CPE 112 Programming with
Data Structures

Linked List

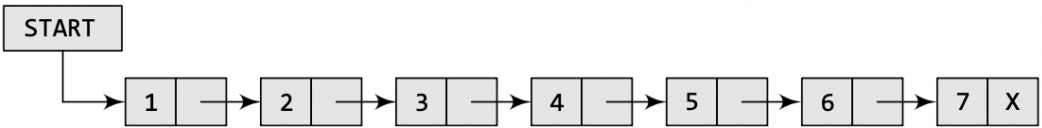
Dr. Piyanit Ua-areemitr
Dr. Taweechai Nuntawisuttiwong

Contents

Linked List

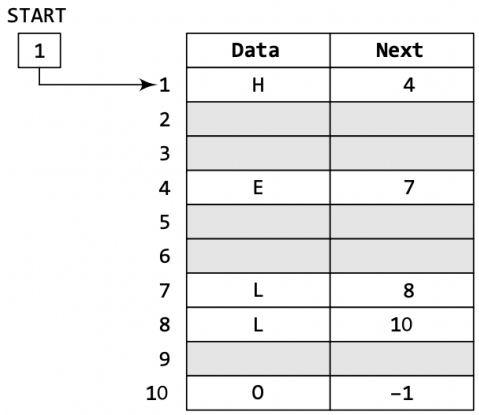
Linked List

- Linked list is a linear collection of data elements.
- These data element are called nodes.
- Linked list is a data structure which in turn can be used to implement other data structures. Thus, it acts as a building block to implement data structures such as stack, queues, and their variations.
- A linked list can be perceived as a train or a sequence of nodes in which each node contains one or more data fields and a pointer to the next node.



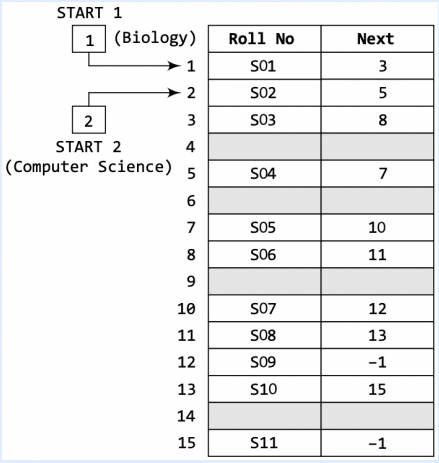
Implement a Linked List

```
struct node{
  char data;
  struct node *next;
};
```



Implement a Linked List

Two linked lists which are simultaneously maintained in the memory



	Roll No	Next
1	S01	3
2	S02	5
3	S03	8
4		
5	S04	7
6		
7	S05	10
8	S06	11
9		
10	S07	12
11	S08	13
12	S09	-1
13	S10	15
14		
15	S11	-1

Implement a Linked List

Students' linked list

START
18

	Roll No	Name	Aggregate	Grade	Next
1	S01	Ram	78	Distinction	6
2	S02	Shyam	64	First division	14
3					
4	S03	Mohit	89	Outstanding	17
5					
6	S04	Rohit	77	Distinction	2
7	S05	Varun	86	Outstanding	10
8	S06	Karan	65	First division	12
9					
10	S07	Veena	54	Second division	-1
11	S08	Meera	67	First division	4
12	S09	Krish	45	Third division	13
13	S10	Kusum	91	Outstanding	11
14	S11	Silky	72	First division	7
15					
16					
17	S12	Monica	75	Distinction	1
18	S13	Ashish	63	First division	19
19	S14	Gaurav	61	First division	8

Linked List VS Array

Linked List

- Linear collection of data elements
- Does not store nodes in consecutive memory locations
- Does not allow random access of data
- Add any number of elements in the list

Array

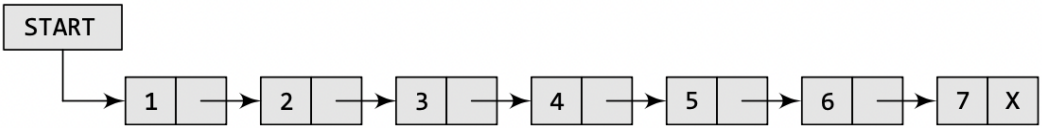
- Linear collection of data elements
- Store data elements in consecutive memory locations
- Allow random access of data
- Add specific number of elements in the list

Linked lists provide an efficient way of storing related data and performing basic operations such as insertion, deletion, and update of information at the cost of extra space required for storing the address of next nodes.

Singly Linked List

Singly Linked List

- A singly linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.
- By saying that the node contains a pointer to the next node, we mean that the node stores the address of the next node in sequence.
- A singly linked list allows traversal of data only in one way.



