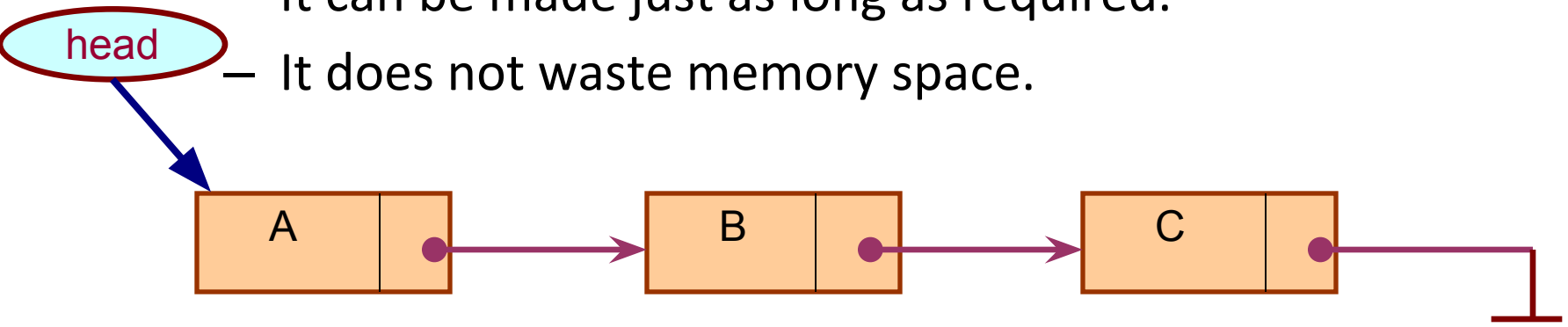
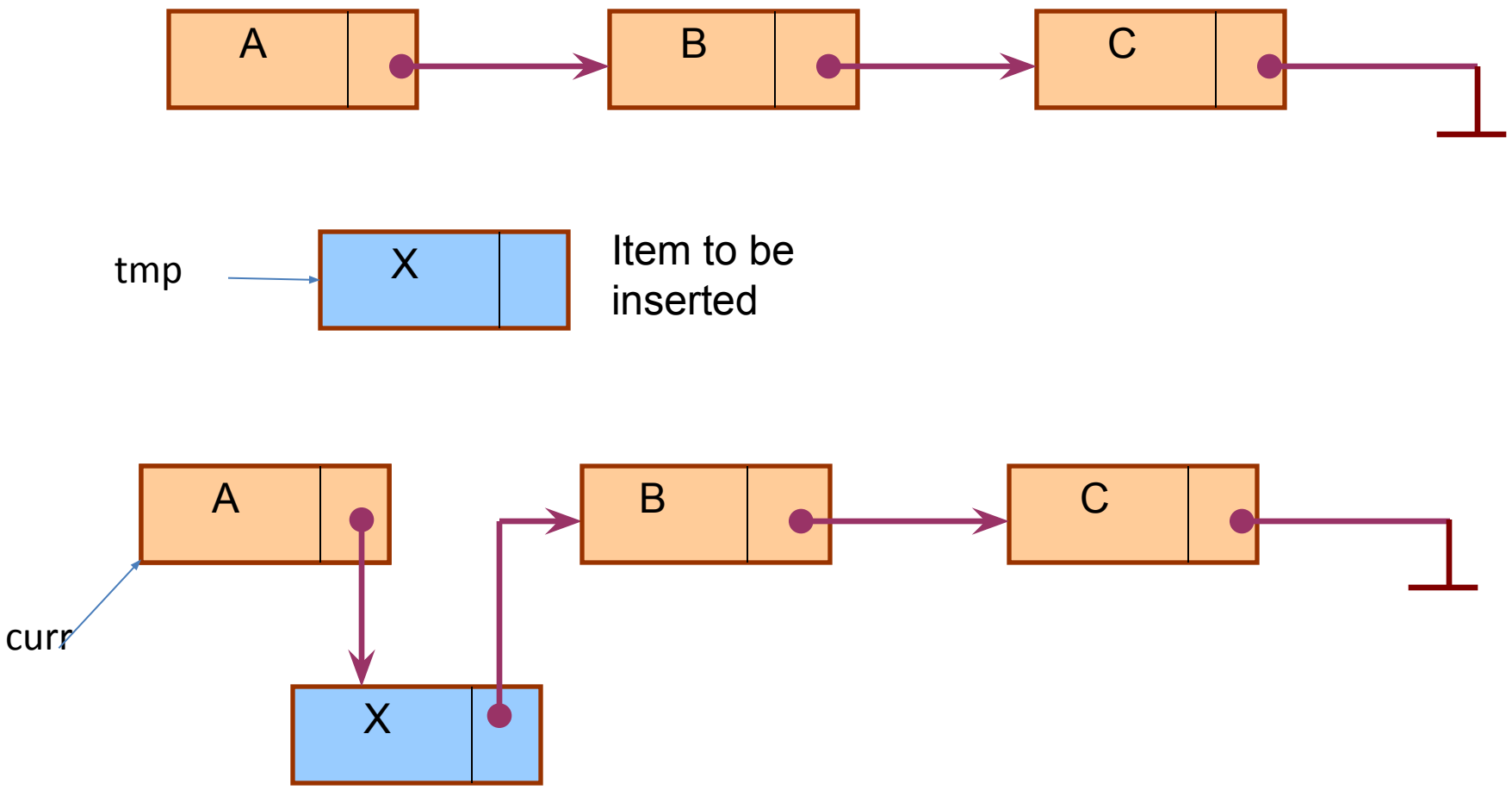
# Linked List

