## ⇒ Forward list / singly linked list:-

Forward list is list in which elements can only be accessed in forward direction not in backward direction. It's iterator is forward iterator, which can't iterate backward and can't access elements randomly.

We use it in chaining in hashing, in graph etc.

We need to include <forward_list> header file to use it.

→ Ways to initialize a forward_list.

```
forward_list<int> l;
l.push_front(5);        5
l.push_front(3);        3 -> 5
l.push_front(10);       10 -> 3 -> 5
```

Push_front() function pushes element at the first position.

10 -> 3 -> 5

forward_list<int> l={10, 30, 50};

10 → 30 → 50

→ ways to traverse through forward list:-

```
for(auto x : l)
{
    cout<<x<<" ";
}
```

```
for(auto it=l.begin(); it!=l.end(); it++)
{
    cout<<(*it)<<" ";
}
```

→ Some function of forward_list :-

- push_front();

Enter an element at front of the forward list.

forward_list<int> l={10, 20};

10 → 20

l.push_front(3);

3 -> 10 -> 20

- pop_front();

Removes one elements from the front of the forward_list.

forward_list<int> l={30,5,7,10}

30 -> 5 -> 7 -> 10

l.pop_front();

5 -> 7 -> 10

- assign()

This function is used to assign new set of values to the forward_list after removing old one, if any. It can also be used to assign value of one forward_list to other, and can be used to assign specific value to all elements mode.

forward_list<int> l={2, 7, 1, 9, 5, 11};

2 → 7 → 1 → 9 → 5 → 11
l

- l.assign({7, 2, 17, 5, 10});

7 → 2 → 17 → 5 → 10
l

- l.assign(5, 10);

10 → 10 → 10 → 10 → 10
l

l.assign({7, 2, 1, 9, 5});

7 → 2 → 1 → 9 → 5
l

forward_list<int> l2;

- l2.assign(l.begin(), l.end());

7 → 2 → 1 → 9 → 5
l2

- remove()

This function removes, all copy of number/element mentioned to removed, from forward_list.

~~It can also remove elements in given range.~~

l.remove(value);

forward_list<int> l={10,7,5,2,10,5};

10 → 7 → 5 → 2 → 10 → 5

l.remove(10);

7 → 5 → 2 → 5

• insert_after() :-

insert_after() accepts an iterator and value and places that value after that iterator.

Syntax. —

l.insert_after(iterator, value);

It returns iterator to the element inserted last.

Ex.- forward_list<int> l={10,12,7};

10 → 12 → 7

auto it = l.insert_after(l.begin, 5);

10 -> 5 -> 12 -> 7

It returns iterator to '5' as it is inseted last.

auto it = l.insert_after(it, {7, 5, 0});

It inserts 3 elements after 'it' iterator and returns iterator to the last element inserted i.e. '0'.

10 -> 5 -> 7 -> 5 -> 0 -> 12 -> 7

- emplace_after() :-

It's same as insert_after() but it does some optimisation over inset_after. It is added in C++11 and it's works better, for big user defined objects than insert_after().

For premitive data type both works same.

forward_list<int> l={7, 5, 9};

auto it = l.emplace_after(l.begin(), 10);

7 → 10 → 5 → 9
l

l.emplace_after(it, 8);

7 → 10 → 8 → 5 → 9
l

• erase_after() :-

erase_after() takes iterator and erases 1 element after it or it can take two iterator starting and end and erases all element b/w them.

It returns iterator to the next element of last erased element.

ex -

forward_list<int> l={17, 7, 49};

auto it = l.insert_after(l.begin(), {8, 6, 16});

17 → 8 → 6 → 16 → 7 → 49
↑ it

it = l.erase_after(it);

17 → 8 → 6 → 16 → 49
↑ it

it=l.erase_after(l.begin()+2, l.end());

17->8->6

now, it will point to l.end() as element after last remove element is l.end().

- clear() :-

It clears the forward_list by removing all element.

forward_list<int> l={9, 8, 7};
l.clear();
no element left.

- empty() :-

Checks if forward_list is empty or not.

- reverse() :-

reverses the forward list.

l.reverse();

• merge() :-

It merges two sorted forward list. It places element of list2 in right position in list1 and SO list2 become empty() at last.

ex.-

forward_list<int> l1 = {10, 20, 30};
forward_list<int> l2 = {5, 15};

l1.merge(l2);

l2 = {5, 10, 15, 20, 30}
l2 = { }

} after merging

• sort() :

It sorts the forward list.

l.sort();

## ⇒ list / doubly linked list :-

In doubly linked list / list, we can move in both direction. It's iterator are bi-directional. It can move forward and backward.

