Linked List (Singly, Doubly, Circular)

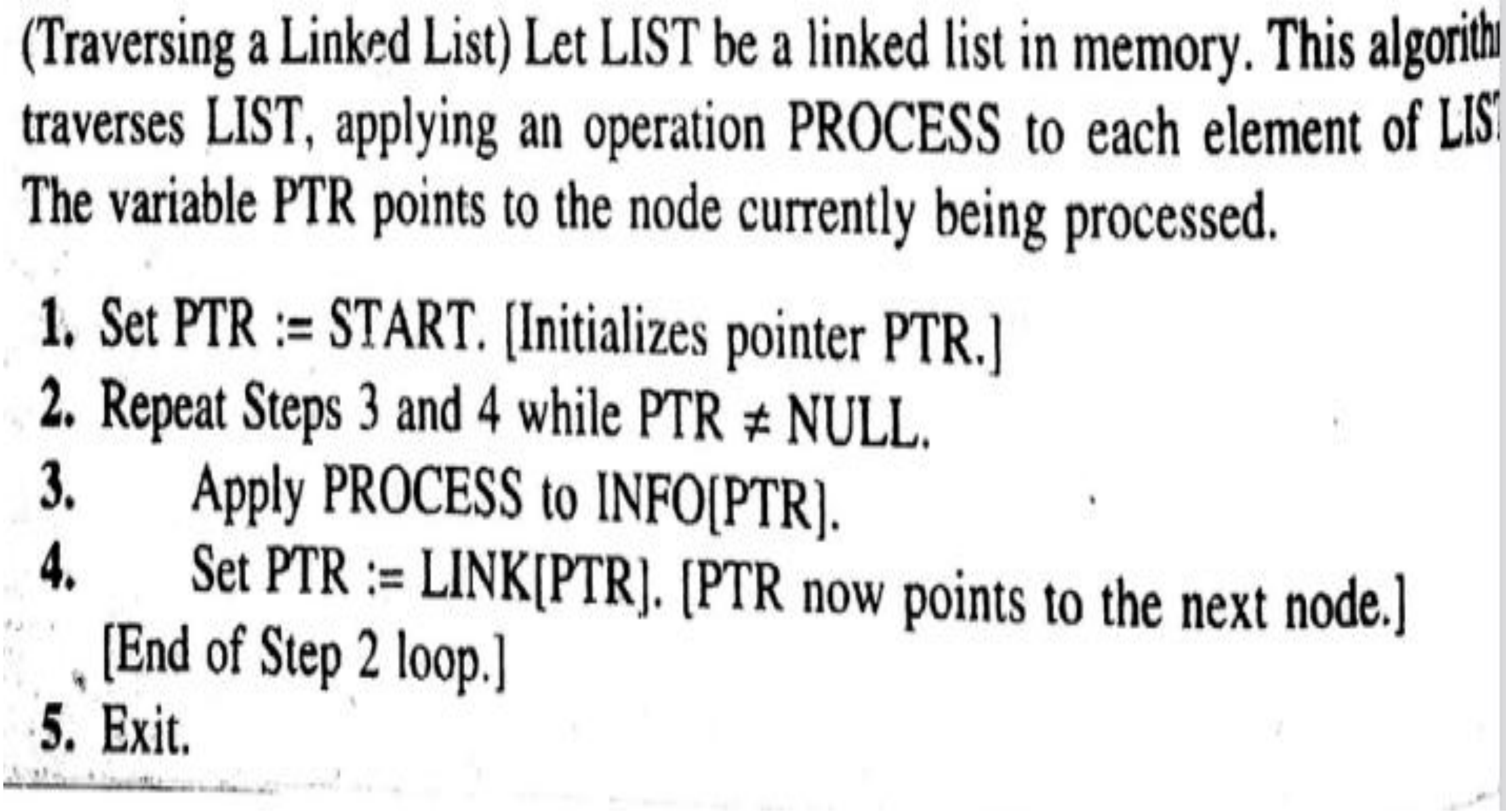
By: Dr. Mamta Kayest

Dept. of CSE, PEC Chandigarh

Singly Linked List

- Traversal
- Display data
- Count number of nodes
- Insertion
- Deletion

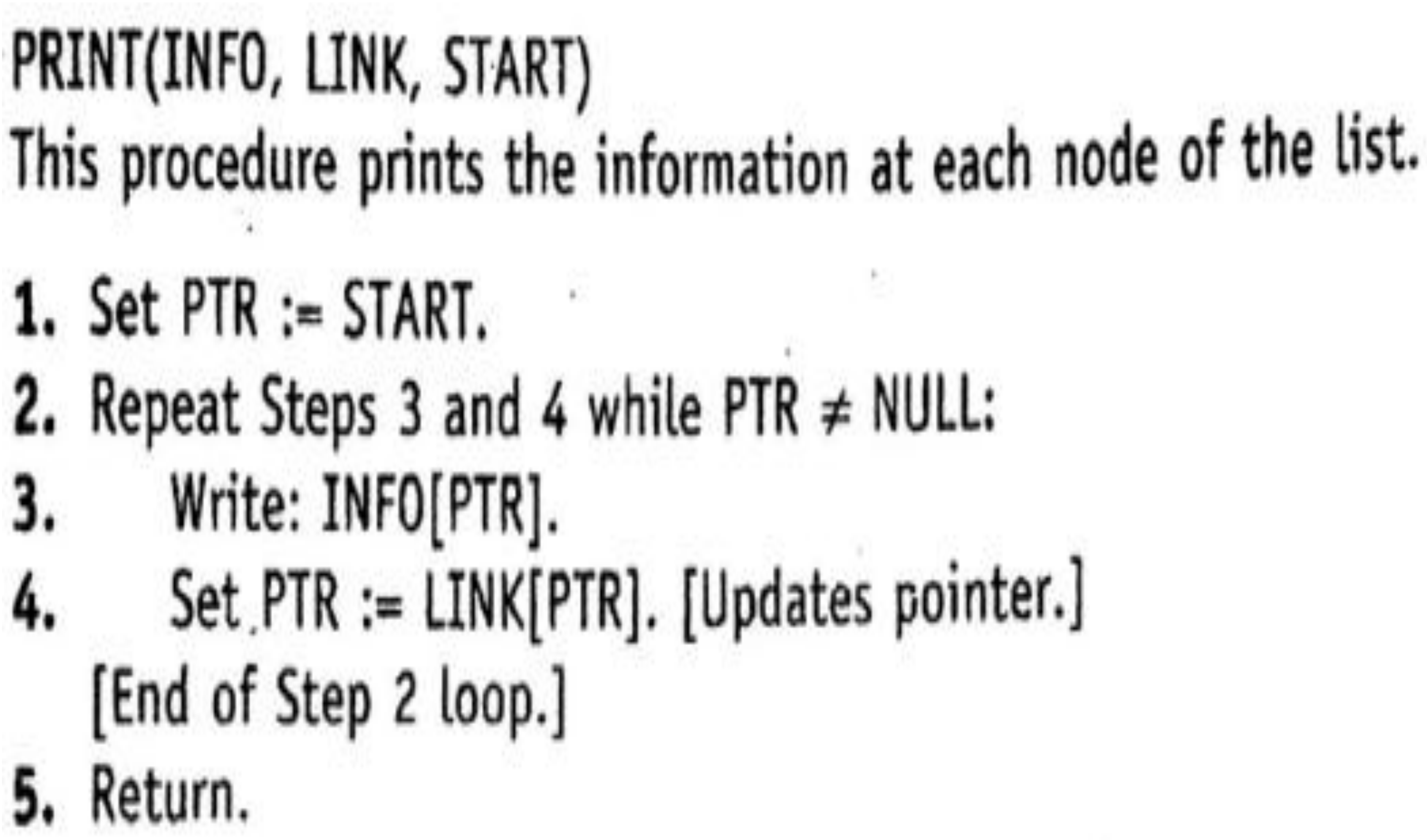
Traversing in Linked List



(Traversing a Linked List) Let LIST be a linked list in memory. This algorith traverses LIST, applying an operation PROCESS to each element of LIS The variable PTR points to the node currently being processed.

```
1. Set PTR := START. [Initializes pointer PTR.]
2. Repeat Steps 3 and 4 while PTR ≠ NULL.
3.        Apply PROCESS to INFO[PTR].
4.        Set PTR := LINK[PTR]. [PTR now points to the next node.]
   [End of Step 2 loop.]
5. Exit.
```

