Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

C Programming

User Defined Data types in C & Data Structures using C

Education and Research

Unit Plan

- Need for Structures
- Introduction to structures
- Pointer to structures.
- Array of structures.
- Passing structures to functions (by value and by reference)
- Unions
- Introduction to data structures
  - Stacks Queues & Link Lists (singly, doubly and circular)
  - Trees, Binary Trees & Binary Search Trees.

Introduction to structures

- **Need for Structures**

  – Built in Data types in C are not sufficient to represent real world entities.

  – Arrays is a collection of variables of similar datatypes. Many a times we need to store variables of dissimilar datatypes.

Introduction to structures

- Structure Declaration

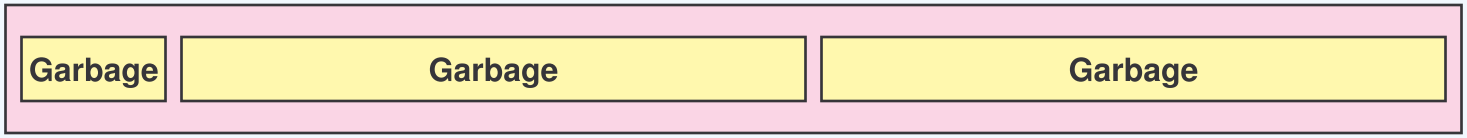
```
struct  Nametag
{
        DataType VariableName;
        DataType VariableName;
        .......
 };
```

- Structure variable declaration

```
struct  Nametag
{
        DataType VariableName;
        DataType VariableName;
        .......
 }Var1, Var2;
```

Introduction to structures (Contd...)

- **Memory Allocation for a structure**
  - Size of the variable is equal to sum of the memory required for the individual data members in the structures**.**



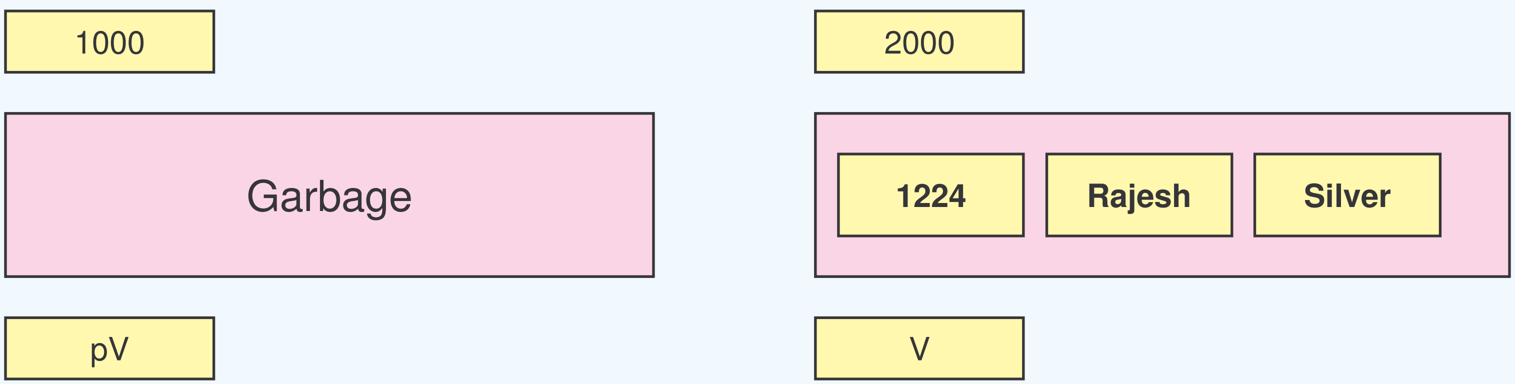
Introduction to structures (Contd...)

Structure variable initialization

1224	Rajesh	Silver

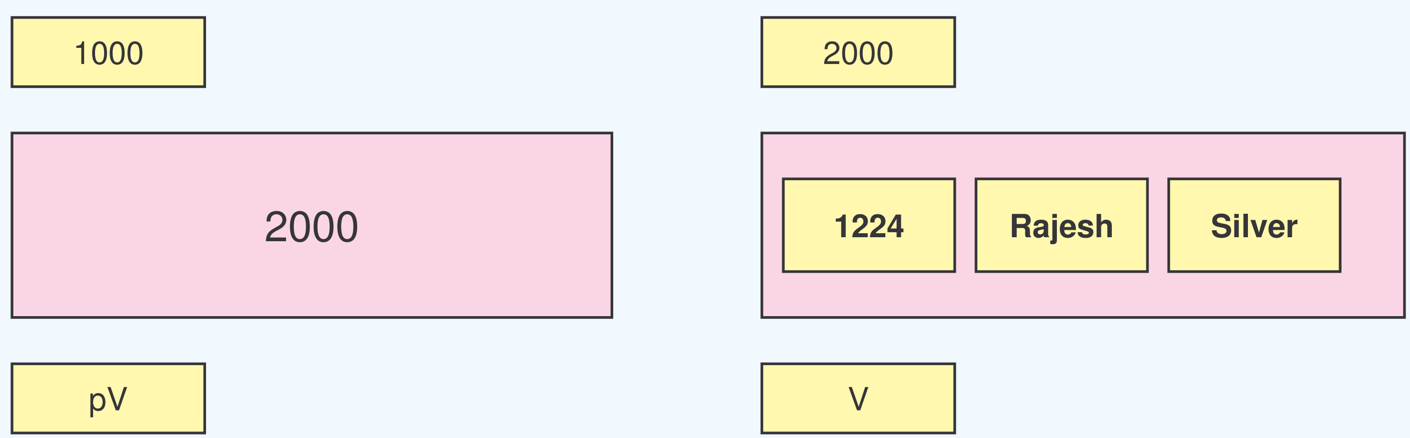
Pointers to structures

- **Declaring a pointer to a structure**
  - struct Vehicle V={1224, “Rajesh”, ”Silver”}
  - struct Vehicle *pV;



Pointers to structures

- **Initializing the Pointer with the address of the structure variable**
  - pV=&V;

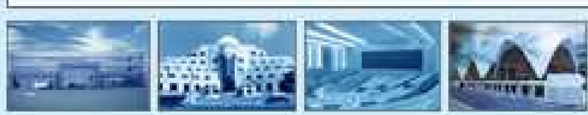


Declaring array of structures

- We can declare an array of structure variable and the syntax for the same is given below.
  - struct Vehicle V[3];
  - V is an array of 3 Vehicle variables

1224	Rajesh	Silver	1225	Ramesh	white	1226	Rakesh	black
	0th element			1st element			2nd element	

- Accessing elements in an array using the array subscript operator.
  - V[0].reg_number=1224;



Passing structure variables to functions by value

- Structures variables can be passed to functions by value.
- Function which accepts a structure variable by value

```
void Display (Vehicle V)
{
    printf("\n The registration number of the vehicle is %d",V.reg_number);
}
```

- Call to a function which accepts a structure variable by value

```
void main()
{
    Vehicle V={1224, "Rajesh", "Silver"};
    Display (V);
}
```

Passing structure variables to Functions by reference

- Structures variables can be passed to functions by reference.
- Function which accepts a structure variable by reference

```
 void Display (Vehicle *V)
{
    printf("\n The registration number of the vehicle is %d",V->reg_number);
}
```

- Call to a function which accepts a structure variable by reference

```
void main()
{
    Vehicle V={1224, "Rajesh", "Silver"};
    Display (&V);
}
```

