ITPC – 203 – Data Structures

Linked Lists

Dr. Lalatendu Behera

Department of CSE

Dr B R Ambedkar National Institute of Technology Jalandhar

- Singly Linked List
- Doubly Linked List
- Circular Linked List
- Representing Stack with Linked List.
- Representing Queue with Linked List.

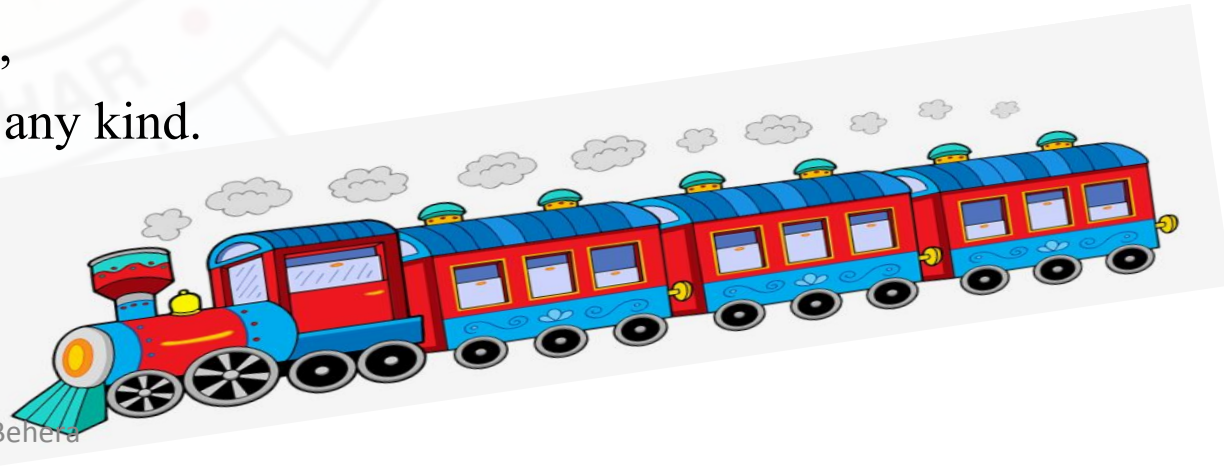
- In **array**, elements are stored in consecutive memory locations.
- To occupy the adjacent space, block of memory that is required for the array should be allocated before hand.
- Once memory is allocated, it cannot be extended any more. So that array is called the **static data structure**.
- Wastage of memory is more in arrays.
- Array has fixed size
- But, **Linked list** is a dynamic data structure, it is able to grow in size as needed.

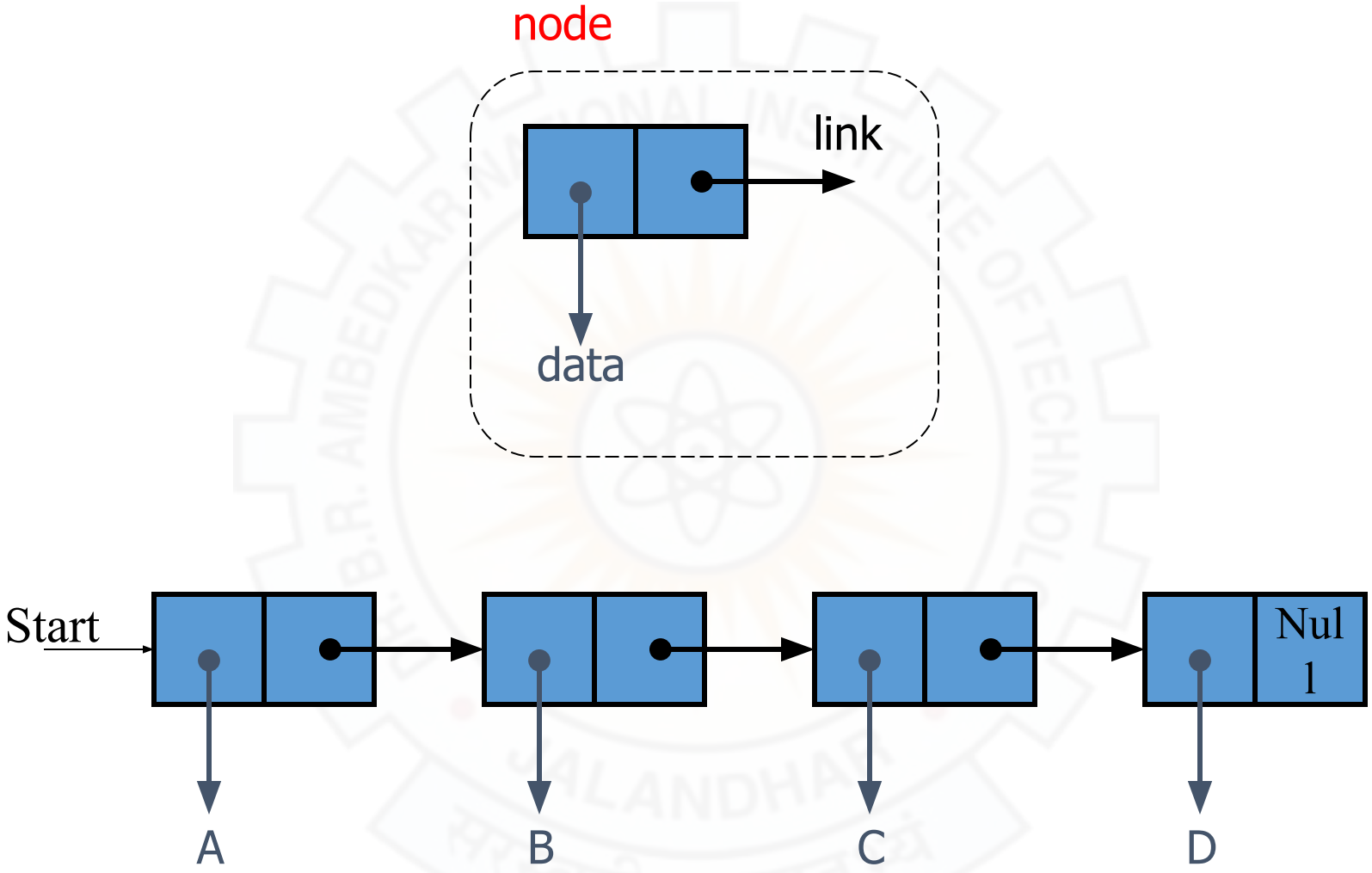
What is Linked List?

- A linked list is a linear collection of homogeneous data elements, called **nodes,** where linear order is maintained by means of links or pointers.

- Each node has two parts:
  - The first part contains the data (information of the element) and
  - The second part contains the address of the next node (link /next pointer field) in the list.

- Data part of the link can be an integer, a character, a String or an object of any kind.




node
link
data
Start
A
B
C
D
Nul
l

Linked Lists

- **Linked list**
  - Linear collection of self-referential structures, called *nodes,* connected by pointer *links.*
  - Accessed via a pointer to the first node of the list.
  - Subsequent nodes are accessed via the link-pointer member stored in each node.
  - Link pointer in the **last node is set to null** to mark the end of list.
  - Data stored dynamically – each node is created as necessary.
  - Length of a list can increase or decrease.
  - Becomes full only when the system has insufficient memory to satisfy dynamic storage allocation requests.

Types of linked lists

- **Singly linked list**
  - Begins with a pointer to the first node
  - Terminates with a null pointer
  - Only traversed in one direction

- **Circular, singly linked list**
  - Pointer in the last node points back to the first node

- **Doubly linked list**
  - Two “start pointers”- first element and last element
  - Each node has a forward pointer and a backward pointer
  - Allows traversals both forwards and backwards

- **Circular, doubly linked list**
  - Forward pointer of the last node points to the first node and backward pointer of the first node points to the last node

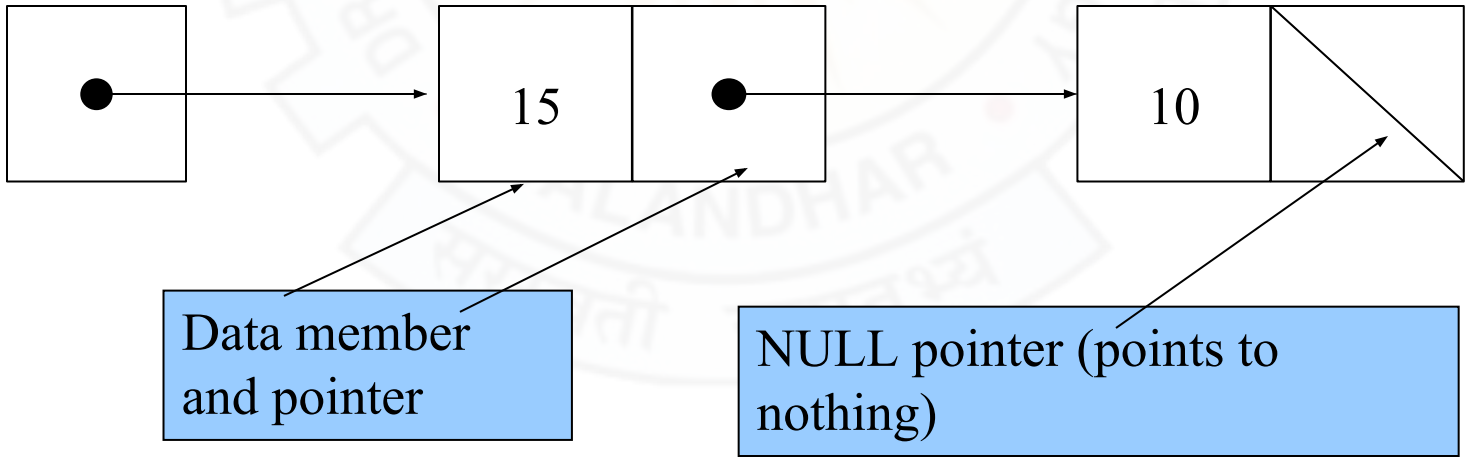
Dynamic Memory Allocation

- Dynamic memory allocation
  - Obtain and release memory during execution
- **malloc**
  - Takes number of bytes to allocate
    - Use **sizeof** to determine the size of an object
  - Returns pointer of type **void ***
    - A **void *** pointer may be assigned to any pointer
    - If no memory available, returns **NULL**
  - **newPtr = (struct node *)malloc(sizeof(struct node));**
  - Here we typecast **void *** into **struct node *.**
- **free**
  - Deallocates memory allocated by **malloc**
  - Takes a pointer as an argument
  - **free (newPtr);**

Self-Referential Structures

- Self-referential structures
  - Structure that contains a pointer to a structure of the same type
  - Can be linked together to form useful data structures such as lists, queues, stacks and trees
  - Terminated with a **NULL** pointer (**0**)

- Two self-referential structure objects linked together



Singly linked list operations

Insertion:

- **Insertion of a node at the front**
- **Insertion of a node at any position in the list**
- **Insertion of a node at the end**

Deletion:

- **Deletion at front**
- **Deletion at any position**
- **Deletion at end**

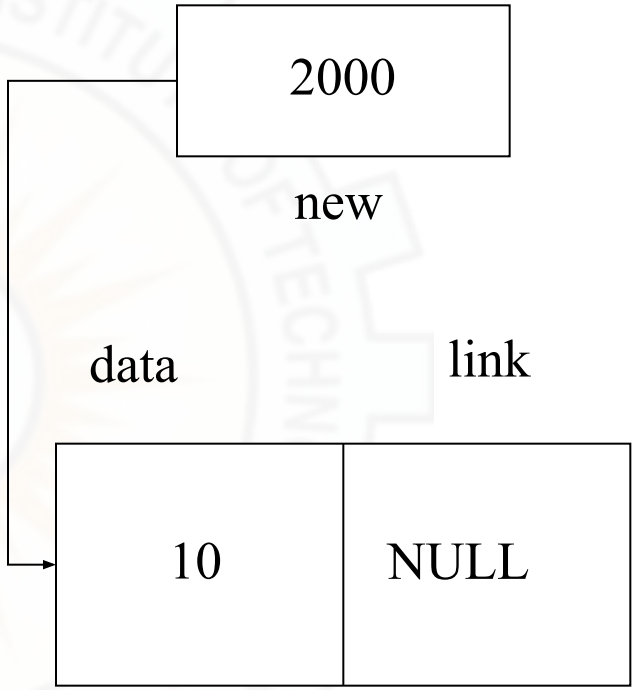
Display:

- **Displaying/Traversing the elements of a list**

Singly linked lists

Node Structure

```
struct node
{
      int data;
      struct node *link;
}*new, *ptr, *start, *ptr1;
```

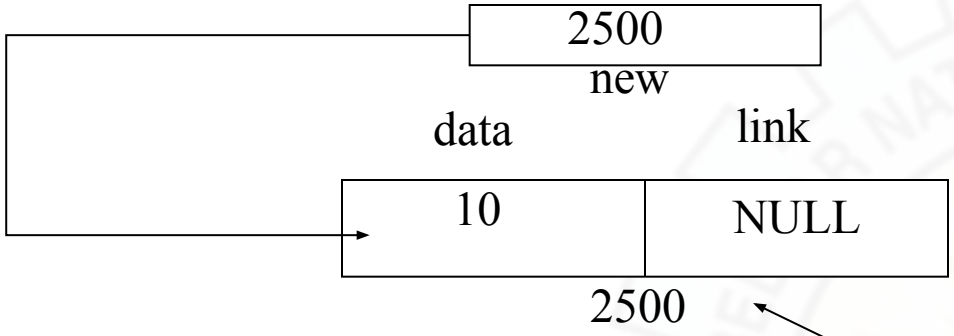


Creating a node

```
new = (struct node *) malloc (sizeof(struct node));
new -> data = 10;
new -> link = NULL;
```

Inserting a node at the beginning

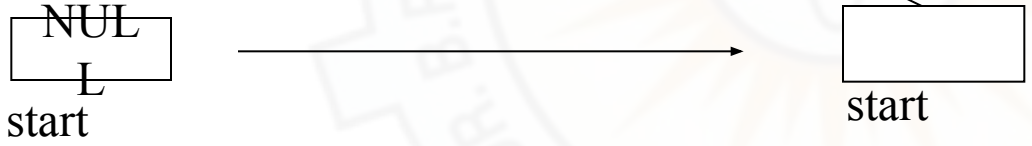
Create a node that is to be inserted



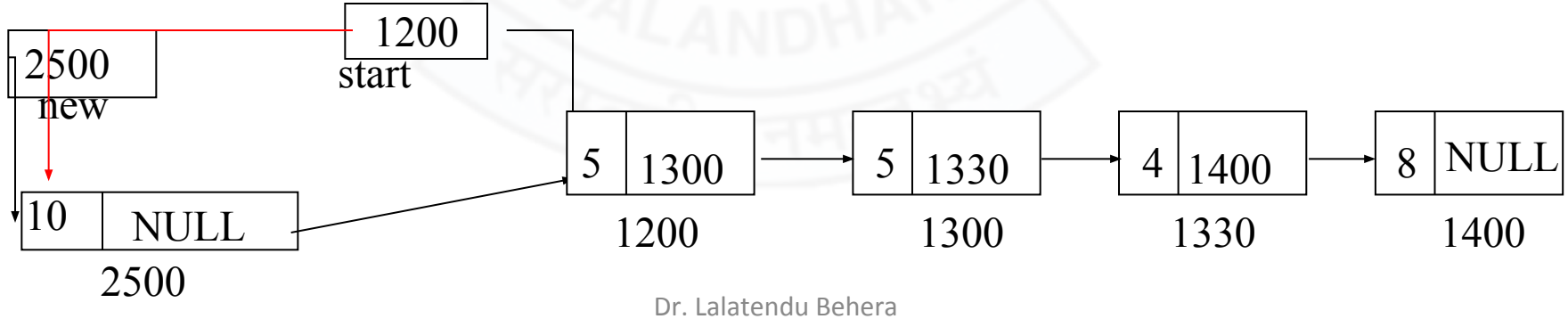
Algorithm:

1.Create a new node.
2.if (start = = NULL)
3. start = new;
4.else
5.{
6. new -> link = start;
7. start = new;
8.}

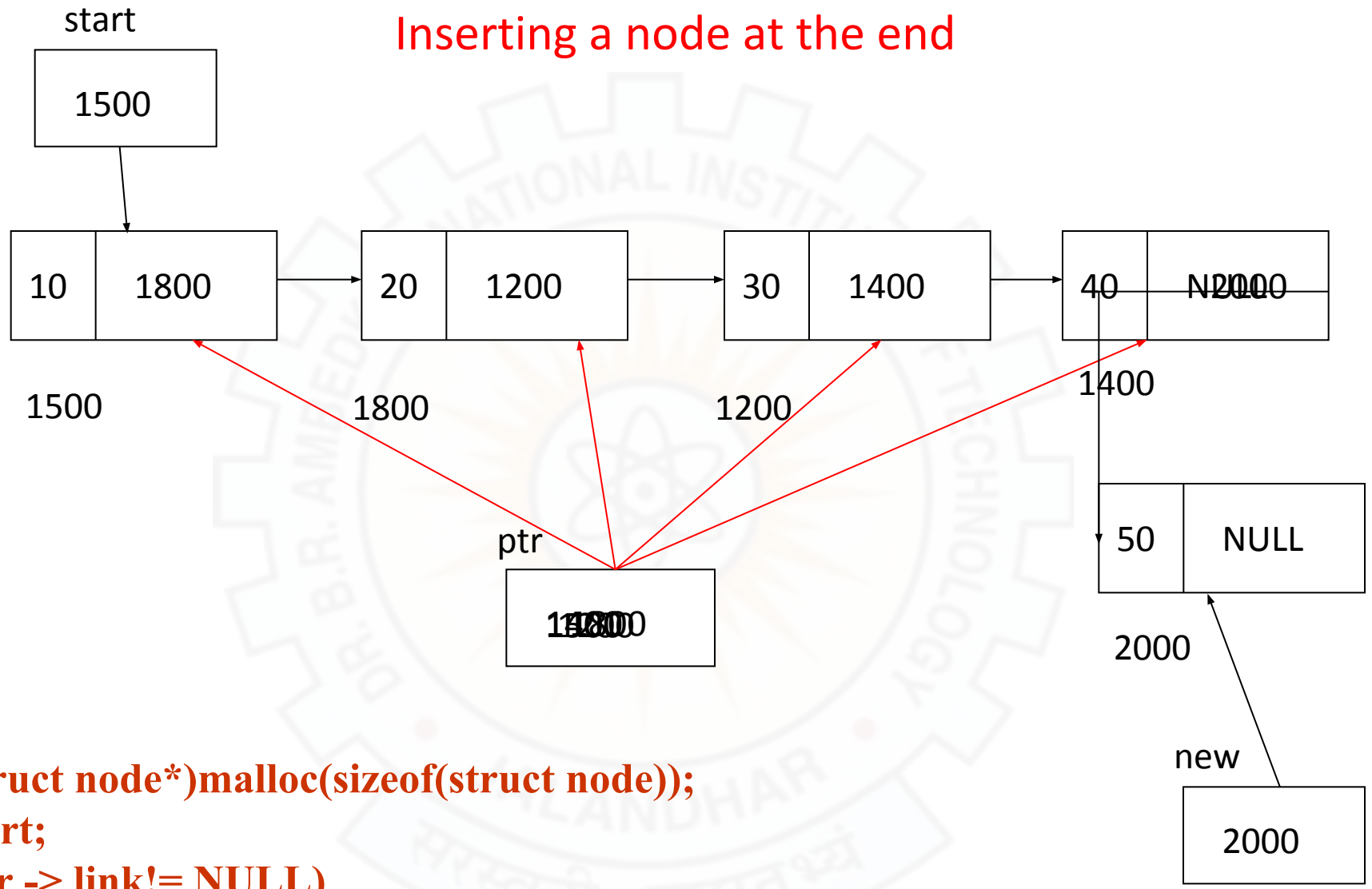
If the list is empty



If the list is not empty



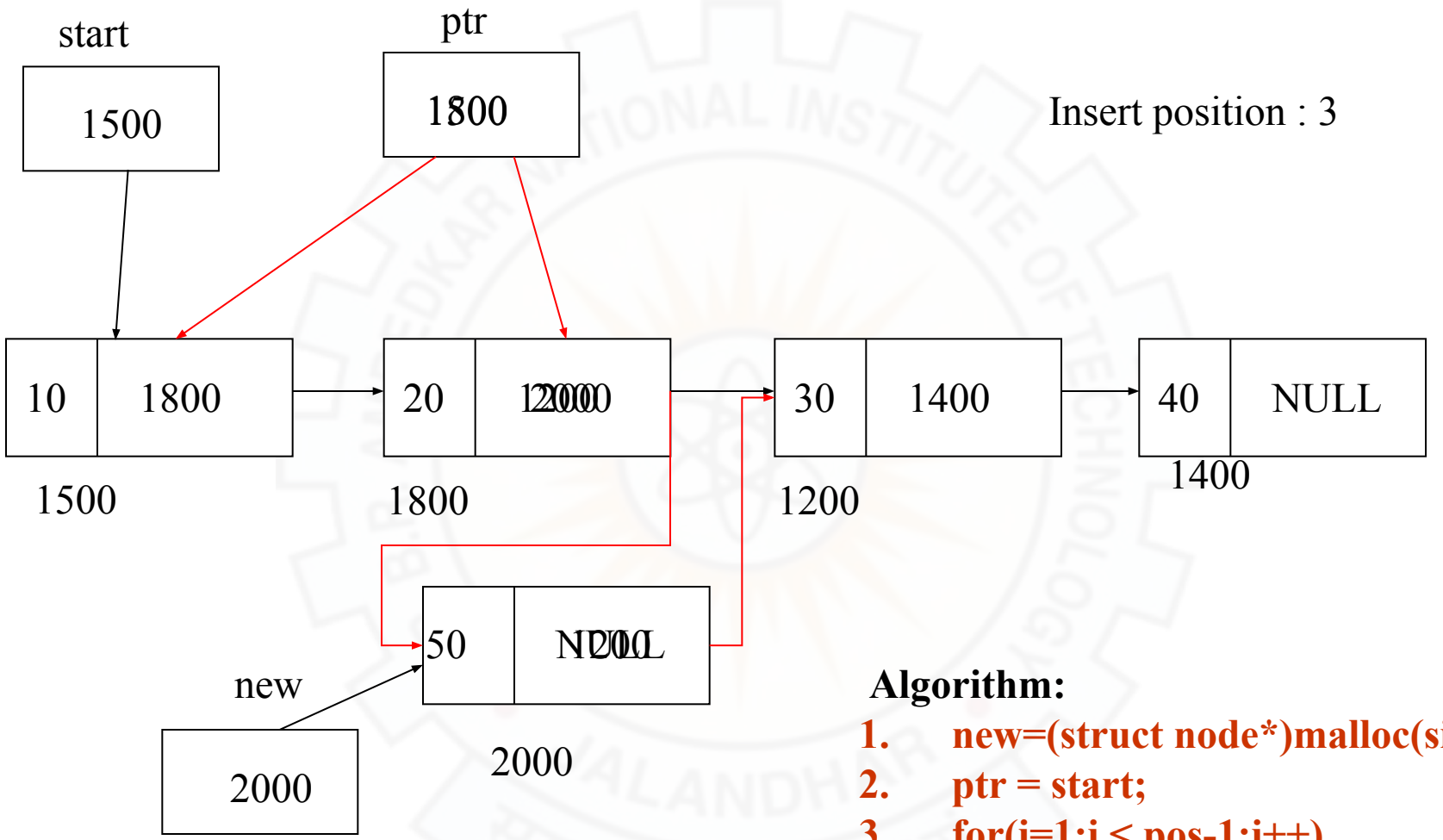
Inserting a node at the end



Algorithm:

1. **new=(struct node*)malloc(sizeof(struct node));**
2. **ptr = start;**
3. **while(ptr -> link!= NULL)**
4. **ptr = ptr -> link;**
5. **ptr -> link = new;**

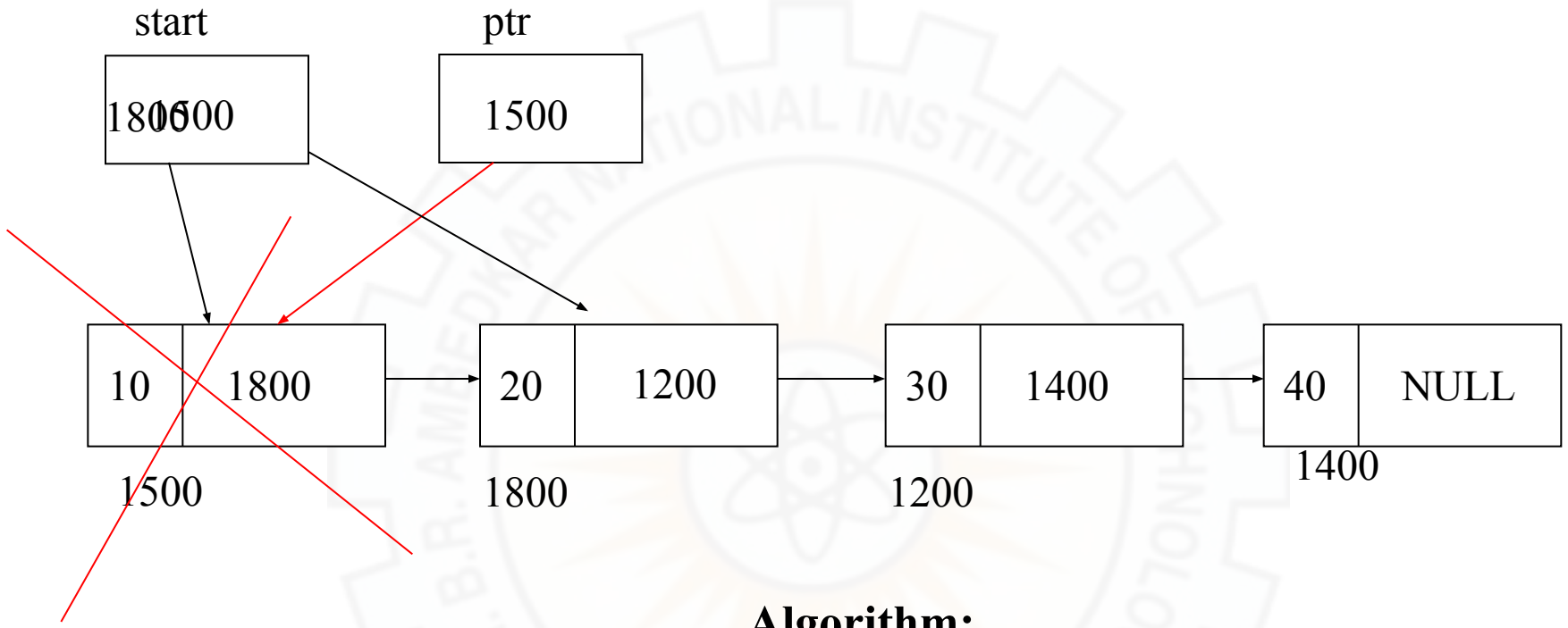
Inserting a node at the given position



Algorithm:

1. `new=(struct node*)malloc(sizeof(struct node));`
2. `ptr = start;`
3. `for(i=1;i < pos-1;i++)`
4. `ptr = ptr -> link;`
5. `new -> link = ptr -> link;`
6. `ptr -> link = new;`

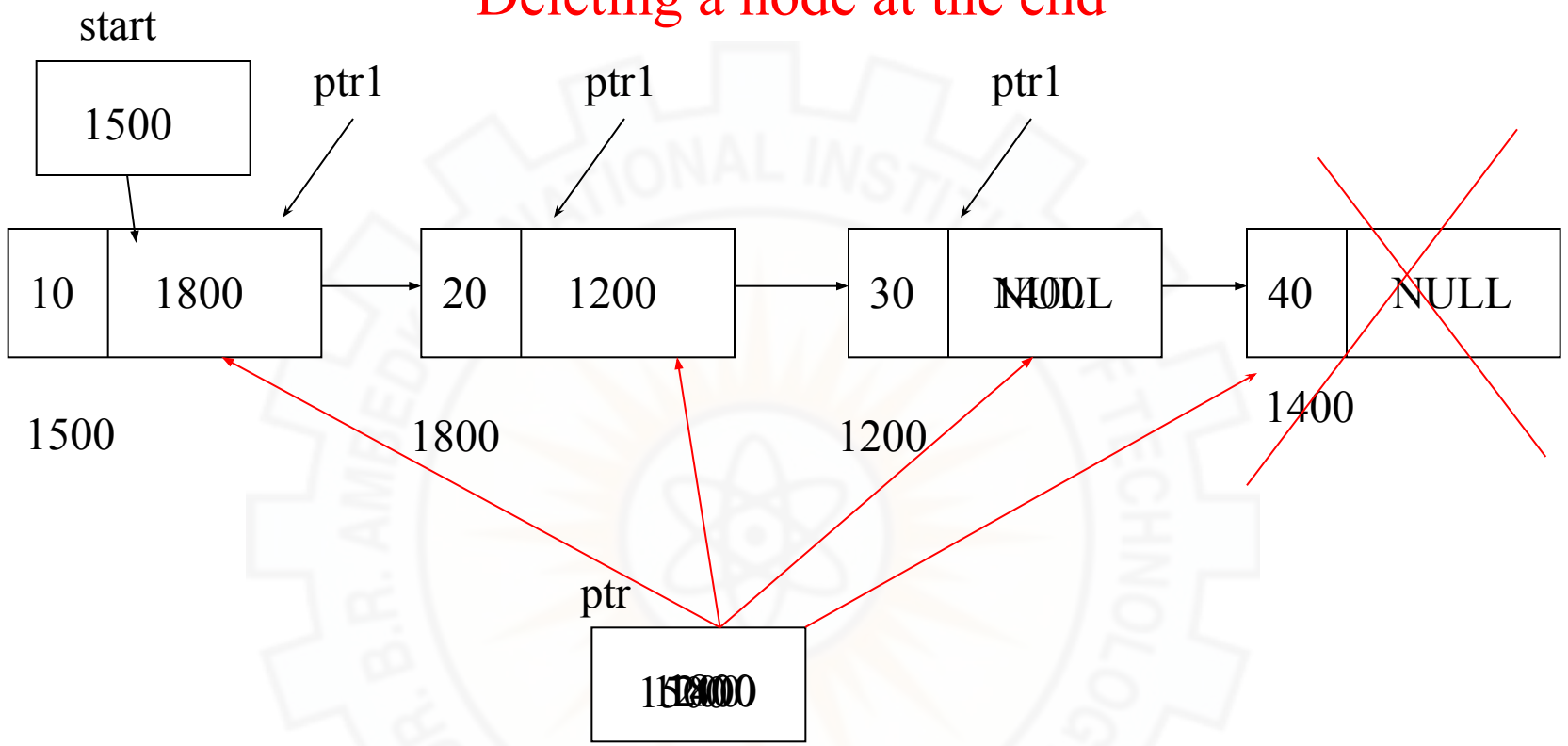
Deleting a node at the beginning



Algorithm:

```
1.  if (start = = NULL)
2.        print  "List is Empty";
3.  else
4.  {
5.        ptr = start;
6.        start = start -> link;
7.        free(ptr);
8.  }
```

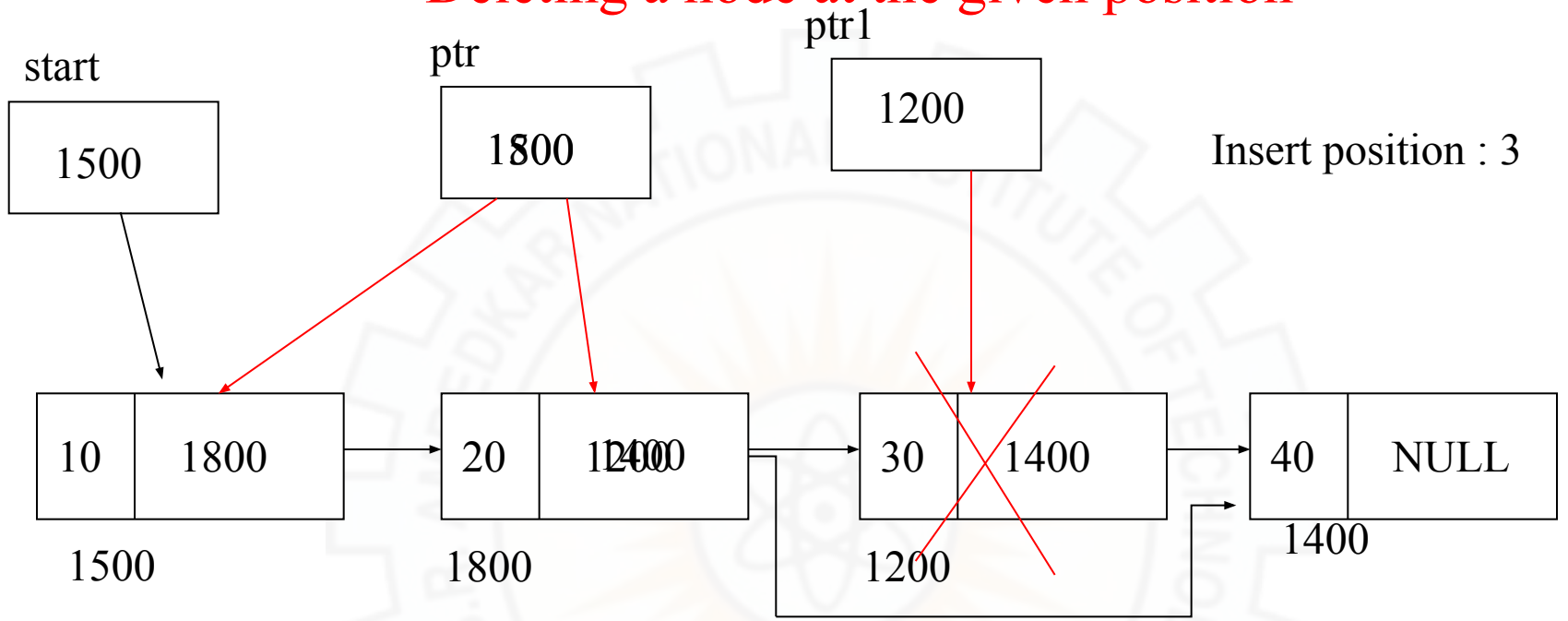
Deleting a node at the end



Algorithm:

1. ptr = start;
2. while(ptr -> link != NULL)
3. ptr1=ptr;
4. ptr = ptr -> link;
5. ptr1 -> link = NULL;
6. free(ptr);

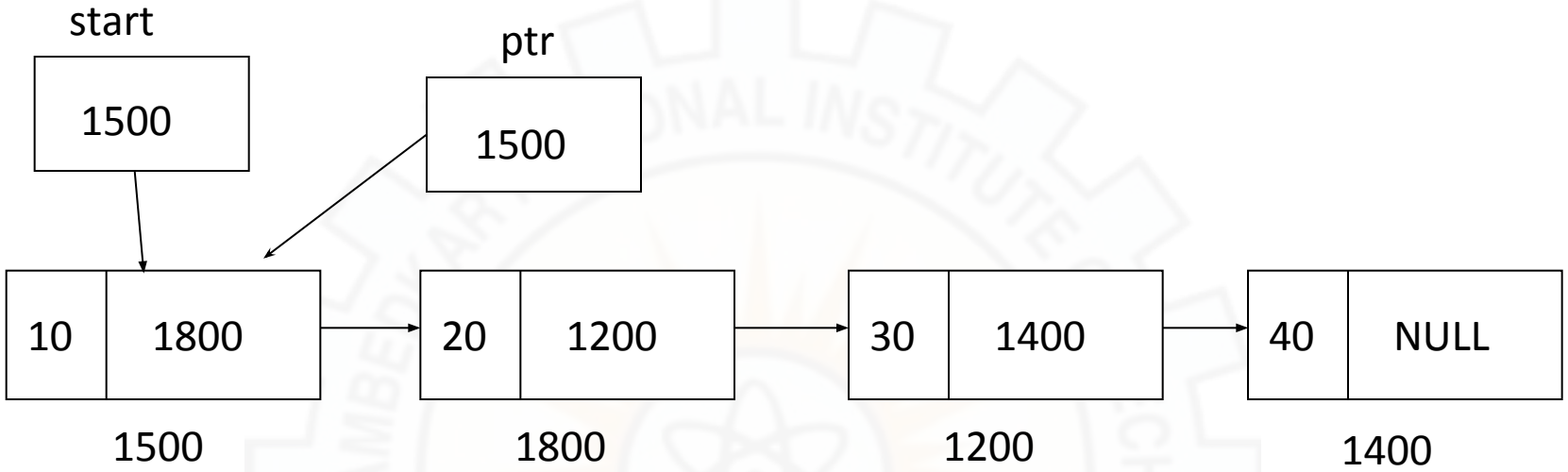
Deleting a node at the given position



Algorithm:

1. ptr = start ;
2. for(i=1;i<pos-1;i++)
3. ptr = ptr -> link;
4. ptr1 = ptr -> link;
5. ptr -> link = ptr1-> link;
6. free(ptr1);

Traversing an elements of a list



Algorithm:

```
1.  if(start = = NULL)
2.        print  "List is empty";
3.  else
4.        for (ptr = start ; ptr != NULL ; ptr = ptr -> link)
5.              print "ptr->data";
```

```
/*Program to implement single linked list*/
#include<stdio.h>        #include<malloc.h>        #include<conio.h>    #include<stdlib.h>
void traverse();        void deletion();        void insertion();
int choice,i,pos,item;
struct node  {
        int data;
        struct node *link;
}*start,*ptr,*ptr1,*new;
void main()   {
        start=NULL;
        ptr=start;
        printf("****Menu****\n");
        printf("\n 1.Insertion\n 2.Deletion\n 3.Traverse\n 4.Search\n 5.Exit\n");
        while(1)    {
                printf("\nenter ur choice");
                scanf("%d",&choice);
                switch(choice)   {
                        case 1:     insertion();        break;
                        case 2:     deletion();         break;
                        case 3:     traverse();         break;
                        case 4:     search();           break;
                        case 5:     exit();
                        default:    printf("\nwrong choice\n");
                }/*end of switch*/
        }/*end of while*/
}/*end of main*/
```

```
void insertion() {
    new=(struct node*)malloc(sizeof(struct node));
    printf("\n enter the item to be inserted\n");
    scanf("%d",&item);
    new->data=item;
    if(start = = NULL)
    {
        new->link=NULL;
        start=new;
    }/*end of if*/
    else
    {
        printf("\nEnter the place to insert the item\n");
        printf("1.Start\n 2.Middle\n 3.End\n");
        scanf("%d",&choice);
        if(choice = = 1)
        {
            new->link=start;
            start=new;
        }
```

```
if(choice = = 2)
{
      ptr=start;
      printf("Enter the position to place an item: ");
      scanf("%d",&pos);
      for(i=1;i<pos-1;i++)
            ptr=ptr->link;
      new->link=ptr->link;
      ptr->link=new;
 }
if(choice = = 3)
{
            ptr=start;
            while(ptr->link!=NULL)
                  ptr=ptr->link;
            new->link=NULL;
            ptr->link=new;
}
}/*end of else*/
}/*end of insertion*/
```

```
void deletion()
{
      ptr=start;
      if(start = = NULL)
      {
            printf("\nThe list is empty");
      }
      else
      {
            printf("\n1.Start \n2.Middle \n3.End");
            printf("\nEnter the place to delete the element from list");
            scanf("%d",&choice);
            if(choice = = 1)
            {
                  printf("\nThe deleted item from the list is -> %d",ptr->data);
                  start=start->link;
            }
```

```
        if(choice = = 2)
        {
                printf("\nEnter the position to delete the element from the list");
                scanf("%d",&pos);
                for(i=0;i<pos-1;i++)
                {
                        ptr1=ptr;
                        ptr=ptr->link;
                }
                printf("\nThe deleted element is ->%d",ptr->data);
                ptr1->link=ptr->link;
        }
        if(choice = = 3)
        {
                while(ptr->link!=NULL)        {
                        ptr1=ptr;
                        ptr=ptr->link;
                }//while
                printf("\nThe deleted element from the list is ->%d", ptr->data);
                ptr1->link=NULL;
        }
    }/*end of else*/
}/*end of deletion*/
```

```
void traverse()
{
      if(start = = NULL)
            printf("List is empty\n");
      else
      {
            printf("\nThe elements in the list are");
            for(ptr=start;ptr!=NULL;ptr=ptr->link)
                  printf("\n\tNode at %d is %d",++i,ptr->data);
      }
}/*end of traverse*/
```

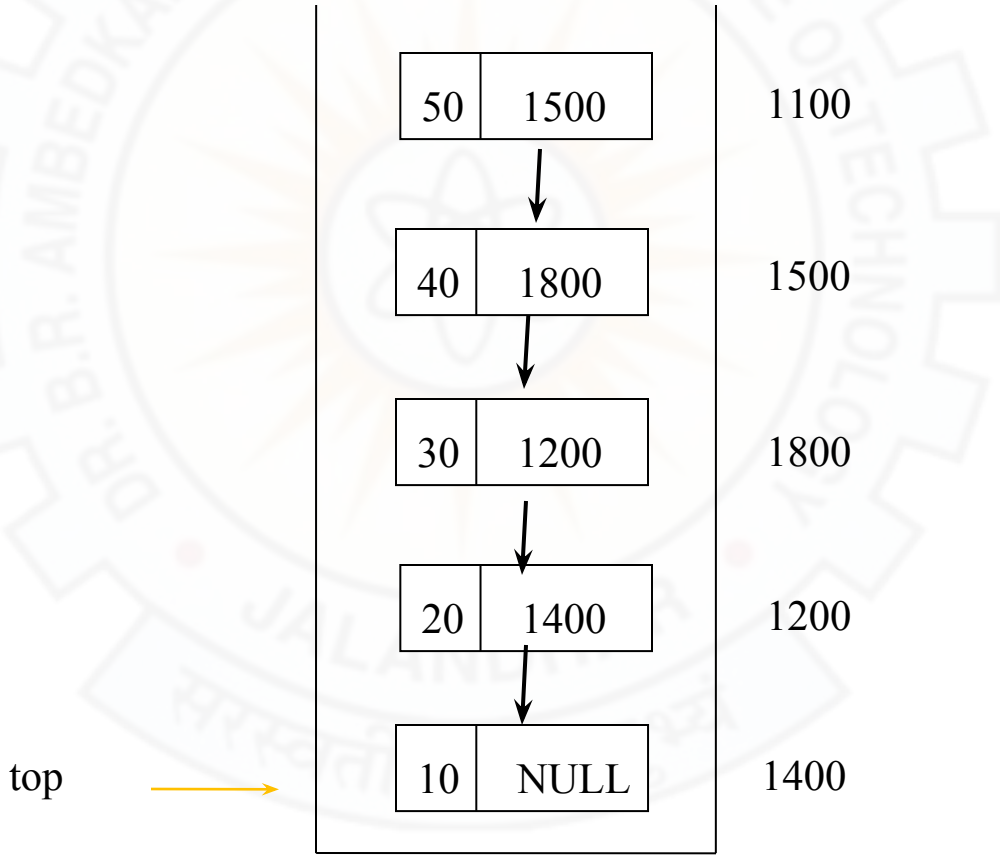
Representing Stack with Linked List

- Disadvantage of using an array to implement a stack or queue is the wastage of space.
- Implementing stacks as linked lists provides a feasibility on the number of nodes by dynamically growing stacks, as a linked list is a dynamic data structure.
- The stack can grow or shrink as the program demands it to.
- A variable **top** always points to top element of the stack.
- top = NULL specifies stack is empty.

Example:

- The following list consists of five cells, each of which holds a data object and a link to another cell.