Display elements of Linked List

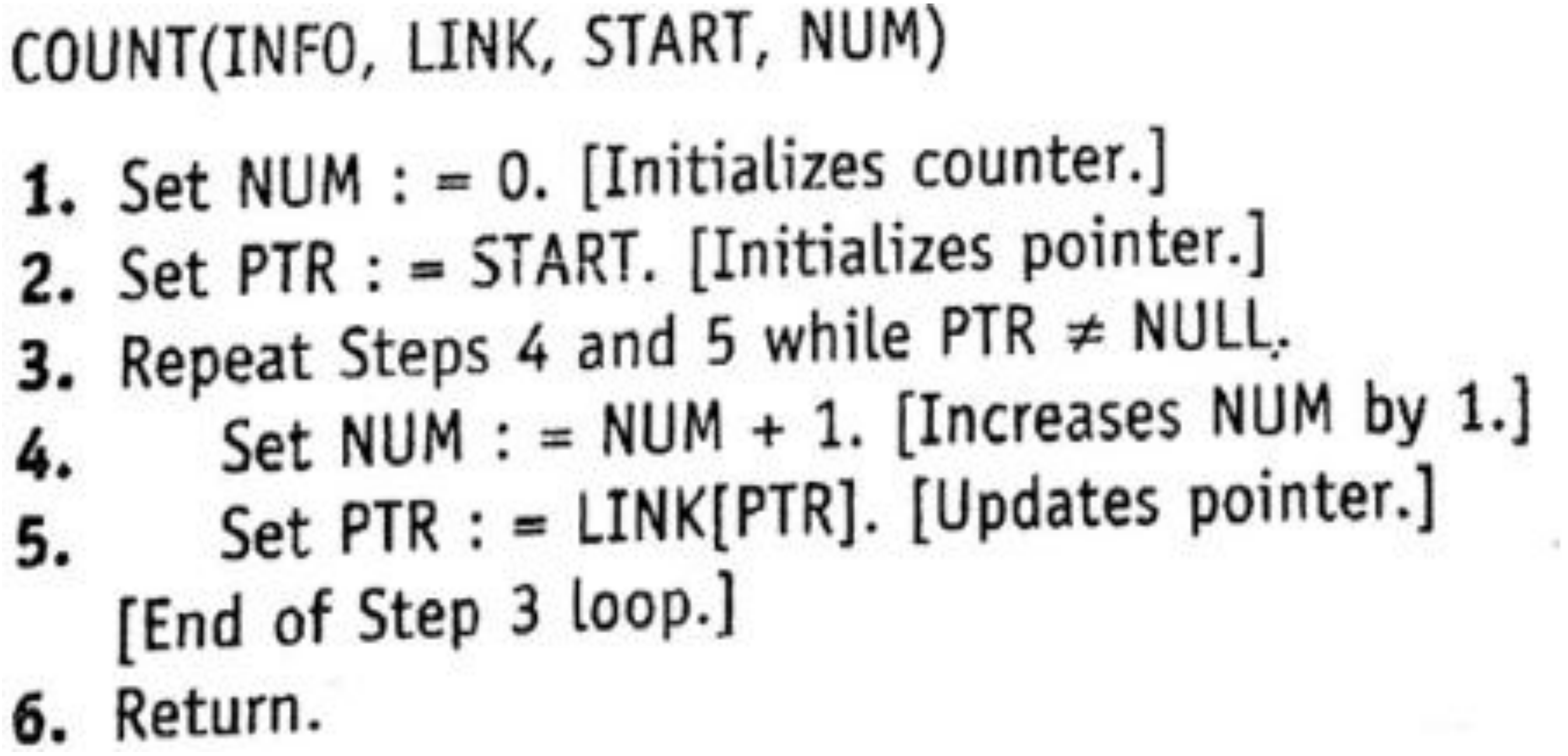


PRINT(INFO, LINK, START)

This procedure prints the information at each node of the list.

```
1. Set PTR := START.
2. Repeat Steps 3 and 4 while PTR ≠ NULL:
3.       Write: INFO[PTR].
4.       Set PTR := LINK[PTR]. [Updates pointer.]
   [End of Step 2 loop.]
5. Return.
```

To count number of elements of Linked List



```
COUNT(INFO, LINK, START, NUM)

1. Set NUM : = 0. [Initializes counter.]
2. Set PTR : = START. [Initializes pointer.]
3. Repeat Steps 4 and 5 while PTR ≠ NULL.
4.        Set NUM : = NUM + 1. [Increases NUM by 1.]
5.        Set PTR : = LINK[PTR]. [Updates pointer.]
   [End of Step 3 loop.]
6. Return.
```

Insertion in Linked List

- Insertion at first
- Insertion at end
- Insertion after a given position

INSERTATBEG (INFO, NEXT, HEAD, TEMP)

This procedure insert node at beginning of Linked List

```
1.  Create a new node.
2.  Set NEW [ DATA ] := INFO [insert data into node]
3.  Set NEW [ LINK ] := HEAD
4.  Set HEAD := NEW
5.  Exit
```

INSERTATEND (INFO, NEXT, TEMP , HEAD)

This procedure insert node at end

```
1.  Create a new node
2.  Set NEW [ DATA ] := INFO [ insert data into node ]
3.  Set NEW[ NEXT ] := 0
4.  Set TEMP  := HEAD
5.  Repeat step 6 while TEMP [ NEXT ] != 0
6.  Set TEMP = TEMP [ NEXT ]
  [ End of step 5 loop ]
7.  Set TEMP [ NEXT ] := NEW
8.  Exit
```

INSERTAFTERPOS (HEAD, TEMP, POS, I)

This procedure insert a node after a given position.

```
1.  Create a node
2.  Set I := 1
3.  Set NEW [ DATA ] := INFO [ insert data into node ]
4.  Enter POS [ position after which node is to be inserted ]
5.  If POS > COUNT,  then:  [count is the total number of nodes in linked list ]
            print "Invalid selection"
    Else:
            Set TEMP := HEAD
    [ End of If structure ]
6.  Repeat step 7 & 8 while I < POS:
7.  Set  TEMP := TEMP [ NEXT ]
8.  Increment I by one
    [ End of step 6 loop ]
9.  Set NEW [ NEXT ] := TEMP [ NEXT ]
10. Set TEMP [ NEXT ] := NEW
11. Exit
```

Deletion from Linked List

- Deletion from first
- Deletion from end
- Deletion after a given position

DELETEBEG (HEAD, NEXT, TEMP)

