06-Apr-20

STL (Standard Template library)

→ For using stack, add following line

```
#include <stack>
```

```
vector < vector < int >> wrong definition
vector < vector < int > > correct definition
                    ↑
                one blank
```

VECTOR :-

It is simplest STL container. vector is just an array with extended functionality. vector is actually array with additional features.

```
vector <int> V(10);
for (int i=0; i<10; i++)
{ V[i] = (i+1)*(i+1);
}
for (int i=9; i>0; i--)
{ V[i] -= V[i-1];
}
```

Actually, when you type:

```
vector <int> V;
```

Empty vector is created. Be careful with constructions like this

```
vector <int> V[10];
```

(not used; use V(10))

most important frequently used feature of vector is that it can report its size.

```
int elements_count = V.size();
```

C++ standard template Library (STL)

It is a library of container classes, algorithms, and iterators.

STL has four components :-

(i) Algorithms (ii) Containers (iii) Function (iv) iterators

Algorithms

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 containers.

Sort in C++ standard template library (STL)

There is a buildin function in C++ STL by name Sort().

Prototype for sort is :-

sort (startaddress, endaddress) → It uses Quicksort, Heapsort, insertionsort.

where, startaddress :- address of first element of array.

endaddress :- address of next contiguous memory location of last element of array.

So it actually sort in range of [startaddress, endaddress).

```
#include <iostream>
#include <algorithm>
using namespace std;
Void show (int a[])
{ for (int i=0 ; i<10; i++)
    cout << a[i] << " ";
}
int main ()
{ int a[10] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
  cout << "\n array before sorting is :";
  show (a) ;
  sort (a, a+10);
  cout << "\n\n array after sorting is :";
  show (a);
  return 0;
}
```

Output:-

1 5 8 9 6 7 3 4 2 0

0 1 2 3 4 5 6 7 8 9

Binary search in C++ Standard template library (STL)

it requires array to be sorted before search applied.
It works by dividing array in half until the elements found.

The prototype of binary search is :-

binary_search (Start address, end address, value to find)

- Start address: address of first element of array.
- end address: address of next contiguous memory of last element of array.
- value to find: target value which has to be found.
- Return: boolean value element present or not.

```
#include <iostream>
#include <algorithm>
using namespace std;
int main()
{ int a[] = {1,5,8,9,6,7,3,4,2,0};
  int asize = sizeof(a) / sizeof(a[0]);
  if (binary_search(a, a+asize, 2))
        cout << "\n Elements found in array";
  Else
     cout << "\n Elements not found in array";
  if (binary_search(a, a+asize, 10))
        cout << "\n Elements found in array";
  Else  cout << "\n Elements not found in array";
  return 0;
}
```

Output:-
Elements found in array
Elements not found in array

C++ magicians STL Algorithm

Non-manipulating algorithms :-

i) sort (first_iterator, last_iterator) :- to sort given vector
ii) reverse (first_iterator, last_iterator) :- reverse given vector
iii) *max_element (first_iterator, last_iterator) :- to find maxm element of
iv) *min_element (first_iterator, last_iterator) :- to find minm element of
v) accumulate (first_iterator, last_iterator, initial value of sum) :- Does the summation of the vector elements.

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <numeric> → for accumulate operation
using namespace std;
int main()
{ int arr[] = { 10, 20, 5, 23, 42, 15 };
  int n = sizeof(a) / sizeof(arr[0]);
  vector<int> vect(arr, arr+n);
  for(int i=0; i<n; i++)
    { cout << vect[i] << " "; }
  sort(vect.begin(), vect.end());
  for(int i=0; i<n; i++)
  { cout << vect[i] << " "; }
  reverse(vect.begin(), vect.end());
  for(int i=0; i<n; i++)
    { cout << v[i] << " "; }
  cout << *max_element(vect.begin(), vect.end());
  cout << *min_element(vect.begin(), vect.end());
  cout << accumulate(vect.begin(), vect.end(), 0);
  return 0;
}
```

vi) count(first_iterator, last_iterator, x) → to count occurances of x in vectors.

vii) find(first_iterator, last_iterator, x) → Point to address of x.

vect.begin() → points to first element.

vect.end() → points to last element.

Return address of first element if element not present in container which is not present in container

first occurance of

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int main()
```

```
{ int arr[] = { 10, 20, 5, 23, 42, 20, 15 };
  int n = sizeof(arr) / sizeof(arr[0]);
  vector<int> vect(arr, arr+n);
  cout << count(vect.begin(), vect.end(), 20);
find(vect.begin(), vect.end(), 5) != vect.end() {
        cout << "\n Element found" : cout << "\n Element not found";
  return 0;
} Output:- Occurance of 20 in vector : 2
                Element found.
```

viii) binary_search (first iterator, last iterator, x) :- Test whether x exists in sorted vector or not.

