Data Structures

Ahsan Ijaz

Linked List

- A linked list is a data structure which can change during execution.
- Successive elements are connected by pointers.Last element points to NULL.
- It can grow or shrink in size during execution of a program.
- It does not waste memory space.

Linked List

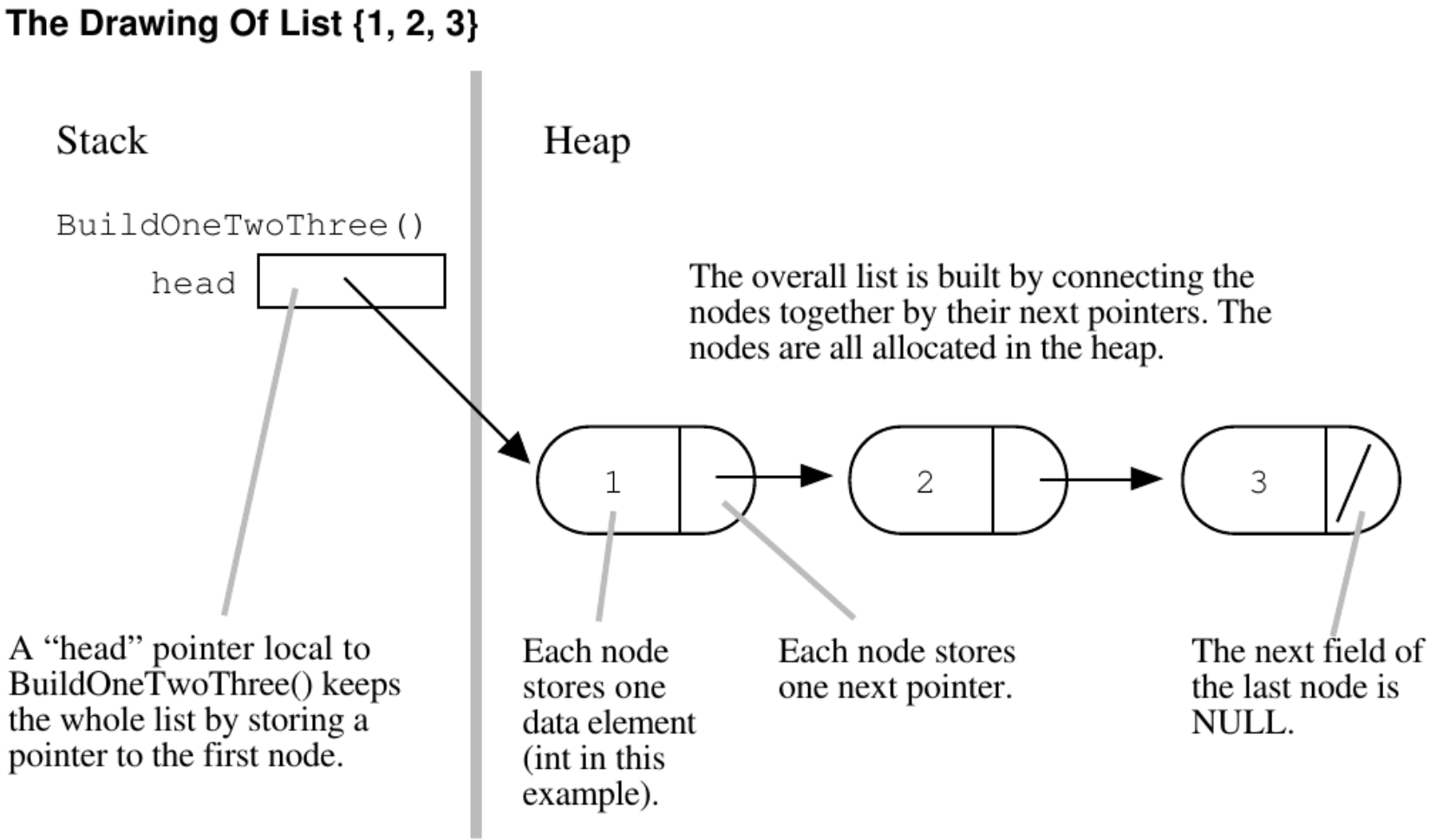

Figure: Linked List

Basic Building Block- Self referential Class

Self referential class containing data member and a pointer for next class of same type.

```
1  class Node
2  {
3  public:
4     int data;
5     Node *next;
6     Node(int a, Node *b)
7     {
8        data=a;
9        next=b;
10    }
11 };
```

Linked List Implementation

- Must know the pointer to the first element of the list (called start, head, etc.)

```
1   class linkedlist
2   {
3   private:
4     Node *head;
5   public:
6     void addnode(int);
7     void deletenode(int);
8     void insertnode(int);
9     void printnode();
10    linkedlist()
11    {
12      head=NULL;
13    }
14  };
```

Deletion

- Get the Node prior of deletion Node (prevnode).
- Get the Node next of deletion Node (nextnode).
- Assign address of nextnode to the next pointer of prevnode.
- Delete the dynamically allocated deletion node (delnode).

