C++ Training Course

Data structure, Algorithm in C++

Lesson Objectives

- Basic data structures
- STL data structures

Section 1

Basic data structures

Basic data structures. Agenda

- Data structures introduction
- Common data structures

Data structures introduction

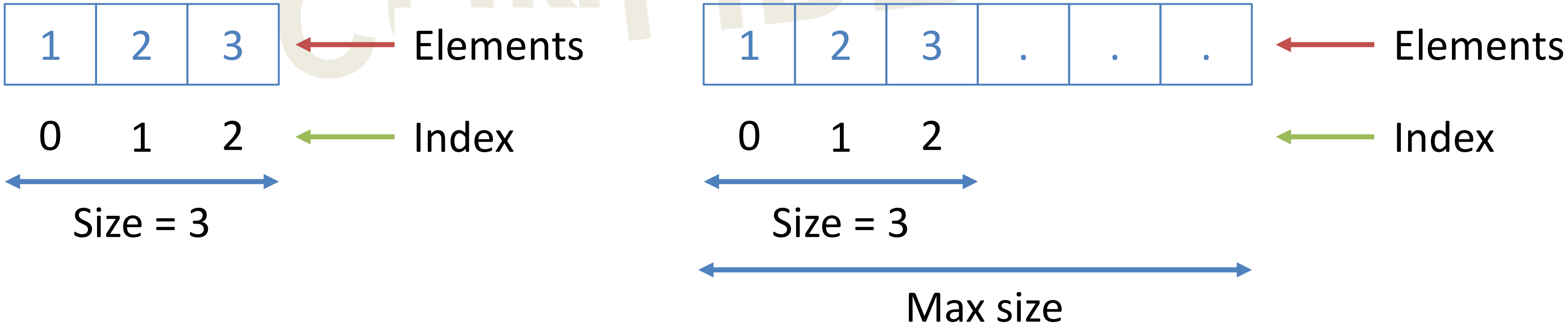
- Data Structures are a specialized means of organizing and storing data in computers in such a way that we can perform operations on the stored data more efficiently
- Operations on data structures include
  - ✓ Access (read): get element at index or id, traverse, search…
  - ✓ Modification (write): update, add, insert, delete…

Common data structures

- Array and Vector
- Linked List
- Stack
- Queue
- Hash Table
- Tree
- Heap
- Graphs

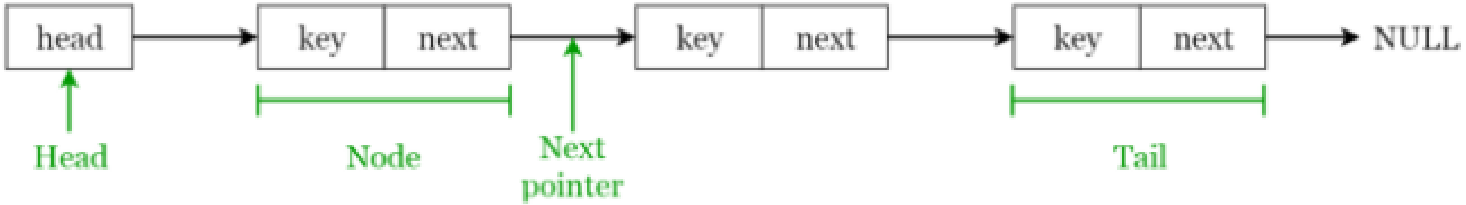
Array and Vector

- Array (static array) is a data structure which stores a fixed-size sequential collection of elements of the same type. Arrays are indexed, meaning that random access is possible
- Vector (dynamic array) is an array with dynamic size which allow add, insert, delete elements without worrying about the size



