→ accumulate() :-

This function takes container range as argument and initial value as argument and calculate sum of container element with initial value.

Its defined in <numeric> header file.

It can also be used to do other things like subtract, multiply using user defined function.

Syntax:-

accumulate(begin add, end add, intial value)

Ex. -

```
int main()
{
    vector<int> v={70, 10, 20, 100};
    int val=0;
    accumulate(v.begin(), v.end(), val);
    cout << val;
    return 0;
}
```

output:- 200

We can use this function to subtract all element of a container from initial number.

```
int main()
{
    vector<int> v={20,7,2,40,10};
    int val= 100;
    accumulate(v.begin(), v.end(), val, minus<int>());
    cout<< val;
    return 0;
}
```

Output:- 21

We can use it to multiply all elements of a container.

```
bool mycomp(int x, int y)
{
    return x* y;
}

int main()
{
    vector<int> v={10,6,100,7};
    int val=1;
    accumulate(v.begin(), v.end(), val, mycomp);
    return 0;
}
```

output:- 42000

→ rand():-

This function generates random number according to seed provided.

If seed is not given / same seed given it will give same random numbers. So, for random random numbers we set seed using srand() function.

It is defined in <cstdlib> header file.

rand() gives random value from 0 to 32767.

ex-

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

int main(){
    for(int i=0; i<5; i++)
        cout<<rand()<<" ";
}
```

Output :-

5 random values.

Now, we will see how we define seed so that we get random values. We use srand() to plant seed.

For one seed, we get same type of random values, so, we need to change seed every time also. For this we use time() function, which returns time_t (which is integer generally) and will be different every time as time is changing.

```
srand(usigned int)
time_t time(time_t *t)
```

time() function is defined in <ctime> header file.

ex-

```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;
```

```
int main()
{
    srand(time(NULL));
    for(int i=0;i<5;i++)
        cout<<rand()<<" ";
    return 0;
}
```

Output :-

5 random values.

If we want to get random numbers from 0 to specific number (say 100). Then we use (%) modulo operator.

If want random number from 0 to 100 then

rand()%100; give numbers 0 to 99.

What if we want to random numbers b/w range (say 10 to 100) in this case we store '10' in low and '100' in high and find range as

range = high - low + 1

and get random numbers as

(rand() % range) + low;

→ Sort() :-

Sort() function sorts elements in containers in specific order. It works on those containers which provide random access iterator in logarithmic time. ex - array, vector, dequeue etc. and for other containers they have their own sort function (Ex - for list → l.sort())

In case of default condition (no comparision function given), it takes range as argument and sort the elements in increasing order.

We can change the order by giving/providing comparision function.

Syntax -

Sort(begin add, end add);

Sort(begin add, end add, comparision function);

Ex.-

```
int main()
{
    vector<int> v={7,2,9,5,30,10,17};
    sort(v.begin(), v.end());
    for(auto x: v)
        cout<<x<<" ";
    return 0;
}
```

Output:-

2 5 7 9 10 17 30

We can sort it in decreasing order using compaision function.

```
int main()
{
    vector<int> v={10,5,9,30,7,2};
    sort(v.begin(), v.end(), greater<int>());
    for(auto x: v)
        cout<<x<<" ";
    return 0;
}
```

Output:- 30 10 9 7 5 2

Case 1 :-

Here greater<int> is a generic function which reverses the order of default sorting.

Now, we will write our own comparision

```
bool mycomp(pair<int,int> p1, pair<int,int> p2)
{
    return p1.first >= p2.first;
}
```

// This comparision function will sort in increasing order.
// To sort in decreasing order (which is default) use '<'

```
int main(){
    int arr[]={7, 31, 20, 17, 5, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    sort(arr, arr+n, mycomp);
    for(int i=0; i<n; i++)
        cout<<arr[i]<<" ";
return 0;
}
```

Output :-

