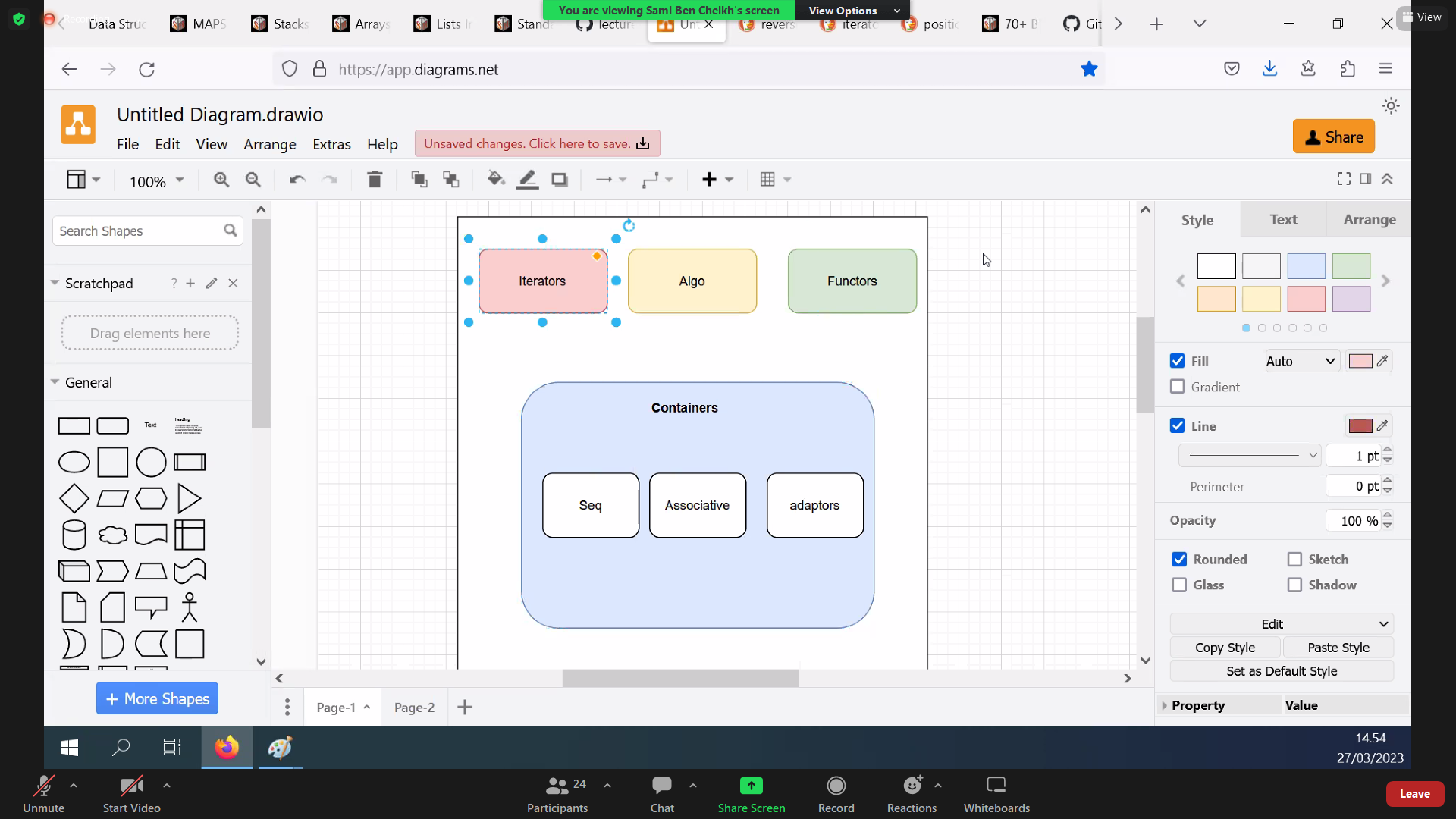
<https://github.com/TT00FE39-3001/lecture8>



