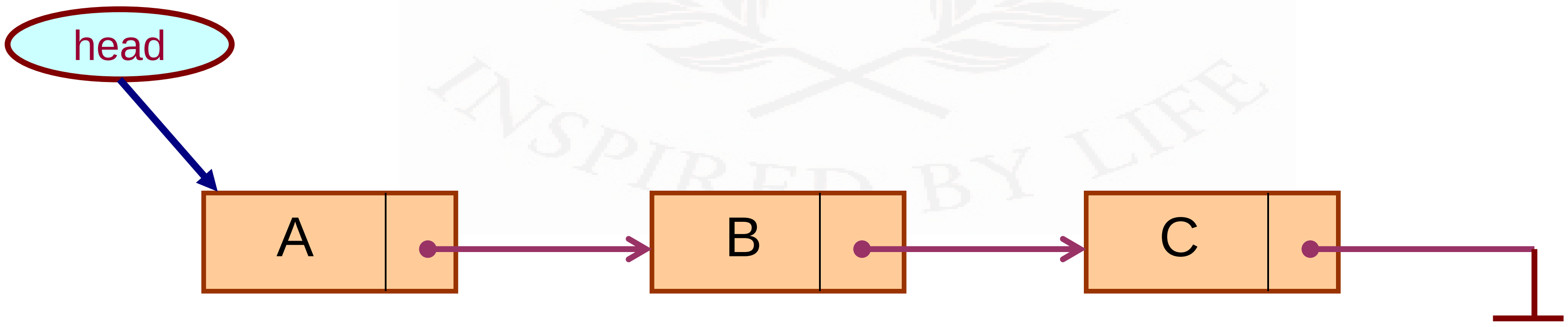
Linked list

Why Linked Lists?

- Advantages of Arrays:
  - ❖ Data access is faster
  - ❖ Simple
- Disadvantages:
  - ❖ Size of the array is fixed.
  - ❖ Array items are stored contiguously.
  - ❖ Insertion and deletion operations involve tedious job of shifting the elements with respect to the index of the array.

Introduction

- 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 can be made just as long as required.
  - It does not waste memory space.



Introduction

- Keeping track of a linked list:
  - Must know the pointer to the first element of the list (called *start*, *head*, etc.).

- Linked lists provide flexibility in allowing the items to be rearranged efficiently.
  - Insert an element.
  - Delete an element.

