Traversal for Binary Tree

Kuan-Yu Chen (陳冠宇)

2020/10/14 @ TR-212, NTUST

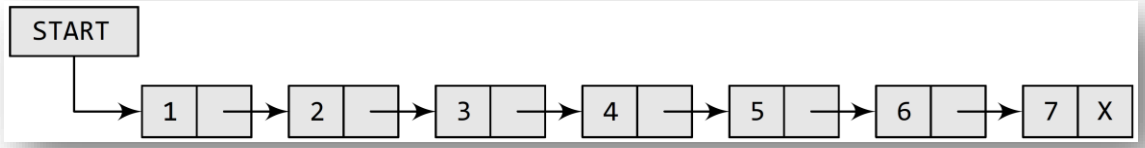
Review

- A tree is a **non-linear** data structure, which is mainly used to store data that is **hierarchical** in nature
  - General Trees
  - Forests
  - Binary Trees
  - Expression Trees
  - Tournament Trees

Linked List.

- A linked list, in simple terms, is a linear collection of data elements
  - Data elements are called **nodes**
  - Each node contains **one or more data fields** and a **pointer** to the next node

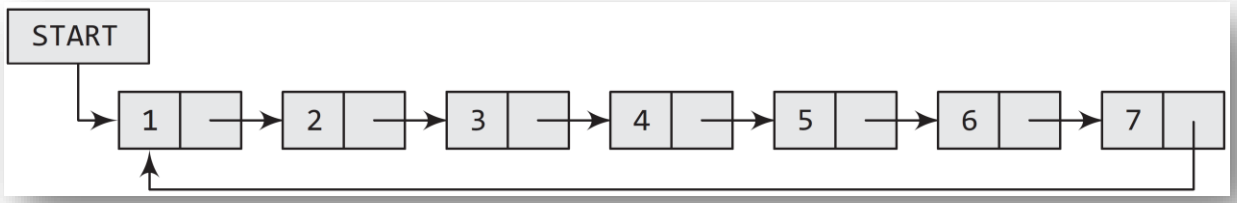
- Singly linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node



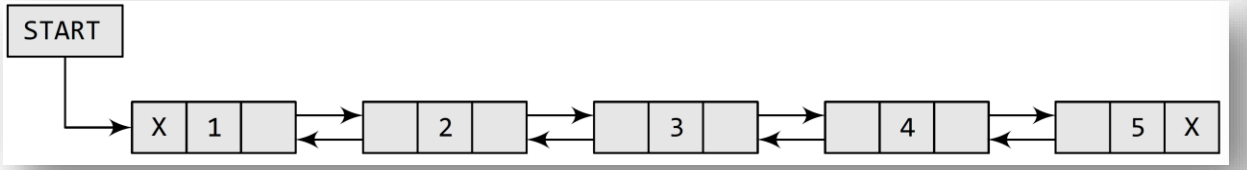

```
struct node
{
        int data;
        struct node *next;
};
```

Linked List..

- Circular linked list is a simple variant, where the last node contains a pointer to the first node of the list



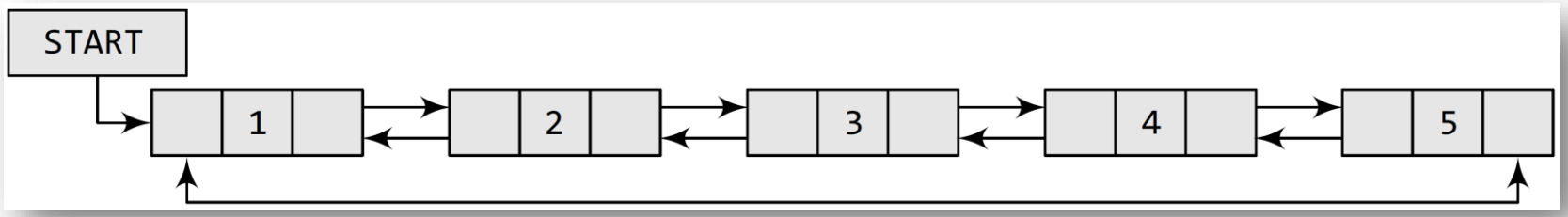
- Doubly linked list or a two-way linked list is a more complex type of linked list
  - It contains a pointer to the next as well as the previous node in the sequence
  - The linked list consists of three parts—data, a pointer to the next node, and a pointer to the previous node




```
struct node
{
        struct node *prev;
        int data;
        struct node *next;
};
```

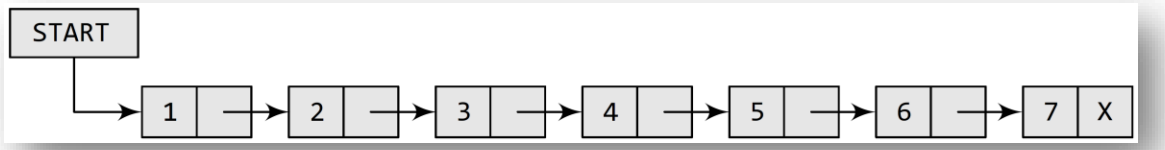
Linked List…

- Circular doubly linked list or a circular two-way linked list is a more complex type of linked list
  - It contains a pointer to the next as well as the previous node in the sequence
    - The next field of the last node stores the address of the first node of the list
    - The previous field of the first field stores the address of the last node