```
31 20 17 7 5 2
```

→ make_heap() :-

This function makes make heap with the given container. It accepts begin and end add. of container.

Syntax :-

make_heap(begin add, end add);

By default it makes max_heap but after passing comparator function it can make min_heap.

make_heap(begin add, end add, greater<int>());

push_heap(v.begin(), v.end(), greater<int>());
pop_heap(v.begin(), v.end(), greater<int>());

sort_heap(v.begin(), v.end());

→ String C++ type-

advantage of C++ type string over C string-

- We don't need to worry about size. We can make a variable of string class and save a string in it of any size and can change the size if we need to.

- We can declair or make a variable of string class now and assign value to it later. We can't do this in C, we have to use strcpy for this.

- We can use +, <, >, ==, <= and >= operator with C++ string but can't use with C string.

- It has richer library function. It supports functions like begin(), end(), sort(), next permutation and all other algorithm functions.

- length() ⇒ returns size of string.
- operator + can be used to concatinate two strings together. Str3 = Str1 + Str2;
- Substr() - This function gives substring. It takes one or two input/argument.

Str.substr(begin index of substring, size of substring)

If only one argument is given (only starting index), it will make substring to the end.

ex-

```
string str = "gfgforme";
cout<<str.substr(1, 3)<<endl;
cout<<str.substr(2);
```

o/p - fgf
gforme

- find() - find function checks if given substring is present in string or not and return index of first occurance of substring if present. If not present returns 'string::npos'.

- operators ==, <, >, <= and >= can be used to compare two string which is lexicographically greater, smaller or equal.

- 'Cin' can be used to take input in string but to take input string with spaces we use getline function.

```
getline(cin, str);
```

now, it will stop taking input if encounters enter.
we can also specify, a character after which, it can stop taking input as 3rd parameter.

```
getline(cin, str, '$');
```

We can use all 3 type of traversal loop with string.

NOTE:- find function has a variation which it takes two argument, first substring which is to be find and second is index, after which given substring is find, in string. If index is not given it searches in whole string.

→ merge() :-

merge() function takes two sorted containers and merges them in 3rd container.

Syntax :-

merge(begin of 1st con., end of 1st con., begin of 2nd con., end of 2nd con., begin of 3rd con.);

ex. -

```
int main()
{
    vector<int> v1 = {10, 20, 40};
    vector<int> v2 = {5, 15, 30};
    vector<int> v3(6); // forming vector of 6 size;

    merge(v1.begin(), v1.end(), v2.begin(), v2.end(),
          v3.begin());

    for(auto x : v)
        cout << x << " ";

    return 0;
}
```

output :- 5 10 15 20 30 40

T.C. = O(m+n)

⇒ next permutation :-

next permutation function arrange the elements in next lexicographical permutation such that next smaller ~~lexicographically~~ smaller permutation that are lexicographically greater than the given sequence.

For sequence of size N, no of permutation will be N!

Ex. -

Given - 1 2 3 4 5

↓

1 2 3 5 4 (lexicographically greater than previous one with smallest possible sequence)

1 2 4 3 5

1 2 4 5 3

1 2 5 3 4

1 2 5 4 3

1 3 2 4 5

⋮

Syntax :-

next permutation (begin add., end add.);

ex. -

```
int main()
{
    vector<int> v = {1, 2, 5, 4, 3};

    next_permutation(v.begin(), v.end());

    for(auto x : v)
        cout << x << " ";

    return 0;
}
```

Output :-

1 3 2 4 5

T.C. - O(n)

→ Reverse() :-

This reverse() function reverse elements in container with bidirectional or higher iterator.

Syntax :-

```
reverse(begin add, end add);
```

Ex.-

```
int main()
{
  vector<int> v={7, 12, 9, 5, 0, 1};
  reverse(v.begin()+1, v.end());
  for(auto x: v)
    cout<<x<<" ";
  return 0;
}
```

output :-

7 1 0 5 9 12