Illustration of delete

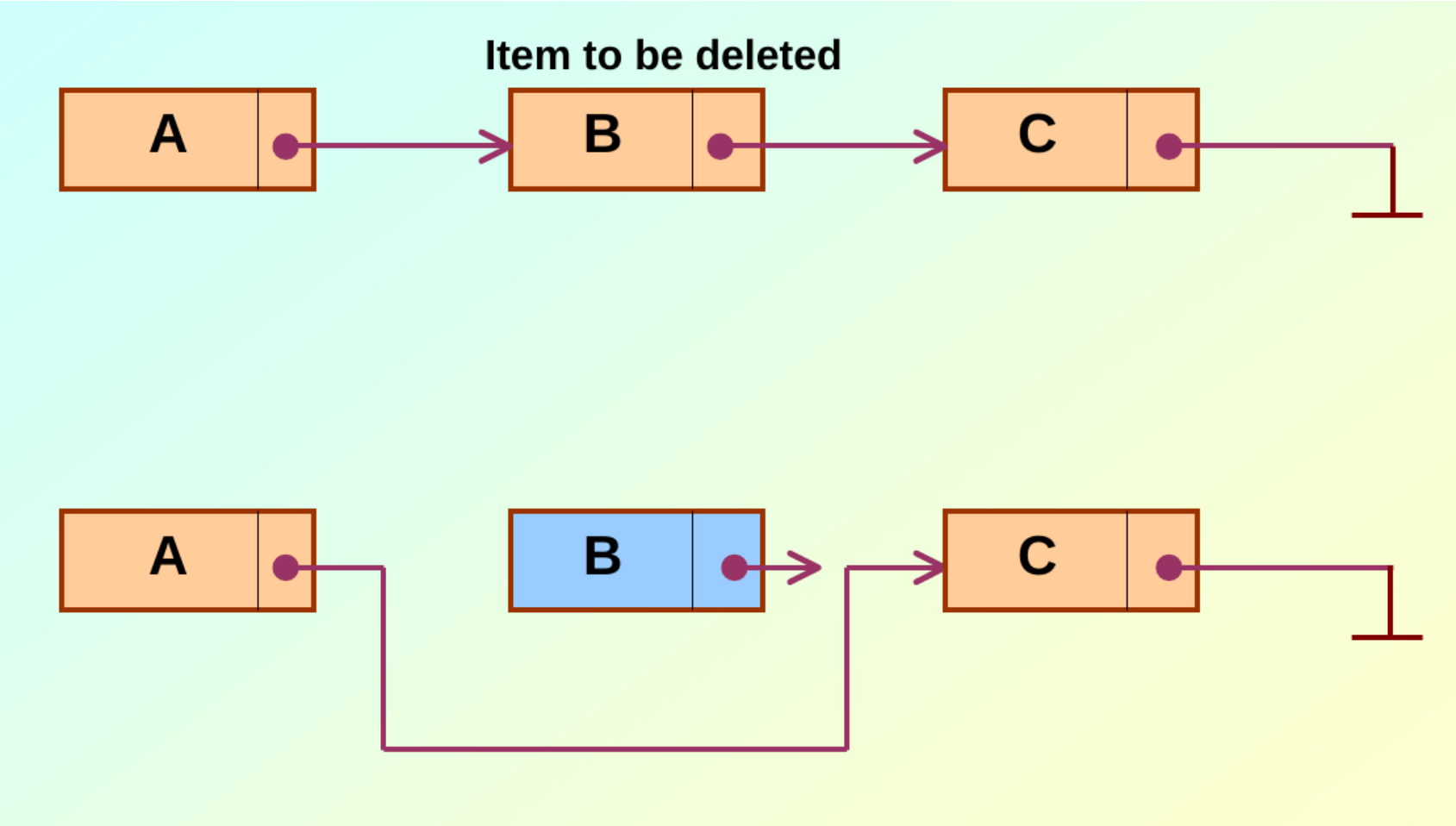

Figure: Deleting a Node

Implementation of Delete

```
1  void linkedlist:: deletenode(int index)
2    {
3      Node *n;     //Dummy variable for traveling through llist
4      Node *delnode;    //Node to be deleted
5      Node *prevnode;   //Previous node
6      Node *nextnode;   //Next Node
7      n=head;
8  for (int i = 1; i < index; i++) //Index is the node which we want
9    {
10     if(i==index-1)
11       {prevnode=n;}
12     n=n->next;      //Main line used to travel
13   }
14 delnode=n;
15 nextnode=delnode->next;
16  prevnode->next=nextnode;
17  delete delnode;
18   }
```

Insertion of Node

- Create a new node with the required data.
- The next pointer of the new node is set to link it to the item which is to follow it in the list.
- The next pointer of the item which is to precede it must be modified to point to the new item.

