Linked List

Prof. Harish D.G.
Dept. of Computer and IT
College of Engineering,Pune
www.harishgadade.com

Linked List

- Introduction
- Definition
- Advantages
- Disadvantages

Node Representation

Key Terms:

- Data Pointer
- Linked Field / Next Address
- Null Pointer
- External Pointer
- Empty List

Linked List

Representation of Node in C

- Simple Linked Structure
- Complex Linked Structure

Linked List

- Simple Linked Structure

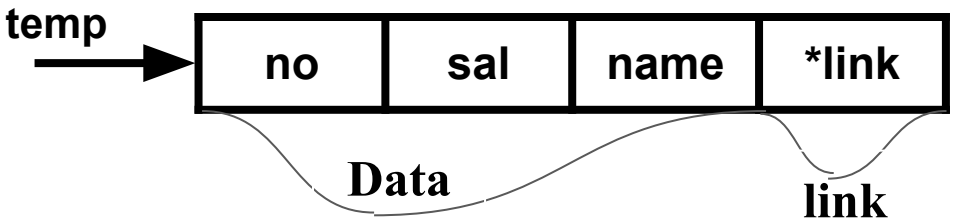
Node Representation

```
Struct node
{
    int data;
    struct node *link;
};

typedef  struct node NODE;
```

Linked List

- Complex Linked Structure



Node Representation

```
Struct student
{
     int no;
     float sal;
     char name[20];
};

Struct node
{
     struct student data;
     struct node *link;
};

typedef struct node NODE;
```

Operations Linked List

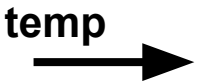
- Create a Node
- Traversing SLL
- Create a SLL
- Append Node(At Beg and at End)
- Search
- Insert
- Delete
- Display

Operations Linked List

- Create a Node

1. Create a pointer, say temp

```
NODE *temp;
```

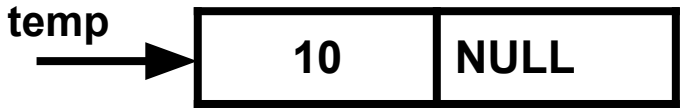


2. Reserve memory for a node

```
temp = (NODE*)malloc(sizeof(NODE));
```

3. Assign value to data field and NULL to Link field

```
temp->data = 10
temp->link = NULL
```

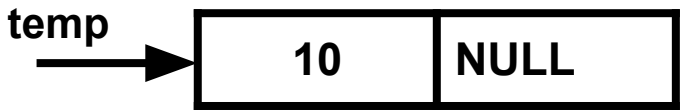


4. Return temp

Operations Linked List

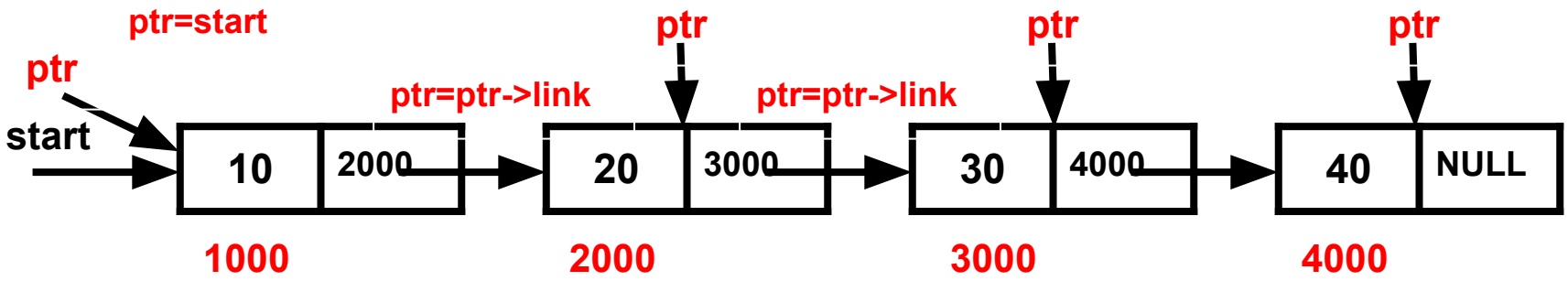
- Create a Node

```
NODE *getnode()
{
    NODE *temp;
    temp = (NODE*)malloc(sizeof(NODE));
    printf ( "Enter Data");
    scanf(" %d ", &temp->data);
    temp->link = NULL;
    return (temp);
}
```