<https://ojp.metropolia.fi/lomakkeet/1/lomake.html?code=VFQwMEZFMzktMzAwMQ==>

# Outline

## Topics

* [Review](https://github.com/TT00FE39-3001/lecture8/blob/main/review.md)
* Standard Template Library (STL)
* Asymptotic Notations
* Course Feedback

## This Week in Points

* Group Activities (Max 15 points)
* Peer reviews (Max 10 points)

## Part 1

* Function templates
* std::vector
* [Activity 1](https://github.com/TT00FE39-3001/lecture8/blob/main/activity1)

## Part 2

* Containers
* Iterators
* C++ STL vs Java Collections Framework
* [Activity 2](https://github.com/TT00FE39-3001/lecture8/blob/main/activity2)

## Part 3

* STL algorithms & [Functors](https://www.go4expert.com/articles/cpp-stl-functors-t34696/)
* �, �, Ω, �, �
* Where to go from here?
* [Activity 3](https://github.com/TT00FE39-3001/lecture8/blob/main/activity3)

## Links

* [#](https://github.com/TT00FE39-3001/lecture8/blob/main/links.md)

links

# Links

* MIT 2011: Introduction to Algorithms <https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-fall-2011/pages/syllabus/> <https://www.youtube.com/watch?v=HtSuA80QTyo&list=PLUl4u3cNGP61Oq3tWYp6V_F-5jb5L2iHb>
* MIT 2020: Introduction to Algorithms <https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-spring-2020/video_galleries/lecture-videos/> <https://www.youtube.com/playlist?list=PLUl4u3cNGP63EdVPNLG3ToM6LaEUuStEY>
* Standford: [Algorithms Specialization](https://www.coursera.org/specializations/algorithms)
* Princeton: [Algorithms, Part I](https://www.coursera.org/learn/algorithms-part1), [Algorithms, Part II](https://www.coursera.org/learn/algorithms-part2)
* Misc
  + <https://www.enjoyalgorithms.com/data-structures-and-algorithms-course/>
  + <https://kalkicode.com/>

**Review**

**Classification of Algorithm techniques**

* Brute Force
  + Linear search
  + Bubble sort
  + Selection sort
* Decrease and conquer
  + Binary Search
  + Insertion sort
* Divide-and-Conquer
  + Quick Sort
  + Merge Sort
* Dynamic Programming
  + Bottom Up: Tabulation
  + Top Down: Memoization
* Greedy Method

**Data Structures & Abstract data types (ADT)**

* Data Structures (Physical)
  + Arrays
  + Linked Lists
* ADT (Logical)
  + Linear
    - Queues
    - Stacks
    - Hash Tables (linear/non-linear)
  + Non Linear
    - Binary Tree
    - Binary Search Tree
    - Heaps
    - Graphs

**Analysis of Algorithm Efficiency**

* Big O Complexity
* Average case vs worst case
* Space vs Time
* Little-O, Theta, Little Omega, Big Omega

**Standard Template Library (STL)**

* Containers
* Algorithms
* Iterators
* Functors

**Misc**

* FIFO vs LIFO
* Recursion vs Iteration
  + The Top-Down Thought Process
* Logarithms vs Exponential

ACTIVITY 1

**# Activities**

---

> [Course Feedback](https://ojp.metropolia.fi/lomakkeet/1/lomake.html?code=VFQwMEZFMzktMzAwMQ==)

---

**## Task 1**

- What is the difference between function overloading and function templates. You can refer to the programs in the `./src/` folder as well as [Links 2 and 3](#links) below.