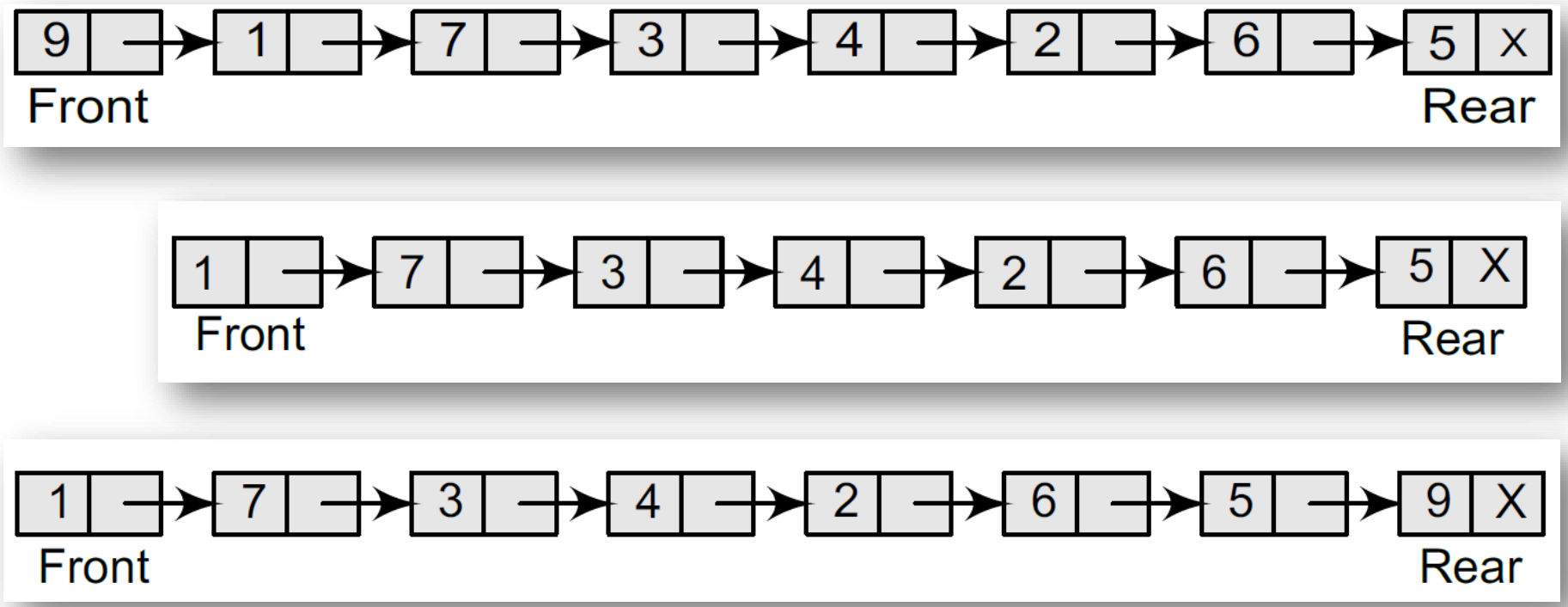
Linked List vs. Array

- Both arrays and linked lists are a linear collection of data elements
  - A linked list does not store its nodes in consecutive memory locations
  - A linked list does not allow random access of data
    - Nodes in a linked list can be accessed only in a sequential manner
  - A linked list can add any number of elements in the list
    - This is not possible in case of an array



Implementation for Queue by Link List.

- Although creating a queue by an array is easy, its drawback is that the array must be declared to have some fixed size
  - If the array size cannot be determined in advance, the linked representation is used



Implementation for Queue by Link List..

- Declare

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node
{
    int data;
    struct node *next;
};
struct queue
{
    struct node *front;
    struct node *rear;
};
struct queue *q;
void create_queue(struct queue *);
struct queue *insert(struct queue *,int);
struct queue *delete_element(struct queue *);
```

Implementation for Queue by Link List...

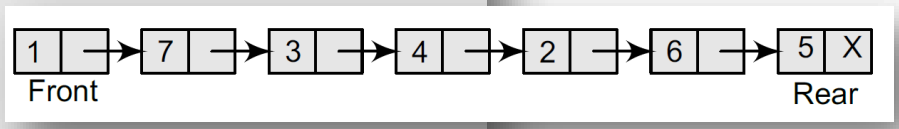
- Create a queue

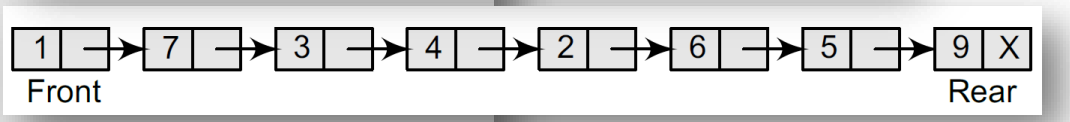
```
void create_queue(struct queue *q)
{
    q->rear = NULL;
    q->front = NULL;
}
```

Implementation for Queue by Link List....

- For insertion

```
struct queue *insert(struct queue *q,int val)
{
    struct node *ptr;
    ptr = (struct node*)malloc(sizeof(struct node));
    ptr->data = val;
    if(q->front == NULL)
    {
        q->front = ptr;
        q->rear = ptr;
        q->front->next = q->rear->next = NULL;
    }
    else
    {
        q->rear->next = ptr;
        q->rear = ptr;
        q->rear->next = NULL;
    }
    return q;
}
```

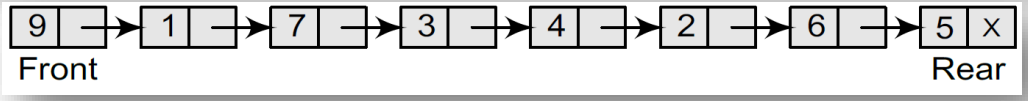


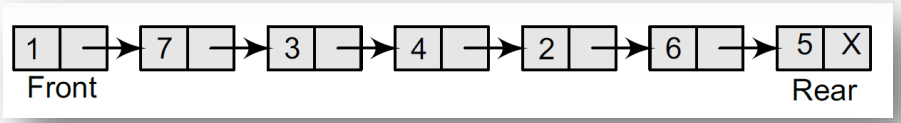


Implementation for Queue by Link List.....

- For deletion