Teaversing a Singly Linked List

```c
typedef struct node{
  int data;
  struct node *next;
}node;
```

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:            Apply Process to PTR -> DATA
Step 4:            SET PTR = PTR -> NEXT
       [END OF LOOP]
Step 5: EXIT
```

```c
void traverse(node *start){
  node *ptr = start;
  while(ptr != NULL){
    //Apply process to ptr->data
    ptr = ptr->next;
  }
}
```

Print a Number of Nodes in a Linked List

```
Step 1: [INITIALIZE] SET COUNT = 0
Step 2: [INITIALIZE] SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR != NULL
Step 4:            SET COUNT = COUNT + 1
Step 5:            SET PTR = PTR -> NEXT
       [END OF LOOP]
Step 6: Write COUNT
Step 7: EXIT
```

```c
void printNumNode(node *start){
  int count = 0;
  node *ptr = start;
  while(ptr != NULL){
    count++;
    ptr = ptr->next;
  }
  printf("Number of nodes = %d.\n",count);
}
```

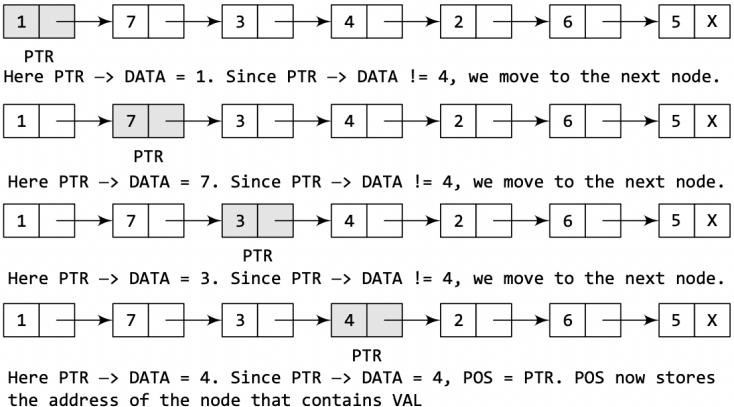
Searching for a Value in a Linked List

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:     IF VAL = PTR -> DATA
                SET POS = PTR
                Go To Step 5
            ELSE
                SET PTR = PTR -> NEXT
            [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

```c
void printNumNode(node *start){
  int count = 0;
  node *ptr = start;
  while(ptr != NULL){
    count++;
    ptr = ptr->next;
  }
  printf("Number of nodes = %d.\n",count);
}
```

Searching for a Value in a Linked List

Consider the linked list shown in figure. If we have $VAL = 4$, then the flow of the algorithm can be explained as shown in figure.



Inserting a New Node in a Linked List

1. The new node is inserted at the beginning.
2. The new node is inserted at the end.
3. The new node is inserted after a given node.
4. The new node is inserted before a given node.

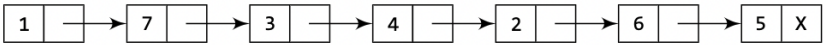
Inserting a New Node at the Begining

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

```c
void insertBegin(node **start){
  node *newNode;
  int val;

  printf("Enter data: ");
  scanf("%d",&val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  newNode->next = *start;
  *start = newNode;
}
```

Inserting a New Node at the Begining

Suppose we want to add a new node with data 9 and add it as the first node of the list.



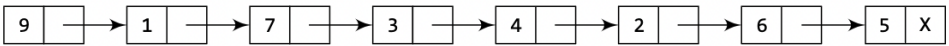
START

Allocate memory for the new node and initialize its DATA part to 9.

Add the new node as the first node of the list by making the NEXT part of the new node contain the address of START.

START

Now make START to point to the first node of the list.



START

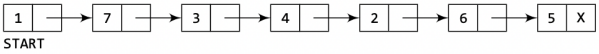
Insert a New Node at the End

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

```c
void insertEnd(node **start){
  node *newNode, *ptr;
  int val;

  printf("Enter data: ");
  scanf("%d",&val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  newNode->next = NULL;

  ptr = *start;
  while(ptr->next != NULL)
    ptr = ptr->next;
  ptr->next = newNode;
}
```

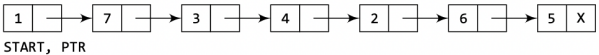
Insert a New Node at the End

Suppose we want to add a new node with value 9 after the node containing data 3.

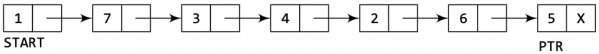


Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.

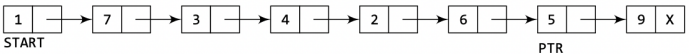
Take a pointer variable PTR which points to START.



Move PTR so that it points to the last node of the list.



Add the new node after the node pointed by PTR. This is done by storing the address of the new node in the NEXT part of PTR.

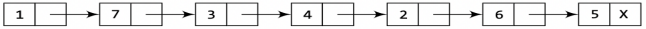


Insert a New Node After a Given Node

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR -> DATA
         != NUM
Step 8:      SET PREPTR = PTR
Step 9:      SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 10: PREPTR -> NEXT = NEW_NODE
Step 11: SET NEW_NODE -> NEXT = PTR
Step 12: EXIT
```

```c
void insertAfter(node **start, int num){
  node *newNode, *ptr, *prePtr;
  int val;

  printf("Enter data: ");
  scanf("%d",&val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  newNode->next = NULL;

  ptr = *start;
  prePtr = ptr;
  while(prePtr->data != num){
    prePtr = ptr;
    ptr = ptr->next;
  }
  prePtr->next = newNode;
  newNode->next = ptr;
}
```