# Introduction

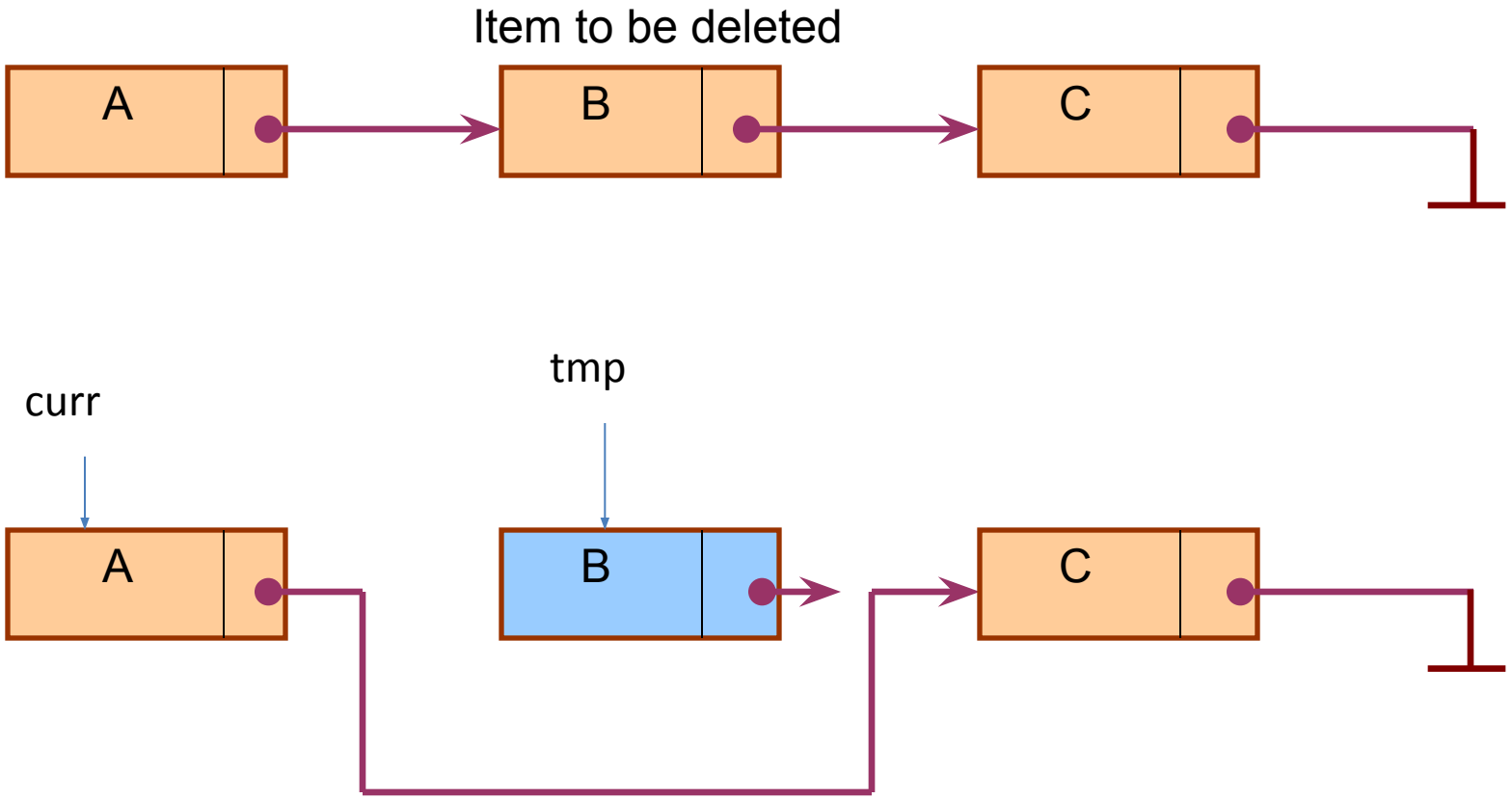
- A linked list is a data structure which can change during execution.
  - Successive elements are connected by pointers.
  - Last element points to `NULL`.
  - It can grow or shrink in size during execution of a program.
  - It can be made just as long as required.
  - It does not waste memory space.