Illustration of insertion

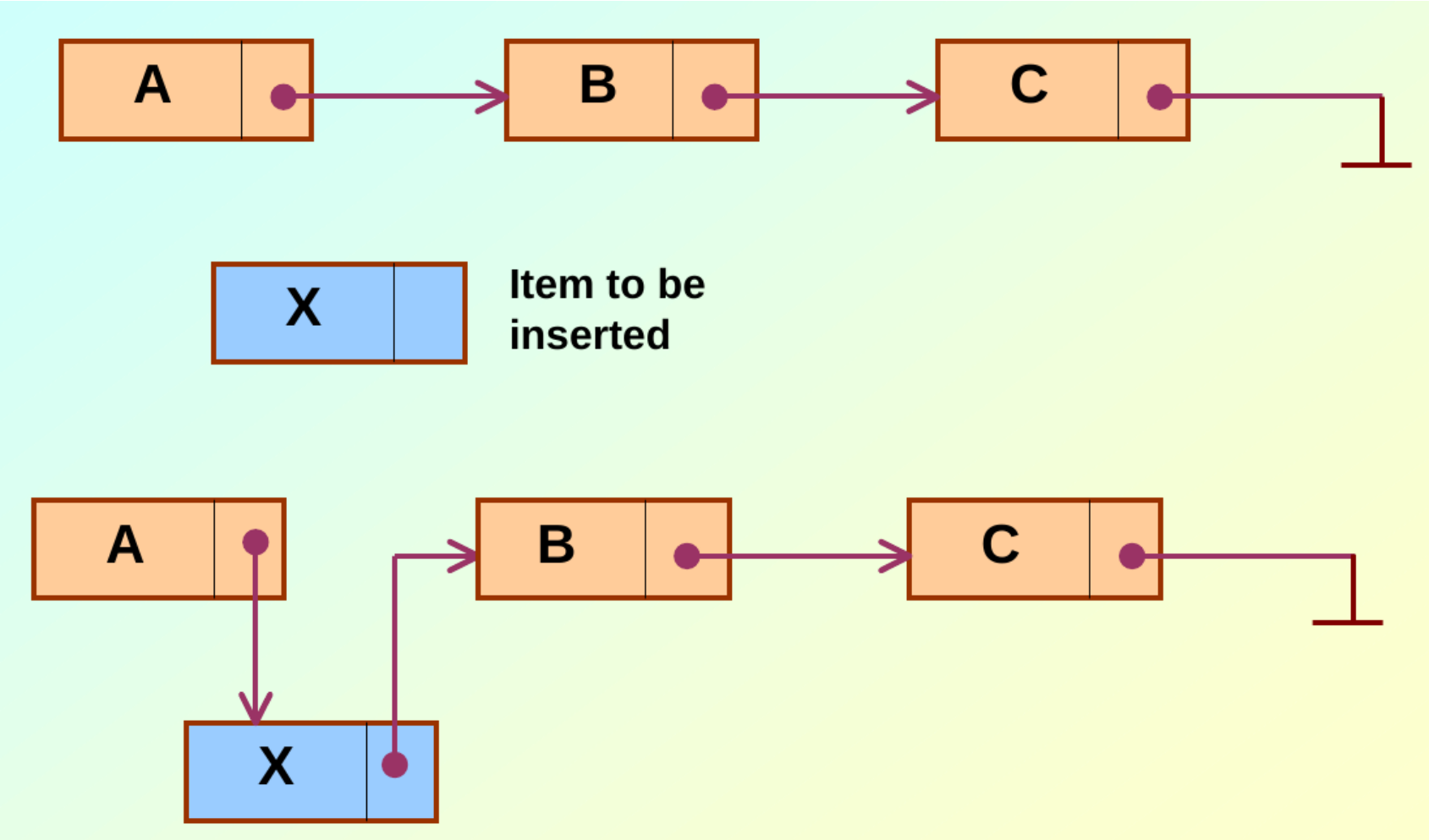

Figure: Inserting a Node

Implementation of Insertion

LMS Assignment to be submitted by Friday

Add Node Function

```
1  void linkedlist::addnode(int val)
2    {
3       Node *n;
4       n=head;
5       if(head==NULL)  //This means list contains no elements
6         {
7  head=new Node(val,NULL); //insert value 'val' in head
8         }
9       else{
10      while(n->next!=NULL)//loop while next pointer of node is valid
11       {
12        n=n->next;   //Go through the list until last node is reached
13       }
14      n->next=new Node(val,NULL); //Create a new node here
15      }
16   }
```

Print Function

```
1   void linkedlist:: printnode()
2     {Node *n;
3       n=head;
4       while(n!=NULL) //loop while n is valid
5         {
6        cout<<"Data = "<<n->data<<endl; //display data in n
7        n=n->next; //travel to next point of list
8         }
9
10    }
```

Circular Linked List

The pointer from the last element in the list points back to the first element.