This procedure is for deleting first node of linked list

```
1.  If HEAD = 0 then:
        print “Linked List doesn't exit”
    Else:
        Set TEMP := HEAD
        Set HEAD := HEAD [ NEXT ]
        Free(TEMP) [de-allocate space of temp]
    [ End of If structure ]
2.  Exit
```

DELETEATEND (HEAD, TEMP, PREV)

This procedure if for deleting last node of linked list

```
1.  Set TEMP := HEAD
2.  Repeat steps 3 & 4 while ( TEMP [ NEXT ] != 0 )
3.  PREV := TEMP
4.  TEMP := TEMP [ NEXT ]
    [ End of step 2 loop ]
5.  If TEMP = HEAD then:
        free ( TEMP ) [ de-allocate space of temp ]
     Else:
        PREV [ NEXT ] := 0
        free ( TEMP )
    [ End of If structure ]
6.  Exit
```

DELAFTERPOS (NEXT, TEMP, POS, I, TODEL)

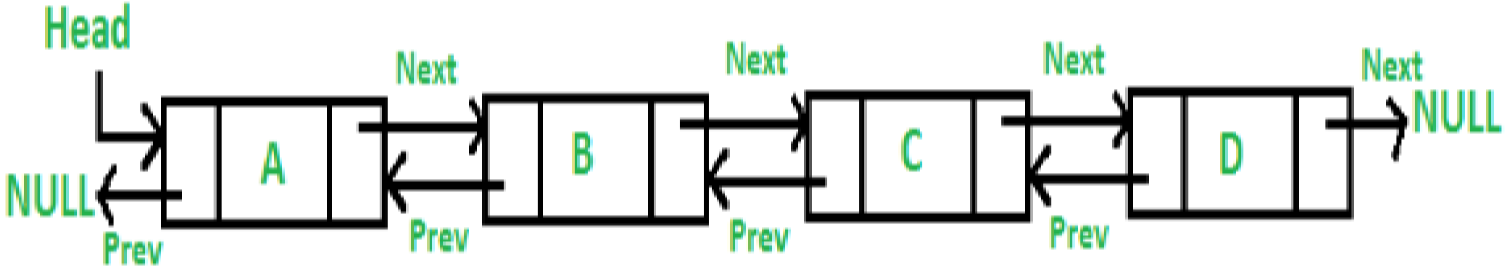
This procedure is for deleting a node after a given position

```
1.  Set I I:= 1
2.  Set TEMP := HEAD
3.  Enter value for POS [ node after Pos will be deleted ]
4.  Repeat steps 5 & 6 while  I < POS
5.  Set TEMP := TEMP [ NEXT ]
6.  Increment I by 1
    [ End of step 4 loop ]
7.  Set TODEL := TEMP [ NEXT ]
8.  Set  TEMP [ NEXT ] := TODEL [ NEXT ]
9.  Free ( TODEL ) [ de-allocate space ]
10. Exit
```

Doubly Linked List

- A doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes.

- Each node contains three fields: two link fields (references to the previous and to the next node in the sequence of nodes) and one data field.



Creation of nodes in Doubly Linked List

```
CREATEDLL( PREV, NEXT, TEMP, NEWNODE, CHOICE)
1.     START
2.     WHILE ( CHOICE ) Then:
3.     Create a node [ NEWNODE ]
4.     Write data into the node
       Set NEWNODE [ DATA ] := INFO
       [ End of step 2 loop ]
5.     Set NEWNODE [ PREV ] := 0
6.     Set NEWNODE [ NEXT ] := 0
7.     If ( HEAD = =0 ) Then
            Set HEAD = NEWNODE = TEMP
       Else
            Set TEMP [ NEXT ] := NEWNODE
            Set NEWNODE [ PREV ] := TEMP
            Set TEMP = NEWNODE
       [ End of If structure ]
8.     Display message " Do you want to continue (0/1)"
9.     Write value in CHOICE
10.    If ( CHOICE == 1) Then:
            Goto step 2
            Else
            Goto step 11
11.    Exit
```

Display data of linked list

```
DISPLAYDLL ( TEMP, HEAD, NEXT )

1.   Start
2.   Set TEMP := HEAD
3.   WHILE ( TEMP ! = 0 ) Then:
4.   Display TEMP [ DATA ]
5.   TEMP = TEMP [ NEXT ] [ Update pointer ]
     [ End of Step 2 Loop ]
5.   Exit
```

Insertion in Doubly Linked List

- Insertion at beginning
- Insertion at end
- Insertion at a given position
- Insertion after a given position

// maintaining tail pointer help in improving time complexity (in insertion and deletion of last node)

```
CREATETAIL( HEAD, TAIL, INFO, NEWNODE)

1.    Start
2.    Create a NEWNODE
3.    Write DATA into NEWNODE
4.    Set NEWNODE [ DATA ] := INFO
5.    Set NEWNODE [ PREV ] := 0
3.    Set NEWNODE [ NEXT ] := 0
4.    If ( HEAD == 0 ) Then
          Set HEAD = TAIL= NEWNODE
      Else
          Set TAIL [ NEXT ] := NEWNODE
          Set NEWNODE [ PREV ] := TAIL
          Set TAIL := NEWNODE
    [ End of If structure ]
```