- Rewrite the following program using templates

```cpp

#include <iostream>

int add(int x, int y)

{

    return x + y;

}

double add(double x, double y)

{

    return x + y;

}

int main()

{

    std::cout << add(1, 2); // calls add(int, int)

    std::cout << '\n';

    std::cout << add(1.2, 3.4); // calls add(double, double)

    return 0;

}

```

**## Task 2**

Refer to the following link:

<https://www.softwaretestinghelp.com/vectors-in-stl/>

Discuss the difference between

- Size() and capacity()

- begin() and cbegin()

- end() and cend()

**## Links**

1. <https://cpp.sh/>

2. <https://www.learncpp.com/cpp-tutorial/function-templates/>

3. <https://www.learncpp.com/cpp-tutorial/function-overload-differentiation/>

4. <https://www.geeksforgeeks.org/design-and-analysis-of-algorithms/>

ANSWERS:

**## Task 1**

- What is the difference between function overloading and function templates. You can refer to the programs in the `./src/` folder as well as [Links 2 and 3](#links) below.

- Rewrite the following program using templates

Both function overloading and templates are examples of polymorphism.Function overloading is used when multiple functions do similar operations; templates are used when multiple functions do identical operations.Templates provide an advantage when you want to perform the same action on types that can be different.

#include <iostream>

int add(int x, int y)

{

    return x + y;

}

double add(double x, double y)

{

    return x + y;

}

int main()

{

    std::cout << add(1, 2); // calls add(int, int)

    std::cout << '\n';

    std::cout << add(1.2, 3.4); // calls add(double, double)

    return 0;

}

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture8-main\activity1\src> .\overloading2

3

4.6

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture8-main\activity1\src> .\template2

3

4.6

#include <iostream>

template <typename T> // this is the template parameter declaration

T add(T x, T y)       // this is the function template definition for add<T>

{

    return x + y;

}

int main()

{

    std::cout << add(1, 2); // calls add(int, int)

    std::cout << '\n';

    std::cout << add(1.2, 3.4); // calls add(double, double)

    return 0;

}

**## Task 2**

Refer to the following link:

<https://www.softwaretestinghelp.com/vectors-in-stl/>

Discuss the difference between

- Size() and capacity()

- begin() and cbegin()

- end() and cend()

Dynamic array will expand on its own as the need arises. STL provides this dynamic array in the form of a vector container.

**Declaring A Vector In C++ With std:: Vector Class**

In STL vector class ‘**std::vector**’ is defined under the header **<vector>**. Thus, in order to use a vector container, we should include this header in our program as shown below:

**#include <vector>**

**We can declare an empty vector as shown below:**

**std::vector<int> myvec;**

The above line of code will create a vector with the elements of the type integer. In memory, this will be laid out as myvec.

**Initialize Vector**

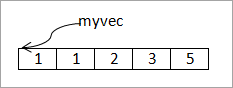
We can initialize a vector with a value at the time of declaring it.

**This is done as follows:**

|  |
| --- |
| #include<vector>  **int** main()  {                   std::vector<**int**> myvec = {1, 1, 2, 3, 5};  } |

In the above code, we declare a vector of type int named myvec containing the first five elements in the Fibonacci sequence.

**The memory layout of this vector will be as below:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/memory-layout.png)

Discuss the difference between

- Size() and capacity()

- begin() and cbegin()

- end() and cend()

**(i) Size of the vector**

The function size() returns the number of elements in the vector container. This is the in-built function of std::vector class and can be used directly to find the size of the vector.

**capacity():** Returns the size of storage space currently allocated. This is returned in terms of the number of elements

#include <iostream>

#include <vector>

using namespace std;

int main()

{

   vector<int> myvec = {1, 1, 2, 3, 5, 8};

   cout << "Vector Size : " << myvec.size();

   cout << " Vector Capacity : " << myvec.capacity();

   return 0;

}

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture8-main\activity1\src> .\vector\_size\_capacity

Vector Size : 6 Vector Capacity : 6

**std::[vector](https://cplusplus.com/reference/vector/vector/)::capacity**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

size\_type capacity() const;

**Return size of allocated storage capacity**

Returns the size of the storage space currently allocated for the [vector](https://cplusplus.com/vector), expressed in terms of elements.  
  