Linked List

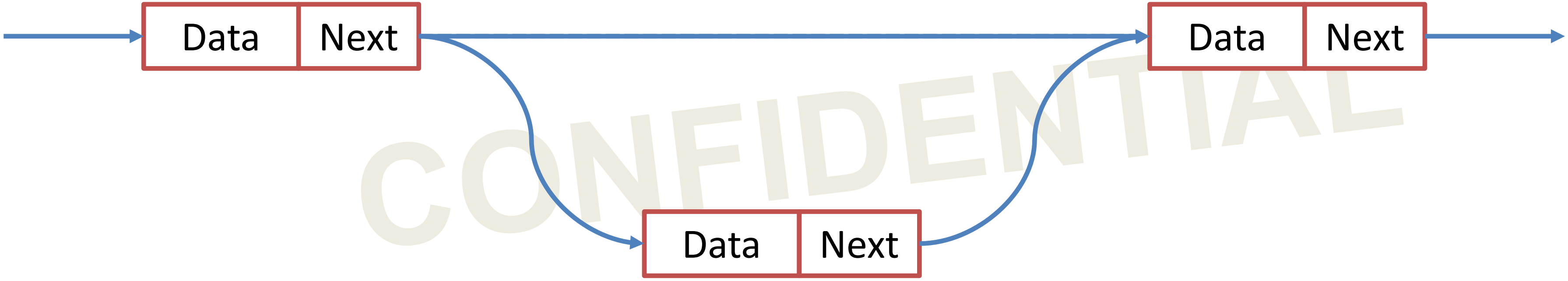
- Linked list is a sequential structure that consists of a sequence of items in linear order which are linked to each other. It only allows access data sequentially, random access is not possible
- Elements in a linked list are known as nodes. Each node contains a key and a pointer to its successor node, known as next
- The attribute named head points to the first element of the linked list. The last element of the linked list is known as the tail



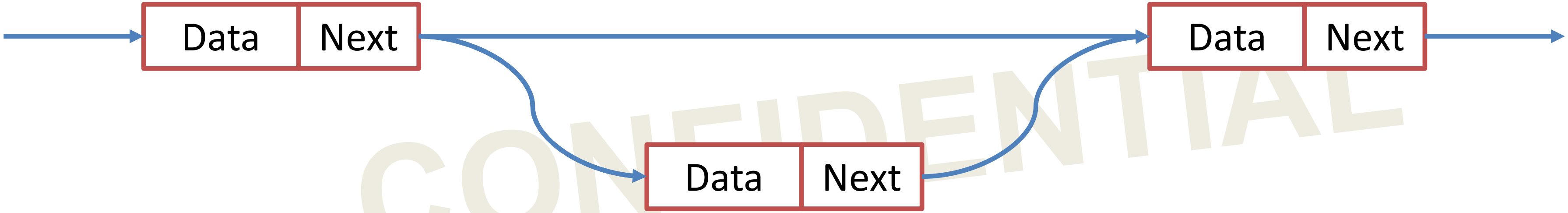
Linked List

- There are three types of linked list
  - ✓ Singly linked list: traversal of items can be done in the forward direction only
  - ✓ Doubly linked list: traversal of items can be done in both forward and backward directions. Nodes consist of an additional pointer known as prev, pointing to the previous node
  - ✓ Circular linked list: linked list where the prev pointer of the head points to the tail and the next pointer of the tail points to the head