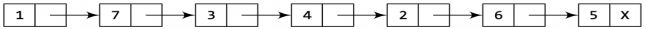
Insert a New Node After a Given Node

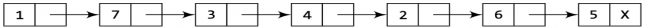


START
Allocate memory for the new node and initialize its DATA part to 9.

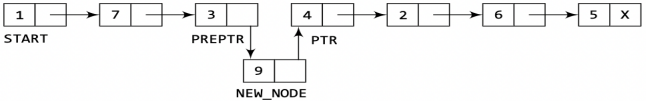
Take two pointer variables PTR and PREPTR and initialize them with START so that START, PTR, and PREPTR point to the first node of the list.

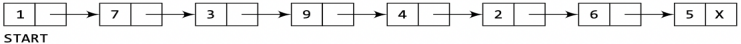
Move PTR and PREPTR until the DATA part of PREPTR = value of the node after which insertion has to be done. PREPTR will always point to the node just before PTR.





Add the new node in between the nodes pointed by PREPTR and PTR.



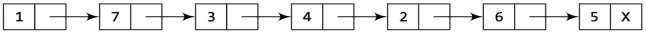


Insert a New Node Before a Given Node

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR -> DATA != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 10: PREPTR -> NEXT = NEW_NODE
Step 11: SET NEW_NODE -> NEXT = PTR
Step 12: EXIT
```

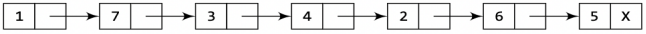
```c
void insertBefore(node **start, int num){
  node *newNode, *ptr, *prePtr;
  int val;

  printf("Enter data: ");
  scanf("%d",&val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  newNode->next = NULL;

  ptr = *start;
  prePtr = ptr;
  while(ptr->data != num){
    prePtr = ptr;
    ptr = ptr->next;
  }
  prePtr->next = newNode;
  newNode->next = ptr;
}
```

Insert a New Node Before a Given Node



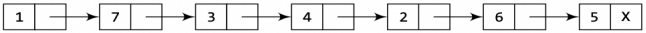
START
Allocate memory for the new node and initialize its DATA part to 9.

Initialize PREPTR and PTR to the START node.



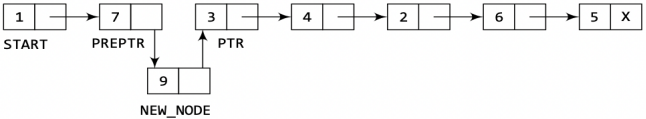
START
PTR
PREPTR

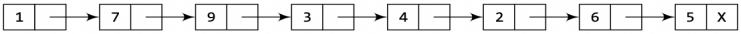
Move PTR and PREPTR until the DATA part of PTR = value of the node before which insertion has to be done. PREPTR will always point to the node just before PTR.



START PREPTR PTR

Insert the new node in between the nodes pointed by PREPTR and PTR.





START

Deleting a Node in a Linked List

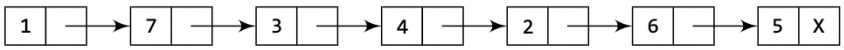
1. The first node is deleted.
2. The last node is deleted.
3. The node after a given node is deleted.

Deleting the First Node in a Linked List

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: FREE PTR
Step 5: EXIT
```

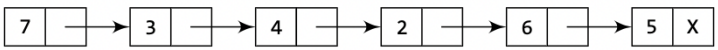
```c
void deleteFirst(node **start){
  node *ptr;

  if(*start == NULL){
    printf("Underflow\n");
    return;
  }
  ptr = *start;
  *start = (*start)->next;
  free(ptr);
}
```

Deleting the First Node in a Linked List



START

Make START to point to the next node in sequence.