Operations Linked List

- Traversing SLL




```
NODE *findlast(NODE *start)
{
     NODE *ptr;
     for(ptr=start;ptr->link!=NULL;ptr=ptr->link);
     return (ptr);
}
```

Operations Linked List

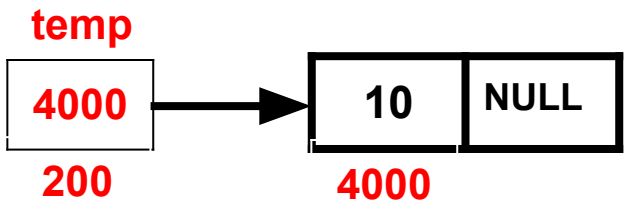
- Creation of SLL

```
1. Create a new Node
2. Check the value of start
   If start = = NULL (it means, list is empty )
      Attach newly created node to start
      (start = temp )
   else
      (Attach node at the end of list)
      - take one external pointer, say ptr
      - use findlast() function to travel ptr at last node
           ptr = findlast(start)
      - store new node address in last node link part
           ptr->link = temp
3.  Return Start
```

Operations Linked List

- Creation of SLL

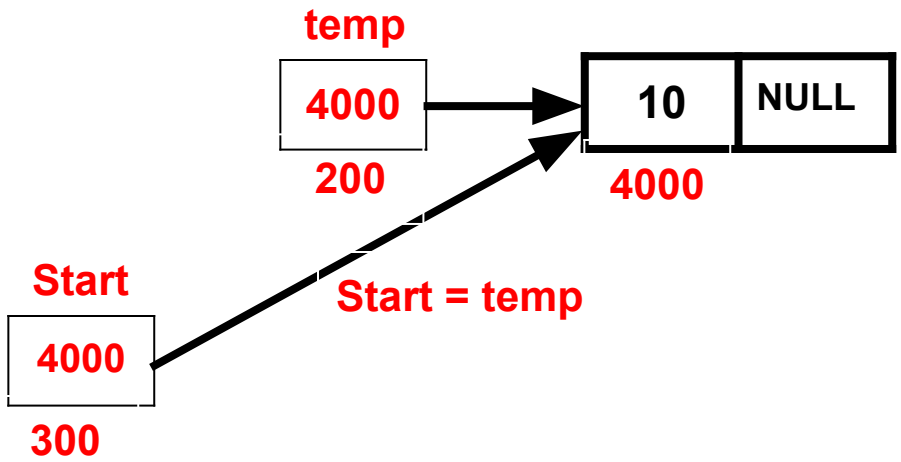
Step-I



Operations Linked List

- Creation of SLL

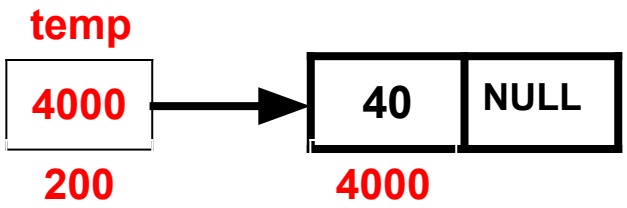
Step-I

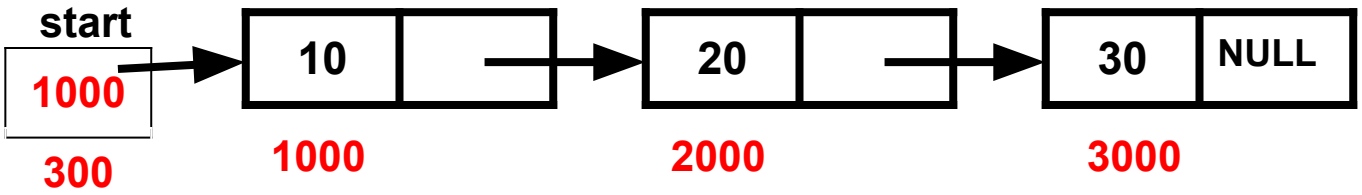