```
struct queue *delete_element(struct queue *q)
{
    struct node *ptr;
    ptr = q -> front;
    if(q -> front == NULL)
        printf("\n UNDERFLOW");
    else
    {
        q -> front = q -> front -> next;
        printf("\n The value being deleted is : %d", ptr -> data);
        free(ptr);
    }
    return q;
}
```



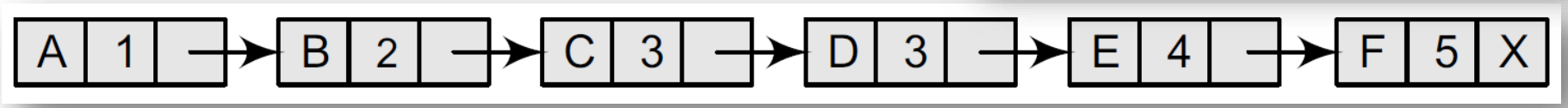


Priority Queue

- Linked Representation of a Priority Queue
  - Every node of the list will have three parts:
    1. the information or data part
    2. the priority number of the element
    3. the address of the next element

FRONT	REAR
3	3
1	3
4	5
4	1

	1	2	3	4	5
1			A		
2	B	C	D		
3				E	F
4	I			G	H

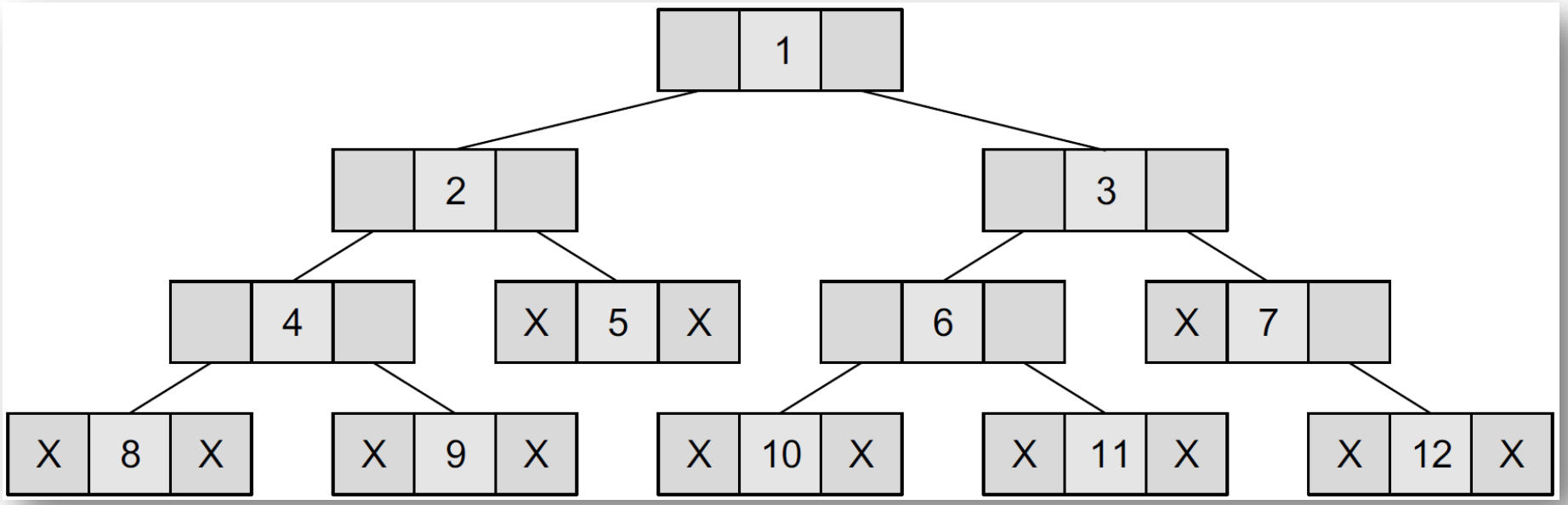


  - From the example
    - Since *A* has a priority number 1 and *B* has a priority number 2, then *A* will be processed before *B* as it has higher priority than *B*
    - We cannot make out whether *A* was inserted before *E* or whether *E* joined the queue before *A*
    - We can definitely say that *C* was inserted in the queue before *D* because when two elements have the same priority

Binary Trees

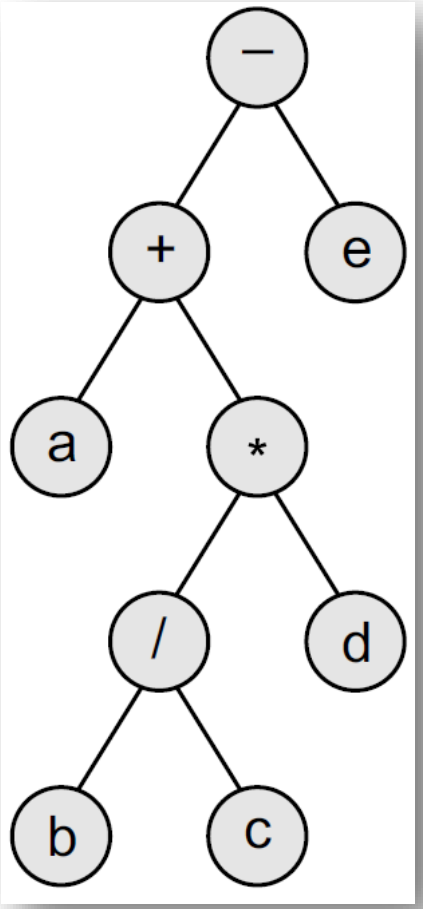
- In the linked representation of a binary tree, every node will have three parts: the data element, a pointer to the left node, and a pointer to the right node