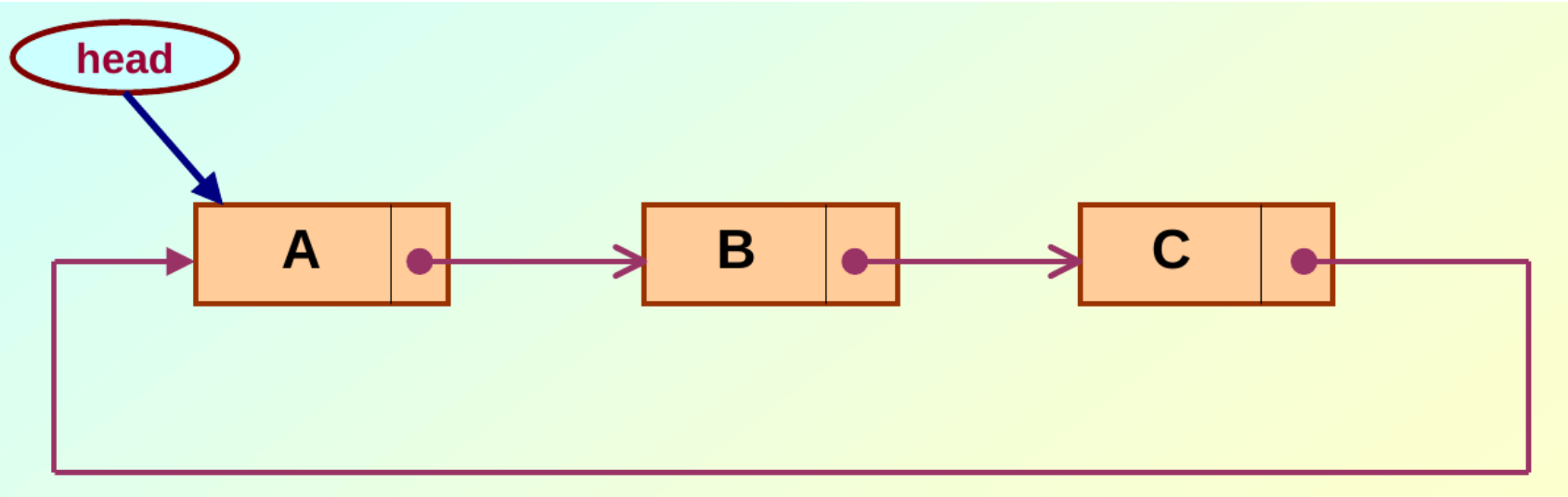

Figure: Circular Linked List

```
1  last->next=head;
```

Doubly linked List

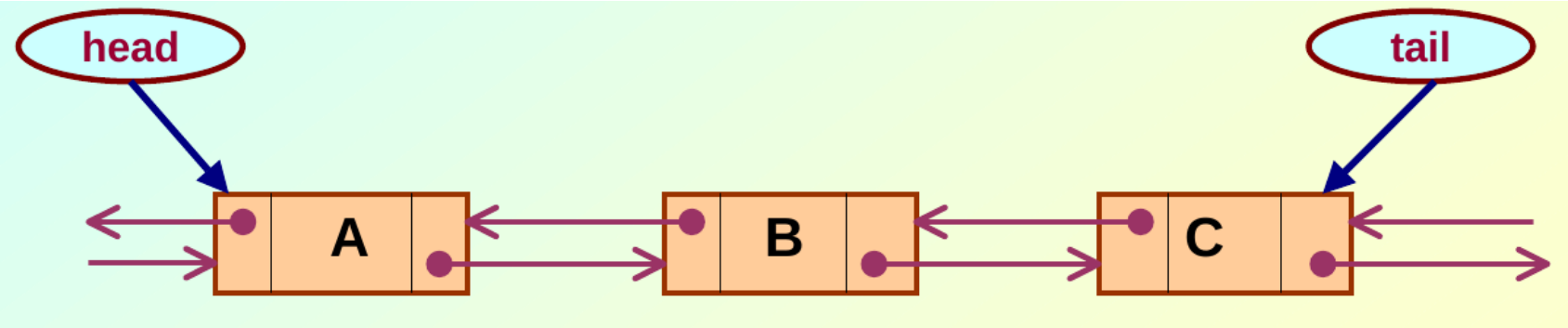

Figure: Double linked list

```
1  Node *previous; // This should be part of
2                  // self referential class
```

Queue Implementation using linked list

