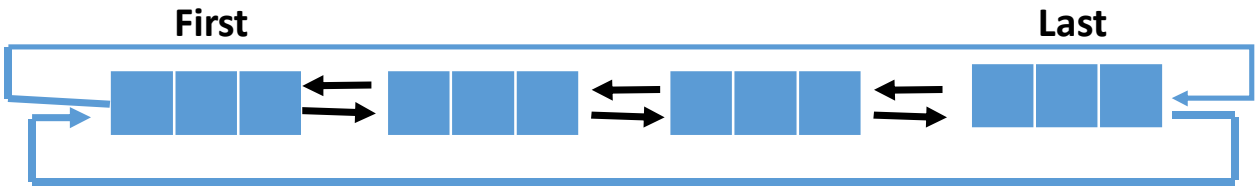
Circular Doubly Linked List

Nirma University

CE403

Even 2017-2018

Links And Data Structure



Right(last) <- First
Left(first)<-last

```
struct CDNode
{
    int info;
     struct CDNode *right,*left;
};
```

Operations

Insert
- Beginning
- Ending
- After/before element X

Delete
- Beginning
- Ending
- Element X
- After/before element X

Other operations
- Traversing
- Copy
- Merge

Insert operation : Beginning

Insert_DCList_Begin(First , X)

1. Creates a new node with the following links.

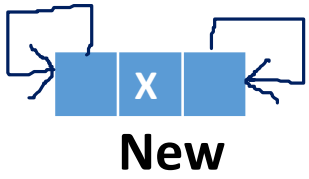
```
new = getnode(X)
```

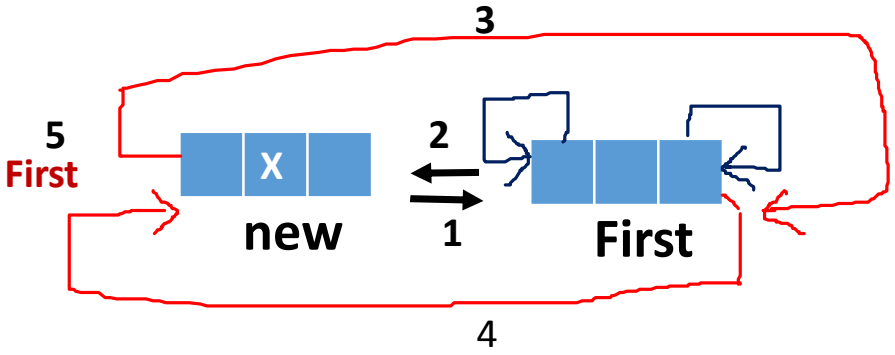
2. Checks for empty list

```
if first = null
   then first<-new
 return (first)
```

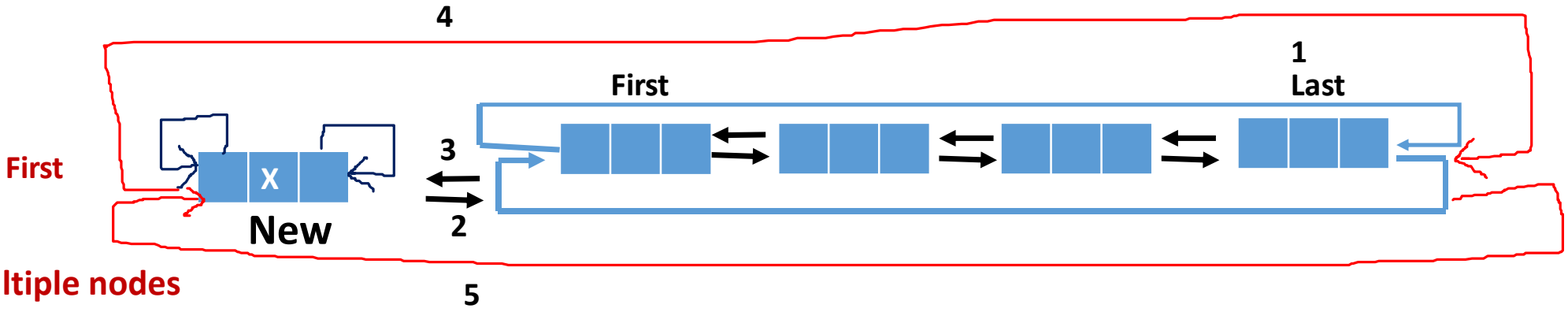
3. Single node

```
if right(first) = first
        right(new)<-first
        left(first)<-new
        left(new)<-first
        right(first)<- new
        first<-new
return(first)
```