Illustration: Insertion

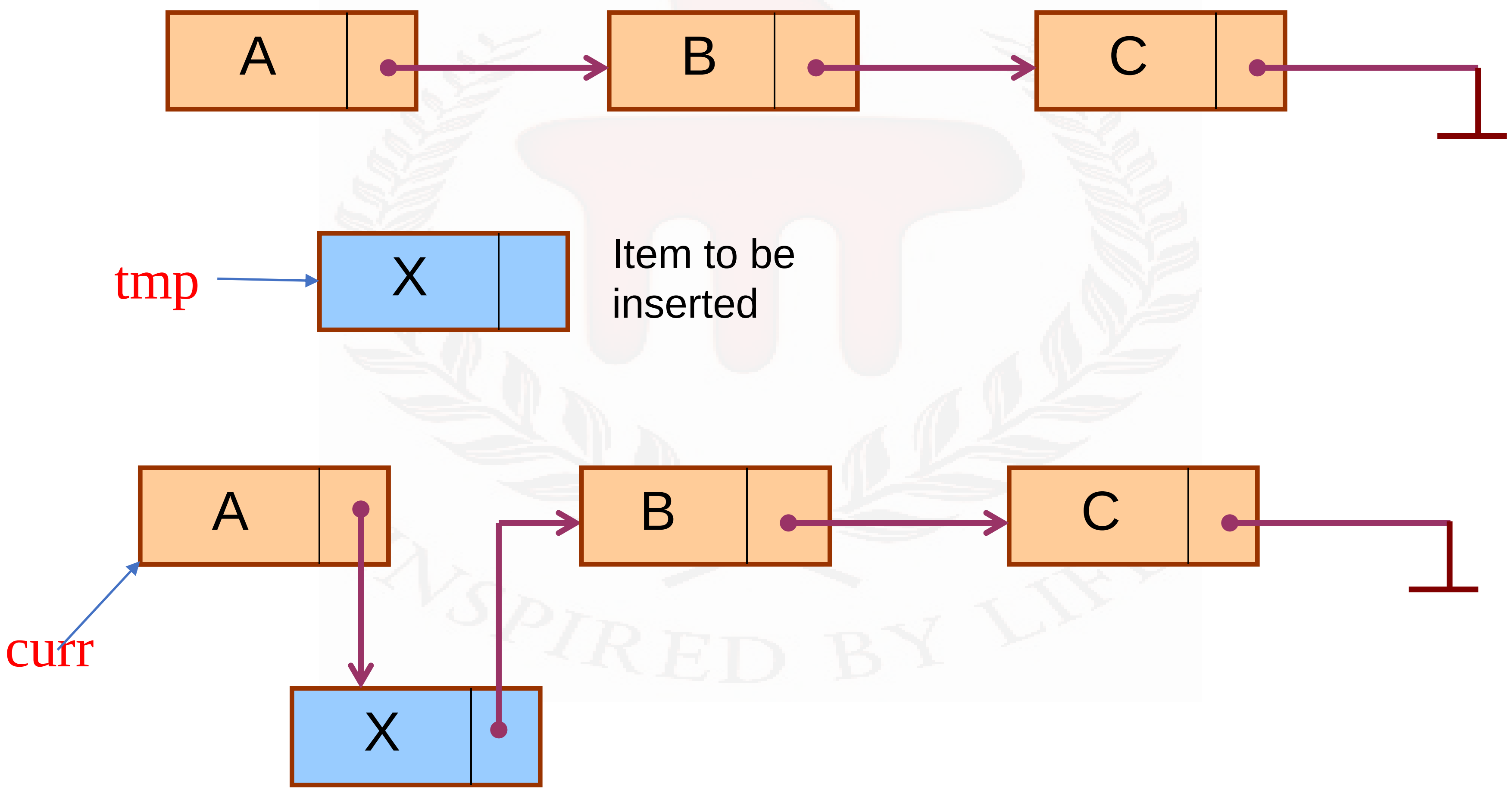
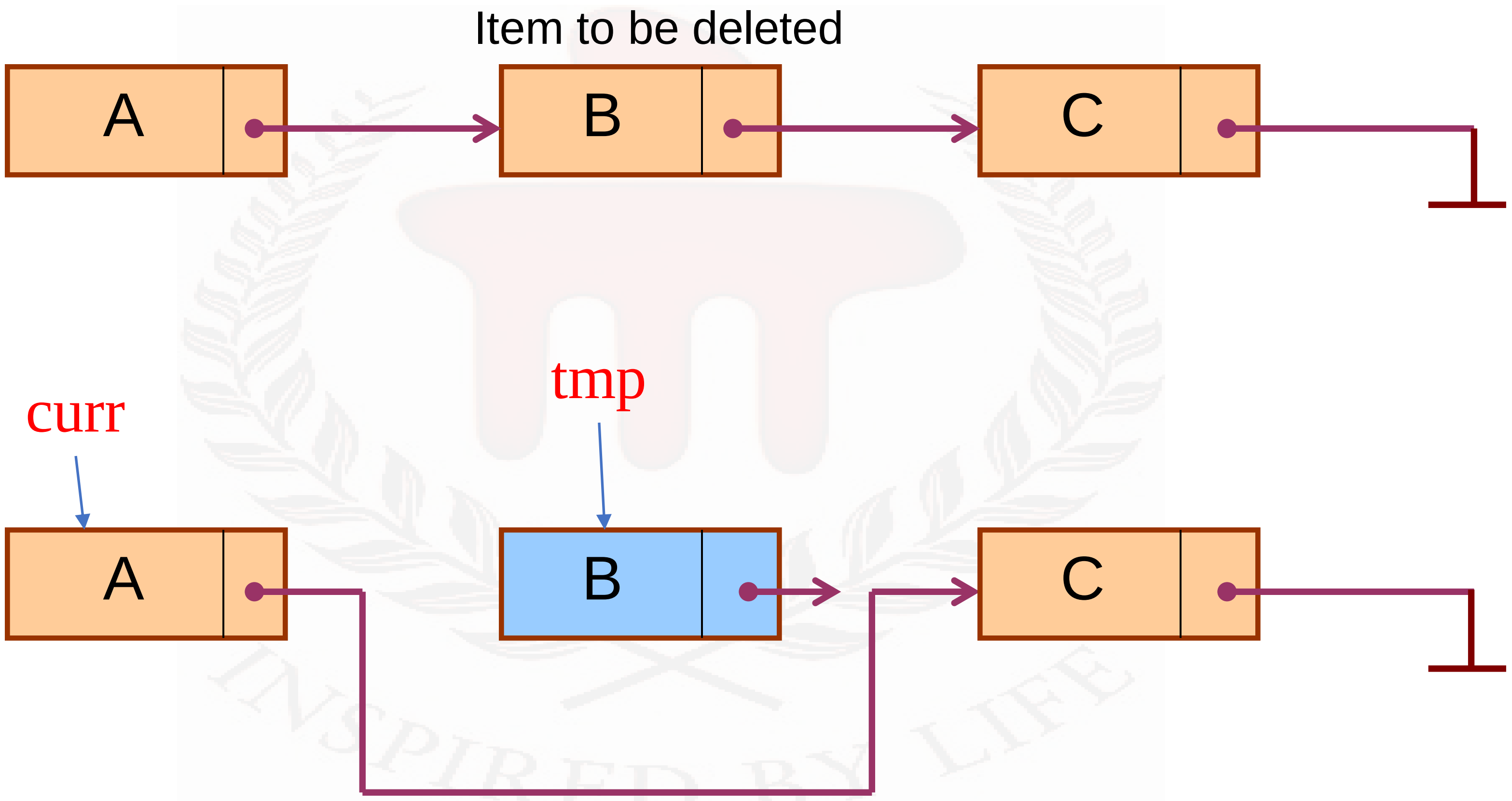