```
struct node {
        struct node *left;
        int data;
        struct node *right;
};
```



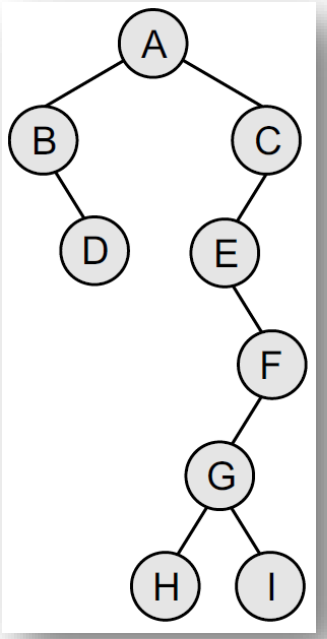
Traversing Binary Tree.

- Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way
  - There are different algorithms for tree traversals
    - Pre-order Traversal
    - Post-order Traversal
    - In-order Traversal
    - Level-order Traversal
  - Take $a + b \div c \times d - e$ for example
    - $(\{a + [(b \div c) \times d]\} - e)$
    - Pre-order: $- + a \times \div bcde$
    - Post-order: $abc \div d \times + e -$
    - In-order: $a + b \div c \times d - e$
    - Level-order: $- + ea \times \div dbc$



Traversing Binary Tree..

- Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way
  - There are different algorithms for tree traversals
    - Pre-order Traversal
      - $ABDCEFGHI$
    - Post-order Traversal
      - $DBHIGFECA$
    - In-order Traversal
      - $BDAEHGIFC$
    - Level-order Traversal
      - $ABCDEFGHI$

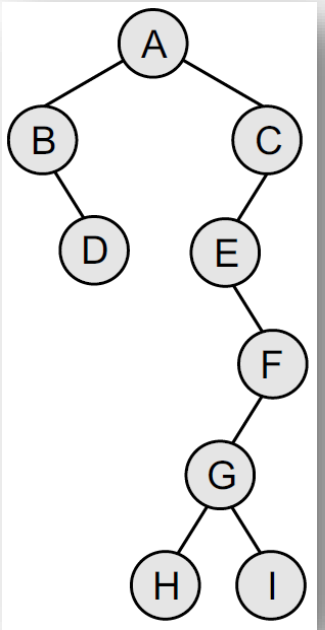


  - Different algorithms differ in the order in which the nodes are visited

In-order

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:             INORDER(TREE -> LEFT)
Step 3:             Write TREE -> DATA
Step 4:             INORDER(TREE -> RIGHT)
        [END OF LOOP]
Step 5: END
```

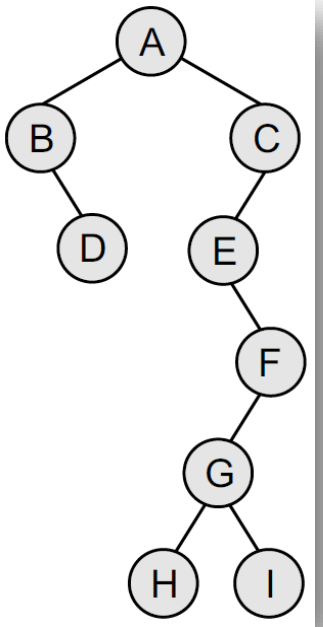
- In-order: BDAEHGIFC



Pre-order

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:             Write TREE -> DATA
Step 3:             PREORDER(TREE -> LEFT)
Step 4:             PREORDER(TREE -> RIGHT)
        [END OF LOOP]
Step 5: END
```

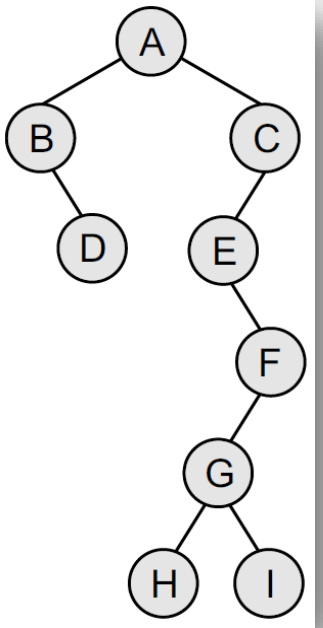
- Pre-order: ABDCEFGHI



Post-order

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:             POSTORDER(TREE -> LEFT)
Step 3:             POSTORDER(TREE -> RIGHT)
Step 4:             Write TREE -> DATA
        [END OF LOOP]
Step 5: END
```

