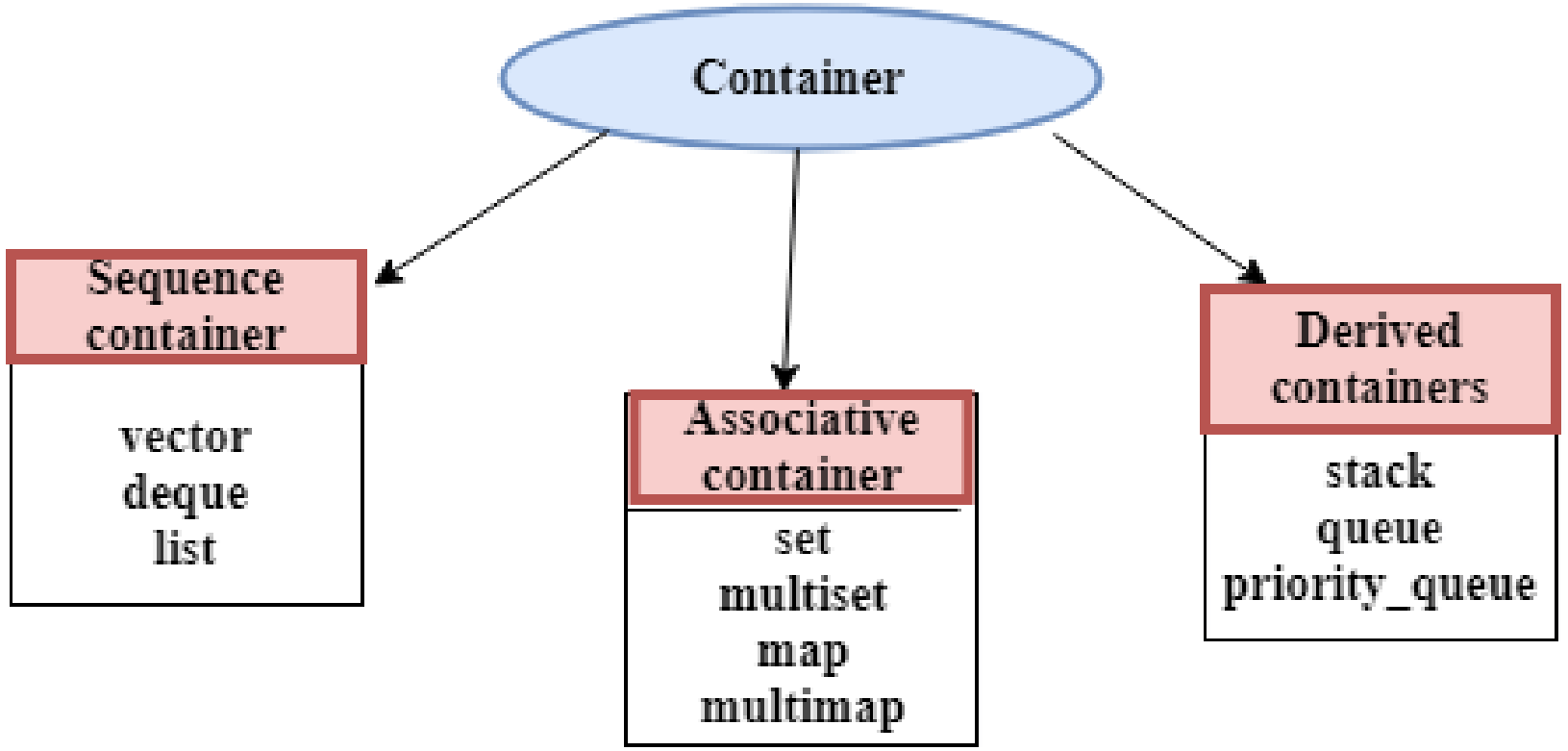
List of Few Containers :-

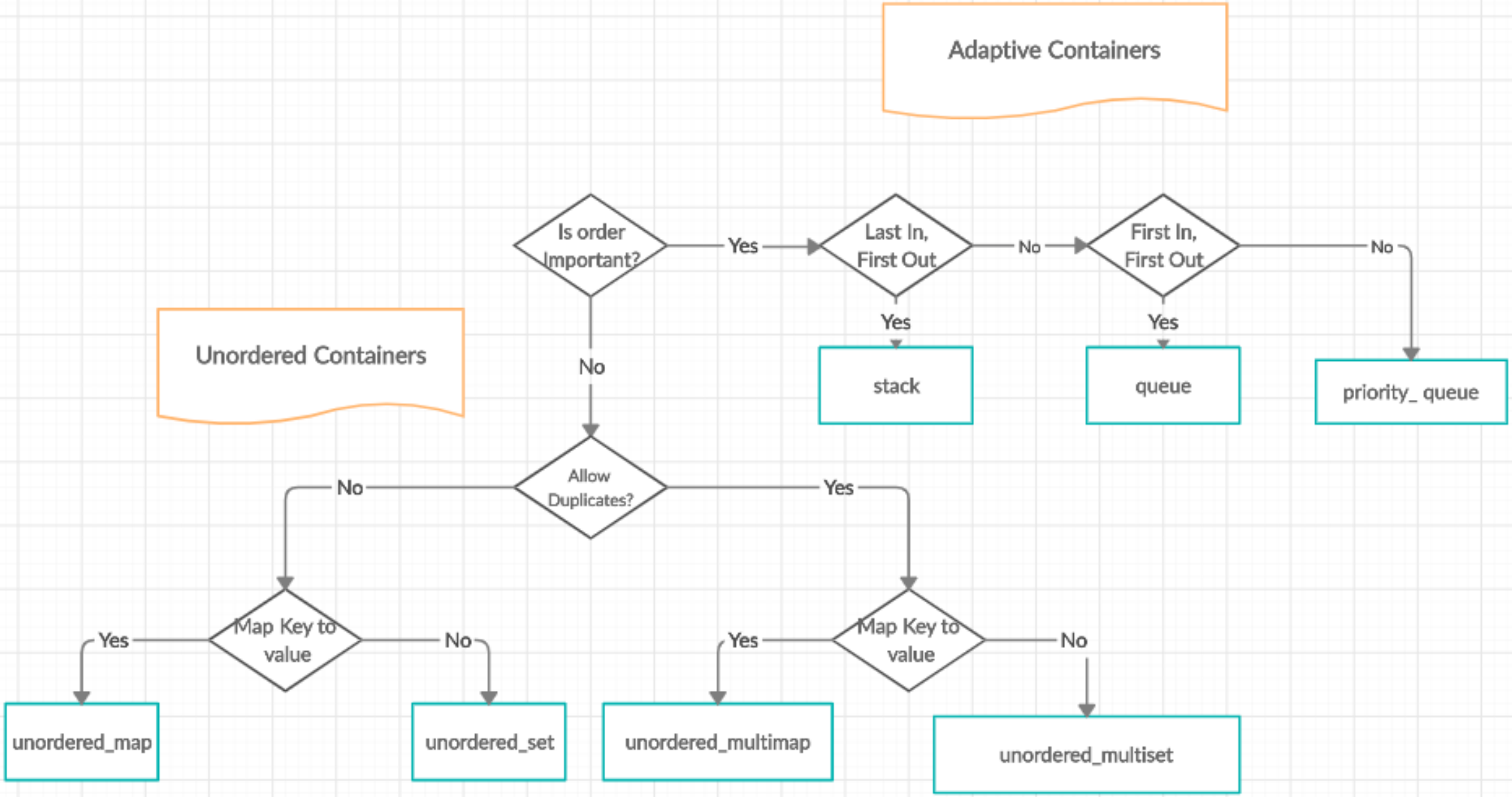
- ✓ **Vector**
- ✓ **List**
- ✓ **Set**
- ✓ **Multiset**
- ✓ **Map**
- ✓ **Multimap**
- ✓ **Stack**
- ✓ **Queue**
- ✓ **Priority Queue**

Classification of Containers

- ❑ Sequence containers
- ❑ Associative containers
- ❑ Derived containers

✓ Each container class contains a set of functions that can be used to manipulate the contents.