Operations Linked List

- Creation of SLL

Step-II

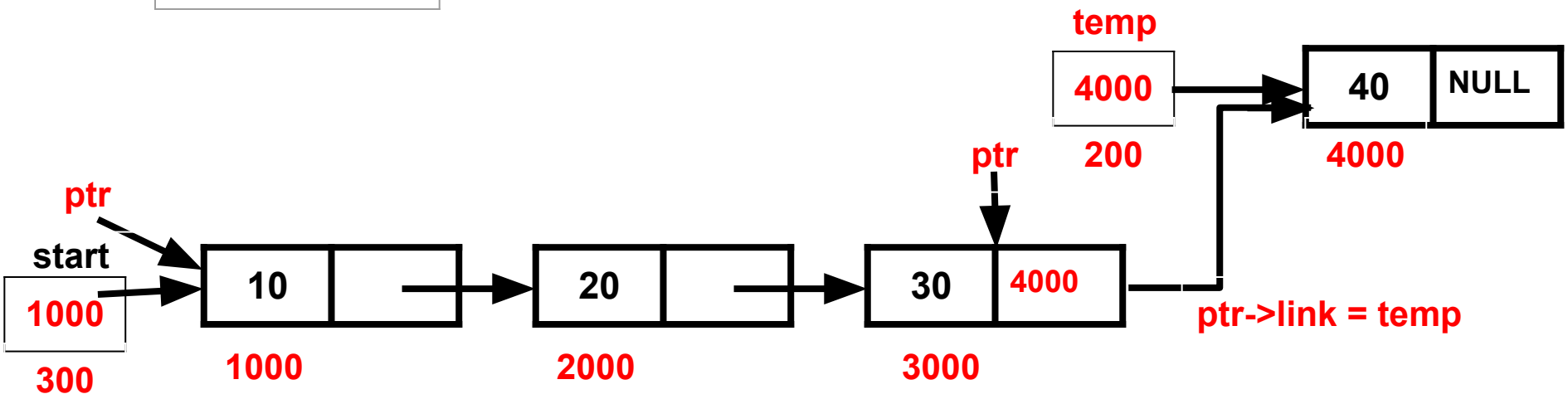




Operations Linked List

- Creation of SLL

Step-II



Operations Linked List

- Creation of SLL

```
NODE *create(NODE *start)
{
    NODE *start=NULL,*temp,*ptr;
    int n,i;
    printf (" Enter the number of node ");
    scanf ( " %d  ",  &n);
    for ( i = 0; i < n; i + +)
    {   temp = getnode( );
        if ( start = = NULL)
            start=temp;
        else
        {   ptr=findlast(start);
            ptr->link=temp;
        }
    }
    return (start);
}
```

Operations Linked List

- Display

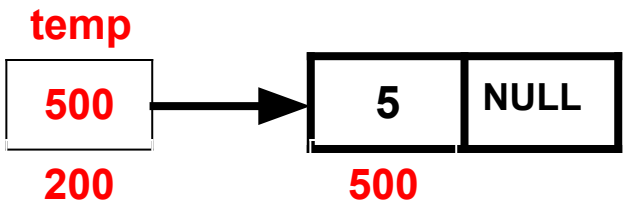
```
Void display (NODE *start)
{
    NODE *ptr;
    for ( ptr = start; ptr ! = NULL; ptr = ptr -> link )
        printf(" %d -> %p ", ptr -> data, ptr ->  link);
}
```

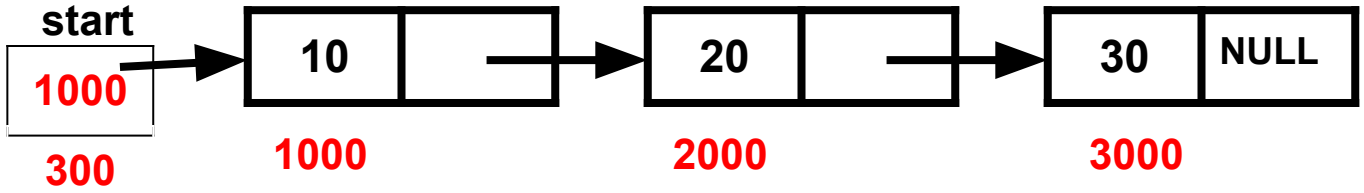
Operations Linked List

- Append Node(At Beg and at End)