- Post-order: DBHIGFECA

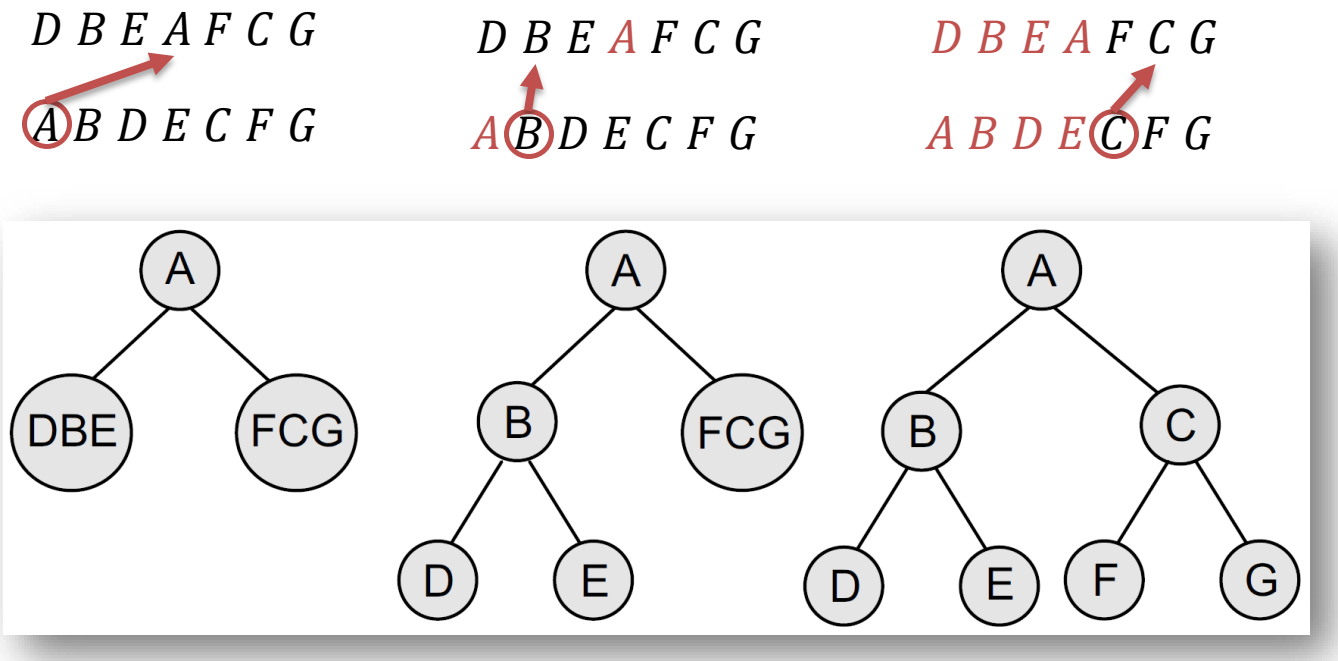


Constructing Binary Tree from Traversal.

- We can construct a binary tree if we are given at least two traversal results
  - In-order traversal
    - The in-order traversal result will be used to **determine the left and the right child nodes**
  - Either pre-order or post-order traversal
    - The pre-order/post-order can be used to **determine the root node**

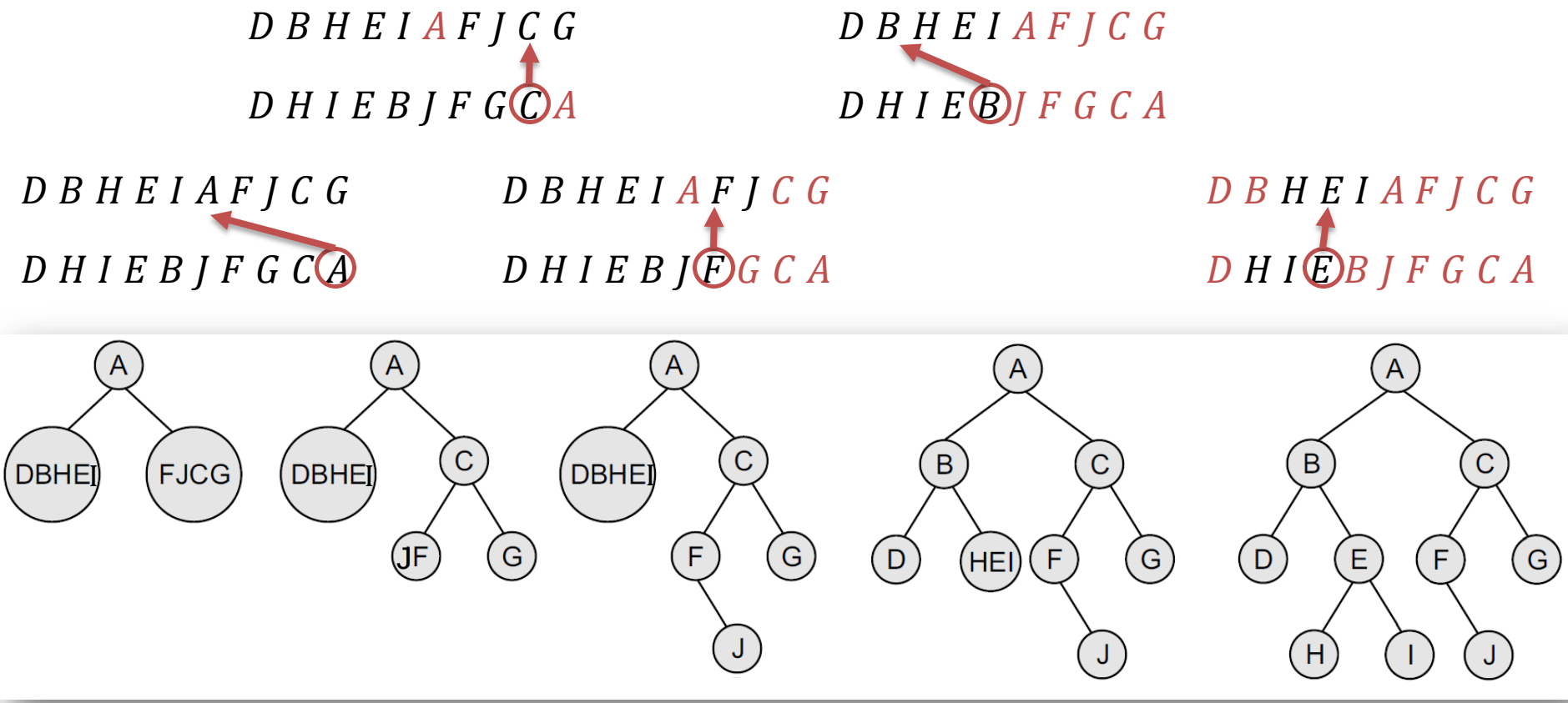
Constructing Binary Tree from Traversal..