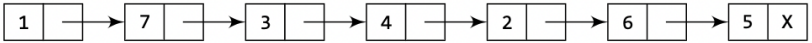
START

Deleting the Last Node in a Linked List

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != NULL
Step 4:     SET PREPTR = PTR
Step 5:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 6: SET PREPTR -> NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT
```

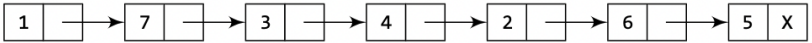
```c
void deleteLast(node **start){
  node *ptr, *prePtr;

  if(*start == NULL){
    printf("Underflow\n");
    return;
  }
  ptr = *start;
  while(ptr->next != NULL){
    prePtr = ptr;
    ptr = ptr->next;
  }
  prePtr->next = NULL;
  free(ptr);
}
```

Deleting the Last Node in a Linked List



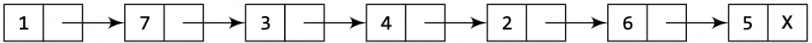
START

Take pointer variables PTR and PREPTR which initially point to START.



START
PREPTR
PTR

Move PTR and PREPTR such that NEXT part of PTR = NULL. PREPTR always points to the node just before the node pointed by PTR.



START PREPTR PTR

Set the NEXT part of PREPTR node to NULL.

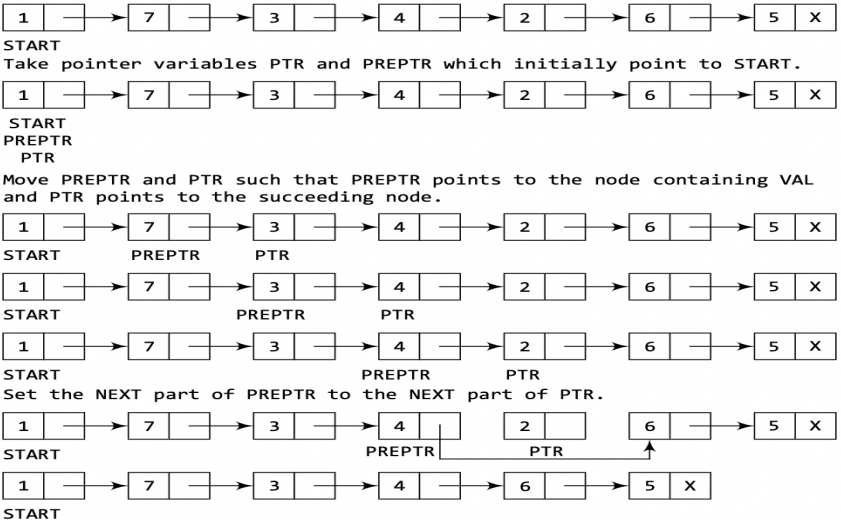
START

Deleting the Node After a Given Node

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR -> NEXT = PTR -> NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

```c
void deleteAfter(node **start, int num){
  node *ptr, *prePtr, *temp;

  if(*start == NULL){
    printf("Underflow\n");
    return;
  }
  ptr = *start;
  prePtr = ptr;
  while(prePtr->data != num){
    prePtr = ptr;
    ptr = ptr->next;
  }
  temp = ptr;
  prePtr->next = ptr->next;
  free(temp);
}
```

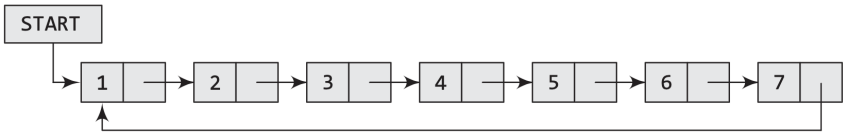
Deleting the Node After a Given Node



Circular Linked List

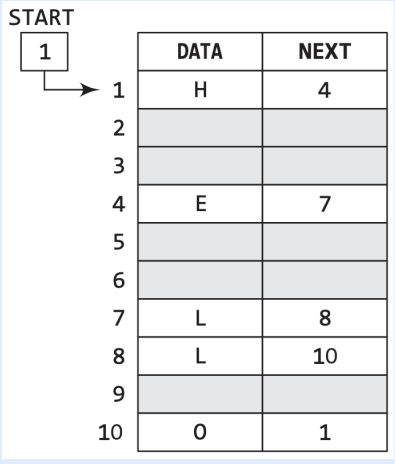
Circular Linked List

- In a circular linked list, the last node contains a pointer to the first node of the list.
- We can have a circular singly linked list as well as a circular doubly linked list.
- While traversing a circular linked list, we can begin at any node and traverse the list in any direction, forward or backward, until we reach the same node where we started.
- Thus, a circular linked list has no beginning and no ending.



Circular Linked List

Memory representation of a circular linked list



Inserting a Node to Circular Linked List

In this section, we will see how a new node is added into an already existing linked list. We will take two cases and then see how insertion is done in each case.

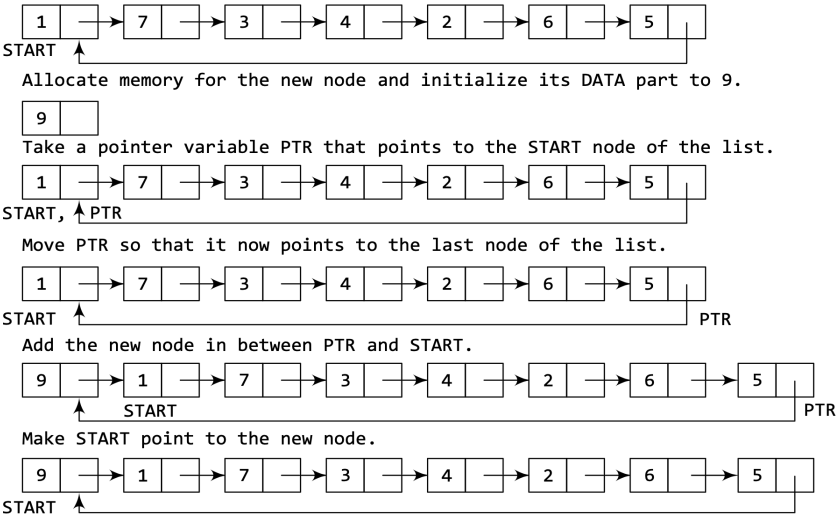
1. The new node is inserted at the begining of the circular linked list.
2. The new ndoe is inserted at the end of the circular linked list.

Inserting a Node at the Begining

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:     PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = START
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET START = NEW_NODE
Step 11: EXIT
```

```c
void insertBegin(node **start){
  node *newNode, *ptr;
  int val;

  printf("Enter data: ");
  scanf("%d", &val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  ptr = *start;
  while(ptr->next != *start)
    ptr = ptr->next;
  newNode->next = *start;
  ptr->next = newNode;
  *start = newNode;
}
```

