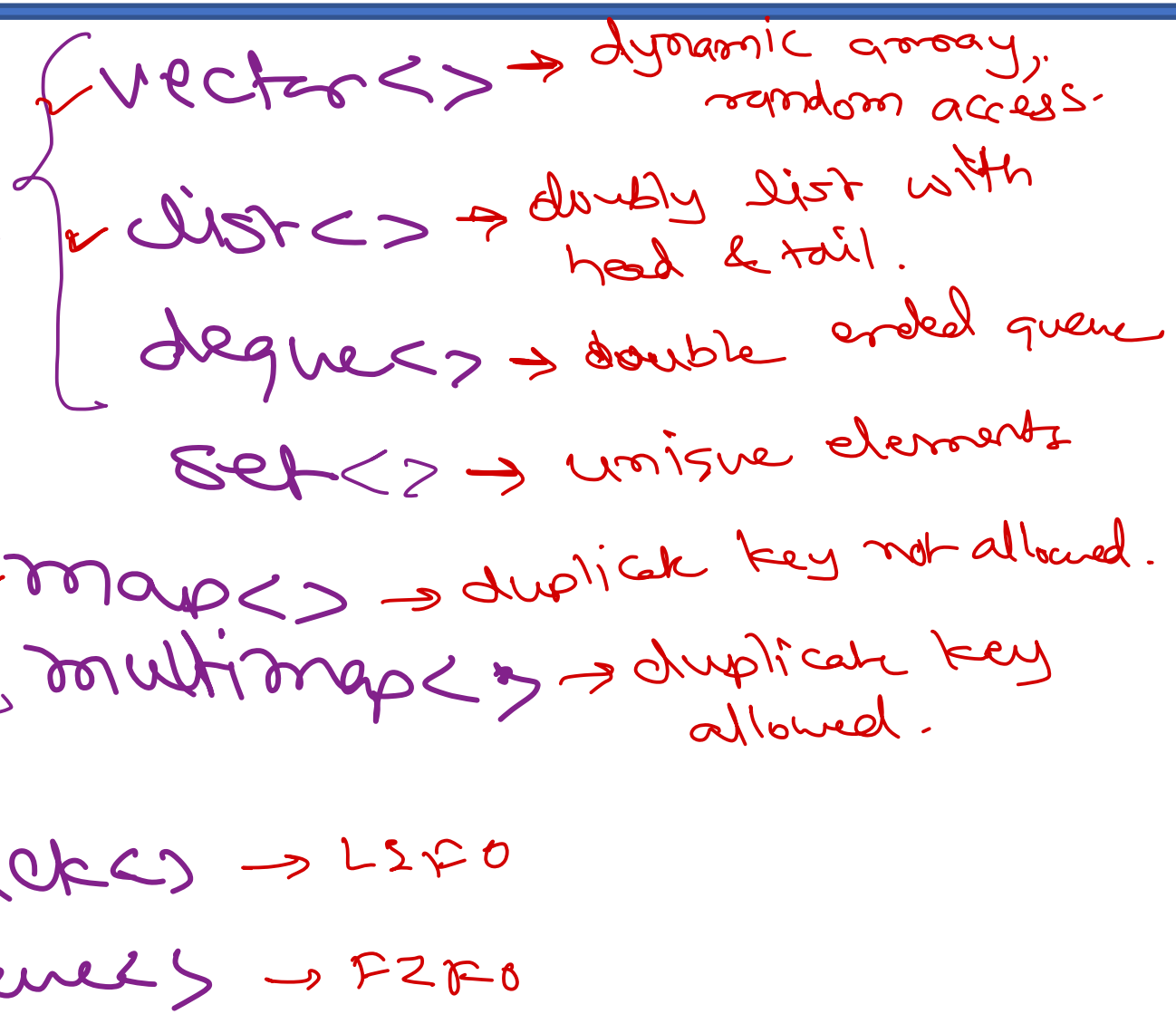
Data Structure & Algorithms

Sunbeam Infotech

STL

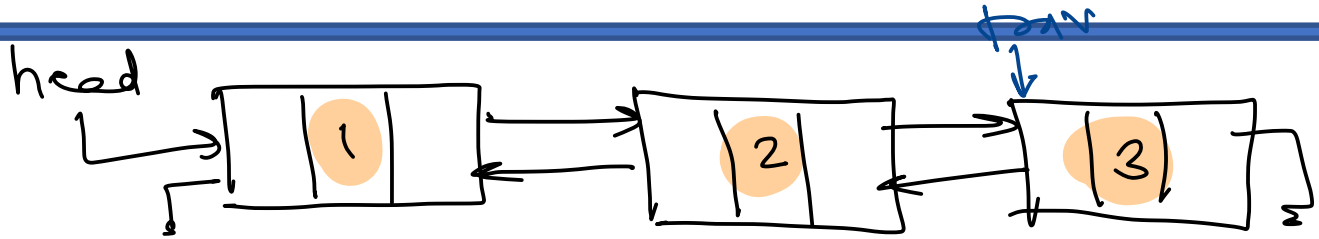
- Containers hold data and operations to be performed on data.
- STL containers are of three types
  - Sequential: Linear collection
    - vector, list, deque
  - Associative: Key-value pair collection
    - set, map, multimap
  - Adapters: Limited container functionality
    - stack, queue



STL

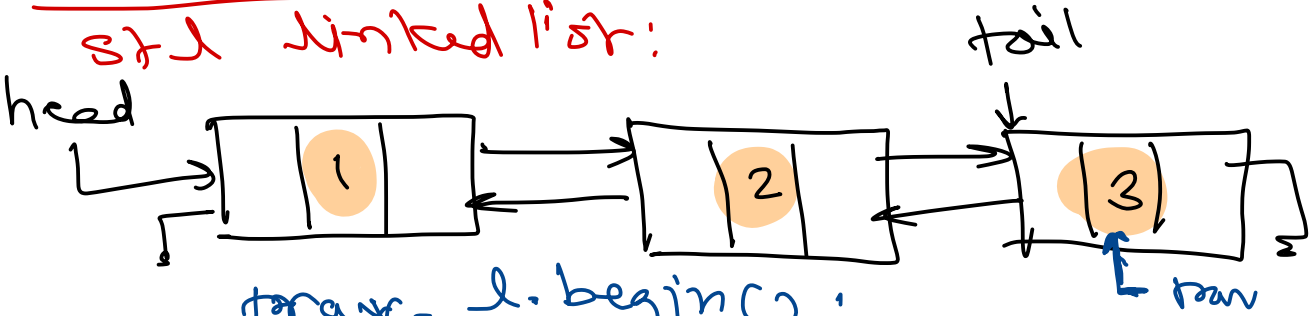
our linkedlist;

- Containers are traversed using iterators.
- Usually iterators are implemented as nested classes in containers.
- Iterators are smart pointers (with -> and * operators overloaded).
- There are six types of iterators
  - Input iterator (read ops, fwd)
  - Output iterator (write ops, fwd)