Illustration: Deletion



Summary

- For insertion:
  - A record is created holding the new item.
  - The next pointer of the new record is set to link 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.
- For deletion:
  - The next pointer of the item immediately preceding the one to be deleted is altered, and made to point to the item following the deleted item.

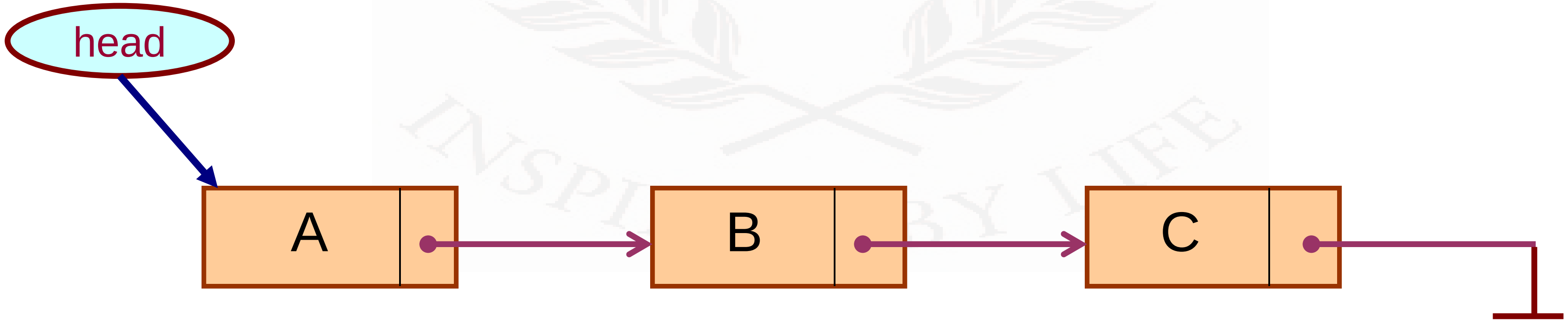
Array versus Linked Lists

- Arrays are suitable for:
  - Inserting/deleting an element at the end.
  - Randomly accessing any element.
  - Searching the list for a particular value.
- Linked lists are suitable for:
  - Inserting an element.
  - Deleting an element.
  - Applications where sequential access is required.
  - In situations where the number of elements cannot be predicted beforehand.

