Practical Lecture : STL Day 1

Quick Recap

Let’s take a quick recap of previous lecture –

- Introduction to templates
- Function template
- class template

Today’s Agenda

Today we are going to cover –

- Introduction to STL
- Containers
- Algorithms and iterators
- Container - Vector and List.

Let’s Get Started-

Introduction

We have already understood the concept of C++ Template

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.

It is a generalized library and so, its components are parameterized.

At the core of the C++ Standard Template Library are following three well-structured components −

1. Containers
2. Algorithms
3. Iterators

Learning STL is important for every C++ programmer as it saves a lot of time while writing code.

Components

All the three components have a rich set of pre-defined functions which help us in doing complicated tasks in very easy fashion.

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.

Algorithms
Algorithms act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.

Iterators
Iterators are used to step through the elements of collections of objects. These collections may be containers or subsets of containers.

Will discuss about each component in detail soon

Containers

Containers

Containers are library used to manage collections of classes and objects of a certain kind.

The containers are implemented as generic class templates.

Containers help us to implement and replicate simple and complex data structures very easily like arrays, lists, trees , stack, queues, etc.

For example you can very easily define a linked list in a single statement by using list container of container library in STL , saving your time and effort. It means a linked list template is already defined. You have to simply use it by creating objects from it and calling methods of it.

Containers can be used to hold different kind of objects. It means same container can be operated on any data types , you don’t have to define the same container for different type of elements.

Data structures

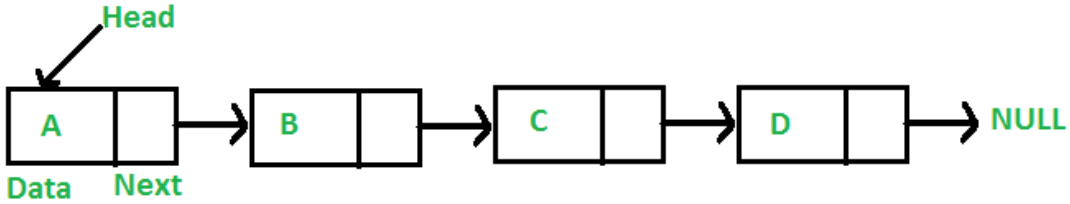
Array: is a linear collection of elements of similar data types. Operations possible on array : addition of elements. Addition can be done randomly.

Stack: collection of items arranged on top of each other in the form of pile where elements are inserted and extracted only from one end of the pile.. Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out) or FILO (First In Last Out)

Queue: A Queue is a linear structure which follows a particular order in which the operations are performed. specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the container and extracted from the other

Data structures

Linked list: A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.

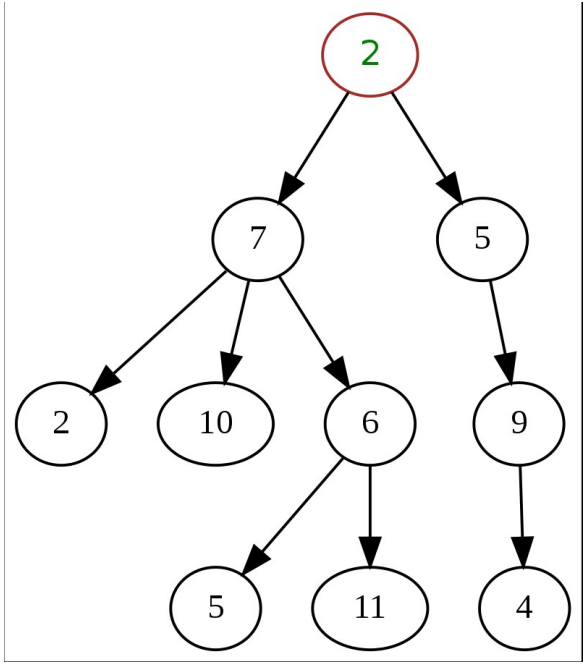


To create a linked list, following structure need to be created. Program will work around this structure.

```
// A linked list node
struct Node {
    int data;
    struct Node* next;
};
```

Data structures

Tree: A tree is a nonlinear data structure, compared to arrays, linked lists, stacks and queues which are linear data structures. It is a collection of nodes connected by directed (or undirected) edges.



Common containers

- vector: replicates arrays
- queue: replicates queue
- stack: replicates stack
- list: replicates linked list
- set: replicates trees
- maps: associative arrays
- And many more…

How to use container library

When we use container library, then we have to include that header file first and use the constructor to initialize the object.