Linked List - Insertion Operation



Linked List - Deletion Operation



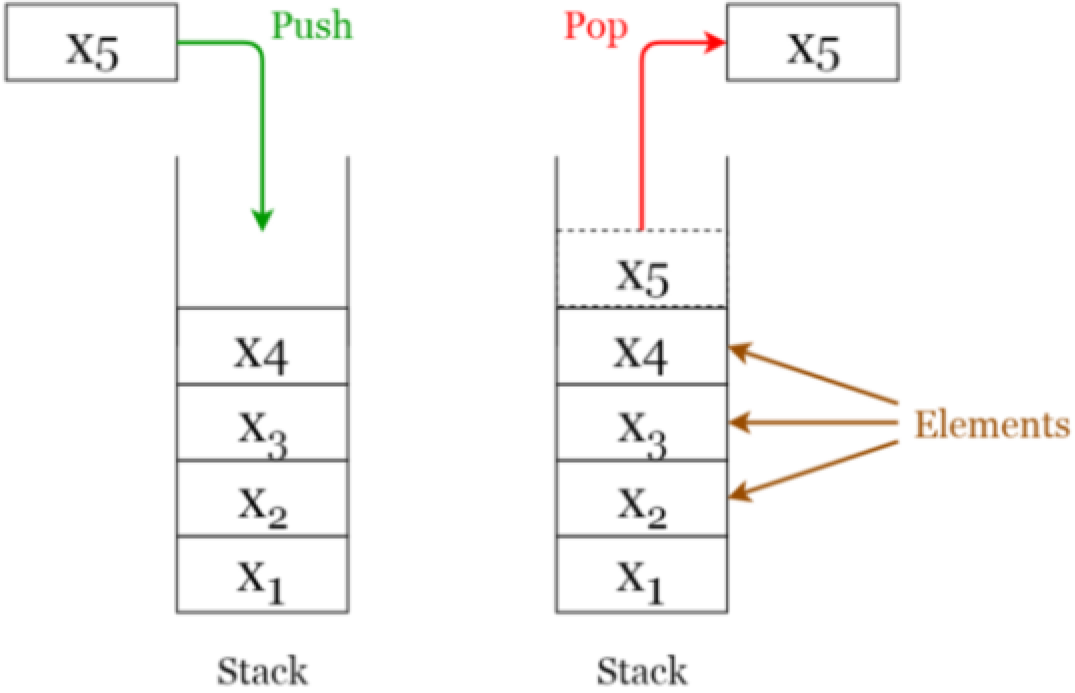
Vector vs Linked List

Operation	Vector	Linked List
Access	Random -> Fast	Sequentially -> Slow
Update	Random -> Fast	Sequentially -> Slow
Insert	Shift elements -> Slow	Update pointer -> Fast
Delete	Shift elements -> Slow	Update pointer -> Fast
Add	If size > max size, need create and copy to new vector -> Slow	Don’t care about max size -> Fast

Stack

- A stack is a LIFO (Last In First Out — the element placed at last can be accessed at first) structure which can be commonly found in many programming languages
- Stack operations
  - ✓ push: insert an element on to the top of the stack
  - ✓ pop: delete the topmost element and return it
- Additional functions
  - ✓ peek: return the top element of the stack without deleting it
  - ✓ isEmpty: check if the stack is empty
  - ✓ isFull: check if the stack is full