ix) lower_bound (first_iterator, last_iterator, x) :-

Manipulating algorithm :-

distance (first_iterator, desired_position) :- return distance of desired position from the first iterator. This function is very useful while finding the index.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{ int arr[] = { 5, 10, 15, 20, 20, 23, 42, 45 };
  int n = sizeof(arr) / sizeof(a[0]);
  vector<int> vect(arr, arr+n);
cout << distance(vect.begin(), max_element(vect.begin, vect.end()));
  return 0;
}
```

Output :- 7

Array algorithms in c++ STL (all_of, any_of, none_of, copy_n and iota)

i) all_of()

if checks for a given property on every elements and returns true when each elements in range satisfies property, specified else returns false.

```
#include <iostream>
#include <algorithm>
using namespace std;
int main()
{ int ar[6] = {1,2,3,4,5,-6};
//checking if all elements are positive
all_of(ar, ar+6, [](int x){ return x>0;}) ? cout<<"all positive" :
                                            cout<<"all not positive";

return 0;
}
```

ii) any_of()

This function checks for given range if there is even one elements satisfying a given property mentioned in function. Returns true if at least one element satisfies property else returns false.

```
#include <iostream>
#include <algorithm>
using namespace std;
int main()
{ int ar[6] = {1,2,3,4,5,-6};
// Checking if any element is negative
any_of(ar, ar+6, [](int x){ return x<0;}) ? cout<<"exist a negative element"
                                          : cout<<"all elements are non negative";

return 0;
}
```

iii) none_of ()

this function returns true if none of elements satisfies the given condition else return false.

iv) Copy_n ()

It copies one array elements to new array. This function takes 3 arguments, source array name, size of array and target array name.

```
#include <iostream>
#include <algorithm>
using namespace std;
int main ()
{ int ar[6] = {1, 2, 3, 4, 5, 6};
  int ar1[6];
  Copy_n (ar, 6, ar1);
}
```

v) iota ()

this function used to assigns continuous values to array.

It accepts 3 arguments, the first iterator, last iterator, starting numbe

```
#include <iostream>
#include <algorithm>
using namespace std;
int main ()
{ int ar[6] = {0};
  iota (ar, ar+6, 20);
  for (int i=0; i<6; i++)
  cout << ar[i] << " ";
  return 0;
}
```

Output:- 20, 21 22 23 24 25

Circular linked list connected to form

Partition in C++ STL

Partition refers to act of dividing elements of containers depending upon a given condition.

Partition operations:-

1. Partition (beg, end, condition) → used to partition element on basis of given condition
2. is_Partitioned (beg, end, condition) :- function returns boolean true if container is partitioned else return false.

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main ()
{ vector <int> vect = {2,1,5,6,8,7};
  // checking if vector is partitioned using is_partitioned ()
  is_partitioned (vect.begin (), vect.end (), [] (int x) { return x%2==0; })
      ? cout << "vector is partioned" : cout << "Not partitioned ";
  cout << endl;
  // partitioning vector using partition ()
  partition (vect.begin (), vect.end (), [] (int x) { return x%2==0; });
  is_partitioned (vect.begin (), vect.end (), [] (int x) { return x%2==0; }) {
      cout << "Now vector is partitioned" : cout << "Not partitioned" ;
  cout << endl;
  for (int i=0 ; i< vect.size () ; i++)
      cout << vect[i] << "  " ;
  return ;
}
```

Output:- Not partitioned
Now vector is partitioned
2 8 6 5 1 7

3. Stable_partition(beg, end, condition) :- partition on basis of Condition such that relative order of elements is preserved.

4. partition_point (beg, end, condition) :- function returns an iterator pointing to the partition point of container. i.e first element in partioned range [beg, end] for which condition is not true. Container should already be partitioned for this function to work.

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main()
{ vector<int> vect ={2,1,5,6,8,7};
  Stable_partition ( vect.begin(), vect.end(), [](int x){return x%2==0;});
  for(int i=0; i< vect.size(); i++)
    { cout << vect[i]; }
    return 0;
}
```

Valarray Class in C++

Public member functions in valarray class :-

1. apply() :- This function applies the manipulation given in its arguments to all valarray elements at once and returns a new valarray with manipulated values.

2. sum() :- function returns summation of elements of valarrays at once.