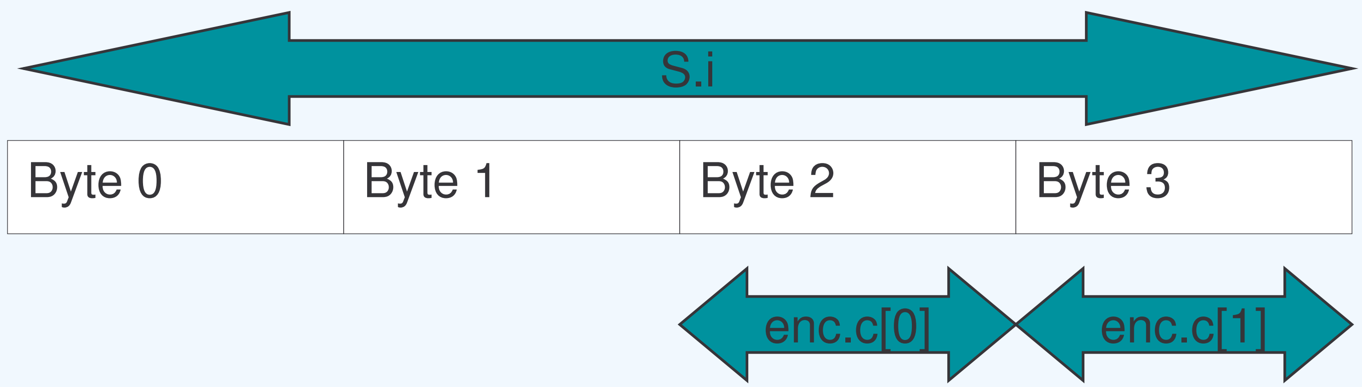
Introduction to unions

- Union is another mechanism to create user defined data types in C
- Union is a single piece of memory that is shared by 2 or more variables.
- The variables that share the memory may be of the same type or different types.
- Only one variable in the union can be in use at any point of time.

Introduction to unions (Contd…)

- Size of the union is fixed at compile time and it’s large enough to accommodate the largest element in the union.

```
union encrypt
{
    int i;
    char c[2];
}enc;
```



Introduction to unions (Contd…)

- Accessing elements of a union is similar to accessing elements of a structure

```
union encrypt
{
    int i;
    char c[2];
}enc;
enc.i=10;
enc.c[0]='a';
enc.c[1]='b';
```

Introduction to Data Structures

- Linear Data Structures
  - Arrays.
  - Stacks.
  - Queues.
  - Linked Lists.
- Hierarchical Data Structures
  - Trees.
  - Graphs.

Introduction to Data Structures (Stacks)

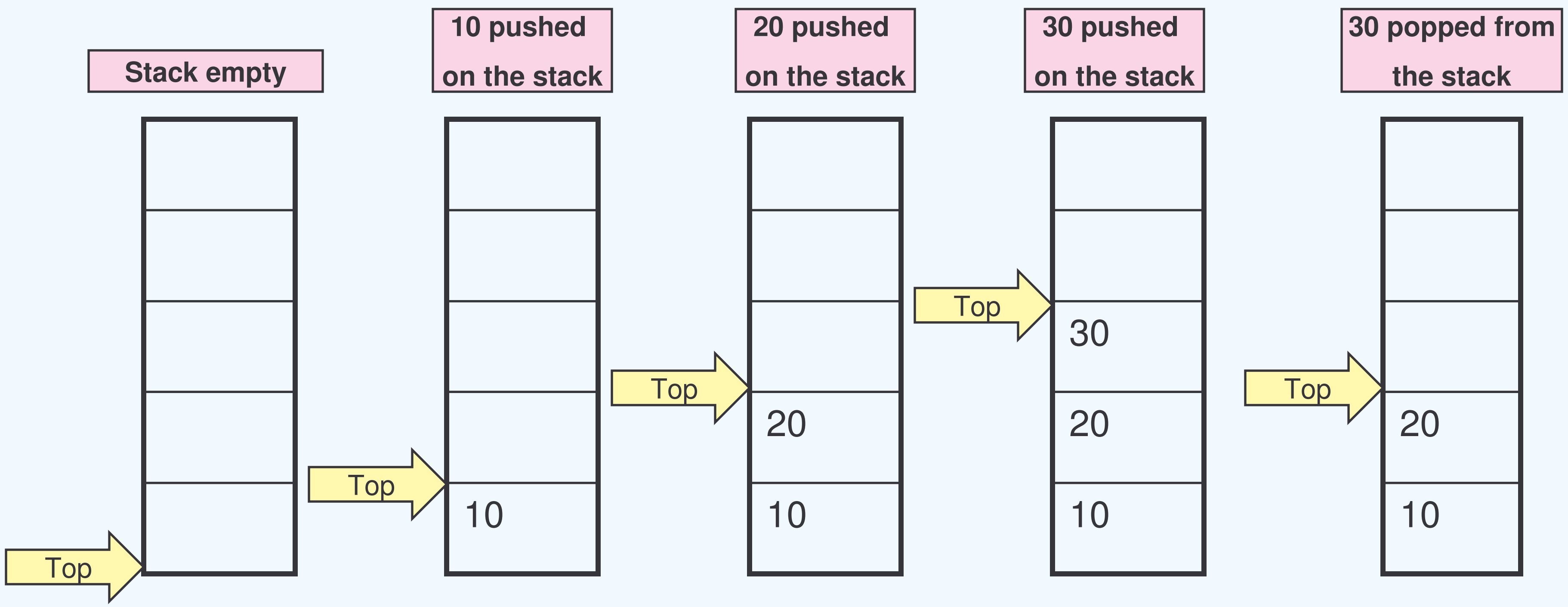
- Stack is a Linear data structure in which addition of new element or deletion of existing element always takes place at the same end.

- The end at which addition and deletion takes place is called as the top of the stack

- The Element which is entered most recently is the first to be removed from the stack i.e. Last in First Out (LIFO)

Introduction to Data Structures (Stacks Contd…)

- Illustration of a Stack



Introduction to Data Structures (Queues)

