I. STL (Standard Template Library) - C++

- Three Components
  - Containers: Objects to store data - vector, list, stack, queue, set, map, etc
  - Iterators: "smart" pointers to access data in containers
  - Algorithms: Function templates for operating on containers - sort, find, search, etc.

II. Containers

- Sequence Containers
  - Vector, list, deque
- Associative Containers
  - Set, Multiset, Map, Multimap, unordered_set, unordered_map, etc
- Container Adapters
  - Use other containers to implement it
  - Stack, Queue, Priority_queue

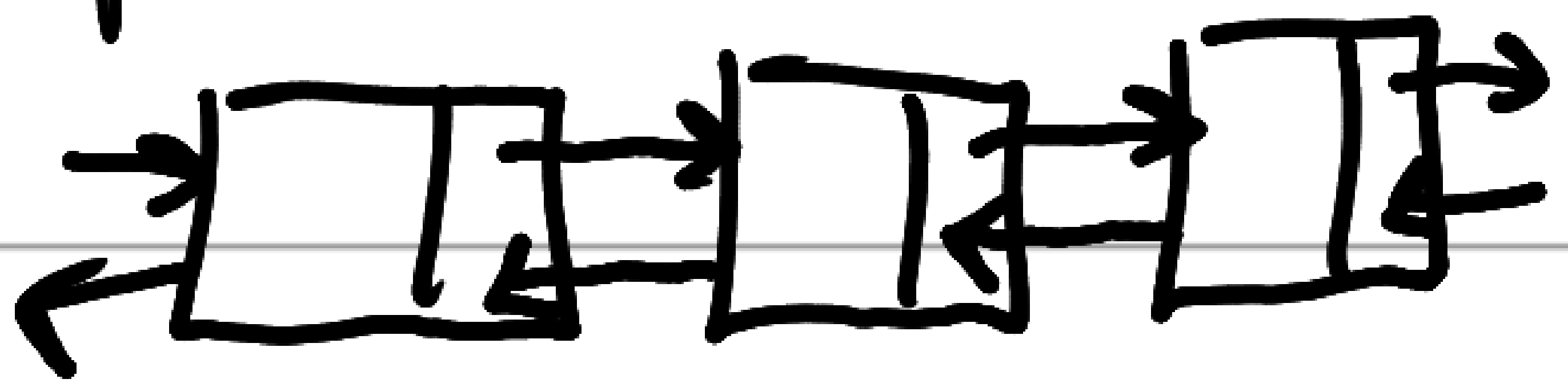
- **Vector** Container (C++)
  - A Vector is a smart array
    - Δ Can grow & shrink capacity while program is running

```
vector<string> names;
names.push_back("Joe");
```

- **ArrayList** (Java)
  - Java equivalent to vector

```
ArrayList<String> names = new ArrayList<>();
names.add("Joe");
```

- Both Vector & ArrayList are implemented using a dynamic array
- **List** Container
  - The list in STL is implemented as a doubly linked list



```
#include <list>
list<int> myList = {20,30,40,50};
myList.push_front(10);
myList.push_back(60);
```

- Operations
  - front(), back()
  - push_front(), push_back()
  - pop_front(), pop_back()
  - empty(), insert(), erase()
  - etc.

- LinkedList (Java)
  - Equivalent of List in Java

```
LinkedList<Integer> myList = new LinkedList<>();
```

- Iterator for List: How to Access Elements?
  - In linked list you should have a pointer to the data in the next node