- Take in-order + pre-order for example
  - In-order: $D\ B\ E\ A\ F\ C\ G$
  - Pre-order: $A\ B\ D\ E\ C\ F\ G$



Constructing Binary Tree from Traversal…

- Take in-order + post-order for example
  - In-order: *D B H E I A F J C G*
  - Post-order: *D H I E B J F G C A*

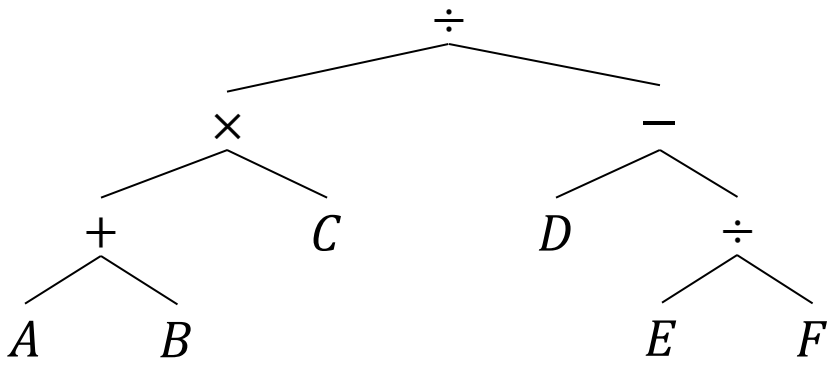


Constructing Binary Tree from Traversal….

- Steps for constructing a binary tree from traversal sequences
  1. Use the pre-order/post-order sequence to determine the root node of the tree
  2. Elements on the left side of the root node in the in-order traversal sequence form the left sub-tree of the root node
  3. Similarly, elements on the right side of the root node in the in-order traversal sequence form the right sub-tree of the root node
  4. Recursively select each element from pre-order/post-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence

By Looking!.

- Given a infix expression $(A + B) \times C \div (D - E \div F)$, please write down the prefix and postfix expressions

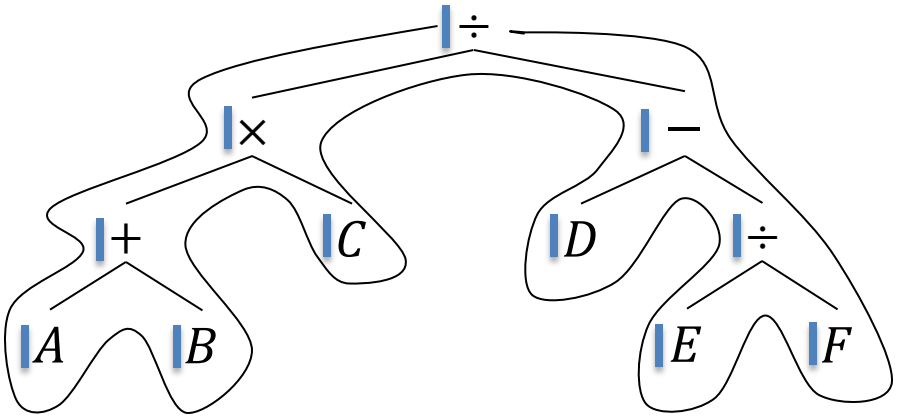


prefix | *element* | postfix

infix

By Looking!..

- Given a infix expression $(A + B) \times C \div (D - E \div F)$, please write down the prefix and postfix expressions
  - Prefix



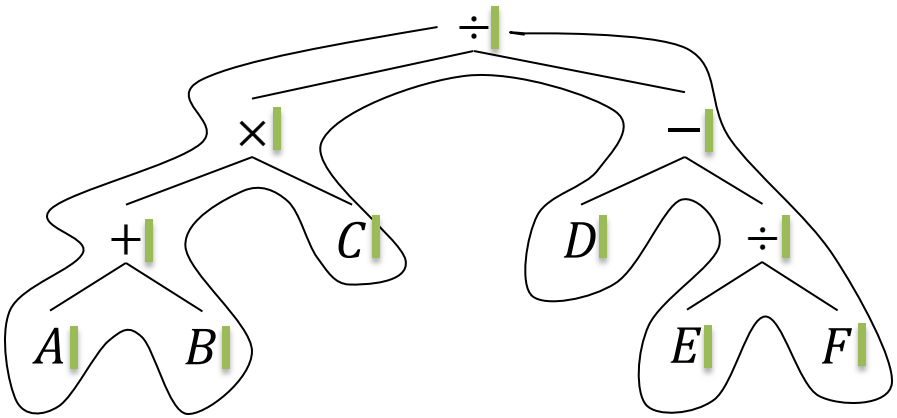
prefix | *element* | postfix

infix

$\div \times + ABC - D \div EF$

By Looking!...