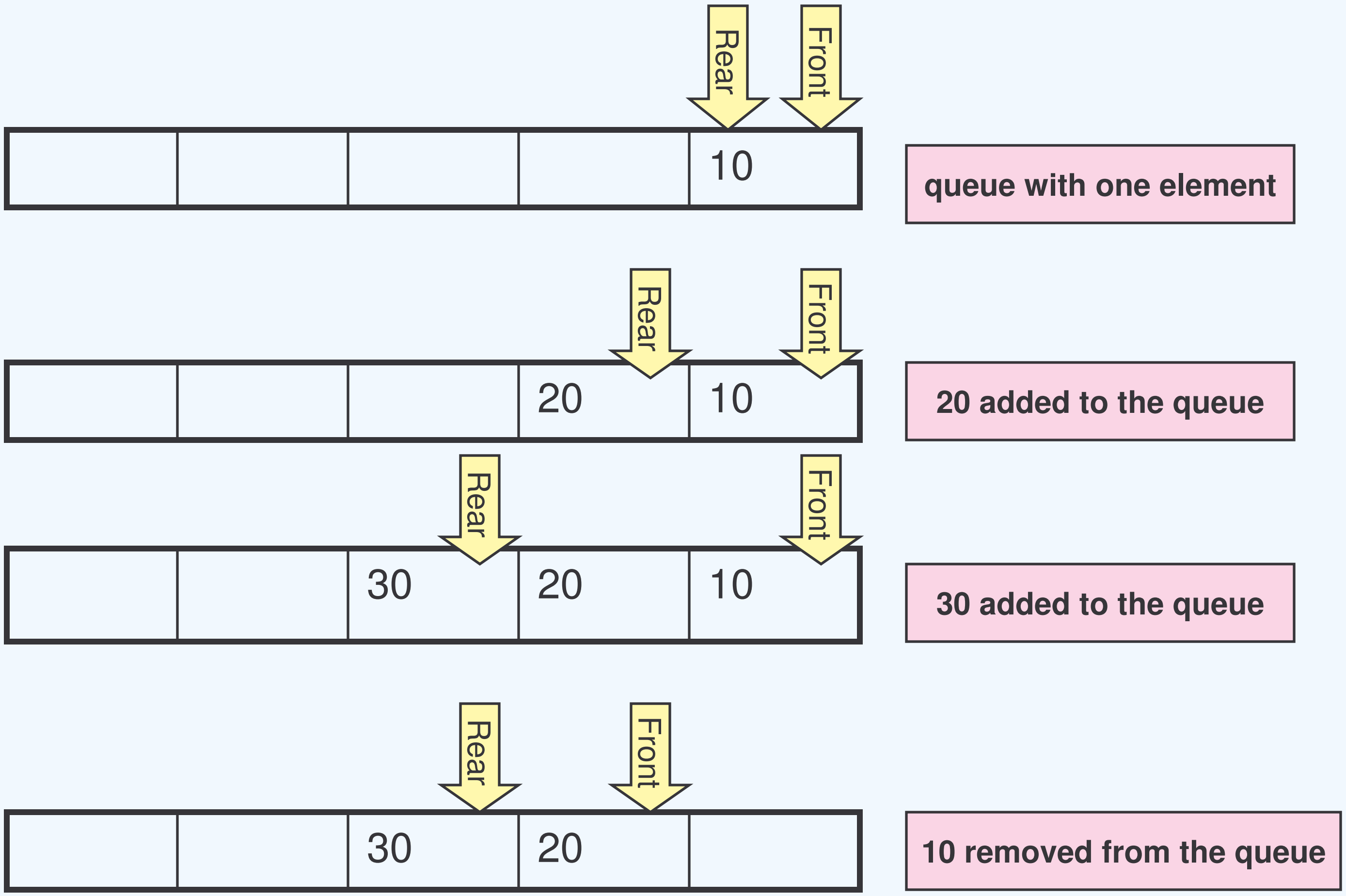
- Queue is a data structure in which addition of new element is done at the rear of the queue and deletion of an element is done from the front of the queue.

- The first element to enter the queue is the first one to go out i.e. First in First out (FIFO).

- Queue is a Linear Data structure.

Introduction to Data Structures (Queues)

- Illustration of a Queue



Introduction to Data Structures (Linked List)

Singly Linked List

- It is a linear data structure which stores data in nodes.

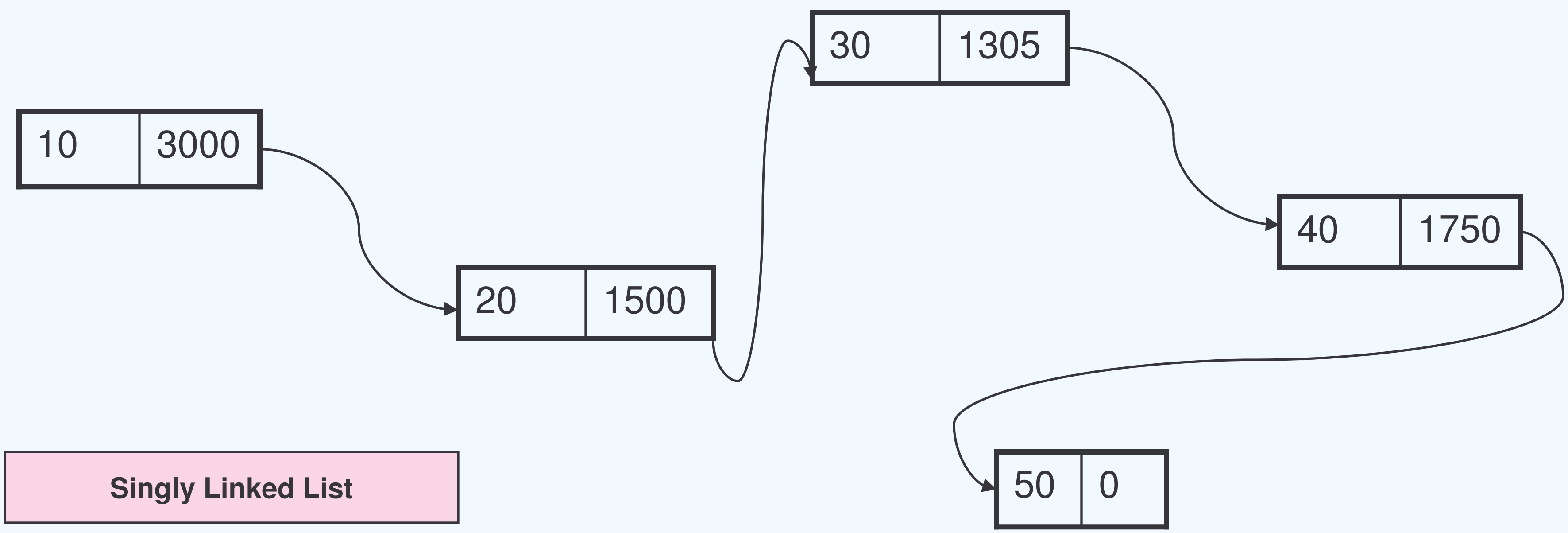
- Every node has 2 sections.
  - Data Section (stores the data).
  - Address section (stores the address of the next node).

- If we have the address the first node we can traverse through the entire linked list.

- The last node maintains a null in its address section to indicate the end of the linked list.

Introduction to Data Structures (Linked List Contd…)

- Illustration of a singly linked list



Singly Linked List

Introduction to Data Structures (Linked List Contd…)

Limitations of a Singly Linked List.

- The traversal of the nodes is possible in only one direction as the node doesn't store the address of the previous node.

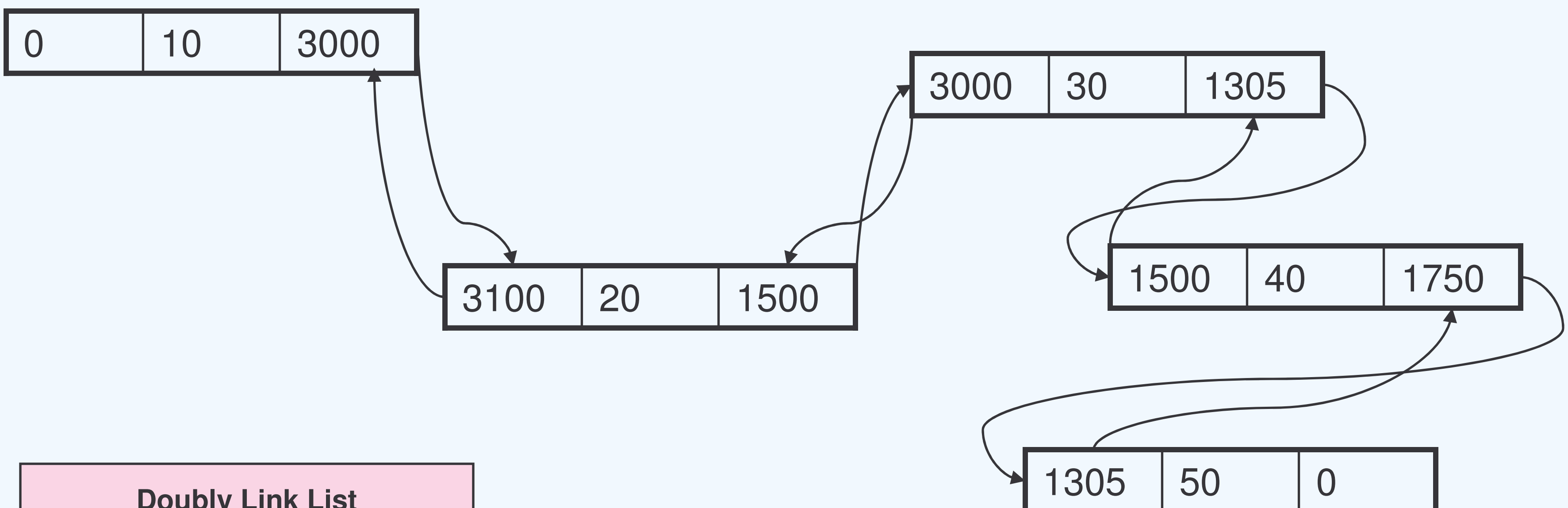
Introduction to Data Structures (Link List Contd…)