  - Append Node at Beg

1. Create New Node

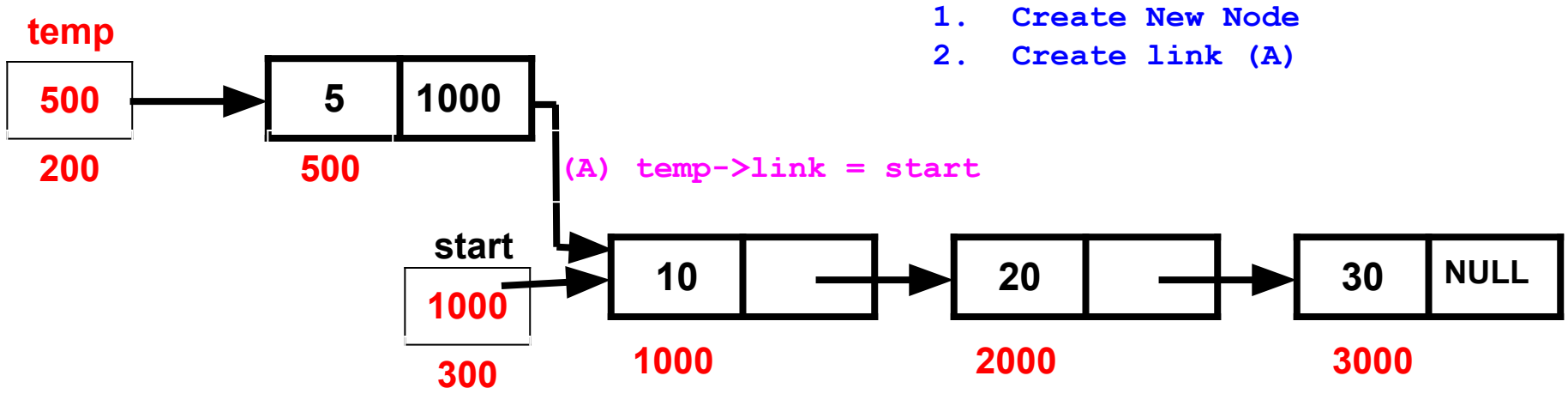




Operations Linked List

- Append Node(At Beg and at End)

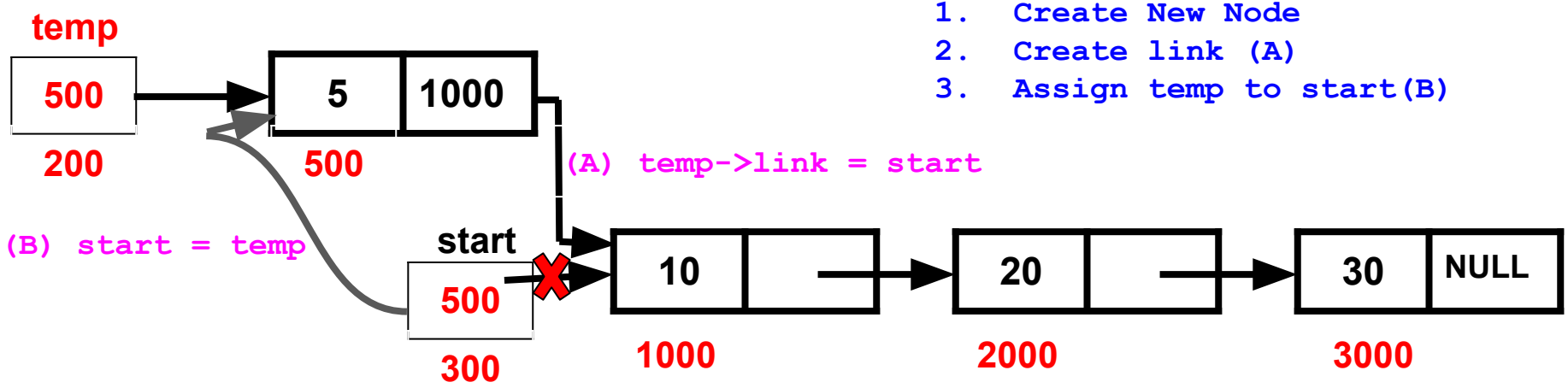
  - Append Node at Beg



Operations Linked List

- Append Node(At Beg and at End)