# Illustration: Insertion

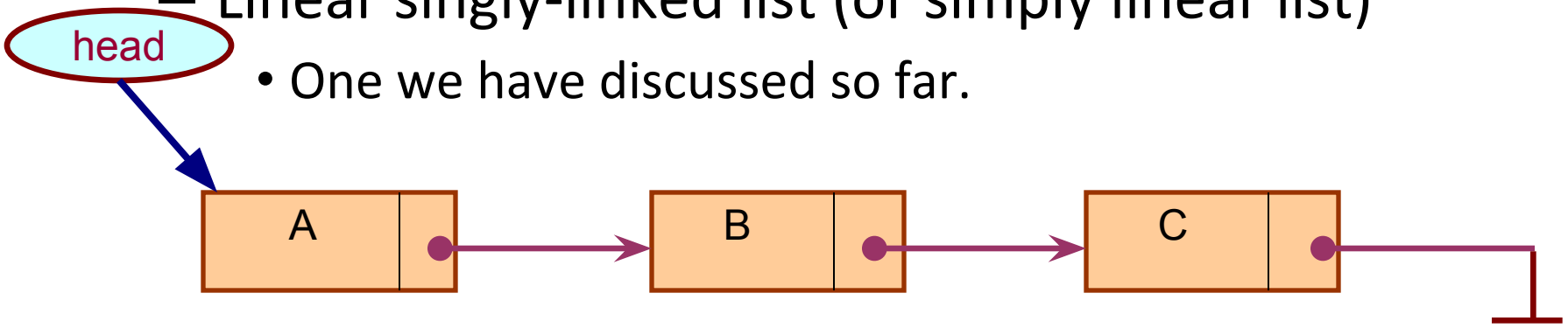


# Illustration: Deletion



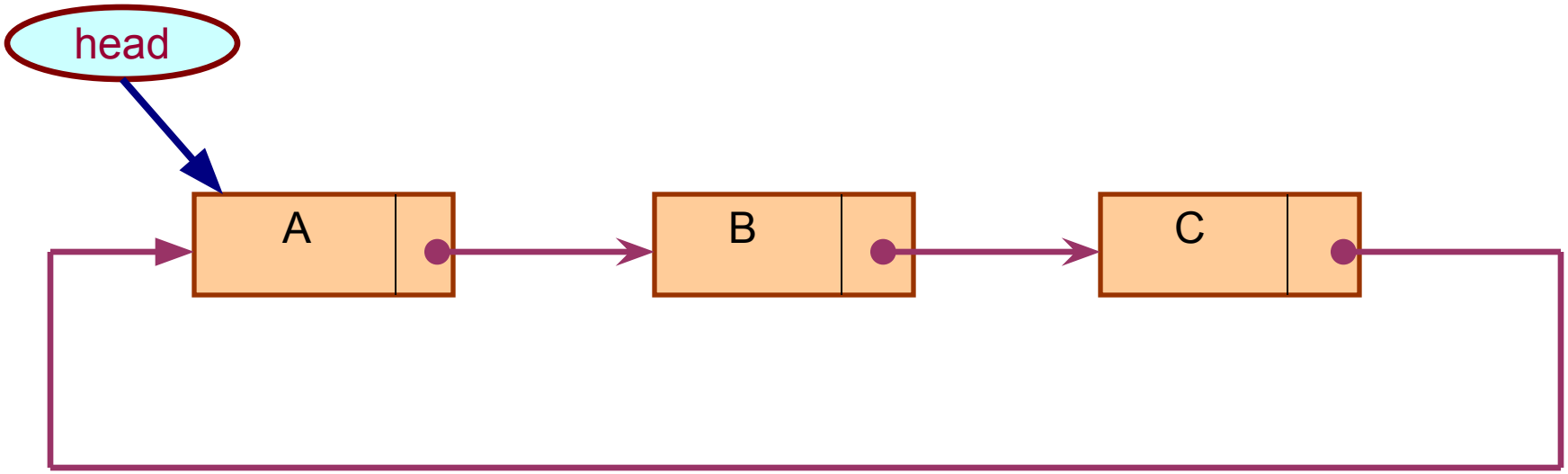
# Types of Lists

- 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.

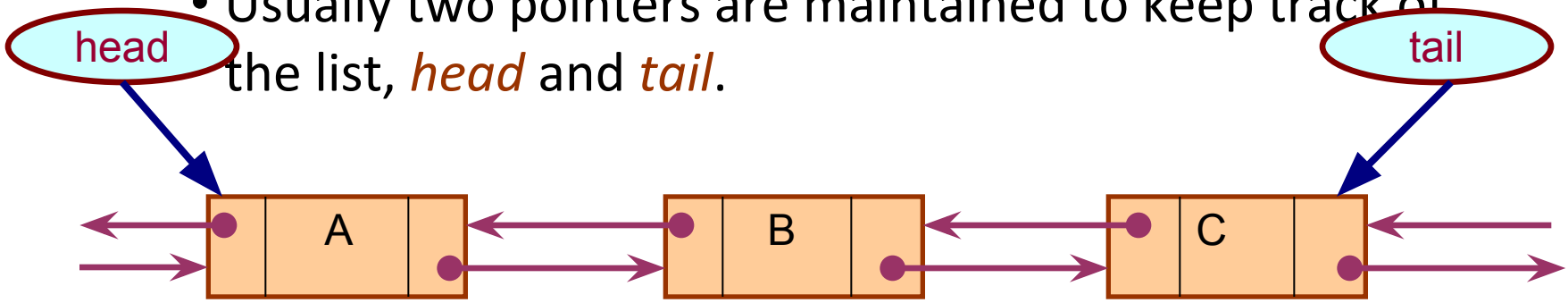


– Circular linked list

  • 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.
  - Usually two pointers are maintained to keep track of the list, *head* and *tail*.



# 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

# Algorithm of insertion at the beginning

- Algorithm of insertion at the beginning
- Create a new node
- Assign its data value
- Assign newly created node’s next ptr to current head reference. So, it points to the previous start node of the linked list address
- Change the head reference to the new node’s address

# Creating and displaying a list

```
Struct node
{
int data;
Struct node *next;
};
int main()
{
stuct node *head ,*newnode ,*temp;
newnode=(struct  node *)malloc(size(struct node));
printf(“Enter data”);
scanf(“%d”,&newdata->data);
newnode->next=NULL;
```

```
if(head==NULL)
{
head=temp=newnode;
}
else
{
temp->next=newnode;
temp=newnode;