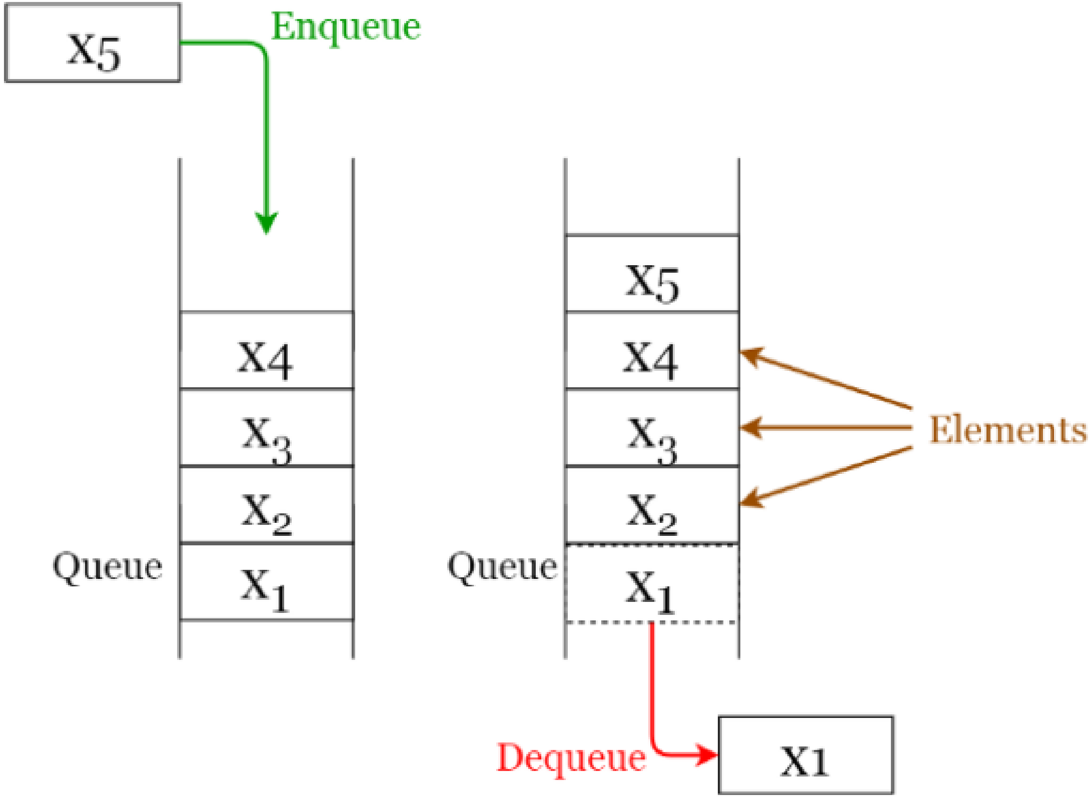
Stack



Queue

- A queue is a FIFO (First In First Out — the element placed at first can be accessed at first) structure which can be commonly found in many programming languages
- Queue operations
  - ✓ enqueue: insert an element to the end of the queue
  - ✓ dequeue: delete the element from the beginning of the queue
- Additional functions
  - ✓ peek: gets the element at the front of the queue without removing it
  - ✓ isEmpty: checks if the queue is empty
  - ✓ isFull: checks if the queue is full

Queue

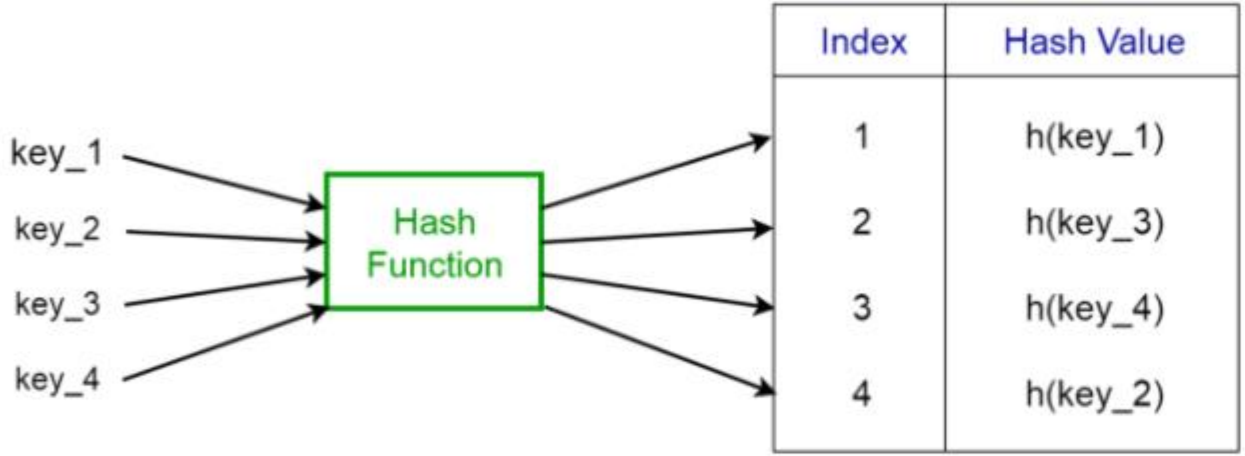

X5
Enqueue
X4
X_3
X_2
Queue
X_1
X5
X4
X_3
X_2
Queue
X_1
Elements
Dequeue
X1

Hash Table

- A hash table is a data structure that stores values which have keys associated with each of them
- Supports lookup efficiently if we know the key associated with the value. So, it is very efficient in inserting and searching, irrespective of the size of the data
- Hash table uses hash function to calculate the index of the table (slot) to which each value goes. The value calculated using the hash function for a given key is called the hash value which indicates the index of the table to which the value is mapped

  For example*:* **h(k) = k % m**

Hash Table



Index	Hash Value
1	h(key_1)
2	h(key_3)
3	h(key_4)
4	h(key_2)