Need to include <list> header file to use it.

Initialization of list :-

- ```
  list<int> l;

  l.push_back(5);        5
  l.push_back(10);       5 ⇄ 10
  l.push_front(7);       7 ⇄ 5 ⇄ 10
  ```

- ```
  list<int> l = {5, 7, 11, 5};
  ```

  5 ⇄ 7 ⇄ 11 ⇄ 5

Traversing a list:-

- for(auto x: l)
  cout << x << " ";

- ~~for(int i=0; i<l.size(); i++~~

- for(auto it=l.begin(); it != l.end(); it++)
  cout << *(it) << " ";

Functions used with list:-

- push_back():-

  insert an element at end of list.

  ex.-

  list<int> l = {5, 7, 2};

  5 ⇌ 7 ⇌ 2

  l.push_back(9);

  5 ⇌ 7 ⇌ 2 ⇌ 9

- Push_front():-
  like push_back(), but insert an de. at beginning.

list<int> l={5,7,0,9}
l.push_front(3);

3⇄5⇄7⇄0⇄9

- Pop_front():-
  Removes an element from begning.

- Pop_back():-
  Removes an element from end.

- front():-
  Returns element at first position as reference, so that we can modify it.

```
list<int> l={8,0,2};
cout<<l.front()<<" ";
```

O/P - 8

- back() :-

  Returns element refesence of last element in list.

• size() :-

Returns size of list as no. of elemed.

```
list<int> l = {7, 0, 9, 2};
int $ = l.size();
cout<< $;
```

O/P - 4

• insert() :-

insert() function can be used to insert one element or many instance of same element at specific position in list.

It can also be used to insert value of one list to the other.

Syntax to inset elemed :-

list(position, no of times, value);

if no of times not mentioned it is by default 1.

list<int> l = {7, 0, 9, 2, 5, 7};

auto it = l.insert(l.begin, 9);

9 ⇄ 7 ⇄ 0 ⇄ 9 ⇄ 2 ⇄ 5 ⇄ 7

NOTE:- insert() returns iterator to the first inserted element, in this example to the first element '9'.

cout << *it; // will print 9.

l.pop_back();
l.pop_front();
l.pop_front();

9 ⇄ 2 ⇄ 5 l

it = l.insert(l.begin+1, 2, 7);

9 ⇄ 7 ⇄ 7 ⇄ 2 ⇄ 5

it

Syntax to assign value of one list to other:-

l2.insert(l2.begin(), l.begin(), l.end());

position to insert in l2 — range to copy

list<int> l = {8, 6, 4, 0};
list<int> l2;
~~l2.begin~~
l2.insert(l2.begin(), l.begin(), l.end());

8 ⇌ 6 ⇌ 4 ⇌ 0
l2

• erase() :-

It is used to erase one or more than one element in a range.

erasing one element :-

syntax :-

l.erase(position)

list<int> l = {10, 20, 30};

auto it = l.erase(l.begin());

20 ⇌ 30
↑
it

erase() returns iterator to the next element of last erased element.

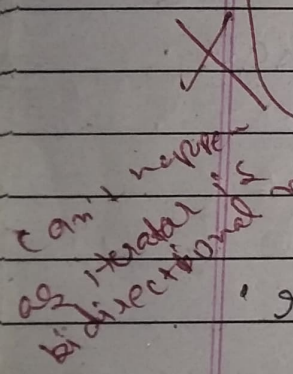
## erasing in range :-

l.erase(first position, last position);

It will erase all element in mid range including element at position first but excluding element at position last.

list<int> l = {12, 7, 9, 8};

auto it = l.erase(l.begin(), l.begin()+2);




## remove() :-

It takes a value as argument and removes all instance of that value.

l.remove(value);

list<int> l = {12, 7, 9, 7};

l.remove(7);

12 ≥ 9

• reverse():-

reverses the list.

• merge():-

It merges two already sorted list in sorted order.

Syntax:-

```
l1.merge(l2);
```

It merges l2 to l1 such that all element of l2, placed in l1 in right position and l2 become empty.

ex-

```
List<Int> l1 = {10, 30, 50};
List<Int> l2 = {20, 40, 60};

l1.merge(l2);
```

10 ≤ 20 ≤ 30 ≤ 40 ≤ 50 ≤ 60

l1

{ } l2