Eg. While using list container, include container list and create object as follows:

```
#include <iostream>
#include <list>

int main()
{
        list <int> mylist;
        list <double> mylist1;
        …..
        …..
}
```

How to use container library

In the above example shown, we don’t have to create list class. It already exists.
There are pre-defined containers in c++, which are list, vector, queues , stacks , etc.

Arrays in STL

SYNTAX of array container:

array<object_type, array_size> array_name;

The above code creates an empty array of **object_type** with maximum size of **array_size**.

e.g. array <int, 4> A;
- **A** is an **array** of **4 integers.**

However, if you want to create an array with elements in it, you can do so by simply using the = operator, here is an example :

e.g. array <int, 5> B ={11,22,33,44,55}

Practice question- array

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

int main()
{
        array <int, 4> A ={ 2,4,6,8};
}
```

You can compile and run above code. It will not have any output.

The above statement will create an array with 2,4,6,8 as data in the array.

Member function of array template

Following are the important and most used member functions of array template.

at function

This method returns value in the array at the given range. If the given range is greater than the array size, out_of_range exception is thrown. Here is a code snippet explaining the use of this operator :

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

int main() {
   array <int ,4> a={1,2,3,4};
   cout << a.at(2) ;    // prints 3
   cout << a.at(3);    //prints 4
}
```

Practice question

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

int main() {
    array <int ,4> a={1,2,3,4};
    cout << a.at(2) ;    // prints 3
    cout<< a[1];    //prints 2
    //cout << a.at(4); //This line when uncommented throws out of range exception
}
```

Note: There are many more functions available in array. However we will restrict ourselves here as we just wanted to learn basic syntax of container class use. We do not have array in syllabus

Points to remember

While using container class template, please remember following points

1. Include respective header file
   e.g #include< array>

2. Use the template class name to create objects from it.
   e.g. array<int, 5> objA;

Containers in STL

We will be studying only two containers in STL

1. Vector
2. Lists

Vector container in STL

- Earlier we have learnt array container in STL (or in general) using
  Array<int, 5> A;
- This array can contain 5 elements in an array named A. It is fixed size array.

- Drawbacks of array:
  -The size of an array is fixed.
  -User must know number of elements to be stored beforehand declaring an array.
  - Defining oversized array is a wastage of memory (to store 10 elements, we are declaring array of size)

- So we need solution which will allow us flexibility to add elements into an array as and when required at runtime

- Solution to this is Vector container in STL

Vector container in STL

As we have identifie solution of the fixed size or static size arrays problem is dynamic arrays!

They have dynamic size, i.e. their size can change during runtime.

Container library provides vectors to replicate dynamic arrays.

SYNTAX for creating a vector is:

```
vector< object_type > vector_name;
```

Vector Container

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

int main() {
   vector <int> v;
}
```

Vector being a dynamic array, doesn't needs size during declaration, hence the above code will create a blank vector. Initially the vector is blank, as it has no data. but as you add data, it grows.

```
Vector <char> V1(5);

//it will create a vector V1 of size 5 initially which can grow dynamically
```

Vector Container

1. There are many ways to initialize a vector

```
#include <iostream>
#include <vector>
using namespace std;
 int main() {
    vector<string> v {“c++" ,“STL" ,“looks" ,“great"};}
}
```

C++	STL	looks	great

2. You can also initialize a vector with one element a certain number of times

```
vector<string> v(4 , “Test");
```

Test	Test	Test	Test

However this is not the end of the vector, still more elements can always be added at the end.

Member functions of vector

push_back function:

push_back() is used for inserting an element at the end of the vector. If the type of object passed as parameter in the push_back() is not same as that of the vector or is not interconvertible an exception is thrown.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   vector<int>  v; //will create a blank vector
   v.push_back(1);  //insert 1 at the back of v
   v.push_back(2);  //insert 2 at the back of v
   v.push_back(3);  //insert 3 at the back of v
}
```

Member functions of vector

Subscript Operator []

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   vector<int>  v; //will create a blank vector
   v.push_back(1);  //insert 1 at the back of v
   v.push_back(2);  //insert 2 at the back of v
   v.push_back(3);  //insert 3 at the back of v
   cout<<v[0];    //prints 1
   cout<<v[1];    //prints 2
   cout<<v[2];  ///prints 3
}
```

Member functions of vector

Subscript Operator []