- A variable, top, holds the address of the first cell in the list.



```
/* Write a c program to implement stack using linked list */
#include<stdio.h>    #include<conio.h>          #include<malloc.h>      #include<stdlib.h>
int push();              int pop();        int display();              int choice,i,item;
struct node {
        int data;
        struct node *link;
}*top,*new,*ptr;
main()  {               top=NULL;
        printf("\n***Select Menu***\n");
        while(1)  {
                printf("\n1.Push \n2.Pop \n3.Display \n4.Exit\n5.Count");
                printf("\n\nEnter ur choice: ");
                scanf("%d",&choice);
                switch(choice) {
                        case 1:    push();          break;
                        case 2:    pop();           break;
                        case 3:    display();       break;
                        case 4:    exit(0);
                        case 5:    count();         break;
                        default: printf("\nWrong choice");
                }/* end of switch */
        }/* end of while */
}/* end of main */
```

```
int push()
{
      new=(struct node*)malloc(sizeof(struct node));
      printf("\nEnter the item: ");
      scanf("%d",&item);
      new->data=item;
      if(top = = NULL)
      {
            new->link=NULL;
      }
      else
      {
            new->link=top;
      }
      top=new;
      return;
}/* end of insertion */
```

```
int pop()
{
      if(top = = NULL)
      {
            printf("\n\nStack is empty");
            return;
      }//if
      else
      {
            printf("\n\nThe deleted element
            is: %d",top->data);

            top=top->link;
      }
      return;
}/* end of pop() */
```

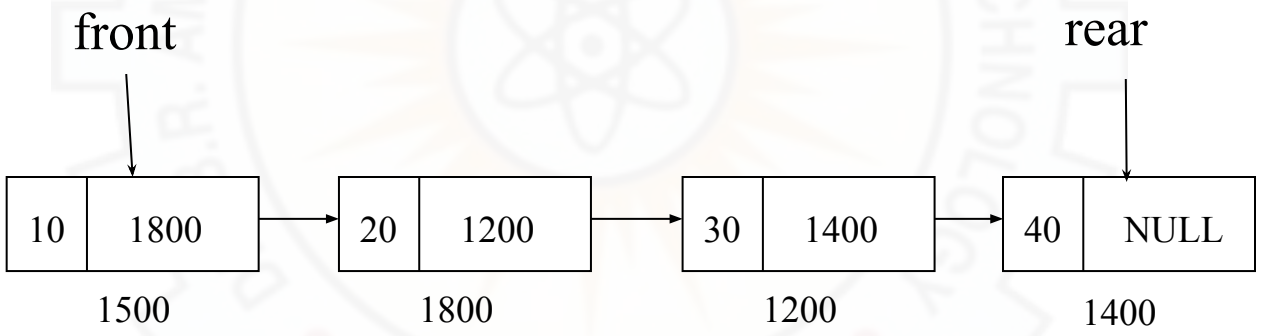
```
int display()
{
        ptr=top;
        if(top = = NULL)
        {
                printf("\nThe list is empty");
                return;
        }
  printf("\nThe elements in the stack are: ");
        while(ptr!=NULL)
        {
                        printf("\n %d",ptr->data);
                        ptr=ptr->link;
        }/* end of while */
        return;
}/* end of display() */
```

```
int count()
{
        int count=1;
        ptr=top;
        if(top = = NULL)
        {
                printf("\nThe list is empty");
                return;
        }
        while(ptr->link!=NULL)
        {
                ++count;
                ptr=ptr->link;
        }
        printf("\n\nThe number of elements in the stack are: %d",count);
        return;
}/* end of count */
```

Representing Queue with Linked List

- New items are added to the end of the list.
- Removing an item from the queue will be done from the front.

- A pictorial representation of a queue being implemented as a linked list is given below.



- The variables **front** points to the front item in the queue and **rear** points to the last item in the queue.

```
/*Write a c program to implement queue using linked list*/
#include<stdio.h>        #include<conio.h>        #include<malloc.h>
#include<stdlib.h>
int choice,i,item;
struct node  {
        int data;
        struct node *link;
}*front,*rear,*new,*ptr;
main()  {
        front=NULL;
        rear=NULL;
        printf("\n\n MENU");
        printf("\n1.Enqueue \n2.Dequeue \n3.Display \n4.Exit");
        while(1)   {
                printf("\nEnter your choice: ");
                scanf("%d",&choice);
                switch(choice)   {
                        case 1:enqueue();          break;
                        case 2:dequeue();          break;
                        case 3:display();          break;
                        case 4:exit(0);
                        default:printf("\nwrong choice");
                }/*end of switch */
        }/*end of while */
```

```
int enqueue()
{
	new=(struct node*)malloc(sizeof(struct node));
	printf("\nenter the item");
	scanf("%d",&item);
	new->data=item;
	new->link=NULL;
	if(front==NULL)
	{
		front=new;
	}
	else
	{
		rear->link=new;
	}
	rear=new;
	return;
}/*end of enqueue */
```

```
display()
{
	if(front==NULL)
		printf("\nThe list is emtpy");
	else
	{
   for(ptr=front;ptr!=NULL;ptr=ptr->link)
		printf("    %d",ptr->data);
	}
	return;
}/* end of display */
```

```
dequeue()
{
    if(front==NULL)
        printf("\nThe list is empty");
    else
    if(front==rear)          /*list has single element*/
    {
        printf("\nThe deleted element is: %d",front->data);
        front=rear=NULL;
    }
    else
    {
        printf("\nThe deleted element is: %d",front->data);
        front=front->link;
    }
    return;
}/*end ofdequeue*/
```

Doubly linked list

- In a singly linked list one can move from the start node to any node in one direction only (left-right).

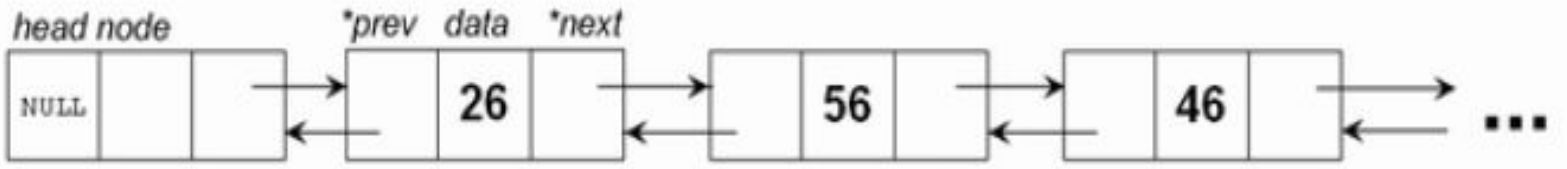
- A doubly linked list is a two-way list because one can move in either direction. That is, either from left to right or from right to left.

- It maintains two links or pointer. Hence it is called as doubly linked list.

Structure of the node

- Where, DATA field - stores the element or data, PREV- contains the address of its previous node, NEXT- contains the address of its next node.

An example of a doubly linked list



Doubly linked list operations

Insertion:

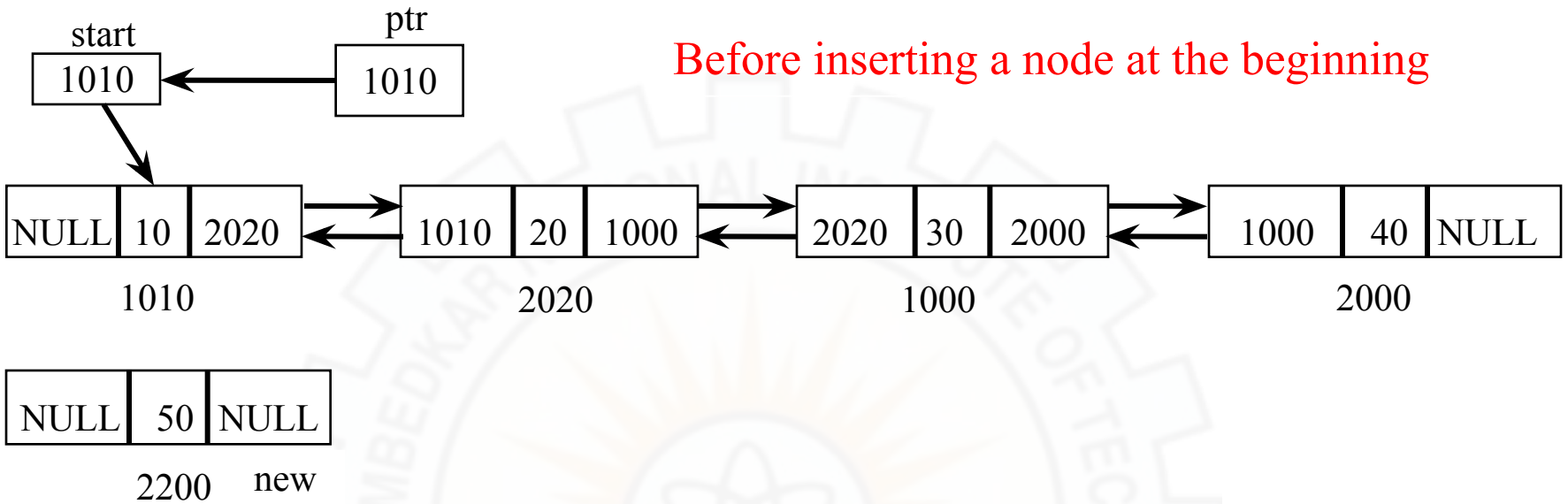
- **Insertion of a node at the front**
- **Insertion of a node at any position in the list**
- **Insertion of a node at the end**

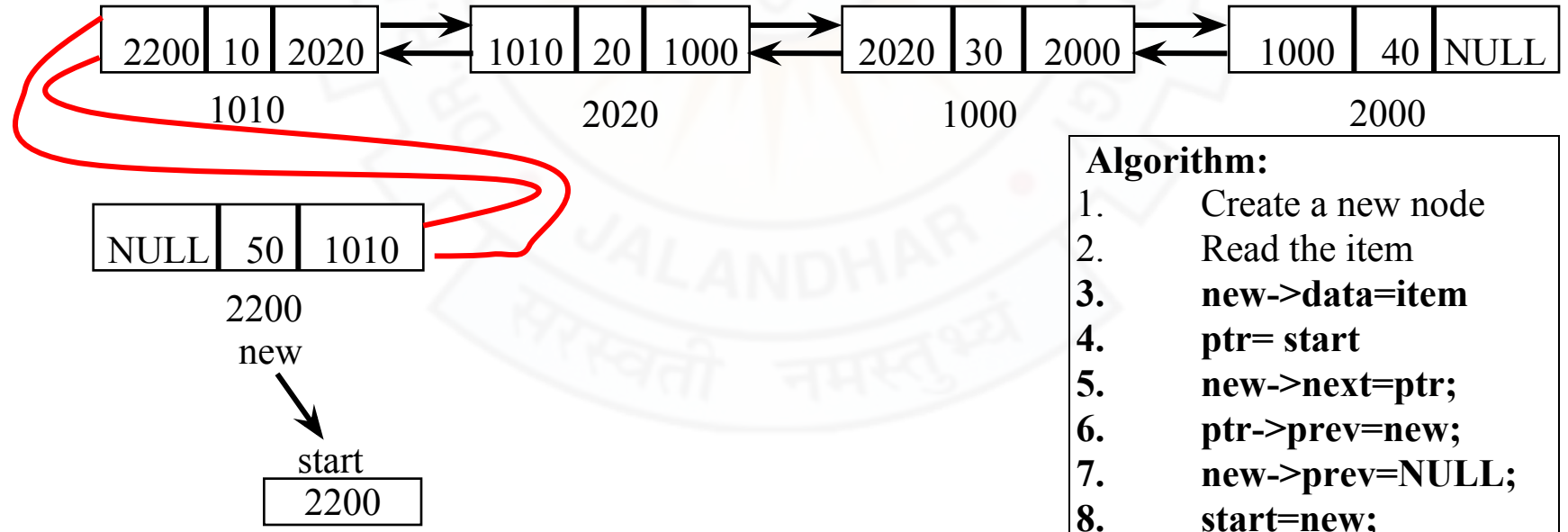
Deletion:

- **Deletion at front**
- **Deletion at any position**
- **Deletion at end**

Display:

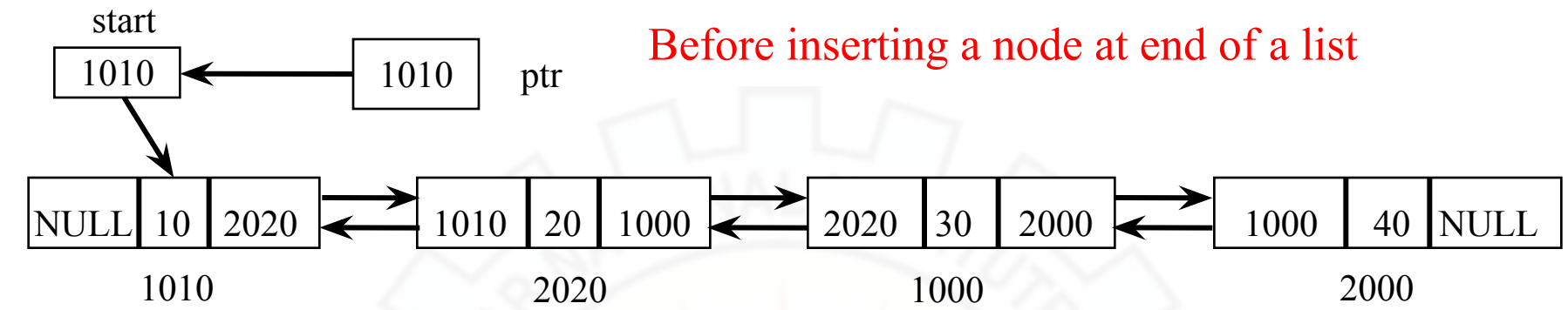
- **Displaying/Traversing the elements of a list**

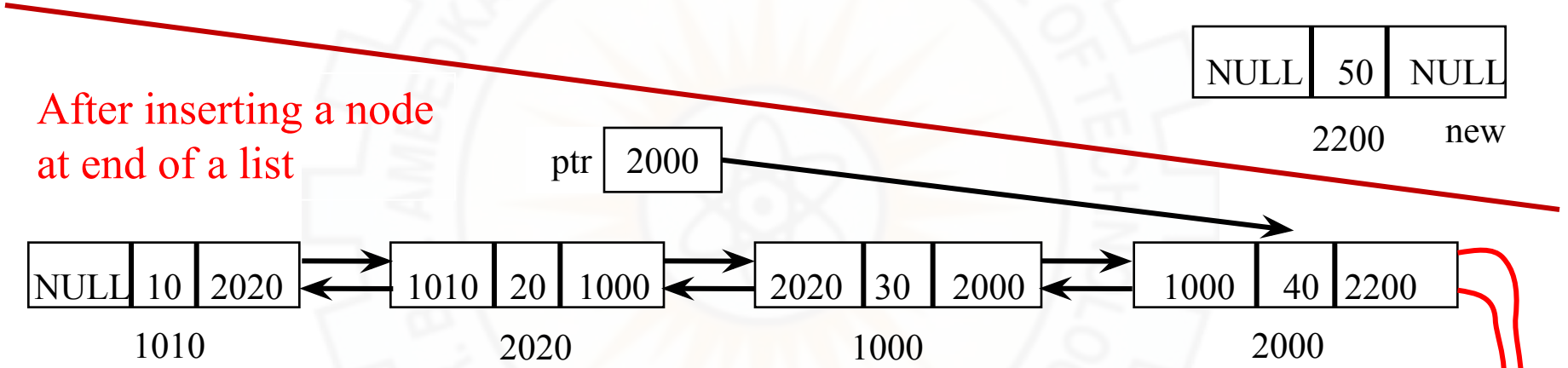




Algorithm:

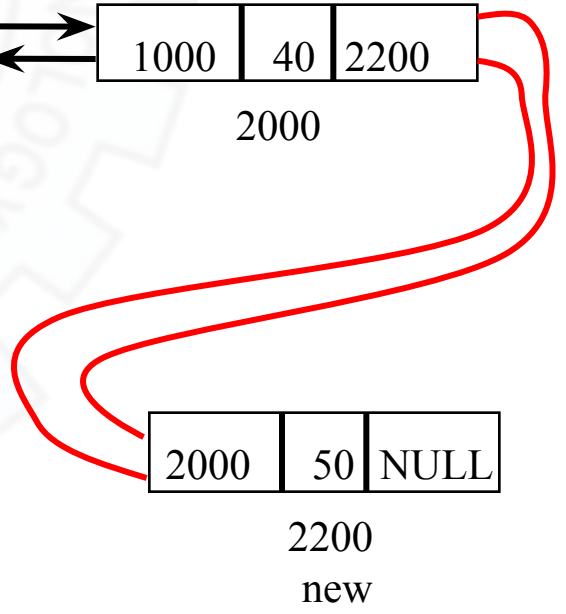
1. Create a new node
2. Read the item
3. **new->data=item**
4. **ptr= start**
5. **new->next=ptr;**
6. **ptr->prev=new;**
7. **new->prev=NULL;**
8. **start=new;**





Algorithm:

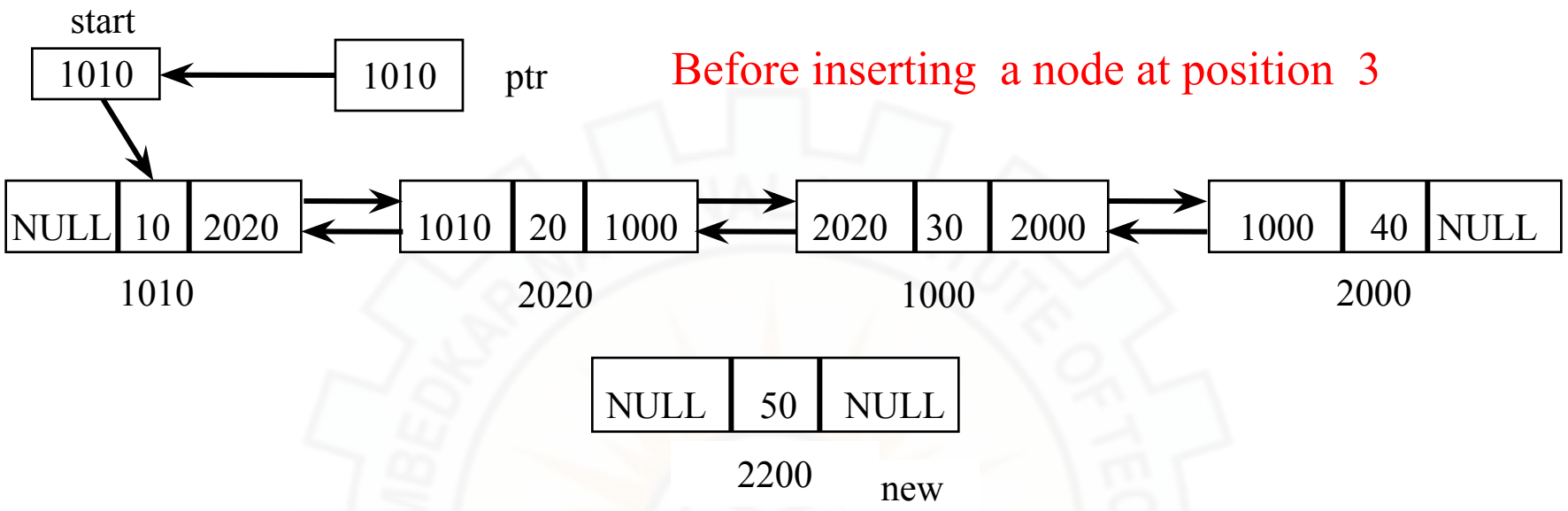
```
1.      Create a new node
2.      Read the item
3.      new->data=item
4.      ptr= start
5.      while(ptr->next!=NULL)
          1.        ptr=ptr->next;
6.      new->next=NULL;
7.      new->prev=ptr;
8.      ptr->next=new;
```

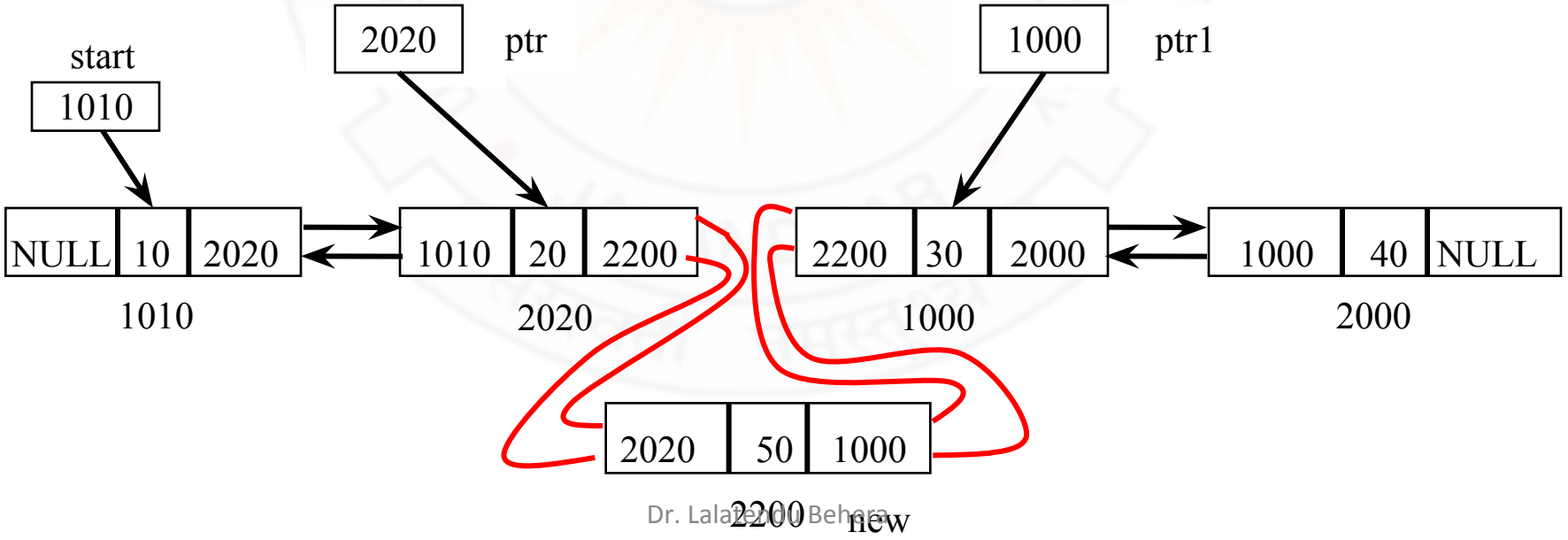


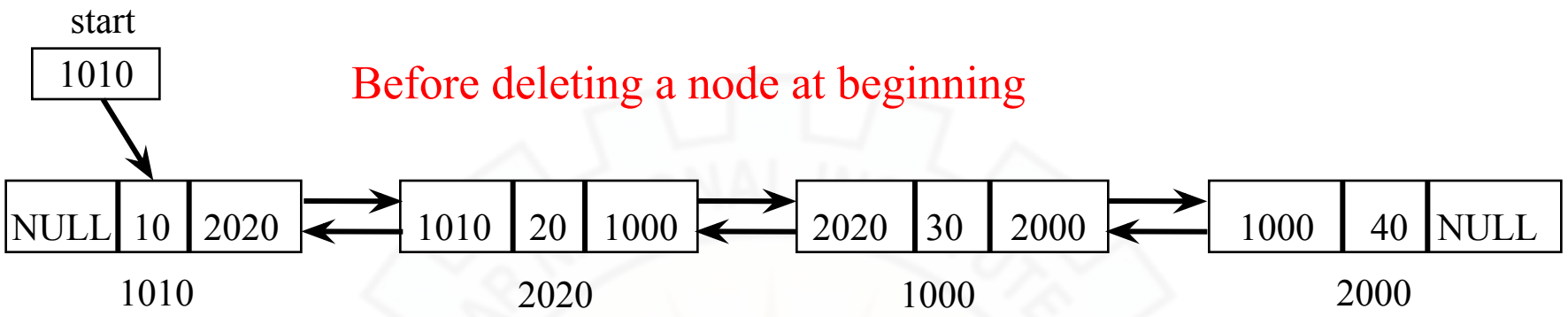
Insertion of a node at any position in the list