III. Iterators

C++

```
container<dataType>::iterator name;
auto name;
```

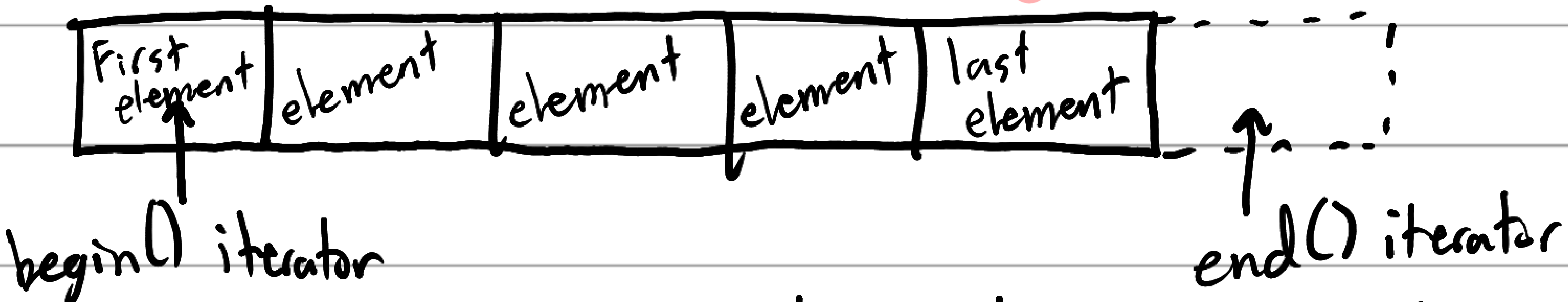
- An iterator is a "smart" pointer to access data in a container
- Iterators are designed to provide uniform interface across different containers in STL
  - Ex: "++" operator points to the next element in a vector/list

- Each container has its "own" iterator type

- begin() & end() functions for containers
  - All containers provide begin() and end() member functions

```
Iterator<Integer> ptr = myList.iterator();
while(ptr.hasNext()){
...}
```

Java



- A begin() member function returns an iterator pointing to the container's first element
- An end() member function returns an iterator pointing to the position after the container's last element
  - Typically used to know when end of container reached

IV. STL Algorithms

- The STL provides several algorithms in the <algorithm> header file ← sort, search, min/max, shuffle, etc
  - The functions perform various operations on a range of elements

- Algorithm libraries
  - #include <algorithm> (C++)
  - import java.util.Collections; (Java)

V. Stacks in STL

- Important functions of a stack
  - push() // add to stack
  - pop() // remove from stack
  - top() // top element of stack

import java.util.Stack;

#include <stack>

```
stack<int> s;
```

```
Stack<Integer> s = new Stack<>();
```

VI. Queues in STL

- Important functions of queue
  - push() //add to back of queue
  - pop() //remove next in queue
  - front() //returns reference of first element in queue
  - back() //returns reference of last element in queue