Continued



Multiple nodes

last<- left(first)

right(new)<-first

left(first)<-new

Left(new)<-last

Right(last)<-new

First<-new

return(first)

Insert operation : End

Insert_DCList_End(First , X)

1. Creates a new node with the following links.

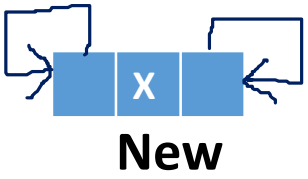
```
new = getnode(X)
```

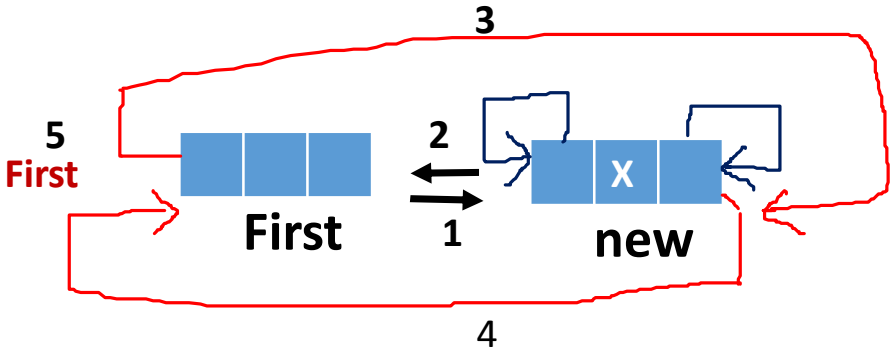
2. Checks for empty list

```
if first = null
   then first<-new
 return (first)
```

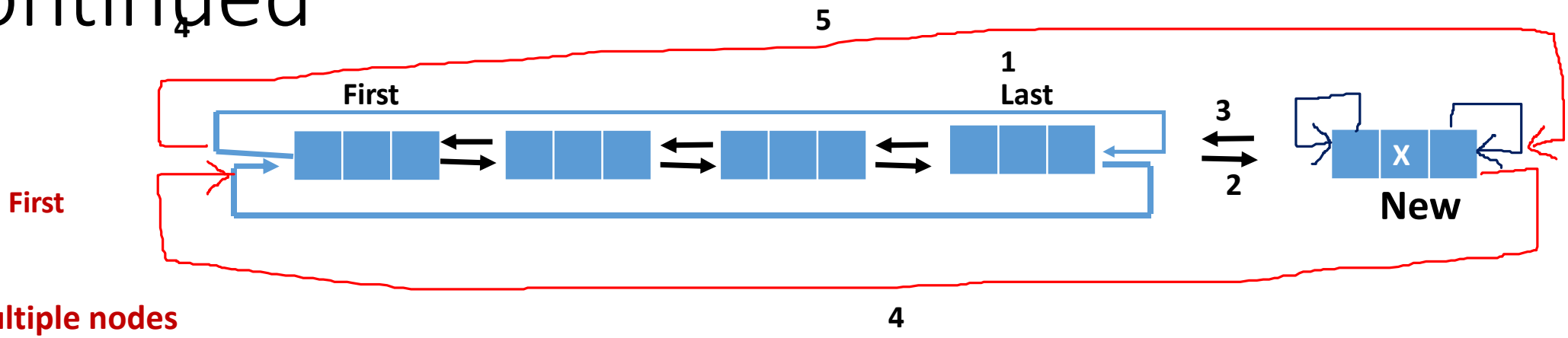
3. Single node

```
if right(first) = first
        right(first)<-new
        left(new)<-first
        Left(first)<-new
        right(new)<- first
        first<-new
return(first)
```