```
#include <iostream>
#include <valarray> // for valarray function
using namespace std;
int main()
{ valarray<int> var = {10,2,20,1,30};
  valarray<int> var1;
```

Circular linked list

Connected to form

// using apply() to increment all elements by 5

```
varr1 = varr.apply ( [](int x){return x=x+5;});
```

// Displaying new element value

```
for(int i=0; i<varr1.size(); i++)
    cout << varr1[i];
```

// Displaying sum of both old and new valarray

```
cout << varr.sum() << endl;
cout << varr1.sum() << endl;
return 0;
}
```

Output :- 15, 7, 25, 6, 35
63,
88

```
});
```

3. min() :- function returns smallest elements if valarray.
4. max() :- " " largest " " ".

```
#include <iostream>
#include <valarray>
using namespace std;
int main()
{ valarray <int> varr = {10,2,20,1,30};
    cout << varr.max() << "  " << varr.min();
    return 0;
}
```

Output :- 30. 1

5. shift() :- function returns new valarray after shifting element by number mentioned in its argument.
if number is positive, left-shift is applied.
if number " negative, right-shift " " .

6. cshift() :- function return new valarray after circularly shifting (rotating) elements by number mentioned in Argument. if number is positive, left circular shift is applied
if number is negative, right circular " " ".

```
#include <iostream>
#include <valarray>
Using namespace std;
int main ()
{
    valarray<int> varr = {10, 2, 20, 1, 30};
    valarray<int> varr1
    varr1 = varr.shift(2); // left shift by 2.
    for (int i=0; i< varr1.size(); i++)
        cout << varr1[i] << "  ";
    cout << endl;
    varr1 = varr.cshift(-3); // Circulary shift elelements to right
    for (int i=0; i< varr1.size(); i++)
        cout << varr1[i] << "  ";
    return 0;
}
```

Output :- 20 1 30 0 0
20 1 30 10 2

7. swap() :- function swaps one valarray with other.

```
#include <iostream>
#include <valarray>
Using namespace std;
int main ()
{ valarray <int> varr1 = {1, 2, 3, 4};
  valarray <int> varr2 = {2, 4, 6, 8};
  varr1.swap (varr2);
  for (int i=0; i< 4; i++)
    cout << varr1[i] << "  ";
  cout << endl;
  for (int i=0; i<4; i++)
    cout << varr2[i] << "  ";
```

Output; 2 4 6 8
1 2 3 4

Vector

vect.size() ⟶ give size of array

vect.empty() ⟶ Return True if array are empty else false.

vect.push_back(i) :- adds an element to end of vector. increasing its size by one.

vect.resize(i) ⟶ resize vector size. and put Zero to ~~empty~~ newly created location.
ex:- 25

vect.clear() ⟶ use to clear a vector. makes vector to contain 0 elements. it does not make elements Zeros it complete erases container.
rotate

∴ initialization of vector
we can create vector from another vector :-

```
Vector < int > V1;
// ....
vector < int > V2 = V1;
vector < int > V3(V1);
```

} initialization of V_2 and V_3 in this example are exactly same.

∴ Creation of vector of specific size, use following constructor :-

```
vector < int > Data (1000);
```

In this data will contain 1000 zeroes after creation.

∴ Two dimensional array via vector is to create vector of vectors.

```
vector < vector < int > > matrix;
```

Example :-
```
int N, M;
// ....
vector < vector < int > > matrix (N, vector < int > (M, -1));
```

Here we create matrix of size N*M and fill it with -1.

∴ To avoid of lot time uses and memory uses

```
Void Some-function (vector < int > v)
{
  ....
}
```

} -> never do it Unless necessary required.

instead, use following construction :-

```
Void Some-function (Const vector < int > &V)
{
  ---
}
```

→ Correct method to pass vector to a function.

if we have to change contents of vector in function just Omit "const" modifier.