import java.util.Queue

#include <queue>

C++

```
queue<string> q;
```

Java

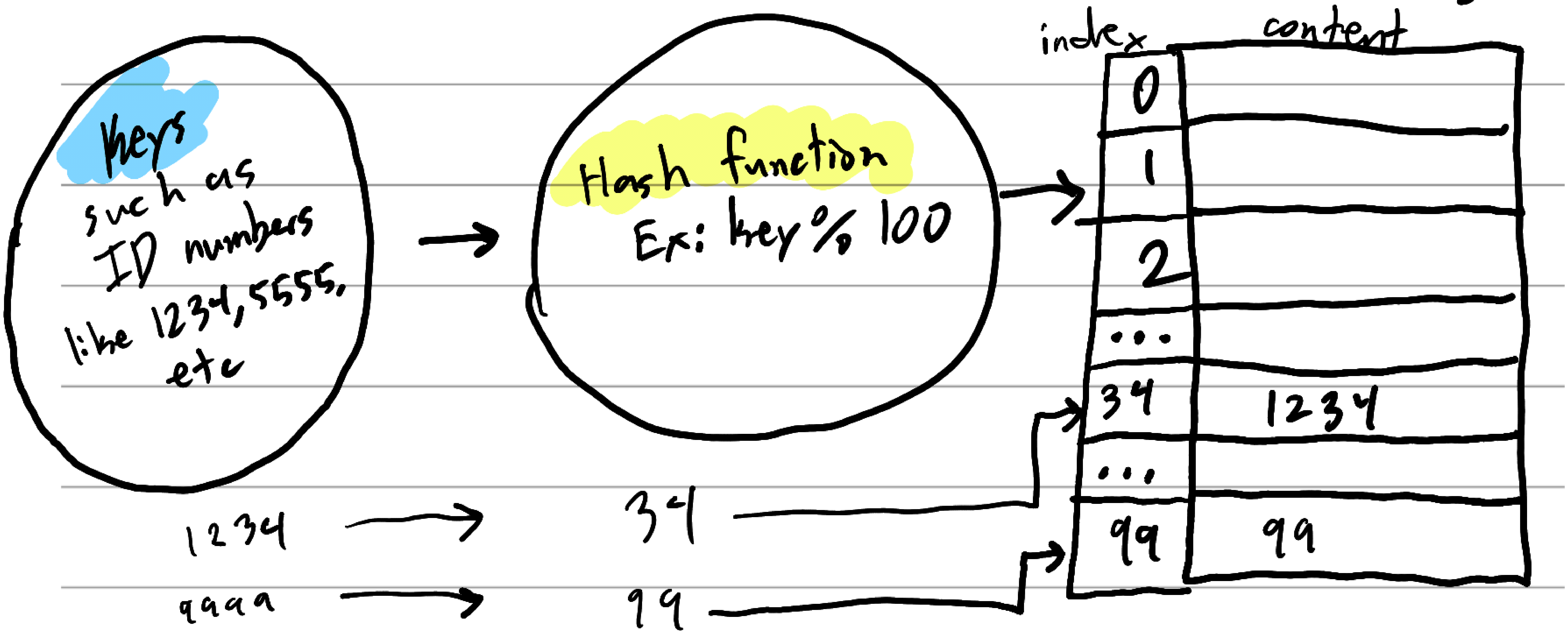
```
Queue <String> q = new LinkedList<>();
```

VII. Set Container

- A set container stores elements without duplicates
    - Adding a duplicate element to set gets ignored
- Two Types of Set Containers
    - In C++ library, there are two types of set containers
        1. unordered_set
            - ▵ No order among elements
            - ▵ Implemented using hashing
        2. set
            - ▵ Automatically ordered when added or deleted
            - ▵ Implemented using Balanced BST
- The unordered_set Container
    - Similar to set container except in two regards
        - ▵ Values in unordered_set are not sorted
        - ▵ unordered_set class has better performance
            - ○ Uses Hashing

set has .erase() method

VIII. Hash Table & Hash Function for Hashing



- Java Set Types: HashSet & TreeSet
    - Two types of sets, like C++
        - TreeSet: An ordered set → set in C++
        - HashSet: An unordered set → unordered_set in C++

IX. Map Container

- A map is an associative container
    - An associative container holds <key, value> pairs
- Each element in a map has a key and its associated value
    - You always retrieve a value that is associated w/ key
    - Keys should be unique (= no duplicates)

- Key/Value Pairs
  - Key: Student ID → Value: Student Records
  - Key: Liscence Plate # → Value: Vehicle Info
  - Key: Zip Code → Value: City Name

II. Two types of map containers (C++)

1. unordered_map
   - Keys are unordered
   - Implemented using hashing
2. map
   - Keys are ordered
   - Implemented using balanced BST