  - Bi-directional iterator (rw, bi-dirn)
  - Forward iterator (rw, fwd)
  - Reverse iterator (rw, rev)
  - Random access iterator (rw, any)




```
trav = head;
while( trav != null) {
    cout << trav->data;
    trav = trav->next;
}
```

STL linked list:




```
traav = l.begin();
while ( trav != l.end()) {
    cout << *trav;
    trav++;
}
```

vector ✓
list ✗

STL

- Algorithms are global functions that operates on containers.
- They can be classified as
  - Search functions
  - Sort functions
  - Manipulaion functions
  - Non-modifying functions
  - Numeric functions

Hash table

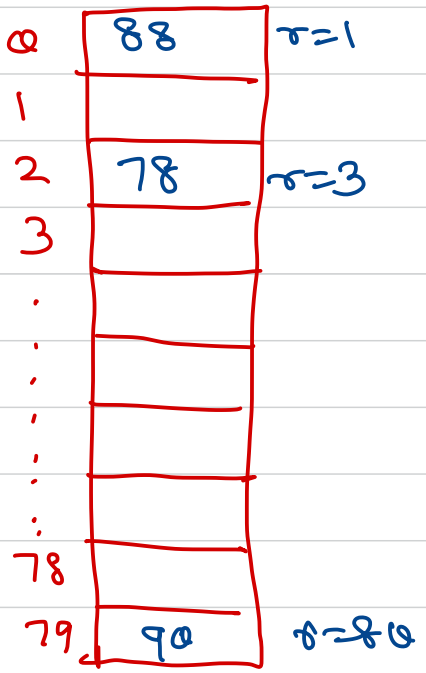
① hashtable is a ds to store key-value pairs, so that for given key, value can be searched in fastest possible time.