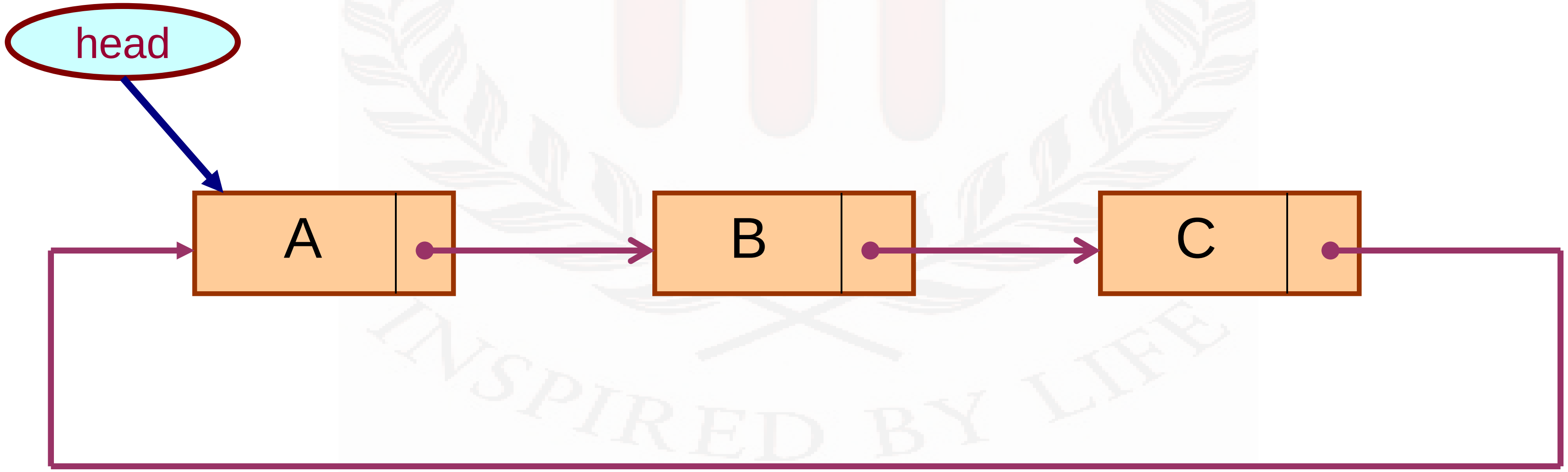
Types of Lists

- Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible.

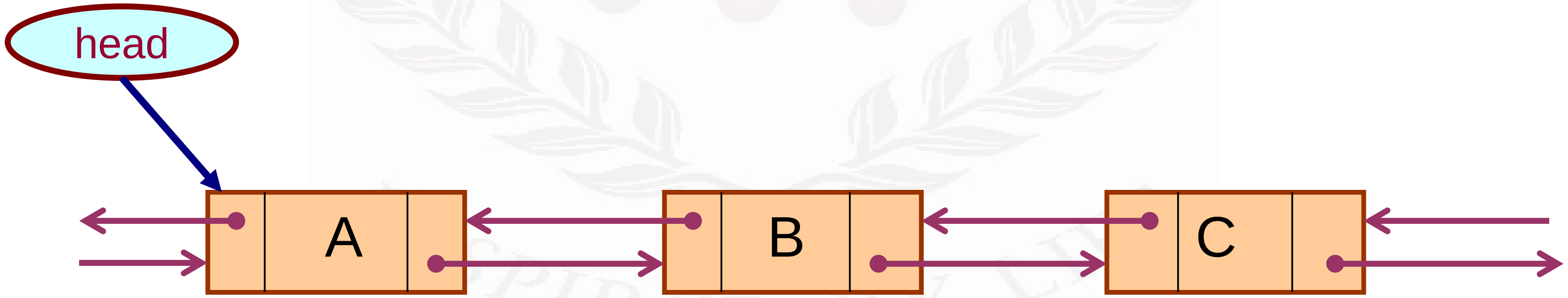
  - Linear singly-linked list (or simply linear list)
    - One we have discussed so far.