This *capacity* is not necessarily equal to the [vector size](https://cplusplus.com/vector::size). It can be equal or greater, with the extra space allowing to accommodate for growth without the need to reallocate on each insertion.  
  
Notice that this *capacity* does not suppose a limit on the size of the [vector](https://cplusplus.com/vector). When this *capacity* is exhausted and more is needed, it is automatically expanded by the container (reallocating it storage space). The theoretical limit on the [size](https://cplusplus.com/vector::size) of a [vector](https://cplusplus.com/vector) is given by member [max\_size](https://cplusplus.com/vector::max_size).  
  
The *capacity* of a [vector](https://cplusplus.com/vector) can be explicitly altered by calling member [vector::reserve](https://cplusplus.com/vector::reserve).

* **begin():**Returns iterator pointed to the first element of the vector container.
* **cbegin():**Returns a constant iterator pointing to the first element in the vector container.

**end():**Returns an iterator pointing to the element that follows the last element in the vector

* **cend():**Returns a constant iterator pointing to the element following the last element of the vector container.

#include <iostream>

#include <vector>

using namespace std;

int main()

   {

      vector<int> v1;

      for (int i = 1; i <= 5; i++)

         v1.push\_back(i+1);

         cout << "Output of Vector with begin and end: ";

      for (auto i = v1.begin(); i != v1.end(); ++i)

         cout << \*i << " ";

         cout << "\nOutput of Vector with rbegin and rend: ";

      for (auto itr = v1.rbegin(); itr != v1.rend(); ++itr)

         cout << \*itr << " ";

         cout << "\nOutput Vector of with cbegin and cend: ";

      for (auto itc = v1.cbegin(); itc != v1.cend(); ++itc)

         cout << \*itc << " ";

         cout << "\nOutput Vector of with crbegin and crend : ";

      for (auto icr = v1.crbegin(); icr != v1.crend(); ++icr)

         cout << \*icr << " ";

         return 0;

}

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture8-main\activity1\src> .\beginnings

Output of Vector with begin and end: 2 3 4 5 6

Output of Vector with rbegin and rend: 6 5 4 3 2

Output Vector of with cbegin and cend: 2 3 4 5 6

Output Vector of with crbegin and crend : 6 5 4 3 2

# std::[vector](https://cplusplus.com/reference/vector/vector/)::cbegin

const\_iterator cbegin() const noexcept;

**Return const\_iterator to beginning**

Returns a const\_iterator pointing to the first element in the container.  
  
A const\_iterator is an iterator that points to const content. This iterator can be increased and decreased (unless it is itself also const), just like the iterator returned by [vector::begin](https://cplusplus.com/vector::begin), but it cannot be used to modify the contents it points to, even if the [vector](https://cplusplus.com/vector) object is not itself const.  
  
If the container is [empty](https://cplusplus.com/vector::empty), the returned iterator value shall not be dereferenced.

ACTIVITY 2

**# Activities**

**## Task 1**

> Refer to the following links while discussing the answer.

- What is the difference between `array` and `std::array`

<https://stackoverflow.com/questions/30263303/stdarray-vs-array-performance>

- What is the difference between `std::array` and `std::vector`

<https://www.softwaretestinghelp.com/arrays-in-stl/>

- What is the difference between `std::list` and `std::vector`

<https://www.softwaretestinghelp.com/lists-in-stl/>

**## Task 2**

- Run the Stack and Queue examples in the following link

<https://www.softwaretestinghelp.com/stacks-and-queues-in-stl/>

> make sure you correct the syntax e.g `&lt;int&gt; becomes  <int>`

**## Task 3**

- Discuss the different types of iterators present in C++. You can refer to the following link

<https://www.geeksforgeeks.org/introduction-iterators-c/>

- What are the Benefits of Iterators

**## Links**

- <https://cpp.sh/>

ANSWERS:

- What is the difference between `array` and `std::array`

<https://stackoverflow.com/questions/30263303/stdarray-vs-array-performance>

std::array has value semantics while raw arrays do not. This means you can copy std::array and treat it like a primitive value. You can receive them by value or reference as function arguments and you can return them by value.