Doubly Linked List

- The nodes in the doubly linked list has 3 sections
  - Address section (address of the previous node)
  - data section (data in the node)
  - Address section (address of the next node)

Introduction to Data Structures (Link List Contd...)

Illustration of Doubly Linked List



Doubly Link List

Introduction to Data Structures (Link List Contd…)

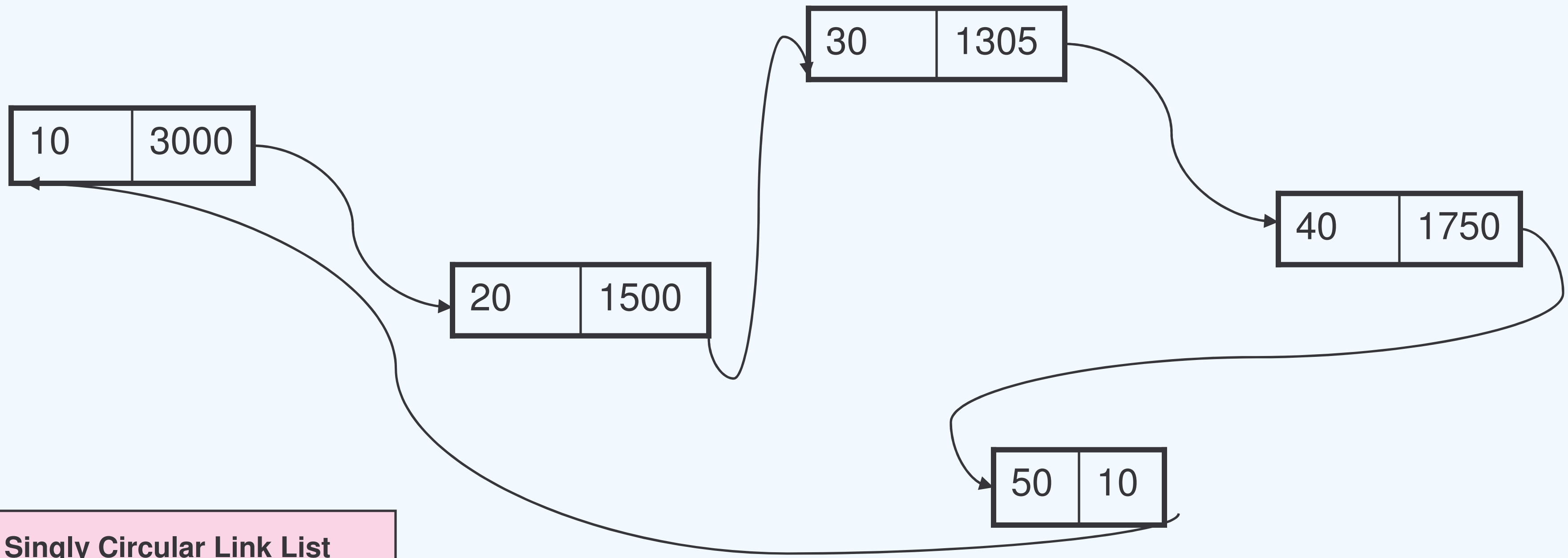
Circular Linked List

- In a singly circular linked list the **next** section of the last node stores the address of the first node

- In a doubly circular linked list the **next** section the last node stores the address of the first node and the **previous** section of the first node stores the address of the last node.

Introduction to Data Structures (Link List Contd…)

- Illustration of a circular linked list.



Singly Circular Link List

Introduction to Data Structures (Trees)

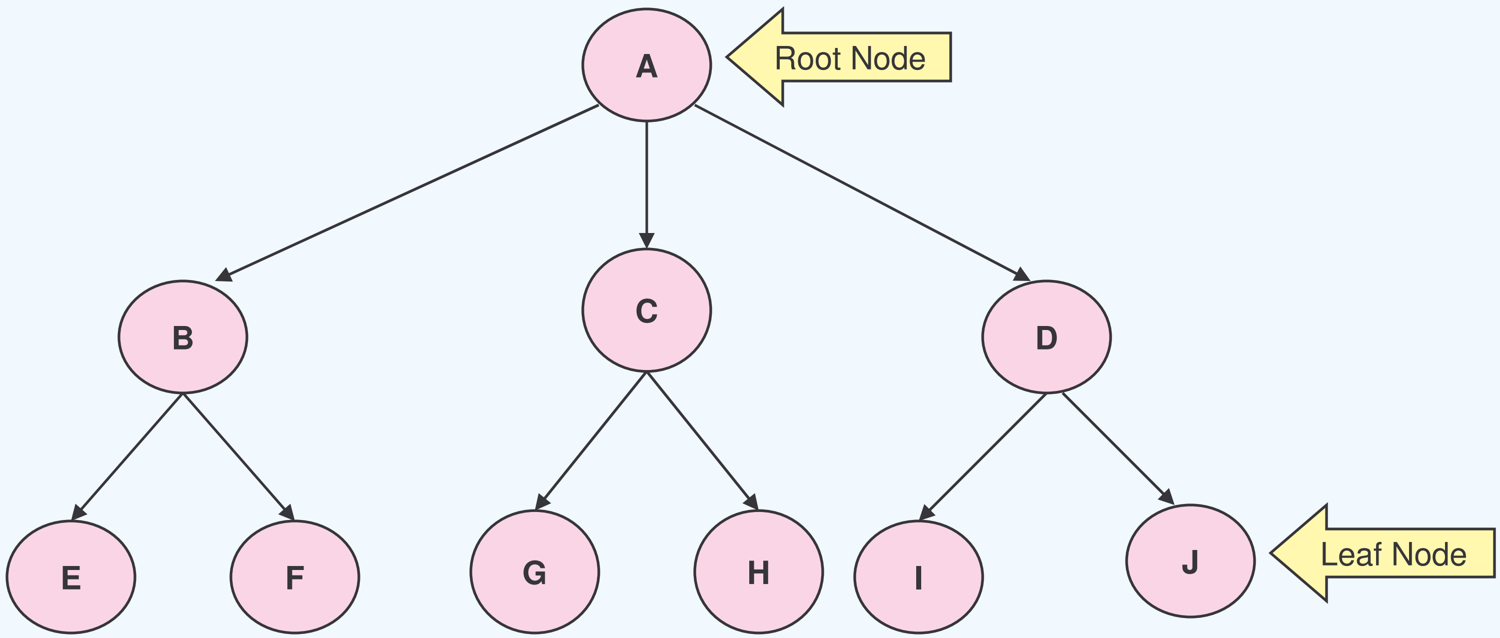
- Tree is a Hierarchical data structures.