// Ask user to enter choice as 0/1 (discussed earlier)

```
INSERTATBEG ( PREV, NEXT, HEAD, INFO, DATA)
//need not to check head as we are assuming there is already a linked list

1.  Start
2.  Create a NEWNODE
3.  Set NEWNODE [ DATA ] := INFO (Enter data into node)
4.  Set NEWNODE [ PREV ] := 0
5.  Set NEWNODE [ NEXT ] := 0
6.  Set HEAD [ PREV ] := NEWNODE
7.  Set NEWNODE [ NEXT ] := HEAD
8.  HEAD := NEWNODE
9.  Exit
```

INSERTATEND (HEAD, TAIL, INFO, DATA)

```
1.  Start
2.  Create a NEWNODE
3.  Write DATA into NEWNODE
    Set NEWNODE [ DATA ] := INFO
4.  Set NEWNODE [ PREV ] := 0
5.  Set NEWNODE [ NEXT ] := 0
// update tail pointer
6.  Set TAIL [ NEXT ] := NEWNODE
7.  Set NEWNODE [ PREV ] := TAIL
8.  Set TAIL := NEWNODE
9.  Exit
```

INSERTATPOS (POS, TEMP, NEXT, PREV, NEWNODE, INFO)

```
1.    Start
2.    Enter POS
3.    Set TEMP := HEAD
4.    Set I := 1
5.    If (POS == -1) Then
          Display message " Invalid "
      Else if  ( POS == 1) Then
          call procedure INSERTATBEG();
      Else
          Create a NEWNODE
          Set NEWNODE [ DATA ] := INFO
          Set NEWNODE [ PREV ] := 0
          Set NEWNODE [ NEXT ] := 0
      [ End of If structure ]
4.    Repeat steps 5 & 6 while ( I < POS – 1)
5.    Set TEMP := TEMP [ NEXT ]
6.    I := I + 1
      [ End of step 4 loop ]
4.    Set NEWNODE [ PREV ] := TEMP
5.    Set NEWNODE [ NEXT ] := TEMP [ NEXT ]
6.    Set TEMP [ NEXT ] := NEWNODE
7.    Set NEWNODE [ NEXT ] [ PREV ] := NEWNODE
8.    Exit
```

INSERTAFTERPOS (POS, NEXT, PREV, TEMP, I)

```
1.  Start
2.  Enter POS
3.  Set I := 1
4.  Repeat steps 5 & 6 while ( I < POS )
5.  Set TEMP := TEMP [ NEXT ]
6.  Set I := I + 1
    [ End of step 4 loop ]
7.  Set NEWNODE [ PREV ] := TEMP
8.  Set NEWNODE [ NEXT ] := TEMP [ NEXT ]
9.  Set TEMP [ NEXT ] := NEWNODE
10. Set NEWNODE [ NEXT ] [ PREV ] := NEWNODE
11. Exit
```

Deletion from Doubly Linked List

- Deletion from beginning
- Deletion from end
- Deletion from a given position

```
DELFROMBEG ( HEAD, TEMP, NEXT, PREV )

1. Start
2. If  ( HEAD == 0 ) Then
           Display " List is empty"
   Else
           Set TEMP := HEAD
           Set HEAD := HEAD [ NEXT ]
           Set HEAD [ PREV ] := 0
           Free ( TEMP )
   [ End of If structure ]
3. Exit
```

DELFROMEND (TAIL, PREV, NEXT, HEAD)

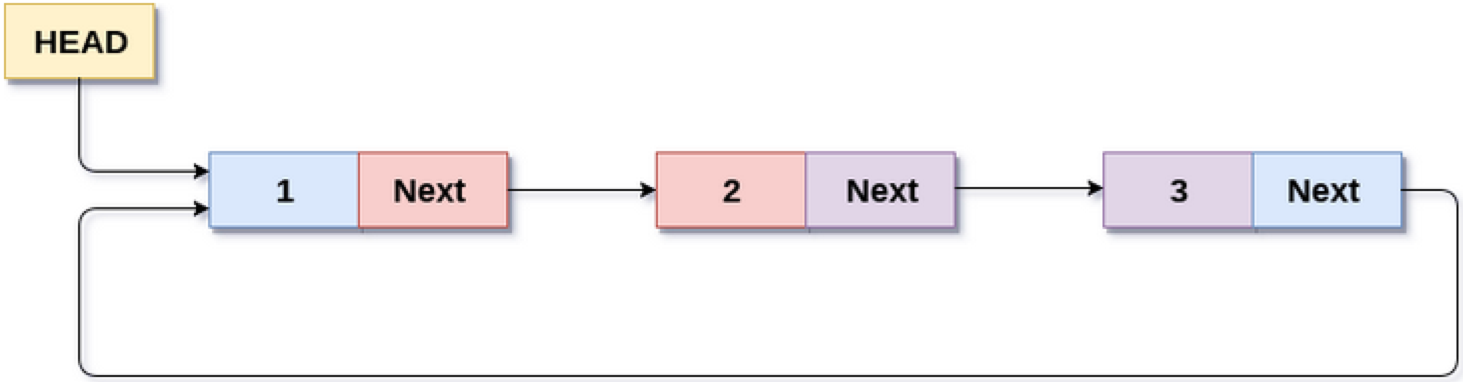
```
1. Start
2. If  ( HEAD == 0 ) Then
           Display " List is empty"
   Else
           Set TEMP := TAIL
           Set TAIL [ PREV ] [ NEXT ] := 0
           Set TAIL := TAIL [ PREV ]
           Free ( TEMP )
   [ End of If structure ]
3. Exit
```