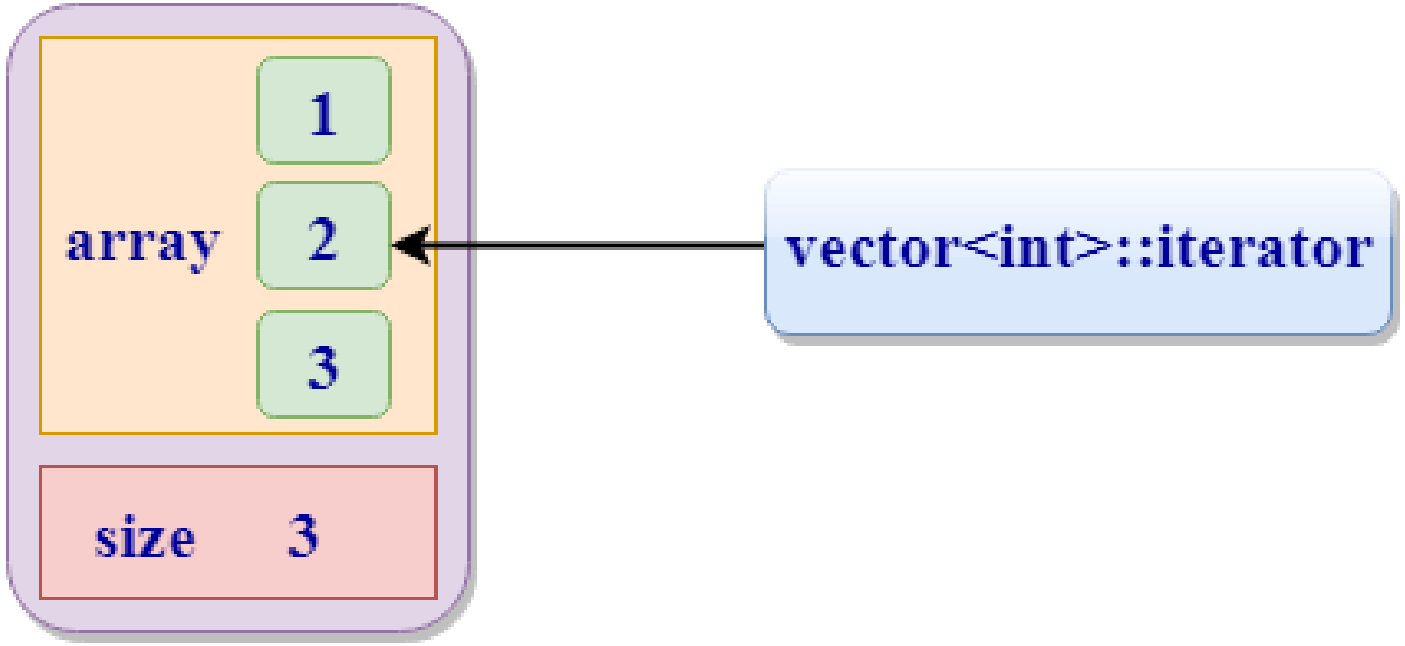



Adaptive Containers
Is order Important?
Yes
Last In, First Out
No
First In, First Out
No
Yes
Yes
stack
queue
priority_ queue
Unordered Containers
No
Allow Duplicates?
No
Yes
Map Key to value
Map Key to value
Yes
No
Yes
No
unordered_map
unordered_set
unordered_multimap
unordered_multiset

Iterator

- Iterators are pointer-like entities used to access the individual elements in a container.

- Iterators are moved sequentially from one element to another element. This process is known as iterating through a container.



Algorithms

Algorithms are the functions used across a variety of containers for processing its contents.

- ✓ Algorithms are not the member functions of a container, but they are the standalone template functions.
- ✓ Algorithms save a lot of time and effort.
- ✓ If we want to access the STL algorithms, we must include the <algorithm> header file in our program.

Function Objects

The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with the help of parameters to be passed.

- ✓ **The C++ Standard Library uses function objects primarily as sorting criteria for containers and in algorithms.**

- ✓ **Function objects provide two main advantages over a straight function call. The first is that a function object can contain state. The second is that a function object is a type and therefore can be used as a template parameter.**

Array class in C++

The introduction of array class from C++11 has offered a better alternative for C-style arrays. The advantages of array class over C-style array are :-

- Array classes knows its own size, whereas C-style arrays lack this property. So when passing to functions, we don't need to pass size of Array as a separate parameter.

- Array classes are generally more efficient, light-weight and reliable than C-style arrays.

Array class in C++

Operations on array :-

1. at() :- This function is used to access the elements of array.

2. get() :- This function is also used to access the elements of array. This function is not the member of array class but overloaded function from class tuple.

3. operator[] :- This is similar to C-style arrays. This method is also used to access array elements.

4. front() :- This returns the first element of array.

5. back() :- This returns the last element of array.

6. size() :- It returns the number of elements in array.

7. max_size() :- It returns the maximum number of elements array can hold i.e, the size with which array is declared. The size() and max_size() return the same value.

8. swap() :- The swap() swaps all elements of one array with other.

9. empty() :- This function returns true when the array size is zero else returns false.

10. fill() :- This function is used to fill the entire array with a particular value.

Vector Container

Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container.

Vector Container

1.begin() – Returns an iterator pointing to the first element in the vector
2.end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector
3.rbegin() – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
4.rend() – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)