- Given a infix expression $(A + B) \times C \div (D - E \div F)$, please write down the prefix and postfix expressions
  - Postfix



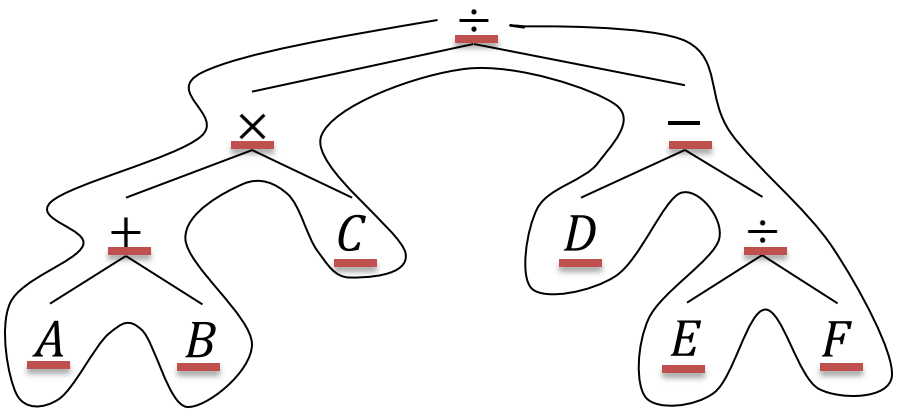
prefix | *element* | postfix

infix

$AB + C \times DEF \div - \div$

By Looking!....

- Given a infix expression $(A + B) \times C \div (D - E \div F)$, please write down the prefix and postfix expressions
  - Infix