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   vector<int>  v; //will create a blank vector
   v.push_back(1);  //insert 1 at the back of v
   v.push_back(2);  //insert 2 at the back of v
   v.push_back(3);  //insert 3 at the back of v
   for (int i=0;i<3;i++)
      {
         cout<<v[i];
      }
}
```

Member functions of vector

- **size function** : This method returns the size of the vector.
- **empty function** :This method returns true if the vector is empty else returns false.
- **at function** : This method works same in case of vector as it works for array. vector_name.at(i) returns the element at **ith** index in the vector **vector_name**.
- **front and back functions** :vector_name.front() retuns the element at the front of the vector (i.e. leftmost element). While vector_name.back() returns the element at the back of the vector (i.e. rightmost element).
- **clear** function:This method clears the whole vector, removes all the elements from the vector but do not delete the vector. SYNTAX: clear() . For a vector **v**, v.clear() will clear it, but not delete it.
- **capacity() function:** This method returns the number of elements that can be inserted in the vector based on the memory allocated to the vector.

Member functions of vector

```
int main()
   {
      vector<int>  v; //will create a blank vector
      cout<<"current capacity =" <<v.capacity()<<endl;
      for(int i=0;i<=9;i++)
      {
         v.push_back(10*(i+1));  //insert 10,20,30, upto 100 etc at the back of v
         cout<<"current capacity =" <<v.capacity()<<endl;
      }
      cout<<" Front element in vector " <<v.front()<<endl;
      cout<<" Back element in vector " <<v.back()<<endl;
```

Member functions of vector

```
for (int i=0;i<v.size();i++)
        {
            cout<<v.at(i) <<" ";
        }
        v.clear();
        cout<<" \n size of vector"<<v.size()<<endl;
        if(v.empty())
            cout<<" Vector is empty " <<endl;
        cout<< " capacity of vector"<<v.capacity();
}
```

Member functions of vector

Output:
current capacity =0
current capacity =1
current capacity =2
current capacity =4
current capacity =4
current capacity =8
current capacity =8
current capacity =8
current capacity =8
current capacity =16
current capacity =16

Front element in vector 10
Back element in vector 100
10 20 30 40 50 60 70 80 90 100
size of vector 0
Vector is empty
capacity of vector 16

Size and capacity difference

Capacity and size are different functions. Size returns current number of elements where capacity function returns the size of the storage space currently allocated for the vector, expressed in terms of elements.

The vector::capacity() function is a built-in function which returns the size of the storage space currently allocated for the vector, expressed in terms of elements.

This capacity is not necessarily equal to the vector size.

It can be equal to or greater, with the extra space allowing to accommodate for growth without the need to reallocate on each insertion.

The capacity does not suppose a limit on the size of the vector.

Size and capacity difference

```
using namespace std;

int main()
{
    vector<int> v;

    // inserts elements
    for (int i = 0; i < 10; i++) {
        v.push_back(i * 10);
    }

    cout << "The size of vector is " << v.size();
    cout << "\nThe maximum capacity is " << v.capacity();
    return 0;
}
```

Size and capacity difference

Current size = 1 Current capacity allocated = 1
Current size = 2 Current capacity allocated = 2
Current size = 3 Current capacity allocated = 4
Current size = 4 Current capacity allocated = 4
Current size = 5 Current capacity allocated = 8
Current size = 6 Current capacity allocated = 8
Current size = 7 Current capacity allocated = 8
Current size = 8 Current capacity allocated = 8
Current size = 9 Current capacity allocated = 16
Current size = 10 Current capacity allocated = 16
The size of vector is 10
The maximum capacity is 16

MCQ

What will the following line do?

vector <int> v3(5,10);

A. Create an integer vector v3 with 2 elements as 5,10.
B. Create an integer vector v3 of size 5 with every element value as 10
C. Compiler Reports an error as two values specified in vector size .
D. Compiler Reports an error as vector does not take size initially.

MCQ

What will the following line do?

vector <int> v3(5,10);

A. Create an integer vector v3 with 2 elements as 5,10.
B. Create an integer vector v3 of size 5 with every element value as 10
C. Compiler Reports an error as two values specified in vector size .
D. Compiler Reports an error as vector does not take size initially.

MCQ

Which of the following is/are component of STL

1. container
2. Algorithm
3. Iterators
4. Vector

A. 1,2,3
B. 1,3.4
C. All
D. 1,4

MCQ

Which of the following is/are component of STL

1. container
2. Algorithm
3. Iterators
4. Vector

A. 1,2,3
B. 1,3.4
C. All
D. 1,4

MCQ

Which of the following is not a function of Vector container in STL

1. at
2. empty
3. throw
4. size

MCQ

Which of the following is not a function of Vector container in STL

1. at
2. empty
3. **throw**
4. size

Assignment

Create a vector of 5 strings and perform following functions and observe the output

1. Pop_back()
2. Pop_front()
3. Front()
4. Back()
5. Size()
6. Capacity()
7. Push_front()
8. Push_back()
9. At
10. empty

QNA Time

Any Questions ??

Any Questions??

Thank You!

See you guys in next class.