If you never copy a std::array, then there is no performance difference than a raw array. If you do need to make copies then std::array will do the right thing and should still give equal performance

If you can use std::array you should use it.

- What is the difference between `std::array` and `std::vector`

<https://www.softwaretestinghelp.com/arrays-in-stl/>

# Difference between std::vector and std::array in C++

[C++](https://www.tutorialspoint.com/articles/category/Cplusplus)[Server Side Programming](https://www.tutorialspoint.com/articles/category/Server-Side-Programming)[Programming](https://www.tutorialspoint.com/articles/category/Programming)

The following are the differences between vector and array −

* Vector is a sequential container to store elements and not index based.
* Array stores a fixed-size sequential collection of elements of the same type and it is index based.
* Vector is dynamic in nature so, size increases with insertion of elements.
* As array is fixed size, once initialized can’t be resized.
* Vector occupies more memory.
* Array is memory efficient data structure.
* Vector takes more time in accessing elements.
* Array access elements in constant time irrespective of their location as elements are arranged in a contiguous memory allocation.

Vectors and arrays can be declared with the following syntax −

Vector declaration:vector<datatype>array name;

Array declaration:type array\_name[array\_size];

Vector initialization:vector<datatype>array name={values};

Array initialization:datatype arrayname[arraysize] = {values};

#include <algorithm>

#include <array>

#include <iostream>

#include <iterator>

using namespace std;

int main() {

  array<int, 5> myarray = {1, 1, 2, 3, 5};

  cout << "Size of array: " << endl;

  cout << myarray.size() << endl;

  cout << "\nmyarray contents: ";

  for (auto i : myarray)

    cout << i << ' ';

  // sort operation

 sort(myarray.begin(), myarray.end());

   cout << "\nsorted myarray : ";

   for (auto i : myarray)

      cout << i << ' ';

   cout<<"\nFirst element of myarray "<<myarray.at(0);

   cout<<endl;

   cout<<"FRONT myarray: "<<myarray.front();

   cout<<endl;

   cout<<"BACK myarray: "<<myarray.back();

   cout<<endl;

   // Filling ar2 with 10

   myarray.fill(8);

  cout << "\nFilled myarray : ";

  for (auto i : myarray)

     cout << i << ' ';

return 0;

}

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture8-main\activity2\src> .\array

Size of array:

5

myarray contents: 1 1 2 3 5

sorted myarray : 1 1 2 3 5

First element of myarray 1

FRONT myarray: 1

BACK myarray: 5

Filled myarray : 8 8 8 8 8

## std::vector:

## Example Code

#include <iostream>

#include <vector>

using namespace std;

int main() {

   vector<vector<int>>v{ { 4, 5, 3 }, { 2, 7, 6 }, { 3, 2, 1 ,10 } };

   cout<<"the 2D vector is:"<<endl;

   for (int i = 0; i < v.size(); i++) {

      for (int j = 0; j < v[i].size(); j++)

         cout << v[i][j] << " ";

         cout << endl;

   }

   return 0;

}

## Output

the 2D vector is:

4 5 3

2 7 6

3 2 1 10

## std:: array:

## Example Code

#include<iostream>

#include<array>

using namespace std;

int main() {

   array<int,4>a = {10, 20, 30, 40};

   cout << "The size of array is : ";

   //size of the array using size()

   cout << a.size() << endl;

   //maximum no of elements of the array

   cout << "Maximum number of elements array can hold is : ";

   cout << a.max\_size() << endl;

   // Printing array elements using at()

   cout << "The array elements are (using at()) : ";

   for ( int i=0; i<4; i++)

      cout << a.at(i) << " ";

      cout << endl;

      // Filling array with 1

      a.fill(1);

      // Displaying array after filling

      cout << "Array after filling operation is : ";

   for ( int i=0; i<4; i++)

      cout << a[i] << " ";

      return 0;

}

## Output

The size of array is : 4

Maximum number of elements array can hold is : 4

The array elements are (using at()) : 10 20 30 40

Array after filling operation is : 1 1 1 1

Arrays are contiguous memory locations. Array container is a sequential homogenous container and is of a fixed size.