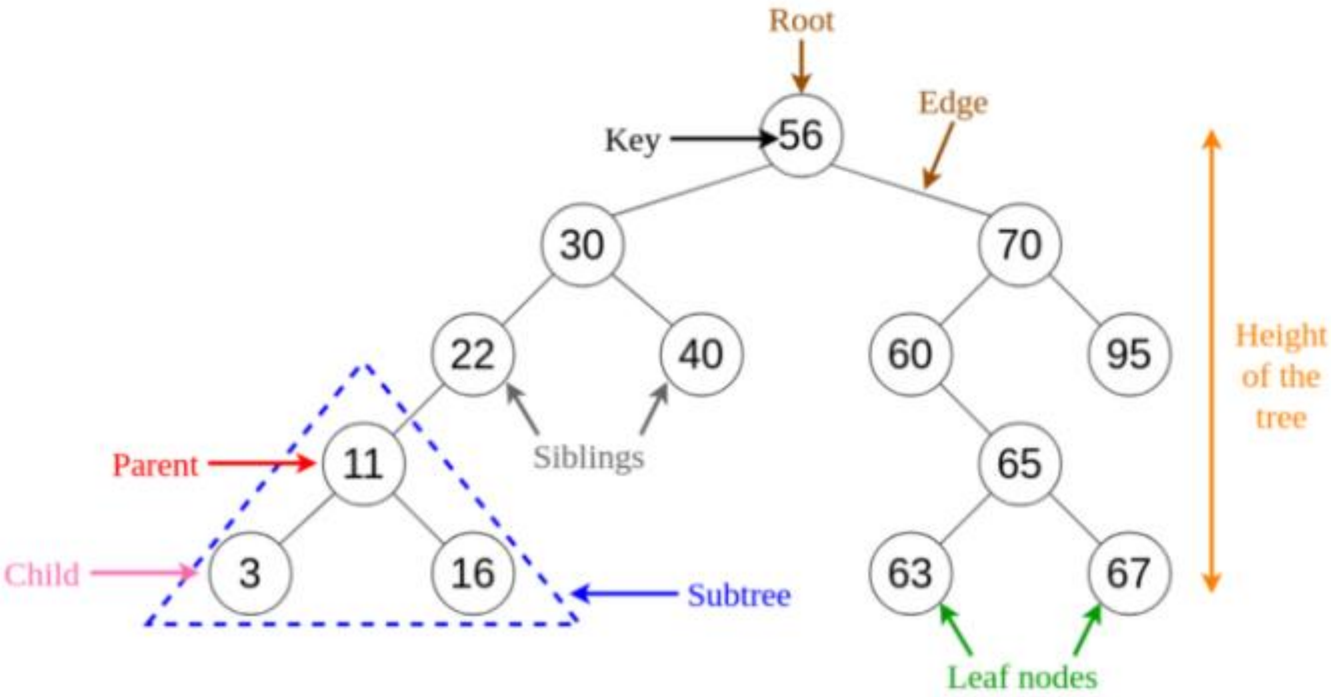
Tree

- A tree is a hierarchical structure where data is organized hierarchically and are linked together
- There are many kinds of tree: binary search tree, B tree, treap, red-black tree, splay tree, AVL tree and n-ary tree…

Tree

- A binary search tree (BST) is a binary tree where data is organized in a hierarchical structure. This data structure stores values in sorted order
- Every node in a binary search tree comprises the following attributes
  - ✓ key: value stored in the node
  - ✓ left: pointer to the left child
  - ✓ right: pointer to the right child
  - ✓ p: pointer to the parent node

Tree

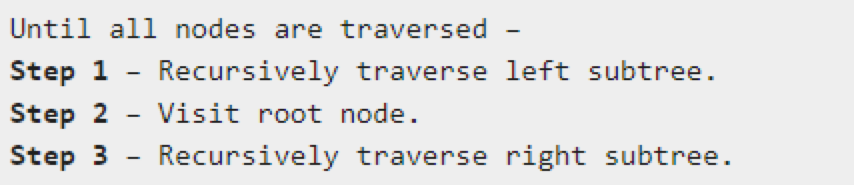

Root
Edge
Key
56
30
70
22
40
60
95
11
65
3
16
63
67
Parent
Child
Siblings
Subtree
Leaf nodes
Height of the tree