DELFROMPOS (POS, TEMP, NEXT, PREV)

// COUNT variable has total number of nodes in the doubly linked list

```
1.  Start
2.  Enter POS
3.  Set TEMP :=HEAD
4.  Set I := 1
5.  If ( POS ==1 ) Then
              call procedure DELFROMBEG();
    Else If ( POS == COUNT ) Then
               call procedure DELFROMEND();
    [ End of If structure ]
5.   Repeat steps 6 & 7 while ( I < POS )
6.   Set TEMP := TEMP [ NEXT ]
7.   Set I := I + 1
     [ End of step 5 loop ]
8.   Set TEMP [ PREV ] [ NEXT ] := TEMP [ NEXT ]
     Set TEMP [ NEXT ] [ PREV ] := TEMP [ PREV ]
     Free ( TEMP )
9.  Exit
```

Circular Singly linked list



CREATECLL (TAIL, NEXT , INFO) // using TAIL

```
1.    START
2.    Repeat steps 3 & 4 WHILE ( CHOICE ) Then:
3.    Create a node [ NEWNODE ]
4.    Write data into the node
      Set NEWNODE [ DATA ] := INFO
      [ End of step 2 loop ]
5.    Set NEWNODE [ NEXT ] := 0
6.    If ( TAIL == 0 ) Then
              Set TAIL := NEWNODE
              Set TAIL [ NEXT ] := NEWNODE
      Else
              Set NEWNODE [ NEXT ] := TAIL [ NEXT ]
              Set TAIL [ NEXT ] := NEWNODE
              Set TAIL := NEWNODE
      [ End of If structure ]
8.    Write value in CHOICE
9.    If ( CHOICE == 1) Then:
              Goto step 2
              Else
              Goto step 10
10.   Exit
```

DISPLAYCLL (TAIL, NEXT, TEMP)// using TAIL and temp

```
1.  Start
2.  If ( TAIL == 0) Then
            Display message “List is Empty”
    Else
            Set TEMP := TAIL [ NEXT ]
3.  Repeat 3 &4 while ( TEMP [NEXT] != TAIL [ NEXT] )   //TRAVERSING UPTO LAST NODE
4.  Display data of TEMP [DATA]
5.  Set TEMP := TEMP [ NEXT ]
    [ End of while loop ]
6.  Display data of TEMP [DATA]
    [ End of If structure ]
7.  Exit
```

Insertion in Circular Singly Linked List

- Insertion at beginning
- Insertion at end
- Insertion at a given position

INSERTATBEG (TAIL, NEXT, INFO)

```
1.  Create a node [ NEWNOD
2.  Write data into the node
    Set NEWNODE [ DATA ] := INFO
3.  Set NEWNODE [ NEXT ] := 0
4.  If ( TAIL == 0 ) Then
        Set TAIL := NEWNODE
        Set TAIL [ NEXT ] := NEWNODE
    Else
        NEWNODE [ NEXT ] := TAIL [ NEXT ]
        TAIL [ NEXT ] := NEWNODE
    [ End of If structure ] // to verify you can print tail [ next ] [data] because it
    will print the first node's data
5.  Exit
```

INSERTATEND(TAIL, NEXT, INFO)

```
1.  Create a node [ NEWNODE ]
2.  Write data into the node
    Set NEWNODE [ DATA ] := INFO
3.  Set NEWNODE [ NEXT ] := 0
4.  If ( TAIL == 0 ) Then
        Set TAIL := NEWNODE
        Set TAIL [ NEXT ] := NEWNODE
    Else
        NEWNODE [ NEXT ] := TAIL [ NEXT ]
        TAIL [ NEXT ] := NEWNODE
        TAIL :=  NEWNODE
    [ End of If structure ] // to verify you can print tail [ next ] [data] because it will print the first node's data
5.
```

```
INSERTATPOS ( POS, NEXT, TAIL, TEMP )  // TEMP is pointer to first node
1.    Enter POS
2.    Set I := 1
3.    If (POS < 0 || POS > COUNT) Then //COUNT is total number of nodes
              Display message " Invalid "
      Else if  ( POS == 1) Then
              call procedure INSERTATBEG();
      Else
              Create a NEWNODE
              Write DATA into NEWNODE
              Set NEWNODE [ DATA ] := INFO
              Set NEWNODE [ NEXT ] := 0
              Set TEMP := TAIL [ NEXT ]
              Repeat step 4  & 5 while ( I < POS-1)
4.    Set TEMP := TEMP [ NEXT ]
5.    Set I := I +1
      [ End of step 3 while loop ]
6.    Set NEWNODE [ NEXT ] := TEMP [ NEXT ]
7.    Set TEMP [ NEXT ] := NEWNODE
      [ End of If structure ]
8.    Exit
```