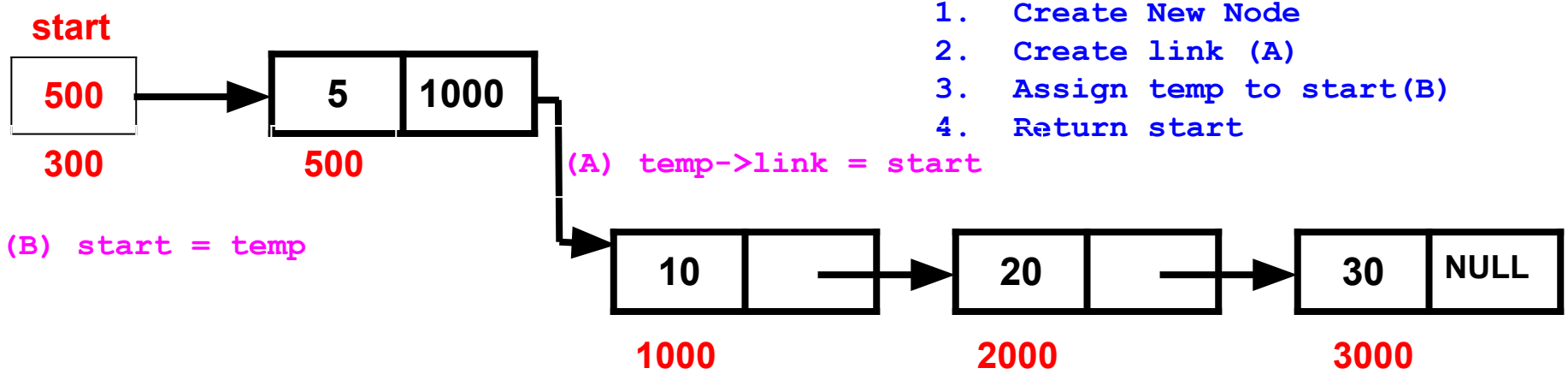
  - Append Node at Beg



Operations Linked List

- Append Node(At Beg and at End)