Actually, in programming, we rarely use such a static container as in real-time scenarios we need containers that can expand or shrink dynamically. Nevertheless, as an array is one of the basic containers, we will begin our discussions about STL containers with arrays.

**The general syntax of declaring an array container is:**

**array<object\_type, size> array\_name;**

The above declaration creates an array container ‘array\_name’ with size ‘size’ and with objects of type ‘object\_type’.

**We can also initialize this array container as shown below,**

**Array<int,5> myarray = {1,1,2,3,5};**

Array container supports various operations that can be carried out to facilitate efficient traversing and manipulation of array container elements.

**Some of the functions that are supported by the array container include:**

* **At:** Returns value in the array container at a given position. ‘Out\_of\_range’ exception is thrown if the position specified is beyond the array limits.
* **Front:** Returns the first element in the array container.
* **Back:** Returns the last element in the array container if the container is completely filled the other returns the rightmost element in the container.
* **Fill:** Assigns a given value to every element in the array container.
* **Swap:** Swaps contents of two arrays with the same type and same size index wise.
* **Empty:** Boolean function to check if an array container is empty or not.
* **Size:** Returns the number of elements in the array container.
* **Max\_size:** Returns the maximum size of the array container.
* **Begin:** Returns the iterator pointing to the beginning of the array container i.e. first element of the array.
* **End:** Returns the iterator pointing to the location next to the last element in the array container.

What is the difference between `std::list` and `std::vector`

<https://www.softwaretestinghelp.com/lists-in-stl/>

Lists are sequential containers. Lists contain elements in non-contiguous locations. We have discussed arrays and vectors in our previous tutorials.

In case of the array and vector containers, as these containers store data in contiguous memory, insert operation in the middle of these containers proves to be very costly as we have to shift the existing elements accordingly to make space for the new element.

### Overview

The list is a container that overcomes this drawback of the array and vector containers. It allows us to insert elements anywhere in the list without causing much of an overhead. But lists are slower than vectors as far as traversal is concerned.

In this tutorial, we will see the implementation of lists in STL along with the various operations of traversal, manipulations and accessing list with examples.

Note that a majority of list operations are similar to those of vectors and hence readers who have already read our tutorial on vectors will not have problems in interpreting list concepts.

### Declaration And Initialization

For implementing list container and using all its benefits, we need to include a header file <list> in our program.

**#include <list>**

**The general declaration for list container is**

**std::list<objectType> listName;**

**For Example, we can declare a list named ‘mylist’ of type int as follows:**

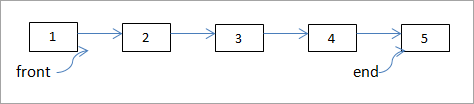
**std::list<int> mylist;**

We can also initialize list at the time of declaration or add elements to it using one of the operations it supports.

**Let’s see how we can initialize the list we created above.**

**std::list<int> mylist = {1, 1, 2, 3, 5};**

**The above initialization will be laid out in memory as shown below:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/Declaration-and-initialization1.png)

Once we initialize the list, we can access the elements of a list using an iterator. The Iterator functions ‘begin’ and ‘end’ help us to traverse through the list elements.

**Note:** Iterator for the list also supports other iterators like reverse iterators (rbegin, rend), constant iterators (cbegin, cend) and constant reverse iterators (crbegin, crend) and can be used in a similar way like vectors.