Inserting a Node at the Begining

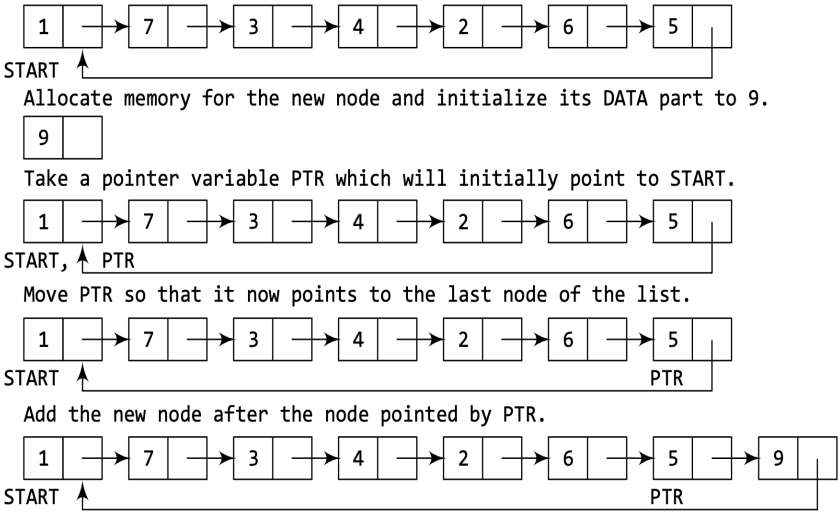


Inserting a Node at the End

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

```c
void insertEnd(node **start){
  node *newNode, *ptr;
  int val;

  printf("Enter data: ");
  scanf("%d", &val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  newNode->next = *start;
  ptr = *start;
  while(ptr->next != *start)
    ptr = ptr->next;
  ptr->next = newNode;
}
```

Inserting a Node at the End



Deleting a Node from Circular Linked List

In this section, we will see how a new node is deleted an already existing node from the linked list. We will take two cases and then see how deletion is done in each case.

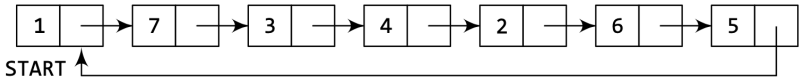
1. The first node is deleted.
2. The last node is deleted.