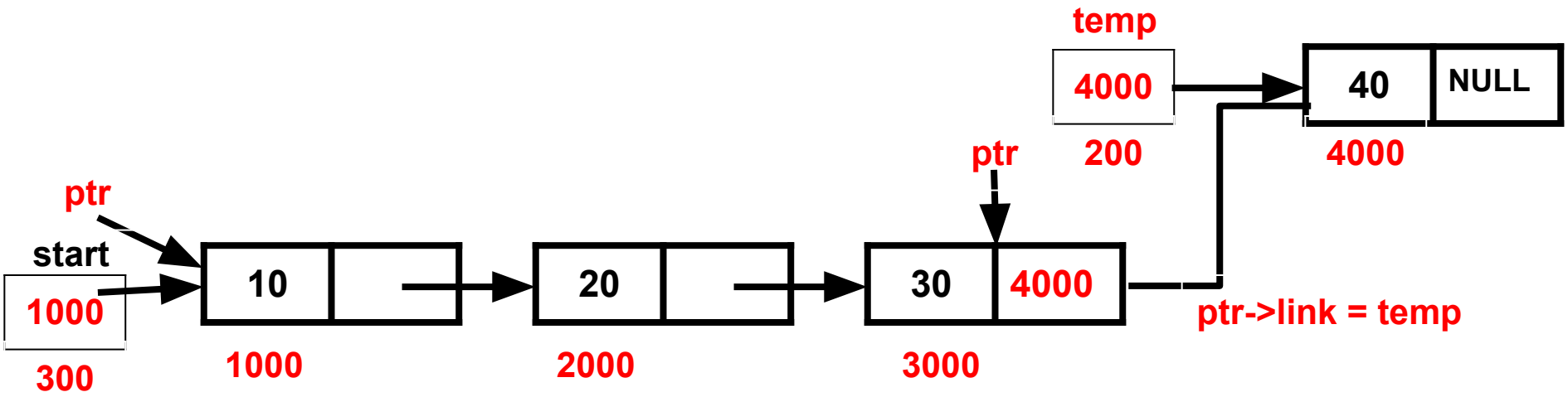
  - Append Node at Beg



Operations Linked List

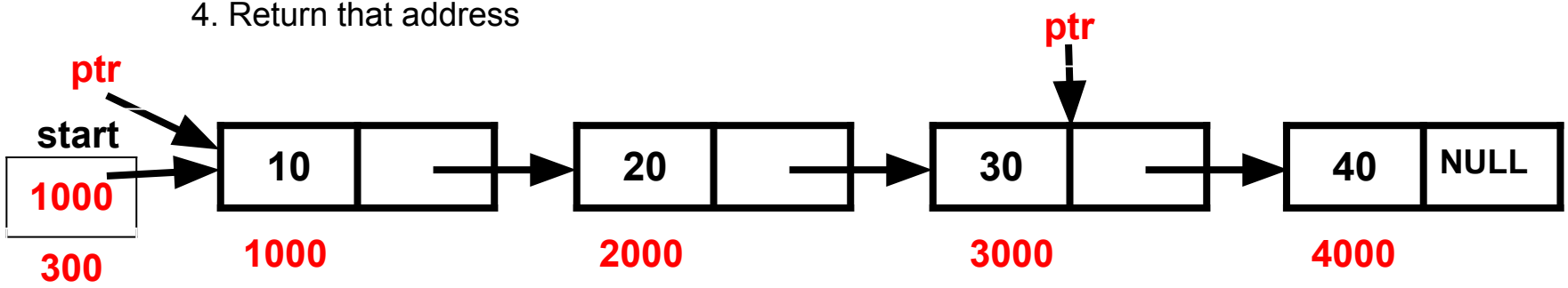
- Append Node(At Beg and at End)

  - Append Node at End



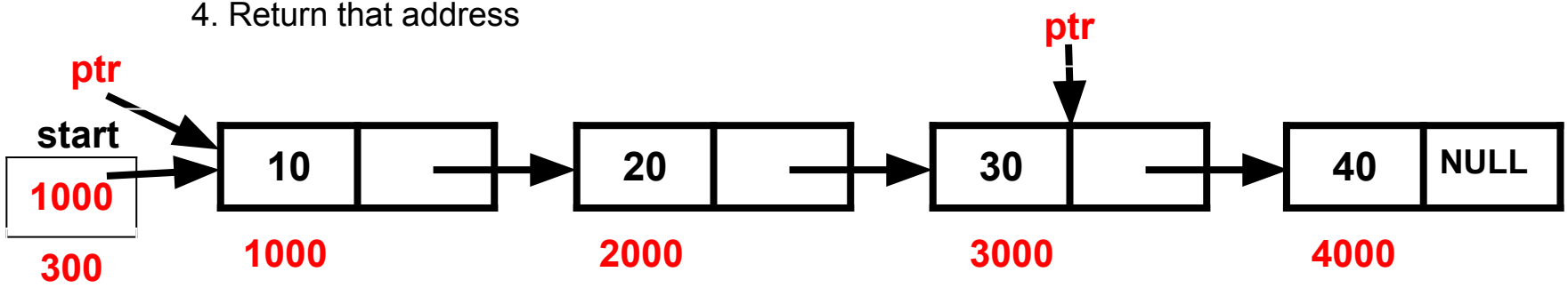
Operations Linked List

- **Search**
  1. Accept value to search
  2. Take one pointer ptr and assign to start
  3. Traverse ptr till that value or end of list
  4. Return that address



Operations Linked List

- **Search**
  1. Accept value to search
  2. Take one pointer ptr and assign to start
  3. Traverse ptr till that value or end of list
  4. Return that address




```
NODE *search (NODE *start, int val )
{
    NODE *ptr;
    for (ptr = start; ptr ! = Null && ptr-> data ! = val; ptr = ptr->l ink);
    return (ptr);
}
```