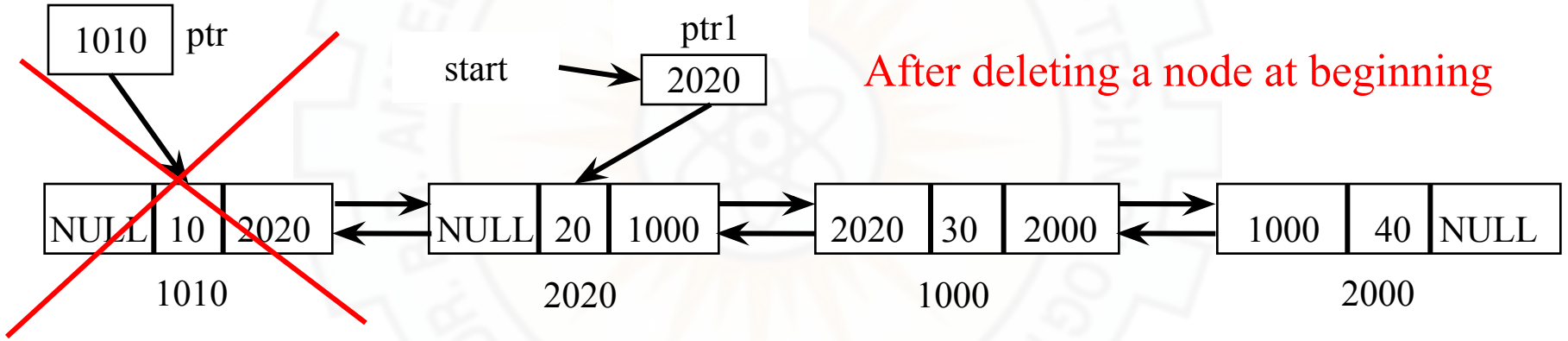
Algorithm:

1. create a node new
2. read item
3. new->data=item
4. ptr=start;
5. Read the position where the element is to be inserted
6. for(i=1;i<pos-1;i++)
   - 6.1 ptr=ptr->next;
7. if(ptr->next = = NULL)
   - 7.1 new->next = NULL;
   - 7.2 new->prev=ptr;
   - 7.3 ptr->next=new;
8. else
   - 8.1 ptr1=ptr->next;
   - 8.2 new->next=ptr1;
   - 8.3 ptr1->prev=new;
   - 8.4 new->prev=ptr;
   - 8.5 ptr->next=new;
9. end


start
1010
1010
ptr
Before inserting a node at position 3
NULL
10
2020
1010
20
1000
2020
30
2000
1000
40
NULL
1010
2020
1000
2000
NULL
50
NULL
2200
new


After inserting a node at position 3
2020
ptr
1000
ptr1
start
1010
NULL
10
2020
1010
20
2200
2200
30
2000
1000
40
NULL
1010
2020
1000
2000
2020
50
1000
2200
new





Algorithm:

1.ptr=start

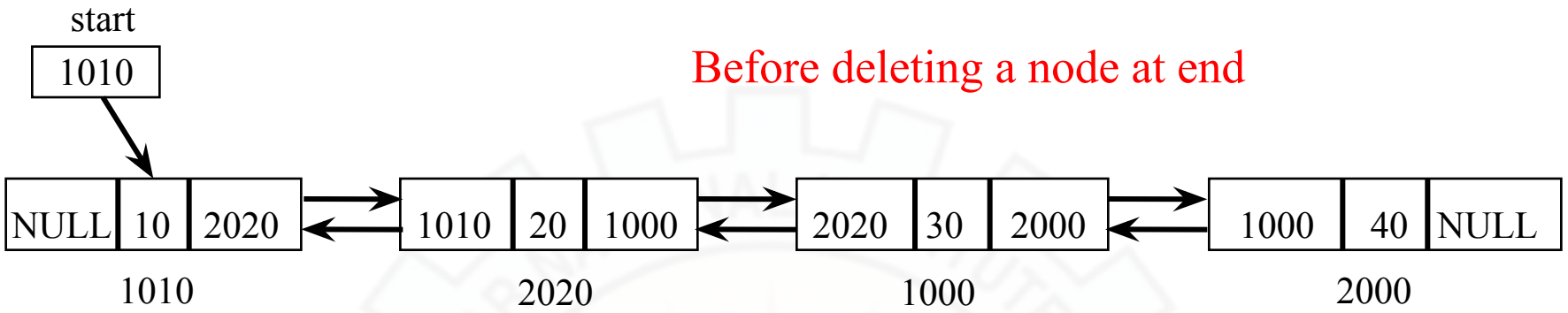
2.ptr1=ptr->next;

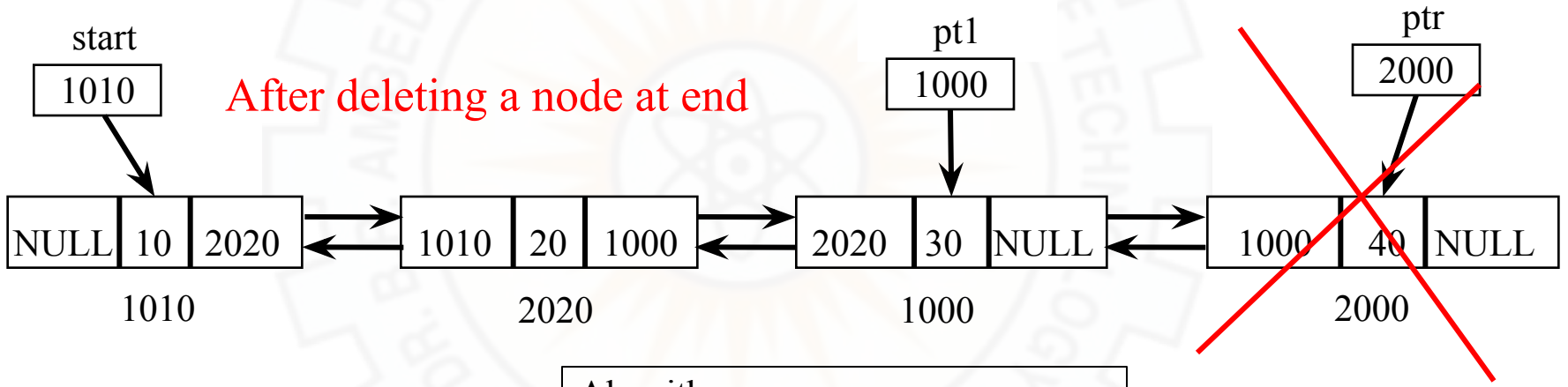
3.start=ptr1;

4.if(ptr1!=NULL)

1.ptr1->prev=NULL;

5. free(ptr);





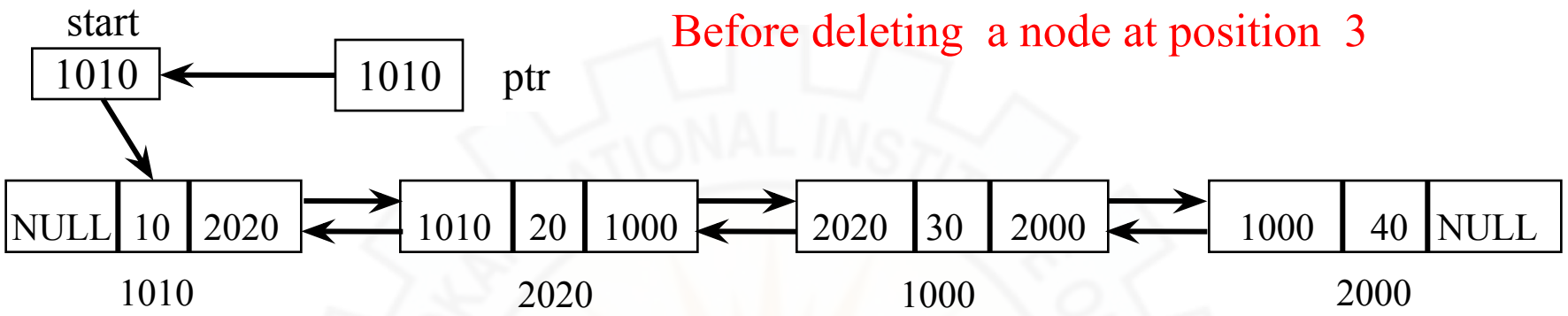
Algorithm:

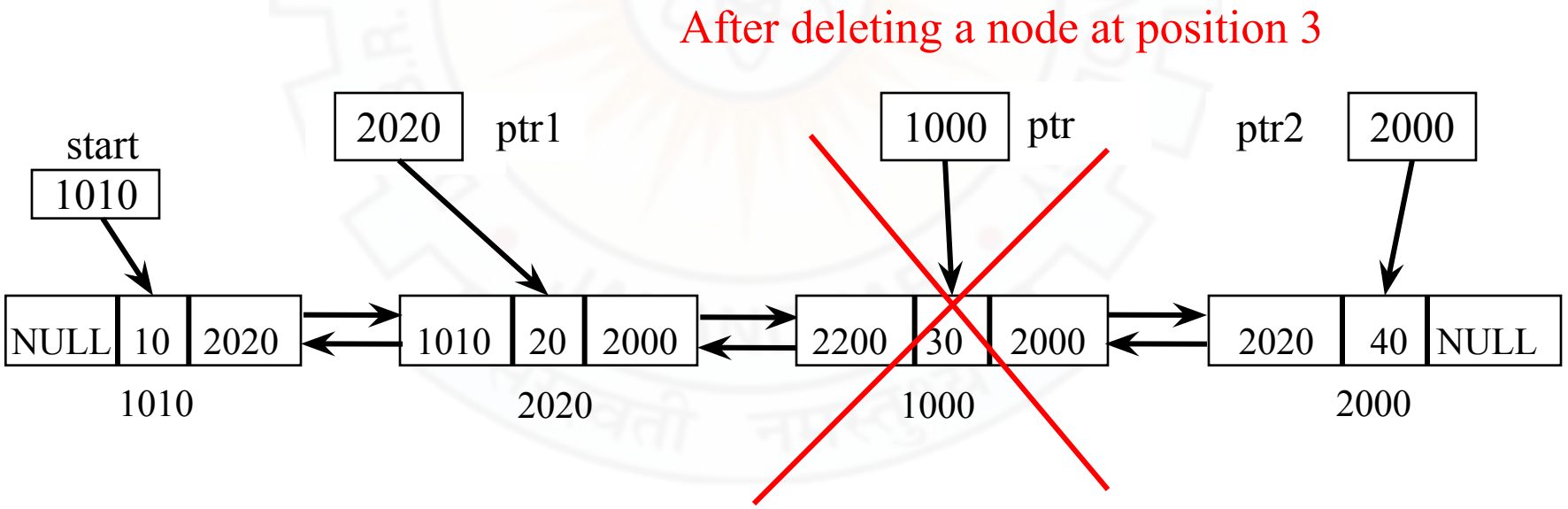
1. ptr=start
2. while(ptr->next!=NULL)
   1. ptr=ptr->next;
3. end while
4. ptr1=ptr->prev;
5. ptr1->next=NULL;

Deletion at any position

Algorithm:

1. ptr=start;
2. **while(ptr->next!=NULL)**
    - **1.for(i=0;i<pos-1;i++)**
        1. ptr=ptr->next;
    - **2.if(i = = pos-1)**
        1. break;
3. end while
4. if(ptr = = start)
    //if the deleted item is first node
    - 4.1 ptr1=ptr->next;
    - 4.2 ptr1->prev=NULL;
    - 4.3 start=ptr1;
    - 4.4 end if
5. else
    - 5.1 ptr1=ptr->prev;
    - 5.2 ptr2=ptr->next;
    - 5.3 ptr1->next=ptr2;
    - 5.4 ptr2->prev=ptr1;
6. end else
7. end if


Before deleting a node at position 3
start
1010
1010
ptr
NULL
10
2020
1010
20
1000
2020
30
2000
1000
40
NULL
1010
2020
1000
2000

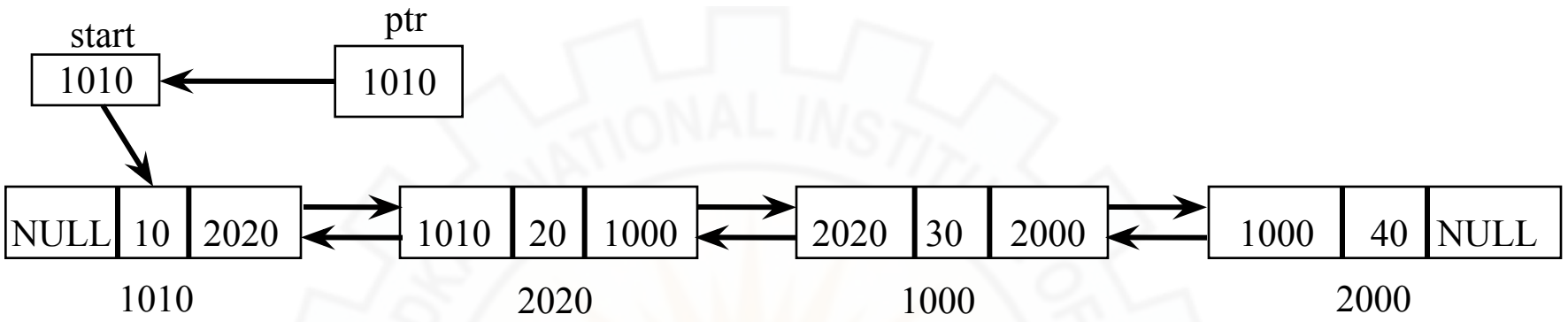

After deleting a node at position 3
start
1010
2020
ptr1
1000
ptr
ptr2
2000
NULL
10
2020
1010
20
2000
2200
30
2000
2020
40
NULL
1010
2020
1000
2000

Displaying elements of a list

Algorithm:

1. ptr=start;
2. if(start = = NULL)
    1. printf("The list is empty\n");
3. else
    1. print “The elements in forward order: “
    2. while(ptr!=NULL)
        1. print “ptr->data”;
        2. if(ptr->next = = NULL)
            1. break;
        3. ptr=ptr->next;
    3. print “The elements in reverse order: “
    4. while(ptr!=start)
        1. if(ptr->next = = NULL)
            1. print “ptr->data”;
        2. else
            1. print “ptr->data”;
            2. ptr=ptr->prev;
            3. print “ptr->data”;

        3.end else
4. end else



Forward Order : 10 20 30 40

Reverse Order : 40 30 20 10

```
/*Program to implement operations of double linked list*/
#include<stdio.h>        #include<conio.h>        #include<malloc.h>
void insertion();        void deletion();         void traverse();
int i,pos,item,choice;
struct node  {
      int data;
      struct node *next;
      struct node *prev;
}*new,*start,*ptr,*ptr1,*ptr2;
void main()  {
      start=NULL;
      printf(" ***** MENU ****");
      printf("\n1.Insertion \n2.Deletion \n3.Traverse \n4.Exit\n");
      while(1) {
            printf("\n\nEnter your choice: ");
            scanf("%d",&choice);
            switch(choice)  {
                  case 1:    insertion();          break;
                  case 2:    deletion();           break;
                  case 3:    traverse();           break;
                  case 4:    exit();
                  default: printf("\nWrong choice");
            }/* end of switch */
      }/* end of while */
```

```
void insertion()  {
        ptr=start;
        new=(struct node*)malloc(sizeof(struct node));
        printf("\nEnter the item to be inserted: ");
        scanf("%d",&item);
        new->data=item;
        if(start==NULL)     {
                new->prev=NULL;
                new->next=NULL;
                start=new;
        }
        else  {
                printf("\nSelect the place:");
                printf("\n1.Start \n2.Middle \n3.End\n");
                scanf("%d",&choice);
                if(choice==1)  {
                        new->next=ptr;
                        ptr->prev=new;
                        new->prev=NULL;
                        start=new;
                }/* choice1 */
```

```
if(choice==2)
{
	printf("\nEnter the position to place the new element: ");
	scanf("%d",&pos);
	for(i=1;i<pos-1;i++)
		ptr=ptr->next;
	if(ptr->next==NULL)
	{
		new->next=NULL;
		new->prev=ptr;
		ptr->next=new;
	}
	else
	{
		ptr1=ptr->next;
		new->next=ptr1;
		ptr1->prev=new;
		new->prev=ptr;
		ptr->next=new;
	}
}/* choice2 */
```

```
		if(choice==3)
		{
			while(ptr->next!=NULL)
				ptr=ptr->next;
			new->next=NULL;
			new->prev=ptr;
			ptr->next=new;
		}/* choice3 */
	}/* end of else */
}/* end of insertion */
```

```
void deletion()
{
        ptr=start;
        if(start==NULL)
                printf("The list is empty\n");
        else
        {
                printf("\Select the place:");
                printf("\n1.Start \n2.Middle \n3.End\n");
                scanf("%d",&choice);
                if(choice==1)
                {
                        printf("\nThe deleted item is: %d",ptr->data);
                        ptr1=ptr->next;
                        start=ptr1;
                        if(ptr1!=NULL)
                                ptr1->prev=NULL;
                }/* choice1 */
```

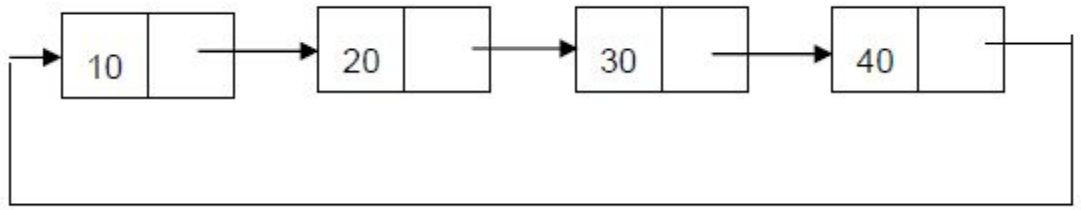
```
if(choice==2)  {
      printf("\nEnter the position to delete the element: ");
            scanf("%d",&pos);
            while(ptr->next!=NULL)  {
                  for(i=0;i<pos-1;i++)
                        ptr=ptr->next;
                        if(i==pos-1)
                              break;
            }//while
            printf("\n\nThe deleted node is: %d",ptr->data);
            if(ptr==start)//deleted item is starting node
            {
                  ptr1=ptr->next;
                  ptr1->prev=NULL;
                  start=ptr1;
            }//if
            else  {
                  ptr1=ptr->prev;
                  ptr2=ptr->next;
                  ptr1->next=ptr2;
                  ptr2->prev=ptr1;
            }
      }/* choice2 */
}/* end of else */
```

```
if(choice==3)
{
      while(ptr->next!=NULL)
            ptr=ptr->next;
   printf("\n\nThe deleted node is: %d",ptr->data);
      ptr1=ptr->prev;
      ptr1->next=NULL;
  }/* choice3 */
}/*end of deletion */
```

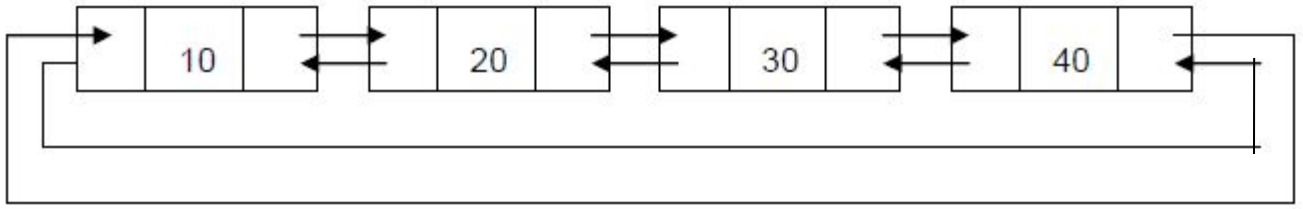
```
void traverse(){
        ptr=start;
        if(start==NULL)
                printf("The list is empty\n");
        else  {
                printf("\n\nThe elements in forward order: ");
                while(ptr!=NULL)  {
                        printf("  %d",ptr->data);
                        if(ptr->next==NULL) {
                                break;
                        }
                        ptr=ptr->next;
                }/* end of while */
                printf("\n\nThe elements in reverse order: ");
                while(ptr!=start)  {
                        if(ptr->next==NULL)
                                printf("  %d",ptr->data);
                        else
                                printf("  %d",ptr->data);
                        ptr=ptr->prev;
                }/* end of while */
                printf("  %d",ptr->data);
        }/* end of else */
}/* end of traverse() */
```

Circular linked list

- The linked list where the last node points the start node is called circular linked list.



Circular singly linked list



Circular doubly linked list

```
/* Write a c program to implement circular linked list*/
#include<stdio.h>      #include<malloc.h>     #include<stdlib.h>
int choice,i,item;
struct node  {
        int data;
        struct node *link;
}*front,*rear,*new,*ptr1,*ptr;
main()  {
        front=rear=NULL;
        printf("\n select menu\n");
        while(1) {
                printf("\n1.Enqueue \n2.Dequeue \n3.Display \n4.Exit");
                printf("\nEnter ur choice: ");
                scanf("%d",&choice);
                switch(choice)  {
                        case 1:    enqueue();           break;
                        case 2:    dequeue();           break;
                        case 3:    display();           break;
                        case 4:    exit(0);
                        default: printf("\nWrong choice.");
                }/*end of switch*/
        }/*end of while*/
}/*end of main*/
```

```
int enqueue()
{
	new=(struct node*)malloc(sizeof(struct node));
	printf("\nEnter the item: ");
	scanf("%d",&item);
	new->data=item;
	if(front==NULL)
		front=new;
	else
		rear->link=new;
		rear=new;
		rear->link=front;
		return;
}/*end of enqueue()*/
```

```
dequeue()
{
	if(front==NULL)
		printf("\nThe circular list is empty.");
	else
	if(front==rear)
	{
	   printf("\nThe deleted element is: %d",front->data);
	   front=rear=NULL;
	}
	else
	{
	   printf("\nThe deleted element is: %d",front->data);
	   front=front->link;
	   rear->link=front;
	}
	return;
}/*end of dequeue*/
```

```
display()
{
        ptr=front;
        ptr1=NULL;
        if(front==NULL)
                printf("\nThe circular list is empty.");
        else
        {
                printf("\nElements in the list are: ");
                while(ptr!=ptr1)
                {
                        printf("  %d",ptr->data);
                        ptr=ptr->link;
                        ptr1=front;
                }/*end of while*/
                return;
        }/*end of else*/
}/*end of display*/
```