Deleting the First Node

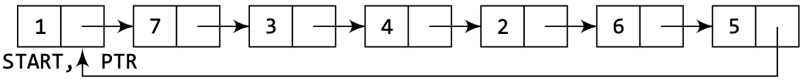
```
Step 1: IF START = NULL
              Write UNDERFLOW
              Go to Step 8
         [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> NEXT = START -> NEXT
Step 6: FREE START
Step 7: SET START = PTR -> NEXT
Step 8: EXIT
```

```c
void deleteBegin(node **start){
  node *ptr;
  if(*start == NULL){
    printf("Underflow\n");
    return;
  }
  ptr = *start;
  while(ptr->next != *start)
    ptr = ptr->next;
  ptr->next = (*start)->next;
  free(*start);
  *start = ptr->next;
}
```

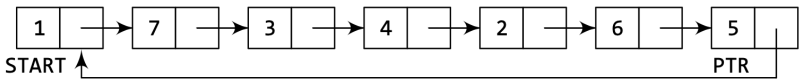
Deleting the First Node



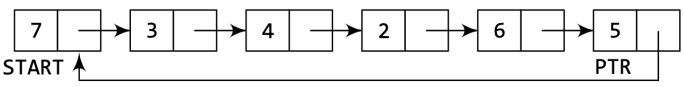
Take a variable PTR and make it point to the START node of the list.



Move PTR further so that it now points to the last node of the list.



The NEXT part of PTR is made to point to the second node of the list and the memory of the first node is freed. The second node becomes the first node of the list.

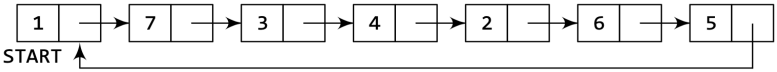


Deleting the Last Node

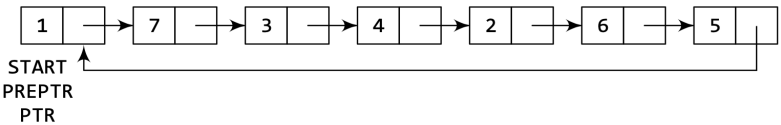
```
Step 1: IF START = NULL
                Write UNDERFLOW
                Go to Step 8
          [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != START
Step 4:          SET PREPTR = PTR
Step 5:          SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 6: SET PREPTR -> NEXT = START
Step 7: FREE PTR
Step 8: EXIT
```

```c
void deleteEnd(node **start){
  node *ptr, *prePtr;
  if(*start == NULL){
    printf("Underflow\n");
    return;
  }
  ptr = *start;
  while(ptr->next != *start){
    prePtr = ptr;
    ptr = ptr->next;
  }
  prePtr->next = *start;
  free(ptr);
}
```

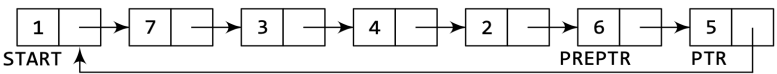
Deleting the Last Node



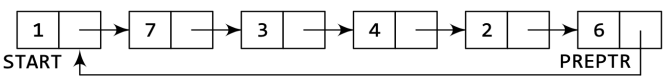
Take two pointers PREPTR and PTR which will initially point to START.



Move PTR so that it points to the last node of the list. PREPTR will always point to the node preceding PTR.



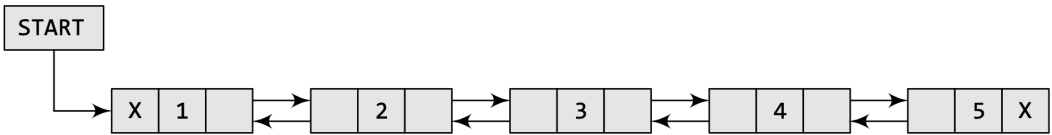
Make the PREPTR's next part store START node's address and free the space allocated for PTR. Now PREPTR is the last node of the list.



Doubly Linked List

Doubly Linked List

- A doubly linked list or a two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence.
- Therefore, it consists of three parts–data, a pointer to the next node, and a pointer to the previous node.

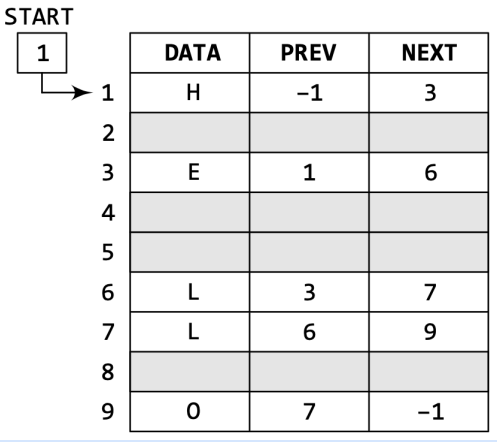


Doubly Linked List

The structure of a doubly linked list

```
struct node
{
        struct node *prev;
        int data;
        struct node *next;
};
```

Memory representation of a doubly linked list



Inserting New Node to Doubly Linked List

1. The new node is inserted at the beginning.
2. The new node is inserted at the end.
3. The new node is inserted after a given node.
4. The new node is inserted before a given node.