DELFROMBEG (TAIL, TEMP, NEXT)

```
1.  Start
2.  Set TEMP : = TAIL [ NEXT ]
3.  If  ( TAIL == 0 ) Then
            Display " List is empty, "
    Else if ( TEMP [ NEXT ] == TEMP ) // only one node is there, PONITING TO ITSELF
            Set TAIL := 0
            free ( TEMP )
    Else
            Set TAIL [ NEXT ] := TEMP [ NEXT ]
            Free ( TEMP )
    [ End of If structure ]
3.  Exit
```

```
DELETEEROMEND ( CURRENT, PREV, NEXT )

1.  Start
2.  Set TEMP : = TAIL [ NEXT ]
3.  If  ( TAIL == 0 ) Then
            Display " List is empty, "
    Else if ( TEMP [ NEXT ] == TEMP ) // only one node is there, PONITING TO ITSELF
            Set TAIL := 0
            free ( TEMP )
    Else
            Repeat while ( TEMP [ NEXT ] != TAIL [ NEXT ]
                    Set PREV= TEMP
                    Set TEMP = TEMP [ NEXT ]
            [ End of loop]
            Set PREV[ NEXT ] := TAIL[ NEXT ]
            Set TAIL := PREV
            free ( TEMP )
            [ End of If structure ]
3.  Exit
```

```
DELETEFROMPOS ( TEMP, NEXTNODE, POS, I, NEXT, PREV )
1.  Enter POS
2.  Set I := 1
3.  SET TEMP:= TAIL [ NEXT ]
4.  If (POS < 0| | POS > COUNT) Then //COUNT is total number of nodes
            Display message " Invalid "
    Else if  ( POS == 1) Then
            call procedure DELFROMBEG();
    Else
     Repeat step 5 & 6 while ( I < POS-1)
5.   Set TEMP := TEMP [ NEXT ]
6.   Set I := I +1
    [ End of step 3 while loop ]
7.  Set NEXTNODE:= TEMP [ NEXT ]
8.  Set TEMP [ NEXT ] := NEXTNODE [ NEXT ]
9.  Free ( TEMP )
    [ End of If structure ]
10. Exit
```

Linked List as STACKS and QUEUES

- While implementing linked list as stack , nodes are to be inserted and deleted from one end ‘TOS’
- While implementing linked list as Queue , nodes are to be inserted from ‘REAR’ and deleted from ‘FRONT’