- A **Tree** *t* is a finite nonempty set of elements.

- One of the element is called as the root and the remaining (if any) are portioned into trees which are called as **subtrees** of *t*

Introduction to Data Structures (Trees)

Example of Tree

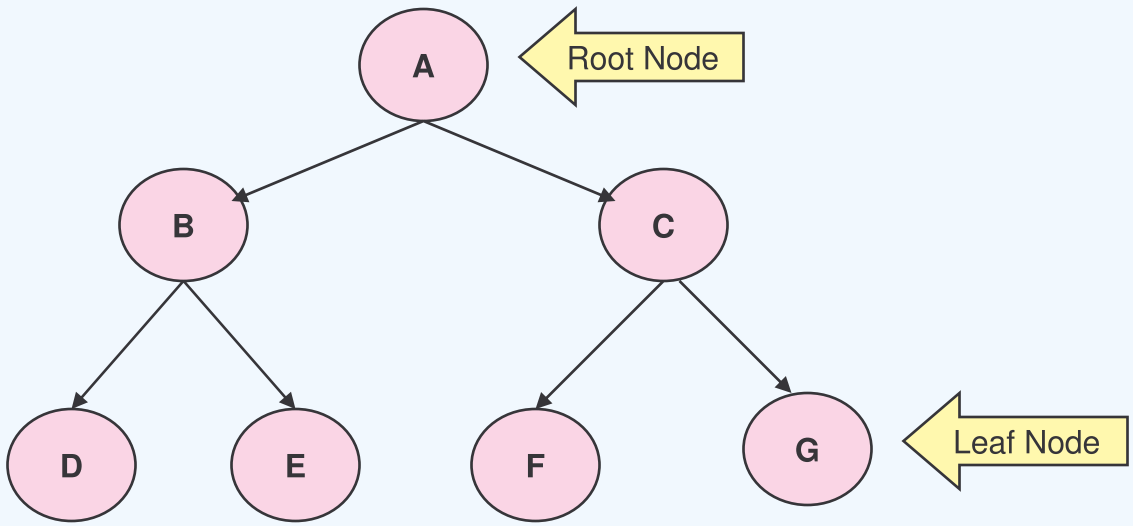


Introduction to Data Structures (Binary Tree)

- Binary Tree is a specialized tree where every node can have maximum of 2 child's.
- Each node in a binary tree has 3 sections.
  - Address of the left child, Data and Address of the right child
- The nodes which don’t have any left & right child's are termed as leaf nodes.
- A tree is said to be Binary tree if left child & right child of the root element are Binary trees

Introduction to Data Structures (Binary Tree)

Example of Binary Tree

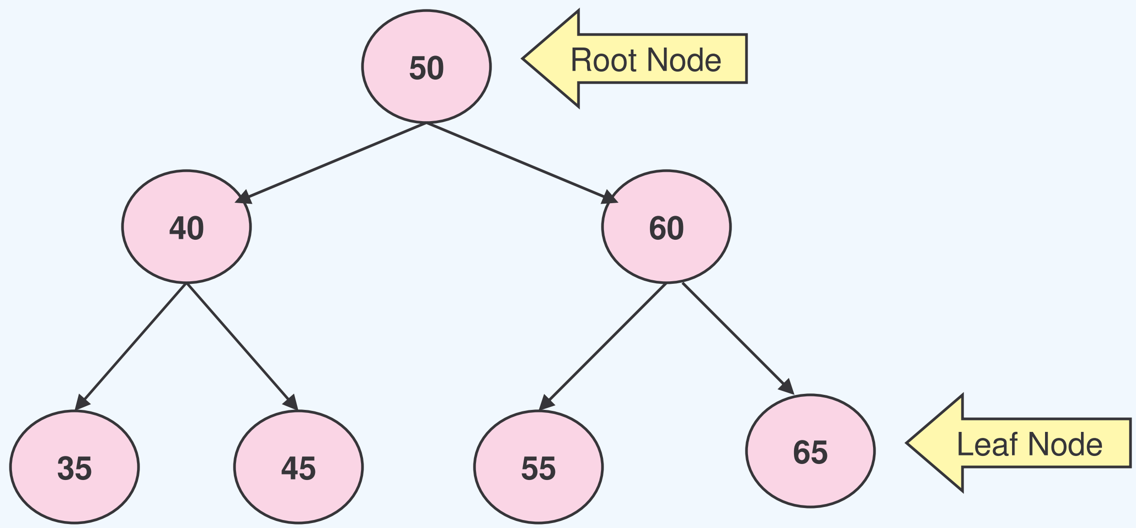


Introduction to Data Structures (Binary Search Tree)

- Binary Search Tree is a specialized Binary Tree where
  - The Elements to the left of the root should have values less than the root element.
  - The Elements to the right of the root should have values greater than the root element.

Introduction to Data Structures (Binary Search Tree Contd…)

- Example of a Binary Search Tree

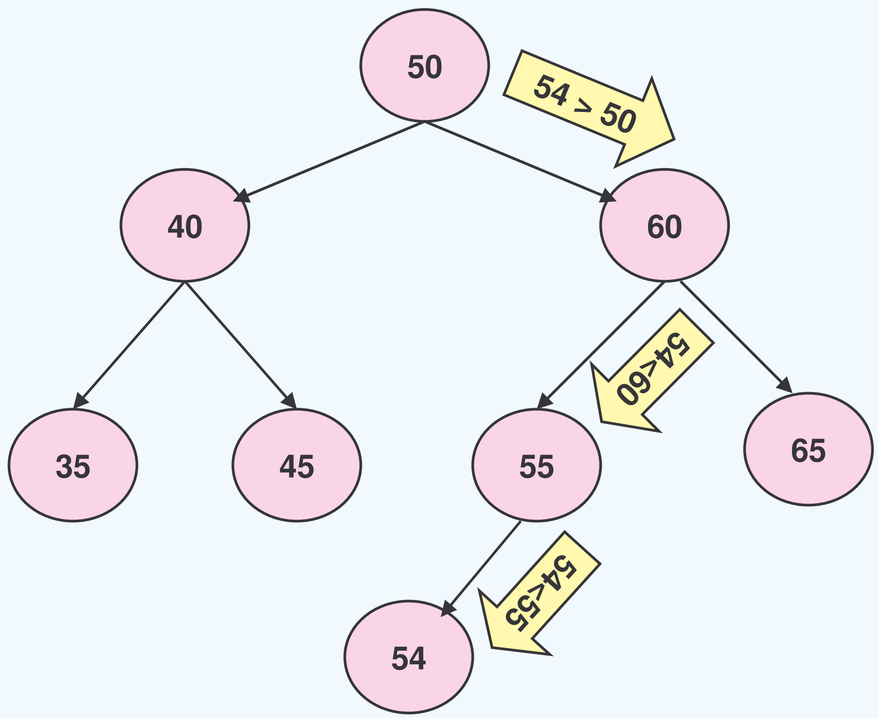


Introduction to Data Structures (Binary Search Tree Contd…)

- Inserting An Element into a Binary Tree.
  - Compare the element to be added in the tree, with the element in the root node.
  - If the element is the less than the root element then
    - Extract the address of the left child from the root node.
    - Search for a appropriate place for the new node in the left sub tree recursively.
  - If the element is the greater than the root element then
    - Extract the address of the right child from the root node.
    - Search for a appropriate place for the new node in the left sub tree recursively.