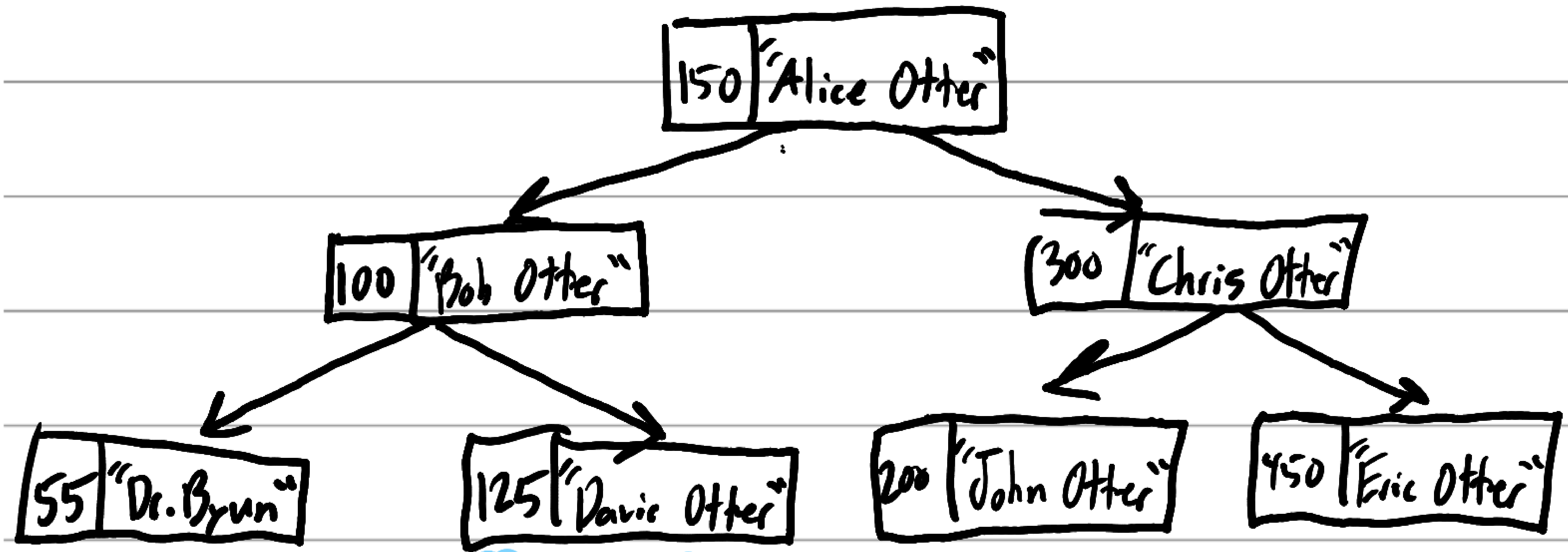
- The pair Type
  - Internally, each element of a map is stored as an instance of the pair type
    - pair is a struct that has two member variables, first & second
    - An element's key stored in first, and the value stored in second

- insert() member function
  - Used to add a pair to a map
  - You need the make_pair() function to construct a pair

    map_name.insert(make_pair());

student.insert (make_pair (7777, "Eric Otter"))

- Implementation of Map Container
  - A map container is implemented using a balanced BST called a red-black tree



- The unordered_map Class
  - Implemented using a hash table
  - Similar to map except in two regards
    - ▲ Keys are not sorted
    - ▲ Better performance

III. Java Map Types: HashMap & TreeMap

- In Java, there are two types of maps
    - TreeMap: An ordered map, like C++'s map
    - HashMap: An unordered map, like C++'s unordered_map

IV. 2-D Arrays

- A 2D Array is a collection of elements organized in a matrix format
    - Useful to represent a graph
- Ex:

```
int graph[4][4] = {
        {0,2,3,0},          graph[0][2] = 3
        {1,0,7,5},          graph[2][1] = 6
        {0,6,0,2},
        {7,0,1,0}
    };
```

- 2D Vectors
    - Same thing but with vectors

```
vector<vector<int>> graph = {...}
```

-2D ArrayList in Java

```
//Declaration of 2D ArrayList
ArrayList<ArrayList<Integer>> graph =new ArrayList<>(n);
for(int i=0; i<n; i++){
    graph.add(new ArrayList<>(m);
}
```

Reading values from user

```
for (int i=0;i<n; i++){
    for(int j=0;j<m;j++) {
        System.out.println("graph["+i+"]["+j+"]: ");
        int value= scanner.nextInt();
        graph.get(i).add(value);
    }
}
```