Continued



Multiple nodes

last<- left(first)

Right(last)<-new

Left(new)<-last

right(new)<-first

Left(first)<- new

return(first)

Insert operation : Before Y

Insert_DCList_Before_Y(First , X, Y)

1. Creates a new node with the following links.

```
new = getnode(X)
```

2. Checks for empty list

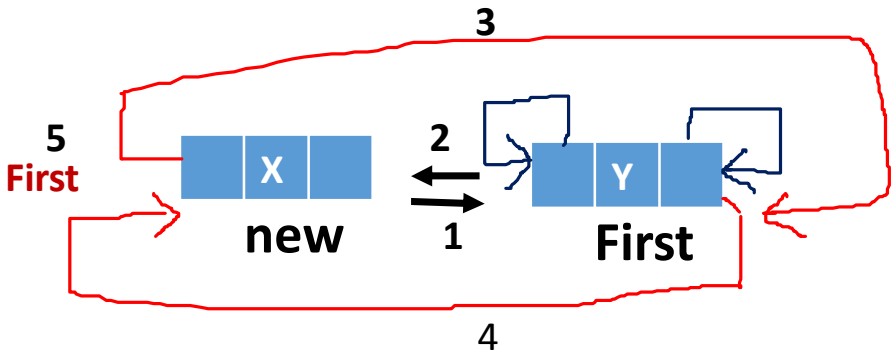
```
if first = null
   then write ( operation not possible)
 return (first)
```

3. Single node

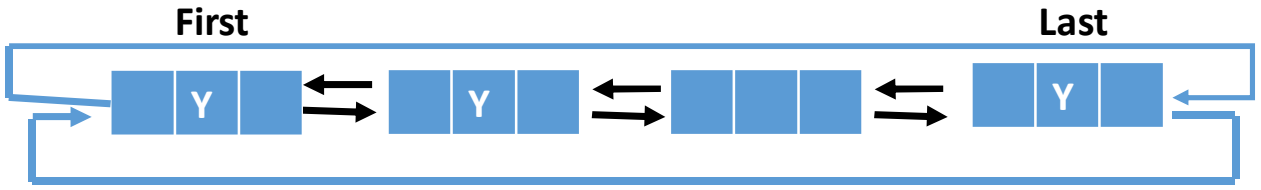
```
if right(first) = first
        if info(first) = Y
                right(new)<-first
                left(first)<-new
                left(new)<-first
                right(first)<- new
                first<-new

         else
                Write ( Data not found)
return(first)
```

Call insert_DCList_Begin(first, X)



Insert operation : Before Y continued



Not Found

Case 3

4. Case 1

```
if info(first)= Y
      Call insert_DCList_begin(First,X)
  return (first)
```

5. Traverse for other cases

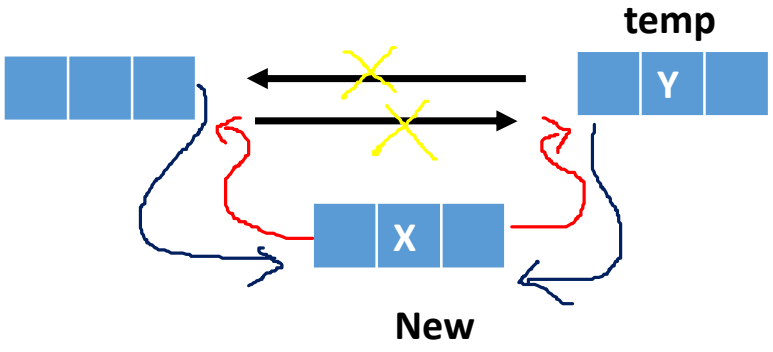
```
temp<-right(first)
repeat while temp # first  and info(temp) # Y
          temp<-right(temp
```