Introduction to Data Structures (Binary Search Tree Contd…)

- Inserting 54 in the Binary Tree

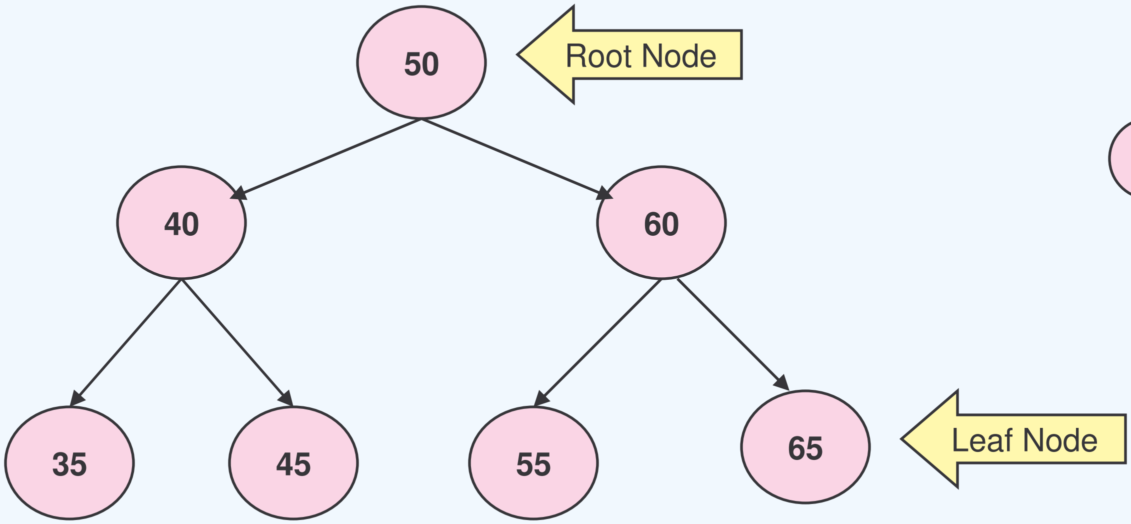


Introduction to Data Structures (Binary Search Tree Contd…)

- Traversal Mechanisms for a Binary Tree.
  - PreOrder Traversal
  - InOrder Traversal
  - PostOrder Traversal

Introduction to Data Structures (Binary Search Tree Contd...)

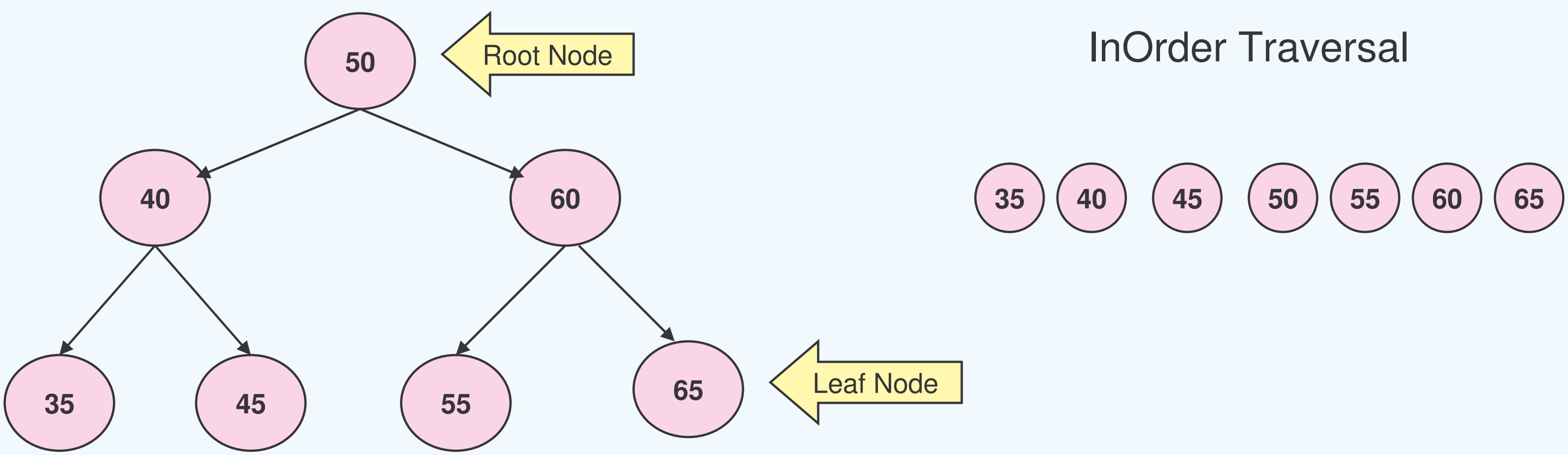
- PreOrder Traversal
  - Traverse the root element then traverse the left sub tree in a PreOrder fashion and then traverse the right sub tree in a PreOrder fashion.



PreOrder Traversal

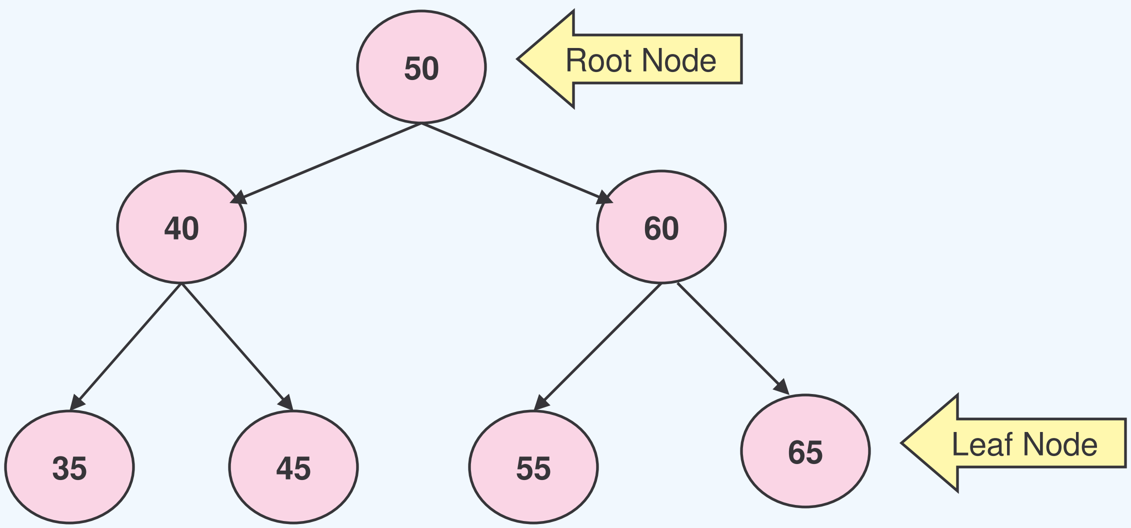
Introduction to Data Structures (Binary Search Tree Contd…)

- InOrder Traversal
  - Traverse the left sub tree in InOrder fashion then traverse the root element and then traverse the right sub tree in InOrder fashion.

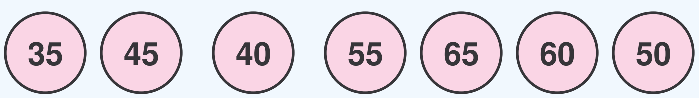


Introduction to Data Structures (Binary Search Tree Contd…)

- PostOrder Traversal
  - Traverse the left sub tree in PostOrder fashion then traverse the right sub tree in PostOrder fashion then traverse the root element.