Inserting New node at the Beginning

```
Step 1: IF AVAIL = NULL
                Write OVERFLOW
                Go to Step 9
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET START -> PREV = NEW_NODE
Step 8: SET START = NEW_NODE
Step 9: EXIT
```

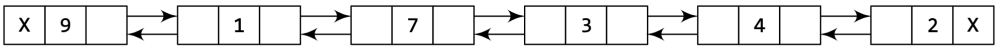
```c
void insertBegin(node **start){
  node *newNode;
  int val;

  printf("Enter data: ");
  scanf("%d", &val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  newNode->prev = NULL;
  newNode->next = *start;
  (*start)->prev = newNode;
  (*start) = newNode;
}
```

Inserting New node at the Beginning

START

Allocate memory for the new node and initialize its DATA part to 9 and PREV field to NULL.

Add the new node before the START node. Now the new node becomes the first node of the list.



START

Inserting New node at the End

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: EXIT
```

```c
void insertEnd(node **start){
  node *newNode, *ptr;
  int val;

  printf("Enter data: ");
  scanf("%d", &val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  newNode->next = NULL;
  ptr = *start;
  while(ptr->next != NULL)
    ptr = ptr->next;
  ptr->next = newNode;
  newNode->prev = ptr;
}
```

Inserting New node at the End

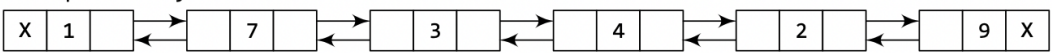
START

Allocate memory for the new node and initialize its DATA part to 9 and its NEXT field to NULL.

Take a pointer variable PTR and make it point to the first node of the list.

START,PTR

Move PTR so that it points to the last node of the list. Add the new node after the node pointed by PTR.



START PTR

Inserting New node after a Given Node

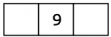
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR -> NEXT
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET PTR -> NEXT = NEW_NODE
Step 11: SET PTR -> NEXT -> PREV = NEW_NODE
Step 12: EXIT
```

```c
void insertAfter(node **start, int num){
  node *newNode, *ptr;
  int val;

  printf("Enter data: ");
  scanf("%d", &val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  ptr = *start;
  while(ptr->data != num)
    ptr = ptr->next;
  newNode->next = ptr->next;
  newNode->prev = ptr;
  ptr->next = newNode;
  ptr->next->prev = newNode;
}
```

Inserting New node after a Given Node

START

Allocate memory for the new node and initialize its DATA part to 9.



Take a pointer variable PTR and make it point to the first node of the list.

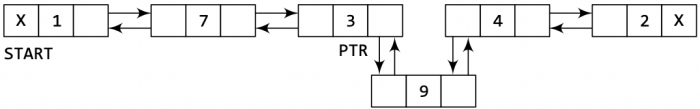
START,PTR

Move PTR further until the data part of PTR = value after which the node has to be inserted.

START

PTR

Insert the new node between PTR and the node succeeding it.



START

Inserting New node before a Given Node

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR
Step 9: SET NEW_NODE -> PREV = PTR -> PREV
Step 10: SET PTR -> PREV = NEW_NODE
Step 11: SET PTR  -> PREV -> NEXT = NEW_NODE
Step 12: EXIT
```

```c
void insertBefore(node **start, int num){
  node *newNode, *ptr;
  int val;

  printf("Enter data: ");
  scanf("%d", &val);
  newNode = (node*)malloc(sizeof(node));
  newNode->data = val;
  ptr = *start;
  while(ptr->data != num)
    ptr = ptr->next;
  newNode->next = ptr;
  newNode->prev = ptr->prev;
  ptr->prev->next = newNode;
  ptr->prev = newNode;
}
```

Inserting New node before a Given Node

START

Allocate memory for the new node and initialize its DATA part to 9.

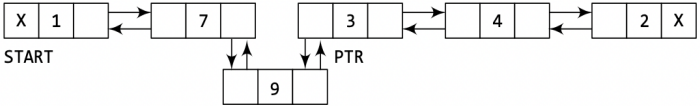
Take a pointer variable PTR and make it point to the first node of the list.

START, PTR

Move PTR further so that it now points to the node whose data is equal to the value before which the node has to be inserted.

START

Add the new node in between the node pointed by PTR and the node preceding it.



START

Deleting a Node in Doubly Linked List

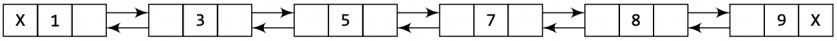
1. The first node is deleted.
2. The last node is deleted.
3. The node after a given node is deleted.
4. The node before a given node is deleted.

Deleting the First Node

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 6
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: SET START -> PREV = NULL
Step 5: FREE PTR
Step 6: EXIT
```

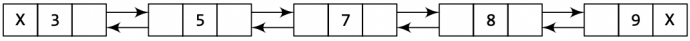
```c
void deleteFirst(node **start){
  node *ptr;
  if(*start == NULL){
    printf("Underflow\n");
    return;
  }
  ptr = *start;
  *start = (*start)->next;
  (*start)->prev = NULL;
  free(ptr);
}
```

Deleting the First Node



START

Free the memory occupied by the first node of the list and make the second node of the list as the START node.