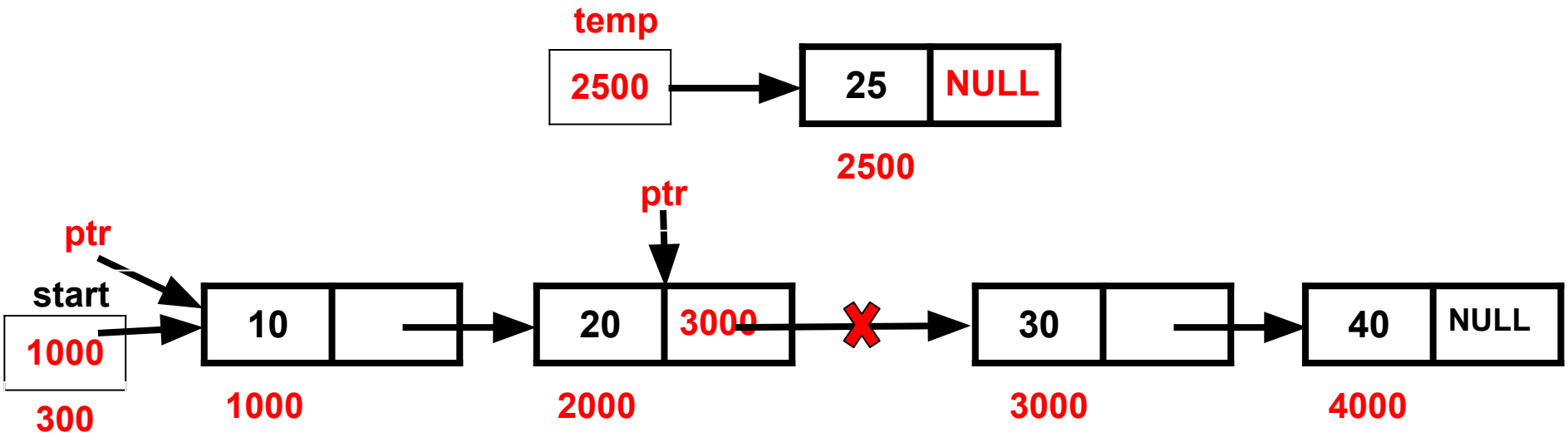
Operations Linked List

- Insert

1. Enter node after which you want to Insert
2. Create a New node to Insert
3. Take one pointer ptr and assign to start Node
4. Traverse ptr till that value or end of list
5. temp->link = ptr->link
   ptr -> link = temp
6. Return start