6. Case 3: Y not present

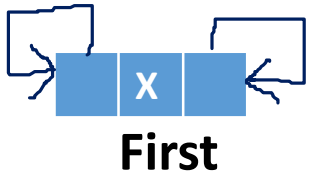
```
if temp = first
    then write (data not found)
return(first)
```

7. Case 2

```
right(new)<-temp
left(new)<-left(temp)
right(left(temp))<-new
left(temp) <- new
return(first)
```



Delete operation : End



Delete_CDList_End(First)

1.Checks for empty list

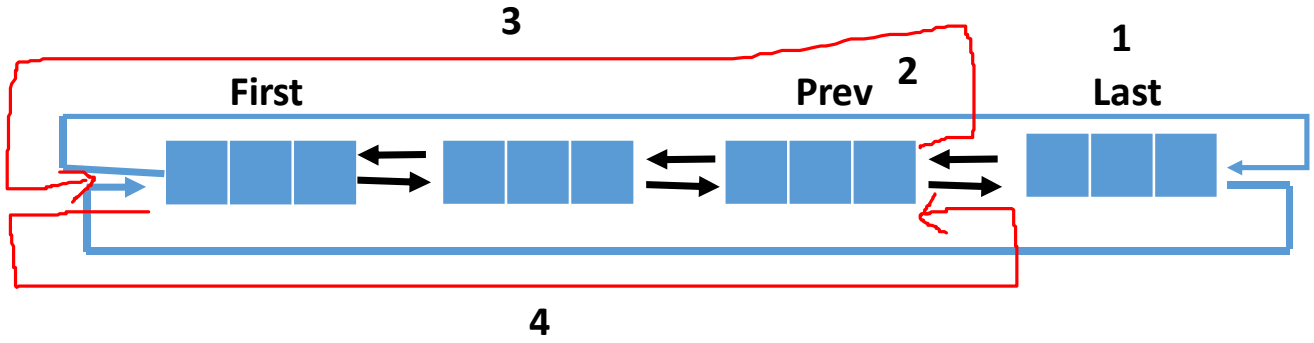
```
if first = null
   then write(empty)
 return (first)
```

2. Single node

```
if right(first) = first
            Temp<-first
            First=null
            Free(temp)
return(first)
```

3. Multiple nodes

```
last<-left(first)
Prev<-left(last)
right(prev)<-first
left(first)<-prev
free(last)
return(first)
```




```
last<-left(first)
Right(left(last)<-first
Left(first)<-left(last)
Free(last)
Return(first)
```

Delete operation : Beginning

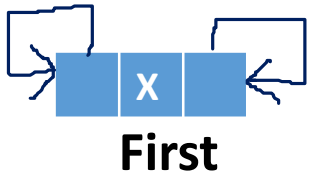
Delete_CDList_Begin(First)

Checks for empty list

```
if first = null
   Then write (empty)
 return (first)
```

3. Single node

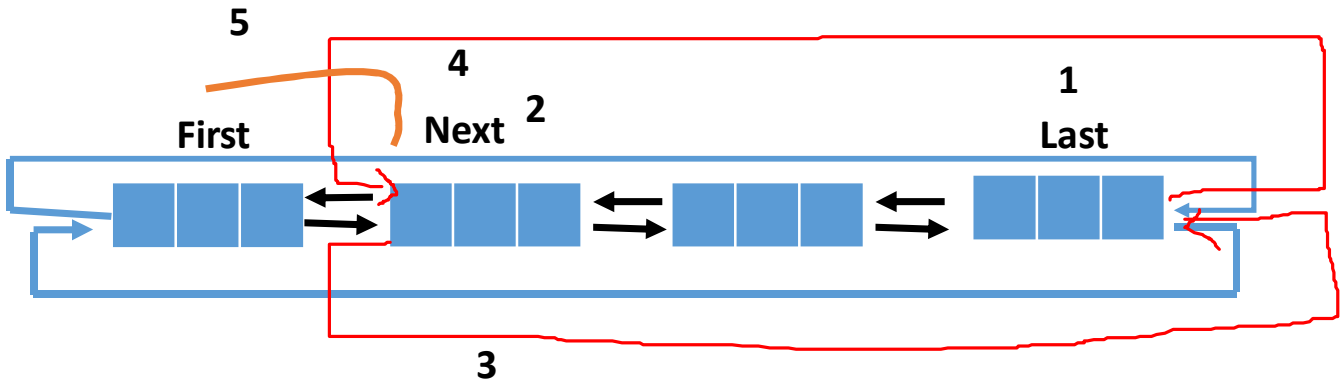
```
if right(first) = first
        Temp<-first
        First=null
        Free(temp)
return(first)
```