```
Void modify_vector (vector < int > &r)
{
  ---
}
```

11-apr-2020

Pairs

Pair is just a pair of elements;

```
template < typename T1, typename T2 > struct pair
{ T1 first;
  T2 second;
};
```

In general pair < int, int > is a pair of integer values.

Pair < string, pair < int, int > > is pair of String and two int.

Usage :- pair < string, pair < int, int > > P;

```
String s = P.first ; // extract string
int x = P.second.first; // extract first int
int y = P.second.second ; // extract second int
```

Iterators

typeof(a+b) X = (a+b);

This will create the variable X of type matching type of (a+b) expression.

vect.insert(1,42) :- insert value 42 at 1 iterator.

All elements from second (index1) to last will be shifted right one element to leave a place for a new element.

For insert many elements, apply following technique:-

```
Vector <int> V;
Vector <int> V2;
V.insert (3, all(v2)); // Shift all elements from second to last
                       // to appropriate number of elements
                       // then copy of contents of V2 into V.
```

V.erase (iterator) :- Single element of vector is deleted.

V.erase (begin iterator, end iterator); interval, specified by two iterators, is erased from vector.

STRINGS

```
String S = "hello";
String S1 = S.substr(0,3), // "hel"
       S2 = S.substr(1,3), // "ell"
       S3 = S.substr(0, S.length()-1), // "hell"
       S4 = S.substr(1); // "ello";
```

Set

Set need a container with following features:-

→ add an element, but do not allow dubels [duplicates ?]

→ Remove elements

→ get count of elements (distinct elements)

→ check wheather elements are present in set

STL provides special container for set — set.

```
Set <int> S;
for(int i=1; i<=100; i++)
  { S.insert(i); // insert 100 elements [1.....100]
  }
S.insert(42); // does nothing, 42 already exist in set
for(int = 0; i<=100; i+=2)
  { S.erase(i); // erase even values
  }
int n = int(S.size()); // n will be 50
```

Note:- Push_back() member not used with set. It make sense since order of element in set does not matter, Push_back() is not applicable here.

Since set is not linear container, it's impossible to take element in set by index. Therefore, only way to traverse element of set is iterators.

```
// Calculate Sum of element in set using iterator
Set<int> s;
//....
int r=0;
for(set<int>::const_iterator it=s.begin(); it!=s.end(); it++)
{ r+=*it;
}
```

Traversing macros use :-

```
Set<pair<string, pair<int, vector<int>>>> ss;
int total = 0;
for(ss, it)
{ total += it->second.first;
}
```

it->second.first ↓ equivalent to '(*it).second.first'

Basic Number Theory-1

1. Modular Arithmetic

Property:-

1. $(a+b)\%c = (a\%c + b\%c)\%c$
2. $(a*b)\%c = ((a\%c)*(b\%c))\%c$
3. $(a-b)\%c = (a\%c - b\%c + c)\%c$
4. $(a/b)\%c = (a\%c * ((b^{-1})\%c))\%c$

2. Modular exponentiation :-

Calculate x^n :-

1.
```
int recursivepower(int x, int n)
{ if(n==0)
      return 1;
  return (* x * recursivepower(x, n-1));
}
```

2.
```
int iterativepower(int x, int n)
{ int result=1;
  while(n>0)
  { result = x * result;
    n--;
  }
  return result;
}
```

Efficient method :-

3.
```
int binaryExponentiation(int x, int n)
{ if(n==0)
      return 1;
  else if(n%2==0)
      return (binaryexponentiation(x*x, n/2));
  else if(n%2)
      return (x*binary exponentiation(x*x, n/2));
}
```

4. int binaryexponentiation (int x, int n)

```
{ if (n == 0)
      return 1;
else if
while (n>0)
  { if (n%.2 == 1)
        result = result * X;

    X = X * X
    n = n / 2;
  }
  return result;
}
```

Greatest Common Divisor (GCD)

using Euclid's algorithm (GCD(A,B) = GCD(B, A%.B))

```
int GCD (int A, int B)
{ if (B == 0)
    return A;
  Else
    return GCD (B, A%.B);
}
```

Extended Euclidean algorithm

Check prime number

if you have two positive number N and D, such that N is divisible by D and D is less than square root of N.

→ N/D must be greater than square root of N.

→ N is also divisible by (N/D).

Ex:- if N = 50 , $\sqrt{50} = 7$

Divisor :- 1, 50 ; 2, 25 ; 5, 10 → total 6.

```
Void checkprime (int N)
{   int count = 0;
    for(int i=1 ; i*i<=N ; ++i)
    {  if (N % i == 0)
       {  if (i*i == N)
              count ++;
          else
              count += 2;
       }
    }
    if (count == 2)
       cout << "prime number" << endl;
    else cout << "Not a prime number" << endl;
}
```

Time complexity $O(\sqrt{N})$

Sieve of Eratosthenes

Find all prime numbers that are less than or equal to given number N, and find wheather a number is prime number.