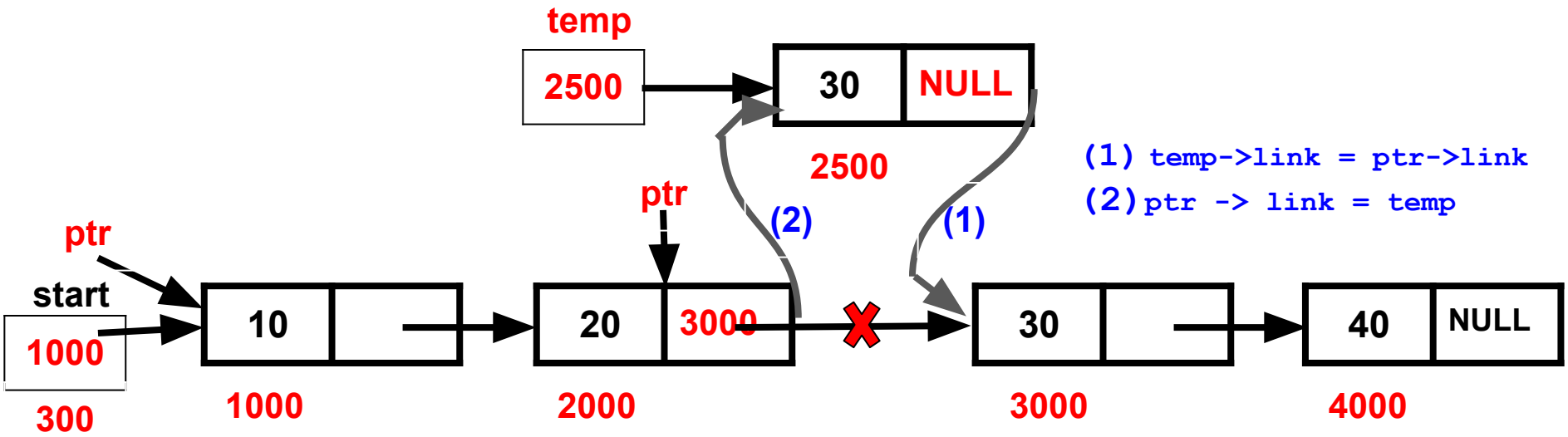
Operations Linked List

- Insert



Operations Linked List

- Insert



Operations Linked List

- Insert

```
Void insert (NODE *start)
{
    NODE *temp,*ptr;
    int val;
    printf ("Enter the node to after which you want to add ");
    scanf (" %d ",  &val);
    ptr = search (start, val);
    if ( ptr-> data = = val)
    {
        temp = getnode();
        temp->link = ptr->link;
        ptr->link = temp;
    }
}
```

Operations Linked List

- Delete

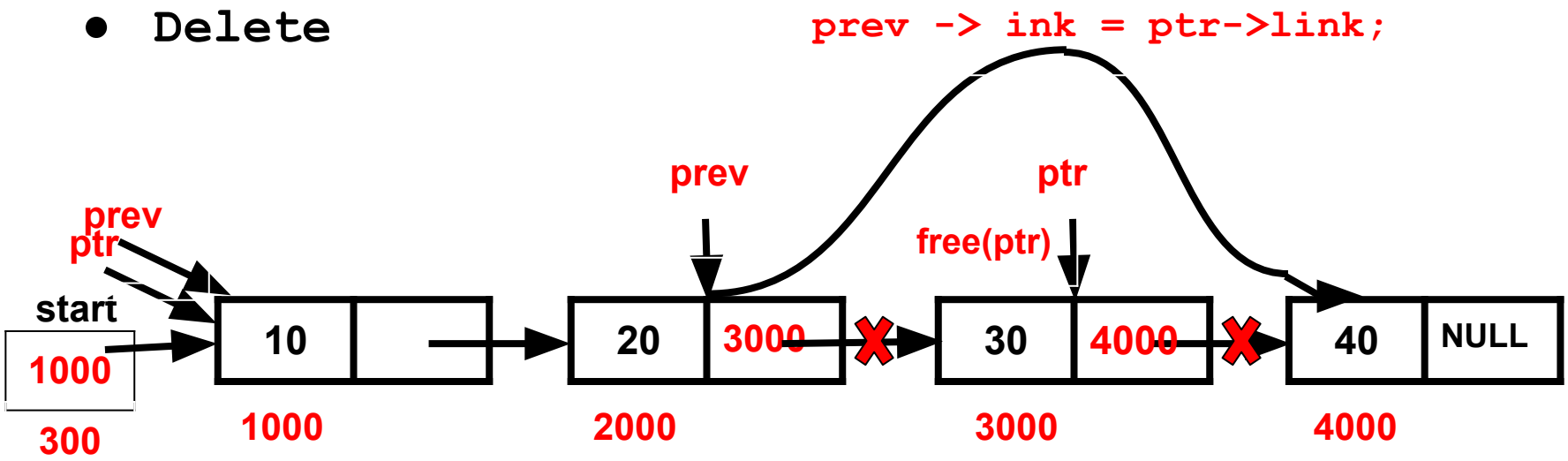
1. Accept node value which you want to Delete
2. Take two pointer ptr and prev and assign to start Node
4. Traverse ptr till that value or end of list and prev pointer
   till previous node of ptr pointer
   for (ptr=start; ptr!=NULL && ptr->data!=val; prev = ptr, ptr=ptr->ink);
5. if (ptr-> data = = val)

```
{    if ( ptr = = start)
          start = ptr->link;
     else
          prev -> ink = ptr->link;
     free ( ptr);
}
```

6. Return start

Operations Linked List

- Delete



Operations Linked List

- Delete

```
Void delet (NODE *start, int val)
{
    NODE *prev, *ptr;
    prev=start;
    for ( ptr=start; ptr!=NULL && ptr->data!=val; prev = ptr, ptr=ptr->ink);
    if (ptr-> data = = val)
    {
        if ( ptr = = start)
            start = ptr->link;
        else
            prev -> ink = ptr->link;
        free ( ptr);
    }
    return (start);
}
```