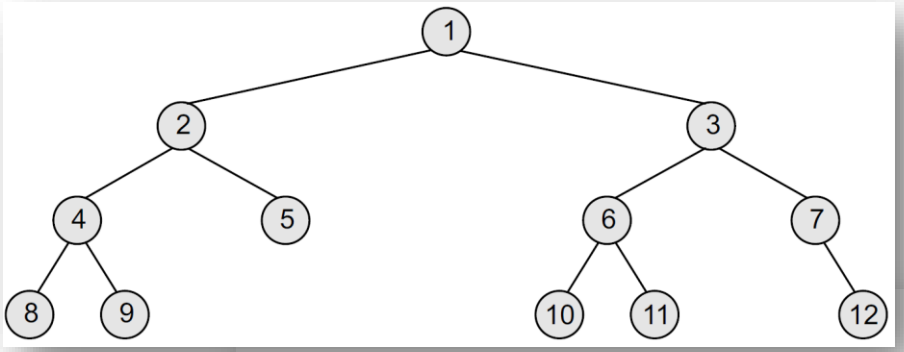
prefix | *element* | postfix

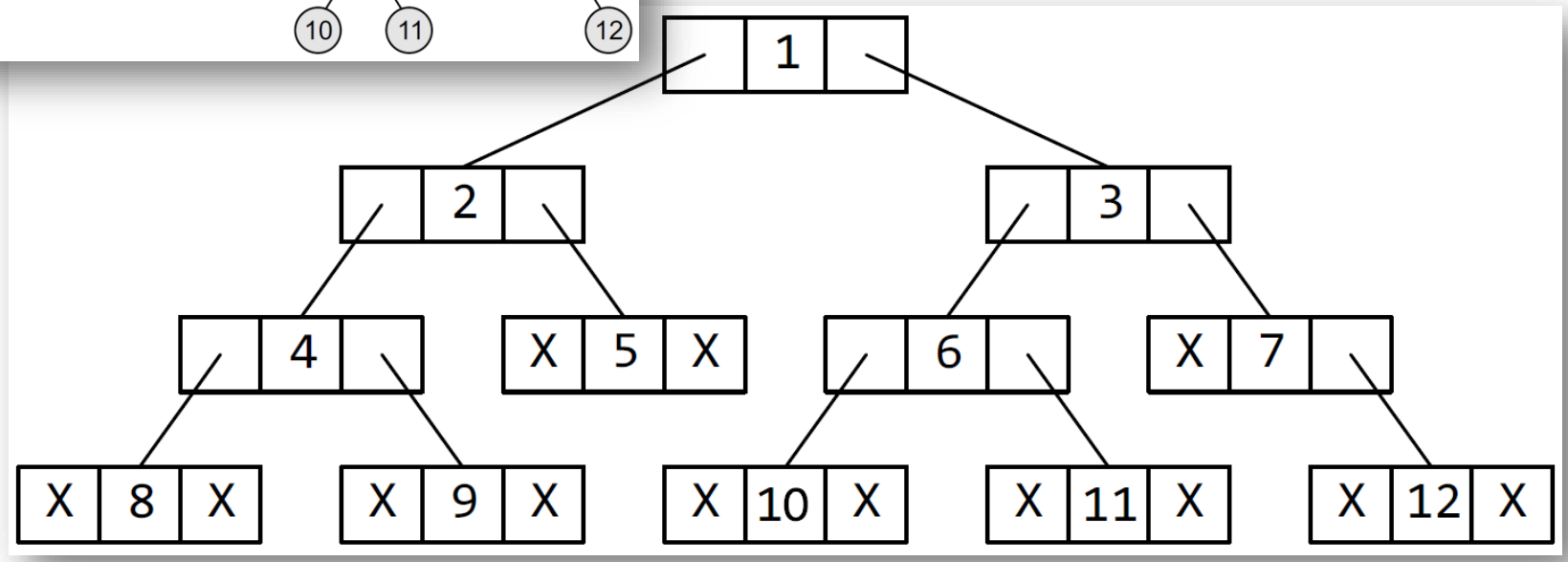
infix

$A + B \times C \div D - E \div F$

Threaded Binary Trees

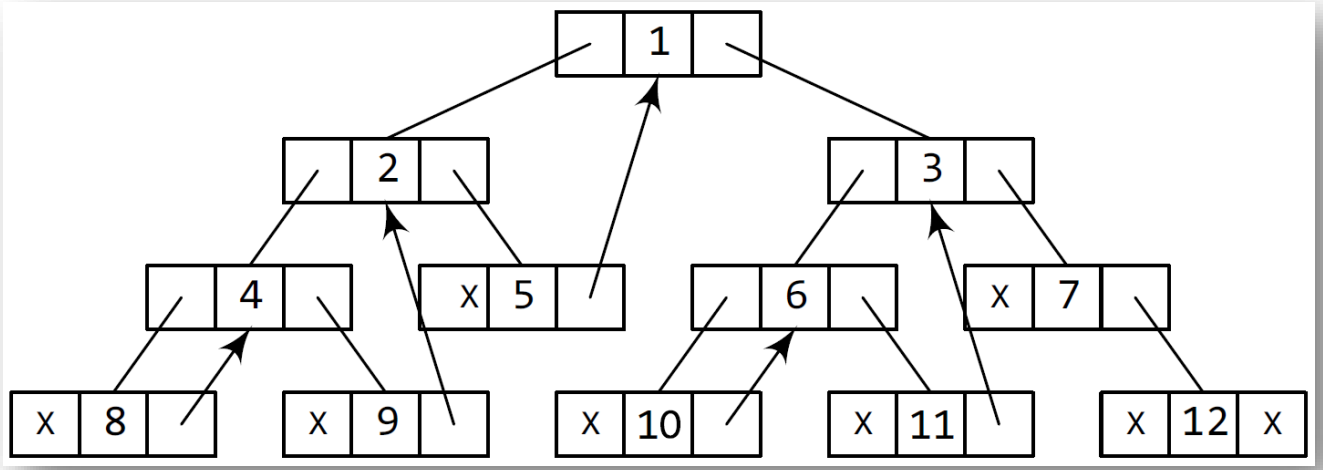
- A threaded binary tree is the same as that of a binary tree but with a difference in storing the NULL pointers
  - The space that is wasted in storing a NULL pointer can be efficiently used to store some other useful piece of information





One-way Threaded Trees

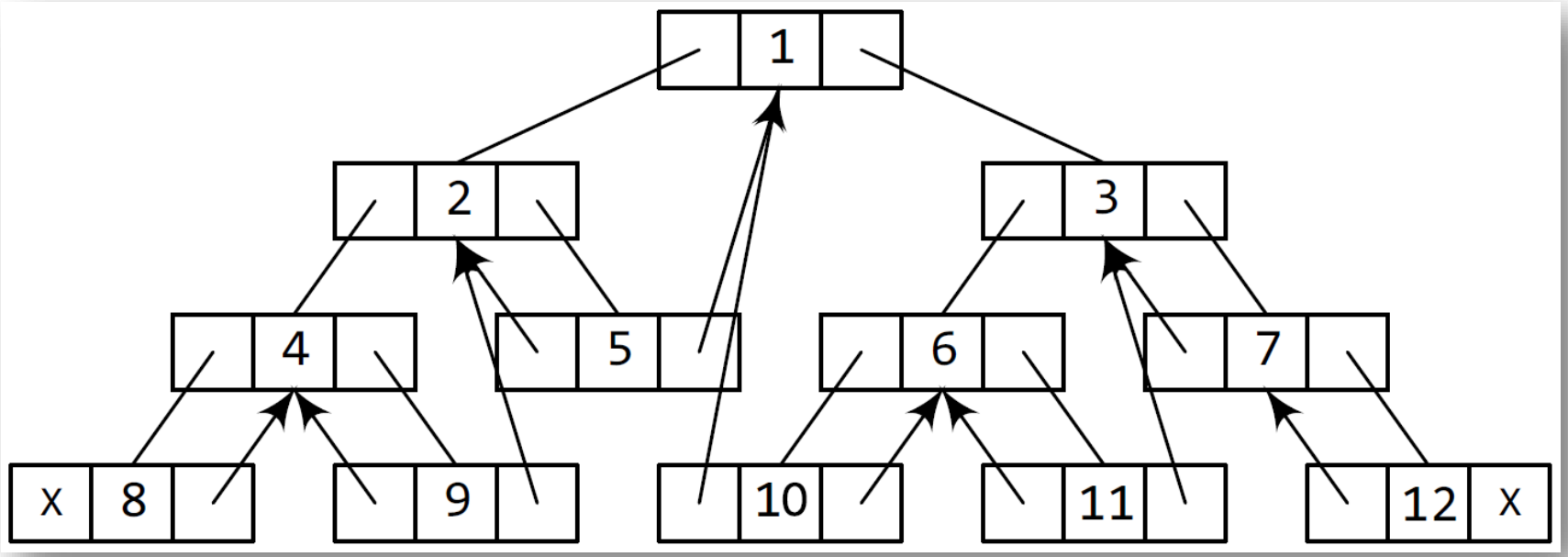
- A **one-way threaded tree** is also called a **single-threaded tree**
  - If the thread appears in the **right field**, then it will point to the **in-order successor** of the node
    - Such a tree is called a **right-threaded binary tree**



  - If the thread appears in the **left field**, then the left field will be made to point to the **in-order predecessor** of the node
    - Such a tree is called a **left-threaded binary tree**

Two-way Threaded Trees

- In a two-way threaded tree, also called a **double-threaded tree**, threads will appear in both the left and the right field of the node
  - The left field will point to the in-order predecessor of the node, and the right field will point to its successor
  - A two-way threaded binary tree is also called a **fully threaded binary tree**



Questions?

kychen@mail.ntust.edu.tw