#include <iostream>

#include <list> // for list operations

using namespace std;

int main()

{

   list<int> mylist = {1,1,2};

   list<int>::iterator it = mylist.begin();

   // iterator to point to 4th position

   advance(it, 3);

   // inserts 3 at 4th position

   mylist.insert(it, 3);

   cout << "The list after inserting"

   << " 1 element using insert() is : ";

   for (list<int>::iterator i = mylist.begin();i != mylist.end();i++)

      cout << \*i << " ";

      cout << endl;

}

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture8-main\activity2\src> .\list

The list after inserting 1 element using insert() is : 1 1 2 3

**## Task 2**

- Run the Stack and Queue examples in the following link

<https://www.softwaretestinghelp.com/stacks-and-queues-in-stl/>

stack:

#include <iostream>

#include <stack>

using namespace std;

void printStack(stack <int> stk)

{

   while (!stk.empty())

      {

         cout << "\t" << stk.top();

         stk.pop();

      }

   cout << "\n";

}

int main ()

{

   stack <int> oddstk;

   oddstk.push(1);

   oddstk.push(3);

   oddstk.push(5);

   oddstk.push(7);

   oddstk.push(9);

   cout << "The stack is : ";

   printStack(oddstk);

   cout << "\nSize of stack: " << oddstk.size();

   cout << "\nTop of stack: " << oddstk.top();

   cout << "\noddstk.pop() : ";

   oddstk.pop();

   printStack(oddstk);

   cout << "\nAnother pop(): ";

   oddstk.pop();

   printStack(oddstk);

   return 0;

}

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture8-main\activity2\src> .\stack

The stack is : 9 7 5 3 1

Size of stack: 5

Top of stack: 9

oddstk.pop() : 7 5 3 1

Another pop(): 5 3 1

queue:

#include <iostream>

#include <queue>

using namespace std;

void printQueue(queue <int> myqueue)

{

   queue <int> secqueue = myqueue;

   while (!secqueue.empty())

   {

      cout << '\t' << secqueue.front();

      secqueue.pop();

   }

   cout << '\n';

}

int main()

   {

      queue <int> myqueue;

      myqueue.push(2);

      myqueue.push(4);

      myqueue.push(6);

      myqueue.push(8);

      cout << "The queue myqueue is : ";

      printQueue(myqueue);

      cout << "\nmyqueue.size() : " << myqueue.size();

      cout << "\nmyqueue.front() : " << myqueue.front();

      cout << "\nmyqueue.back() : " << myqueue.back();

      cout << "\nmyqueue.pop() : ";

      myqueue.pop();

      printQueue(myqueue);

      return 0;

   }

PS C:\Users\Seppo\Downloads\Metropolia\2023\Datastructures\_and\_algorithms\lecture8-main\activity2\src> .\queue

The queue myqueue is : 2 4 6 8

myqueue.size() : 4

myqueue.front() : 2

myqueue.back() : 8

myqueue.pop() : 4 6 8

ACTIVITY 3

**# Activities**

**## Task 1**

- What are the advantages of the C++ Standard Template Library (STL):

- What are the disadvantages of STL?

- What are the main components of STL?

> You can refer to the following link: <https://www.geeksforgeeks.org/the-c-standard-template-library-stl/>

**## Task 2**

- Refer to `./src/non-manipulative.cpp`

  - Discuss how the program works

  - What are the Non-Manipulating Algorithms used in the program?

- Refer to `./src/manipulative.cpp`

  - Discuss how the program works

  - What are the Manipulating Algorithms used in the program?

> You can refer to the following link: <https://www.geeksforgeeks.org/c-magicians-stl-algorithms/>

**## Task 3**

- Refer to the following article. Reflect on the difference between Big Oh and little oh

  https://www.baeldung.com/cs/big-o-vs-little-o-notation

**## Task 4**

Refer to one of the following articles. Reflect on the differences between Big Oh, Big Omega and Theta.

- <https://jarednielsen.com/big-o-omega-theta/>

- <https://www.codeandgadgets.com/big-oh-big-omega-and-theta-definitions/>

- <https://www.geeksforgeeks.org/difference-between-big-oh-big-omega-and-big-theta/>