Tree Traversal

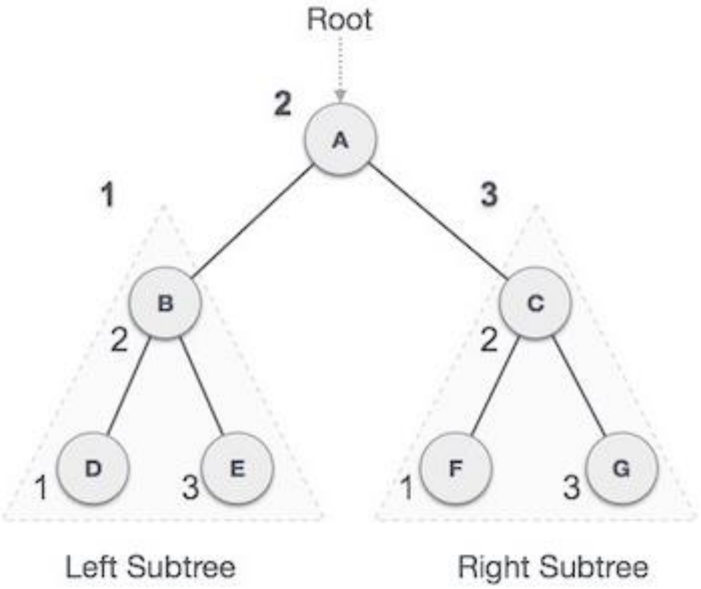
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

In-order Traversal

- Output
  - ✓ $D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$
- Algorithm

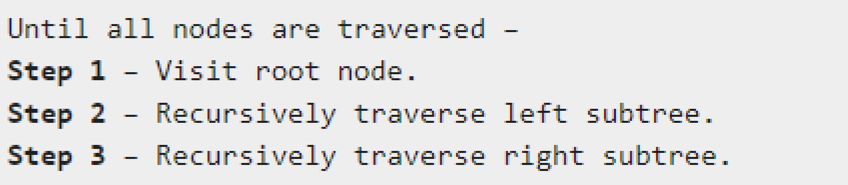


```
Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Visit root node.
Step 3 − Recursively traverse right subtree.
```

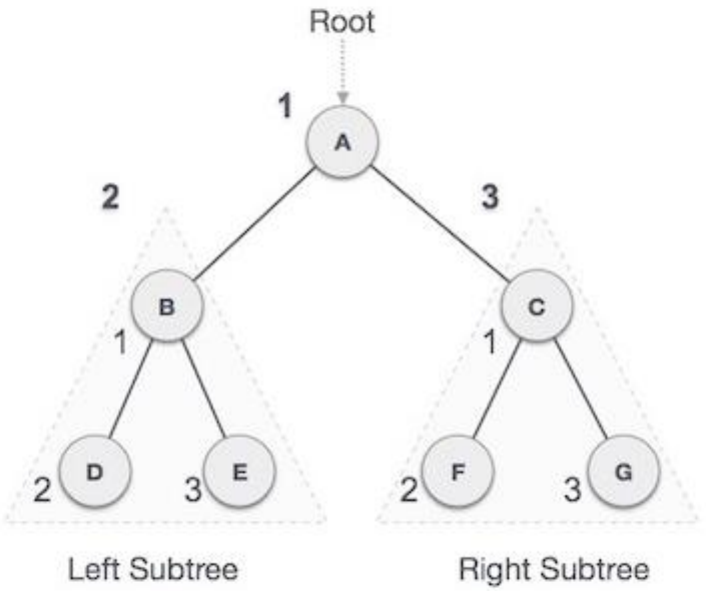


Pre-order Traversal

- Output
  - ✓ $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$
- Algorithm

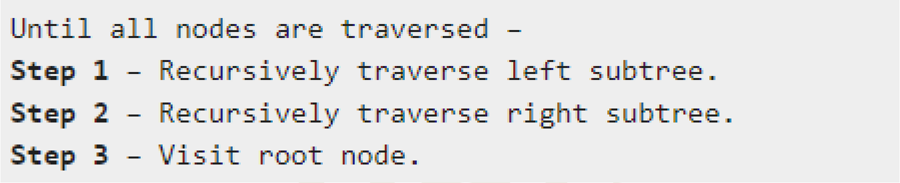


```
Until all nodes are traversed −
Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree.
Step 3 − Recursively traverse right subtree.
```