② key-value pair → associative data-structure.

③ ideal time complexity of search ele → O(1)
i.e. const time (irrespective of num of eles).

Q. Class has 80 students (roll 1 to 80). Want to store their marks so that for a given roll marks can be searched quickly. which basic DS to use?

~~array~~ table



array index = roll - 1

contents = marks.

table slot = roll - 1

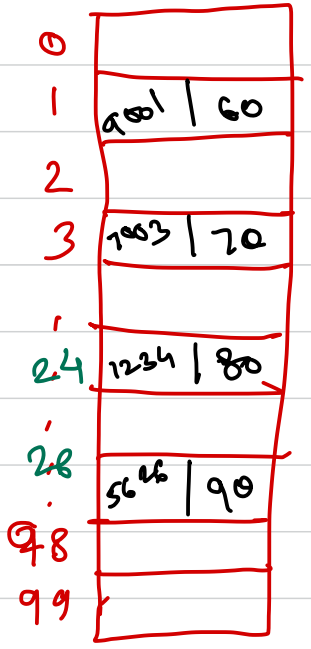
data/contents = marks

key = roll & value = marks

slot = f(key) ← hash fn

e.g. $f(r) = r - 1$

Q. University has n students (PRN ———). Few students decided to meet (m). Want to store their marks, so that for given PRN, marks can be found quickly.



8003 | 75

24 ← 1324 % 100

26 ← 5626 % 100

3 ← 7003 % 100

3 ← 8003 % 100

1 ← 9001 % 100

2102

5403

```
struct student {
    int prn;
    float marks;
};
```

~~array~~ table

~~index~~ slot

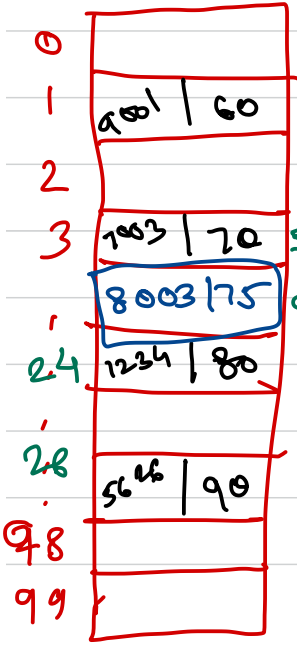
~~formula~~ hash fn

~~clash~~ collision

if multiple keys are yielding same slot in table (due to hash fn), it is called as collision.

There are two ways to handle the collision:

① open addressing ② chaining (bucketing)



(only once)

r = 1

hash fn 8003 | 75

rehash fn. (repeatedly)

(h(r) + 1) % n

find → 7003 % 100 = 3 → 0(1) → towards = 70

find → 8003 % 100 = 3 → not found

rehash = 3+1 = 4 → towards = 75

T + rehash

open addressing can be used only if num of entries < num of slots in table

Load factor of Hash table. = $\frac{\text{num of entries}}{\text{num of slots}}$

① num of entries < num of slots
ie. load factor < 1

② num of entries = num of slots
ie. load factor = 1

③ num of entries > num of slots
ie. load factor > 1

→ open addressing

→ chaining

open addressing

hash fn ① → collision → rehash fn ②

probing / checking next slot ← math fn. (also)

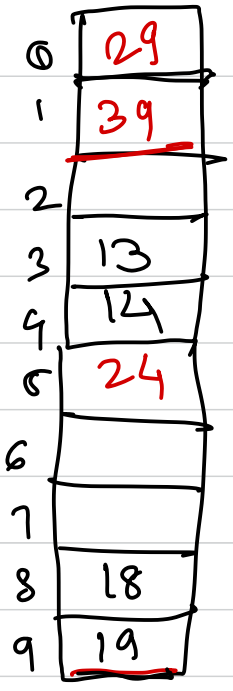
linear probing ← linear fn

quadratic probing ← quadratic fn

" ← polynomial fn

* Ideal hash fn → minimum collision. (depends on data & problem domain).
* It is observed that multiply with prime num gives less collisions.

```
slot = (roll * 31) % slots;
```



13, 19, 18, 14, 24, 29, 39, 68.

$h(n) = n \% 10$

$rh(n) = h(n) + 1$

char int
K V

"abcdab"

(a)

m[ch] = m[ch] + 1;

1 + 1

m['b'] = 2

→ a	2
b	2
c	1
d	1

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>