PostOrder Traversal



Introduction to Data Structures (Binary Search Tree Contd…)

- Deletion of a node from the Binary Tree
  - Before deleting a node from a binary tree it needs to be searched beginning from the root element.
  - If the element is less than the root node then it will be found in the left sub tree.
  - If element is greater than the root then it will be found in the right sub tree.

Introduction to Data Structures (Binary Search Tree Contd...)

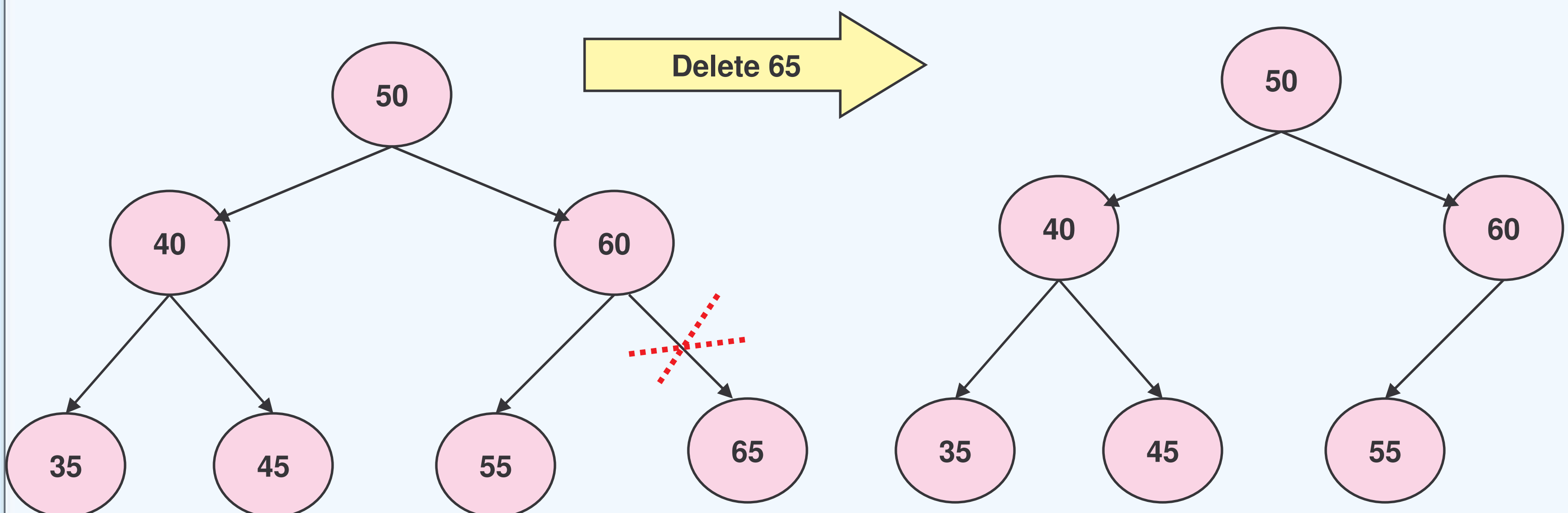
- Deletion of a node from the Binary Tree
  - The node to be deleted could be
    - A leaf node
    - A node having exactly one sub tree (right sub tree or left sub tree)
    - A node having both sub trees (right as well left sub trees)

Introduction to Data Structures (Binary Search Tree Contd…)

- Node to be deleted is a leaf node.
  - If the leaf is a left child of the parent then the parents left address is set to null.
  - If the leaf is a right child of the parent then the parents right address is set to null.

Introduction to Data Structures (Binary Search Tree Contd…)

- Illustration – Delete 65

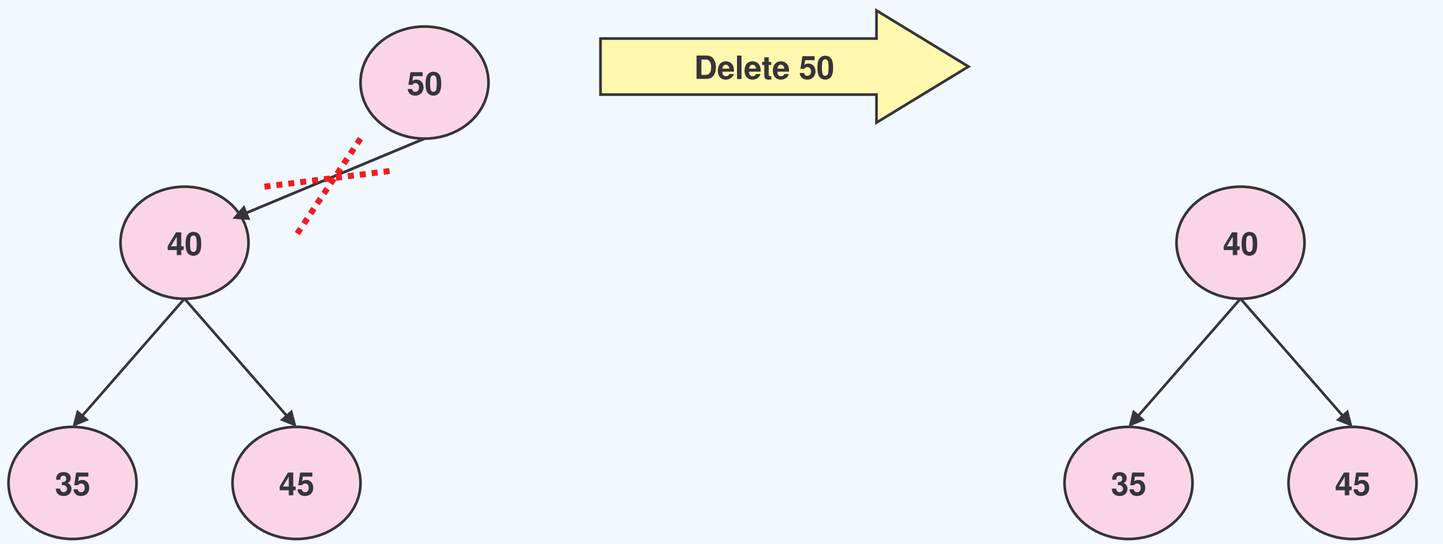


Introduction to Data Structures (Binary Search Tree Contd...)

- Deletion of a node which has exactly one sub tree and it is also a root element.
  - After deleting the root the left child or the right child becomes the root node.

Introduction to Data Structures (Binary Search Tree Contd…)