}

 temp=head;
 while(temp!=NULL)
 {
 printf("%d",temp->data);
 temp=temp->next;
 }
 }
```

# Inserting a node in the list

- Inserting a node at the beginning of list
- Struct node
- {
- int data;
- Struct node *next;
- };
- stuct node *head *newnode *temp;
- newnode=(struct  node *)malloc(size(struct node));
- printf(“Enter data”);
- scanf(“%d”,&newdata->data);
- newnode->next=head;
- head=newnode;

# Inserting a node at the end of the list

```
Struct node
{
int data;
Struct node *next;
};
stuct node *head *newnode *temp;
newnode=(struct  node *)malloc(size(struct node));
printf("Enter data");
scanf("%d",&newdata->data);
newnode->next=NULL;
temp=head;
while(temp!=NULL)
{
temp=temp->next;
}
temp->next=newnode;
```

# Inserting a node at the position in the list

```
struct node
{
int data;
Struct node *next;
};
stuct node *head *newnode *temp;
int pos;
newnode=(struct  node *)malloc(size(struct node));
printf(“Enter position”);
scanf(“%d”,&pos);
if(pos>count)
{
printf(“invalid choice”);
}else
{
temp=head;
while(i<pos)
{
temp=temp->next
i++;
}
printf(“enter data”);
scanf(“%d”,&newnode->data);
newnode->next=temp->next;
}
```