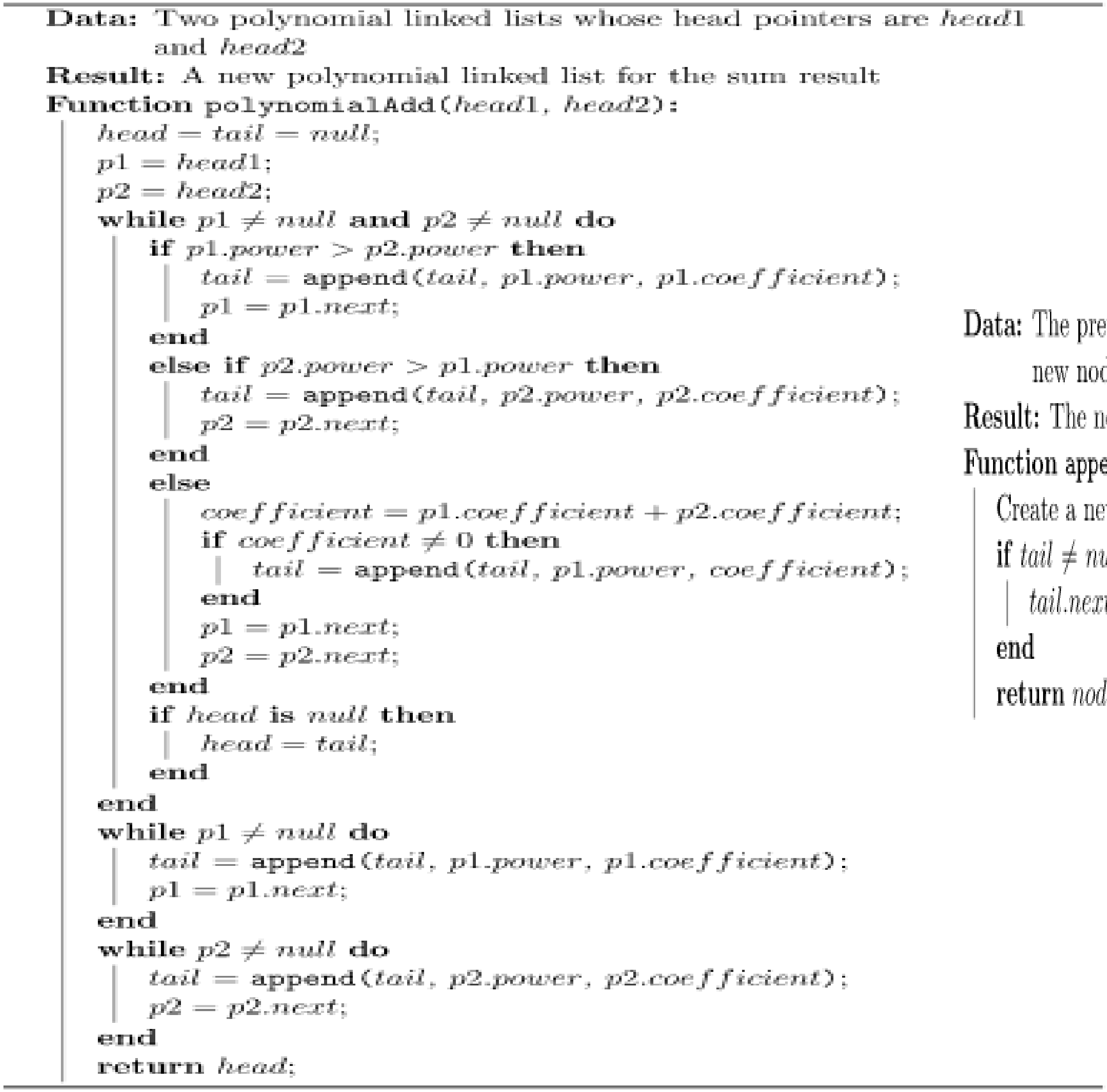
Applications of Linked List

- Polynomial addition
- Representation of Sparse matrix

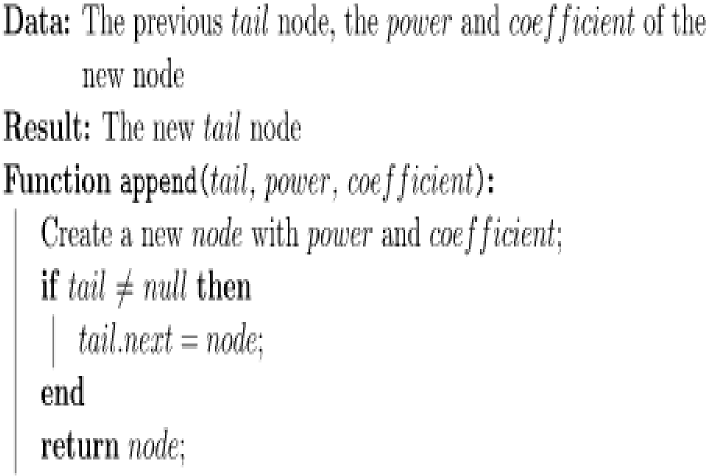
POLYADD (P,Q, NEW)

```
1.  START
2.  Repeat step 3 while ( P! = Q)
3.  If expo of two terms are equal Then
        if the terms do not cancel Then
                insert the sum of terms into sum polynomial
                increment P
                increment Q
    Else if (expo of 1st > expo of 2nd ) Then
        insert the term from 1st polynomial into sum polynomial
        increment P
    Else
        insert the term from 2nd polynomial into sum polynomial
        increment Q
4.  Copy the remaining terms to sum polynomial
```

The third step of algorithm is to be processed till the end of polynomial hasn’t reached



```
Data: Two polynomial linked lists whose head pointers are $head1$
      and $head2$
Result: A new polynomial linked list for the sum result
Function polynomialAdd($head1$, $head2$):
    $head = tail = null$;
    $p1 = head1$;
    $p2 = head2$;
    while $p1 \neq null$ and $p2 \neq null$ do
        if $p1.power > p2.power$ then
            $tail$ = append($tail$, $p1.power$, $p1.coefficient$);
            $p1 = p1.next$;
        end
        else if $p2.power > p1.power$ then
            $tail$ = append($tail$, $p2.power$, $p2.coefficient$);
            $p2 = p2.next$;
        end
        else
            $coefficient = p1.coefficient + p2.coefficient$;
            if $coefficient \neq 0$ then
                $tail$ = append($tail$, $p1.power$, $coefficient$);
            end
            $p1 = p1.next$;
            $p2 = p2.next$;
        end
        if $head$ is $null$ then
            $head = tail$;
        end
    end
    while $p1 \neq null$ do
        $tail$ = append($tail$, $p1.power$, $p1.coefficient$);
        $p1 = p1.next$;
    end
    while $p2 \neq null$ do
        $tail$ = append($tail$, $p2.power$, $p2.coefficient$);
        $p2 = p2.next$;
    end
    return $head$;
```



```
Data: The previous $tail$ node, the $power$ and $coefficient$ of the
      new node
Result: The new $tail$ node
Function append($tail$, $power$, $coefficient$):
    Create a new $node$ with $power$ and $coefficient$;
    if $tail \neq null$ then
        $tail.next = node$;
    end
    return $node$;
```

Sparse Matrix

- Sparse matrix is a matrix having majority of zero values/ elements