Temp<-first // to free/deallocate the node

4. Multiple nodes

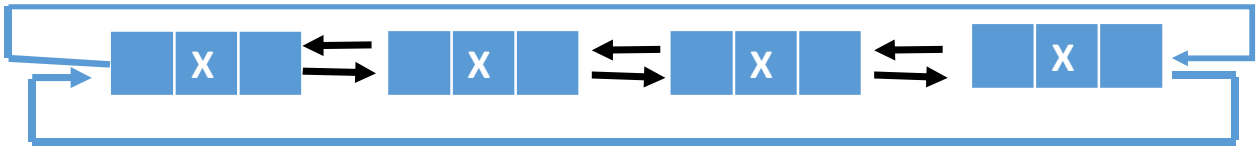
```
last<-left(first)
next<-right(first)
left(next)<-last
right(last)<-next
first<-next
free(temp)
return(first)
```



Delete Operation: After X

Minimum 2 nodes are required for the said operation.

Delete_CDList_After_X(First)



Case 1 Case 4 Case 3 Case 2 Not Found Case 5

1.Checks for empty list

```
if first = null
   then Write(empty)
 return (first)
```

2. Single node

```
if right(first) = first
      then write(not possible)
return(first)
```

3. Case 1: First Node

```
if info(first)=X
    then
     last<-left(first)
     next<-right(first)

right(first)<right(next)
Left(right(next)<-first
Free(next)
```

4. Case 2: last node

```
        last<-left(first)
     if(info(last)=X
        Delete_CDList_begin(first)
        return(first)
```

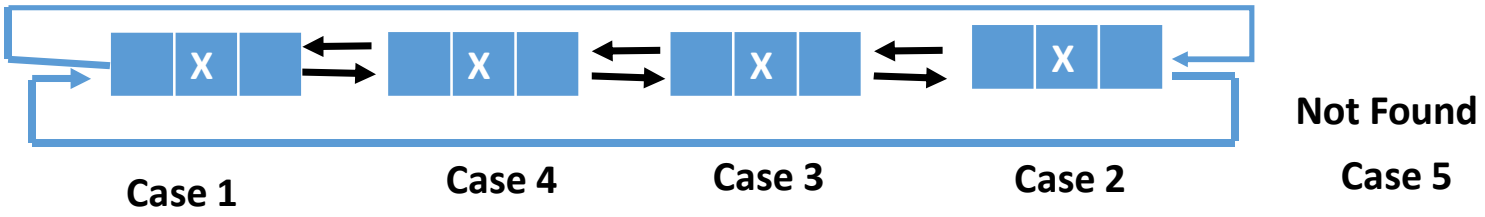
5. Case 3 onwards need to traverse the list

```
        temp<-right(first)
        repeat while temp # first  and
info(temp) # Y
                  temp<-right(temp)
```

6. Case 5: Not Found

```
        if(temp = first)
        then write(data not found)
return(first)
```

Continued



7. Case 3 : last but one node

```
if(right(right(temp)) = first
        Delete_CDList_End(first)
return(first)
```

5. Case 4: X is an intermediate node

```
 next<-right(temp)
right(temp)<-right(next)
left(right(next) <- left(next)
free(next)
return(first)
```