Ideas??

```
1  void pushback(int value);
2  Node* popfront();
```

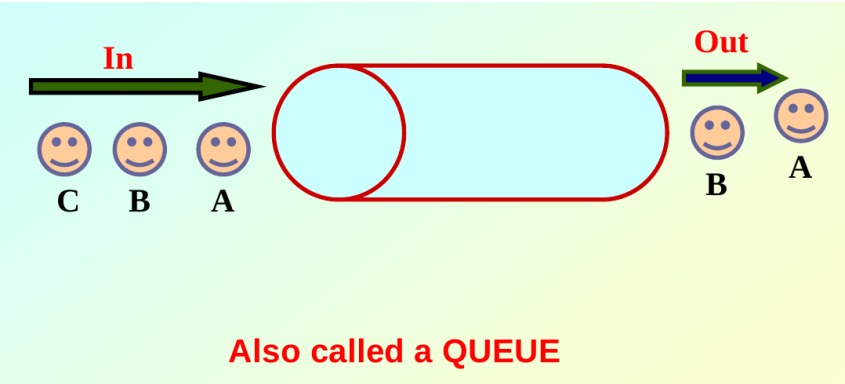

Figure: Queue

Stack Implementation using linked list

```
1  void pushback(int value);
2  Node* popback();
```

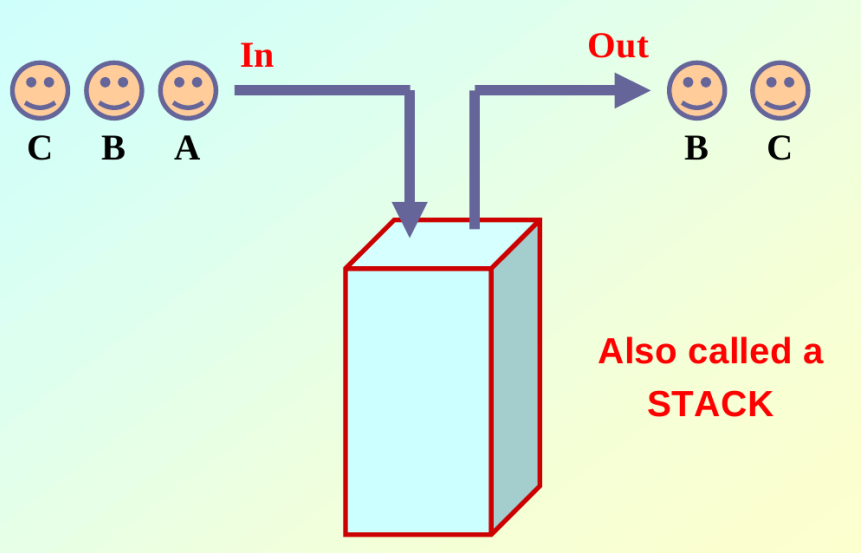

Figure: Stack Implementation