START

Deleting the End Node

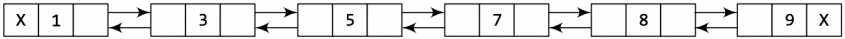
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != NULL
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> PREV -> NEXT = NULL
Step 6: FREE PTR
Step 7: EXIT
```

```c
void deleteLast(node **start){
  node *ptr;
  if(*start == NULL){
    printf("Underflow\n");
    return;
  }
  ptr = *start;
  while(ptr->next != NULL)
    ptr = ptr->next;
  ptr->prev->next = NULL;
  free(ptr);
}
```

Deleting the End Node

START

Take a pointer variable PTR that points to the first node of the list.

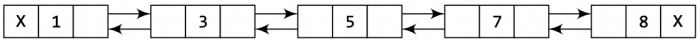


START,PTR

Move PTR so that it now points to the last node of the list.

START PTR

Free the space occupied by the node pointed by PTR and store NULL in NEXT field of its preceding node.



START

Deleting the Node After a Given Node

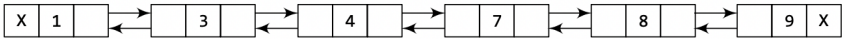
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> NEXT
Step 6: SET PTR -> NEXT = TEMP -> NEXT
Step 7: SET TEMP -> NEXT -> PREV = PTR
Step 8: FREE TEMP
Step 9: EXIT
```

```c
void deleteAfter(node **start, int num){
  node *ptr, *temp;
  if(*start == NULL){
    printf("Underflow\n");
    return;
  }
  ptr = *start;
  while(ptr->data != num)
    ptr = ptr->next;
  temp = ptr->next;
  ptr->next = temp->next;
  temp->next->prev = ptr;
  free(temp);
}
```

Deleting the Node After a Given Node

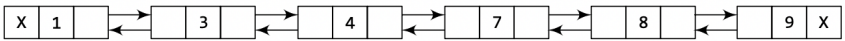
START

Take a pointer variable PTR and make it point to the first node of the list.



START,PTR

Move PTR further so that its data part is equal to the value after which the node has to be inserted.



START PTR

Delete the node succeeding PTR.

START PTR

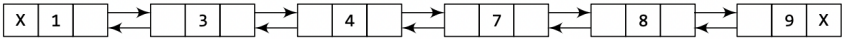
START

Deleting the Node Before a Given Node

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> PREV
Step 6: SET TEMP -> PREV -> NEXT = PTR
Step 7: SET PTR -> PREV = TEMP -> PREV
Step 8: FREE TEMP
Step 9: EXIT
```

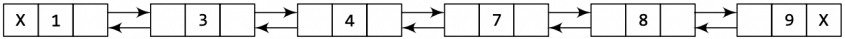
```c
void deleteBefore(node **start, int num){
  node *ptr, *temp;
  if(*start == NULL){
    printf("Underflow\n");
    return;
  }
  ptr = *start;
  while(ptr->data != num)
    ptr = ptr->next;
  temp = ptr->prev;
  temp->prev->next = ptr;
  ptr->prev = temp->prev;
  free(temp);
}
```

Deleting the Node Before a Given Node



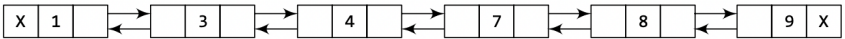
START

Take a pointer variable PTR that points to the first node of the list.



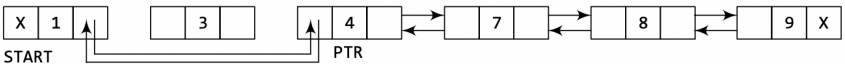
START,PTR

Move PTR further till its data part is equal to the value before which the node has to be deleted.



START PTR

Delete the node preceding PTR.



START

Applications of Linked List

Applications of Linked List

Linked lists can be used to represent polynomials and the different operations that can be performed on them. In this section, we will see how polynomials are represented in the memory using linked lists.

Polynomial Representation

Let us see how a polynomial is represented in the memory using a linked list. Consider a polynomial $6x^3 + 9x^2 + 7x + 1$. Every individual term in a polynomial consists of two parts, a coefficient and a power. Here, 6, 9, 7, and 1 are the coefficients of the terms that have 3, 2, 1, and 0 as their powers respectively. Every term of a polynomial can be represented as a node of the linked list. Figure shows the linked representation of the terms of the above polynomial.