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



- Doubly linked list
  - Pointers exist between adjacent nodes in both directions.
  - The list can be traversed either forward or backward.
  - .



Basic Operations on a List

- Creating a list
- Traversing the list
- Inserting an item in the list
- Deleting an item from the list
- Concatenating two lists into one

```c
#include <stdio.h>
#include <stdlib.h>

// Linked List Node
struct node {
        int info;
        struct node* link;
};
struct node* start = NULL;
```

```c
// Function to create list with n nodes initially
void createList()
{       if (start == NULL) {
        int n;
        printf("\nEnter the number of nodes: "); scanf("%d", &n);
        if (n != 0) {
                int data; struct node* newnode; struct node* temp;
                newnode = malloc(sizeof(struct node));
                start = newnode; temp = start;
                printf("\nEnter number to be inserted : "); scanf("%d", &data);
                start->info = data;
                for (int i = 2; i <= n; i++) {
                        newnode = malloc(sizeof(struct node));
                        temp->link = newnode;
                        printf("\nEnter number to be inserted : "); scanf("%d", &data);
                        newnode->info = data;   temp = temp->link;
                        }
                }
                printf("\nThe list is created\n");
        }
        else printf("\nThe list is already created\n");
}
```

```
// Function to traverse the linked list
void traverse()
{
        struct node* temp;
        if (start == NULL) printf("\nList is empty\n");
        // Else print the LL
        else {
                temp = start;
                while (temp != NULL) {
                        printf("Data = %d\n", temp->info);
                        temp = temp->link;
                }
        }
}
```

```
// Function to insert at the front of the linked list
void insertAtFront()
{
        int data;-
        struct node* temp;
        temp = malloc(sizeof(struct node));
        printf("\nEnter number to"
                " be inserted : ");
        scanf("%d", &data);
        temp->info = data;

        // Pointer of temp will be
        // assigned to start
        temp->link = start;
        start = temp;
}
```

```
// Function to insert at the end of the linked list
void insertAtEnd()
{
        int data;
        struct node *temp, *head;
        temp = malloc(sizeof(struct node));
        printf("\nEnter number to be inserted : ");
        scanf("%d", &data);
        temp->link = 0;
        temp->info = data;
        head = start;
        if(head==NULL)  start=temp;
        else{
        while (head->link != NULL) {
                head = head->link;
        }
        head->link = temp;
        }
}
```

```c
// Function to insert at any specified position in the linked list
void insertAtPosition()
{
        struct node *temp, *newnode;
        int pos, data, i = 1;
        newnode = malloc(sizeof(struct node));
        // Enter the position and data
        printf("\nEnter position and data :");
        scanf("%d %d", &pos, &data);
        temp = start;   newnode->info = data;   newnode->link = 0;
        while (i < pos - 1) {
                temp = temp->link;
                i++;
        }
        newnode->link = temp->link;
        temp->link = newnode;
}
```

```
// Function to delete from the front of the linked list
void deleteFirst()
{
        struct node* temp;
        if (start == NULL)
                printf("\nList is empty\n");
        else {
                temp = start;
                start = start->link;
                free(temp);
        }
}
```

```c
// Function to delete from the end
// of the linked list
void deleteEnd()
{       struct node *temp, *prevnode;
        if (start == NULL)      printf("\nList is Empty\n");
        else if(start->link==NULL) {free(start); start=NULL;}
        else {
                temp = start;
                while (temp->link != 0) {
                        prevnode = temp;
                        temp = temp->link;
                }
                free(temp);
                prevnode->link = 0;
        }
}
```

```c
// Function to delete from any specified position from the linked list
void deletePosition()
{       struct node *temp, *position;   int i = 1, pos;
        if (start == NULL) printf("\nList is empty\n"); // If LL is empty
        else {
                printf("\nEnter index : ");     // Position to be deleted
                scanf("%d", &pos);
                temp = start;
                while (i < pos - 1) {
                        temp = temp->link;
                        i++;
                }
                position = temp->link;
                temp->link = position->link;
                free(position);
        }
}
```