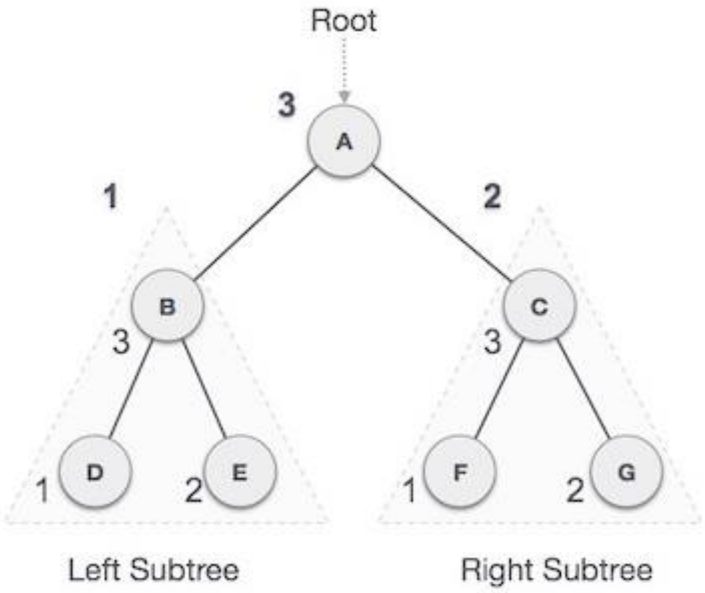


Post-order Traversal

- Output
  - ✓ *D* → *E* → *B* → *F* → *G* → *C* → *A*
- Algorithm



```
Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.
```



Section 2

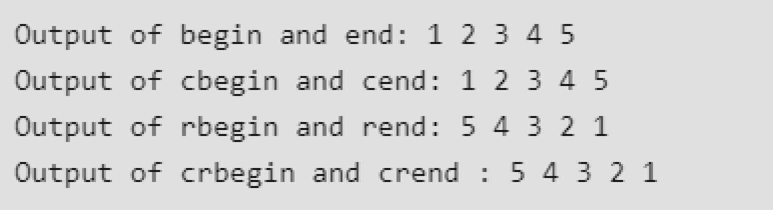
STL data structures

STL data structures. Agenda

- Vector
- Stack
- Queue
- List
- Map

Vector

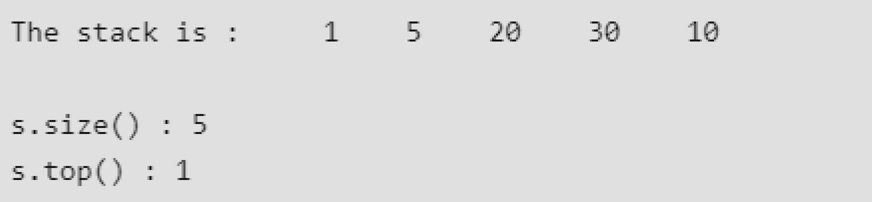
```
int main()
{
   vector<int> g1;
   for (int i = 1; i <= 5; i++)
      g1.push_back(i);

   cout << "Output of begin and end: ";
   for (auto i = g1.begin(); i != g1.end(); ++i)
      cout << *i << " ";

   cout << "\nOutput of cbegin and cend: ";
   for (auto i = g1.cbegin(); i != g1.cend(); ++i)
      cout << *i << " ";

   cout << "\nOutput of rbegin and rend: ";
   for (auto ir = g1.rbegin(); ir != g1.rend(); ++ir)
      cout << *ir << " ";

   cout << "\nOutput of crbegin and crend : ";
   for (auto ir = g1.crbegin(); ir != g1.crend(); ++ir)
      cout << *ir << " ";

   return 0;
}
```



```
Output of begin and end: 1 2 3 4 5
Output of cbegin and cend: 1 2 3 4 5
Output of rbegin and rend: 5 4 3 2 1
Output of crbegin and crend : 5 4 3 2 1
```