- Illustration – Delete 50

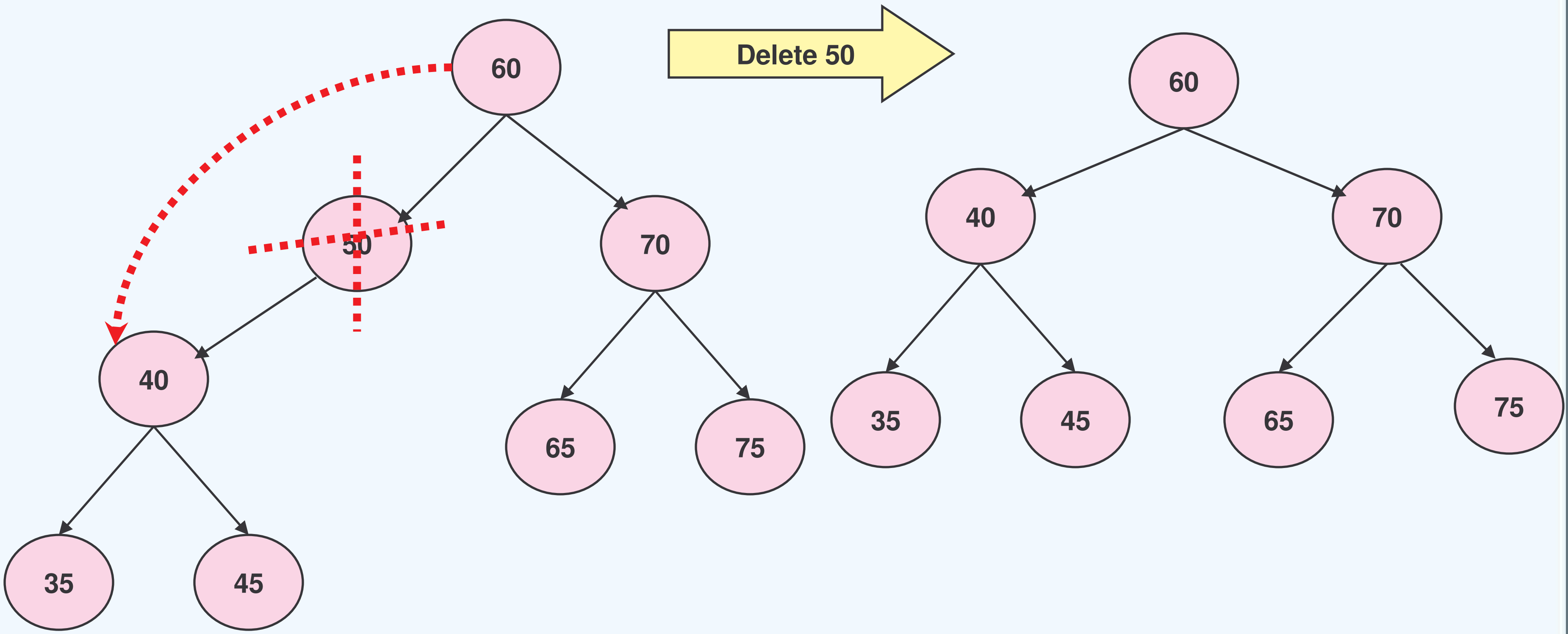


Introduction to Data Structures (Binary Search Tree Contd…)

- Node to be deleted has exactly one sub tree and it's a non root element.
  - If the node to be deleted is a left child of its parent then the left address of this node is copied to the parents left address.
  - If the node to be deleted is a right child of its parent then the right address of this node is copied to the parents right address.

Introduction to Data Structures (Binary Search Tree Contd…)

- Illustration – Delete 50



Introduction to Data Structures (Binary Search Tree Contd...)

- Deletion of a node from the Binary Tree which has exactly 2 sub trees
  - The value of the node to be deleted can be **replaced** with the value of the **greatest** node in the **left sub tree** *or* it can be **replaced** with the value of the **smallest** node in the **right sub tree**.
  - The node with the greatest value in the left sub tree (or node with the smallest value in the right sub tree) will be leaf nodes.
  - Once the value is replaced we can delete the corresponding leaf node and deleting a leaf node is the easiest task.

Introduction to Data Structures (Binary Search Tree Contd…)

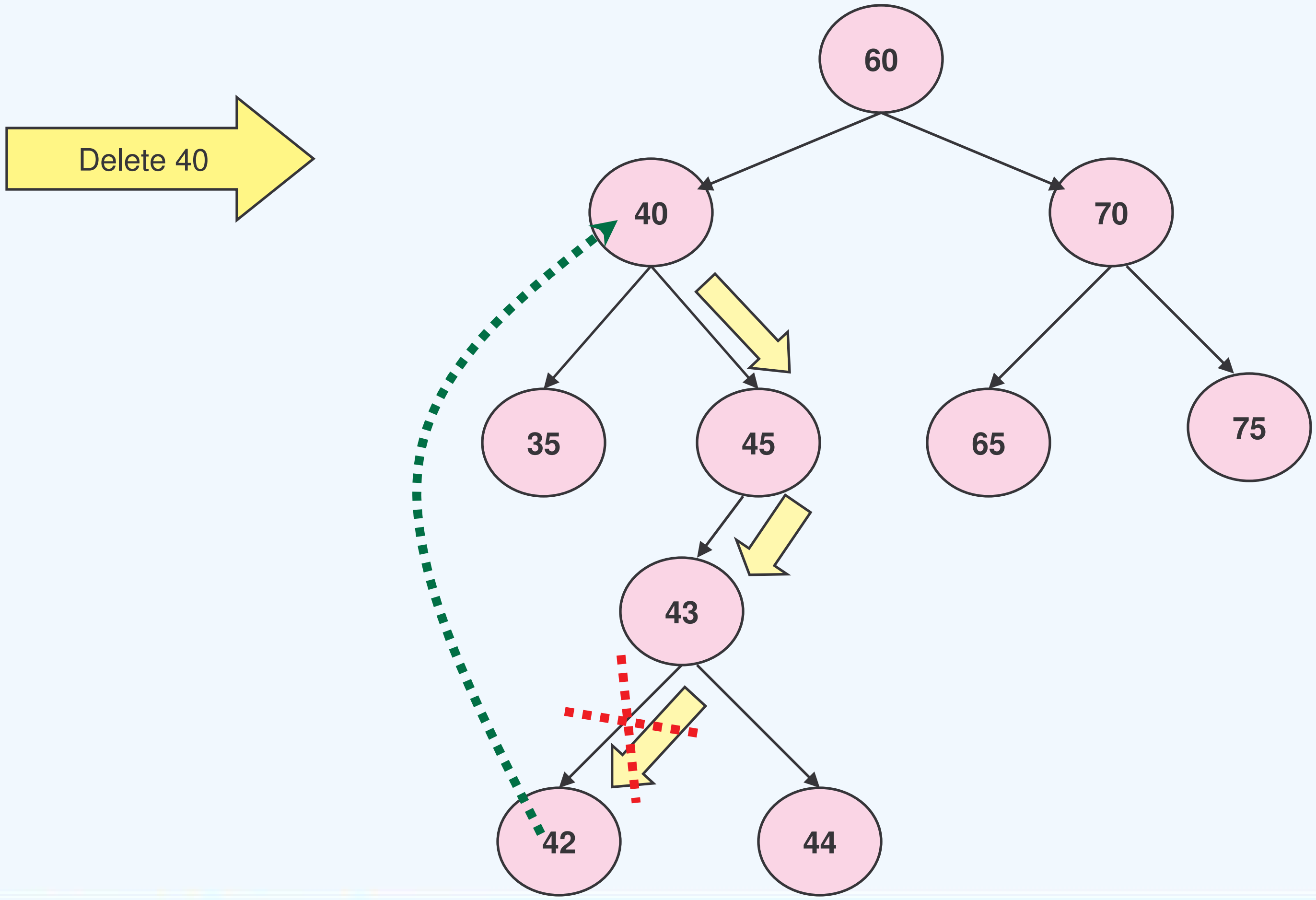
- How to find out greatest node in the left sub tree?
  - Go to the root of the left sub tree (i.e. go to the left child of node to be deleted)
  - Then go to the right till you reach a leaf node.

Introduction to Data Structures (Binary Search Tree Contd…)

- How to find out smallest node in the right sub tree?
  - Go to the root of the right sub tree (i.e. go to the right child of node to be deleted)
  - Then go to the left till you reach a leaf node.

Introduction to Data Structures (Binary Search Tree Contd...)

- Illustration – Delete 40



Summary

- Topics covered in the unit.
  - User Defined data types
    - Structures
    - Unions
- Data Structures
  - Stacks
  - Queues
  - Link List
  - Binary Trees

Thank You!