Stack

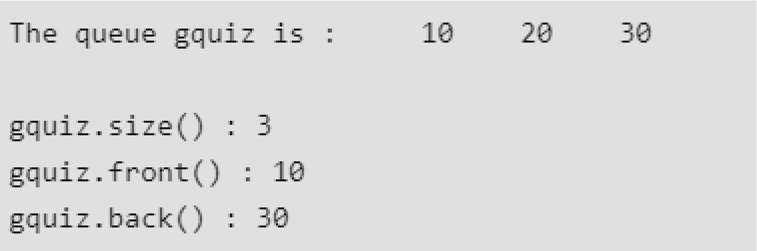
```
void showstack(stack <int> s) {
    while (!s.empty()) {
        cout << '\t' << s.top();
        s.pop();
    }
    cout << '\n';
}

int main () {
    stack<int> s;
    s.push(10);
    s.push(30);
    s.push(20);
    s.push(5);
    s.push(1);

    cout << "The stack is : ";
    showstack(s);

    cout << "\ns.size() : " << s.size();
    cout << "\ns.top() : " << s.top();

    return 0;
}
```



```
The stack is :      1     5     20     30     10


s.size() : 5
s.top() : 1
```

Queue

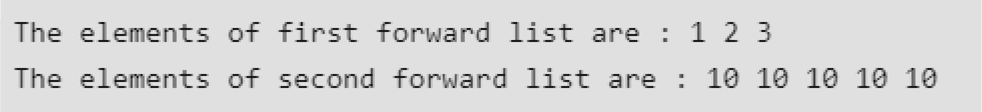
```
void showq(queue <int> gq) {
   queue <int> g = gq;
   while (!g.empty()) {
      cout << '\t' << g.front();
      g.pop();
   }
   cout << '\n';
}

int main() {
   queue<int> gquiz;
   gquiz.push(10);
   gquiz.push(20);
   gquiz.push(30);

   cout << "The queue gquiz is : ";
   showq(gquiz);

   cout << "\ngquiz.size() : " << gquiz.size();
   cout << "\ngquiz.front() : " << gquiz.front();
   cout << "\ngquiz.back() : " << gquiz.back();

   return 0;
}
```



```
The queue gquiz is :     10     20     30


gquiz.size() : 3
gquiz.front() : 10
gquiz.back() : 30
```

List

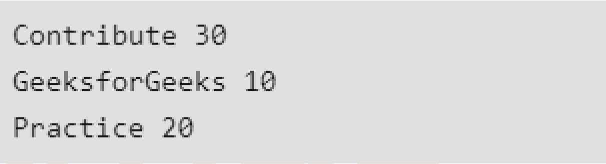
```
int main() {
   // Declaring forward list
   forward_list<int> flist1;
   forward_list<int> flist2;

   // Assigning values using assign()
   flist1.assign({1, 2, 3});

   // Assigning repeating values using assign()
   // 5 elements with value 10
   flist2.assign(5, 10);

   // Displaying forward lists
   cout << "The elements of first forward list are : ";
   for (int&a : flist1)
      cout << a << " ";
   cout << endl;

   cout << "The elements of second forward list are : ";
   for (int&b : flist2)
      cout << b << " ";
   cout << endl;

   return 0;
}
```



```
The elements of first forward list are : 1 2 3
The elements of second forward list are : 10 10 10 10 10
```

Map

```
int main()
{
   // Declaring umap to be of <string, int> type
   // key will be of string type and mapped value will
   // be of double type
   unordered_map<string, int> umap;

   // inserting values by using [] operator
   umap["GeeksforGeeks"] = 10;
   umap["Practice"] = 20;
   umap["Contribute"] = 30;

   // Traversing an unordered map
   for (auto x : umap)
     cout << x.first << " " << x.second << endl;

}
```



```
Contribute 30
GeeksforGeeks 10
Practice 20
```

References

- https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm
- *https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm*
- *https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm*
- *https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_algorithm.htm*
- *https://www.tutorialspoint.com/data_structures_algorithms/circular_linked_list_algorithm.htm*
- *https://www.tutorialspoint.com/data_structures_algorithms/tree_data_structure.htm*
- *https://www.tutorialspoint.com/data_structures_algorithms/tree_traversal.htm*
- *https://www.tutorialspoint.com/data_structures_algorithms/binary_search_tree.htm*
- *https://towardsdatascience.com/8-common-data-structures-every-programmer-must-know-171acf6a1a42*

Lesson Summary

- Basic data structures
- STL data structures

Thank you