MODULE 1-3 [CPE11201]

LINKED-LIST

ลิงค์ลิสต์

Assoc. Prof. Dr. Natasha Dejdumrong

OUTLINES

- Linked List
- Singly Linked List
- Circular Linked List
- Doubly Linked List
- Header Linked List
- Multi-Linked Matrix and Applications of Linked List

Linked List

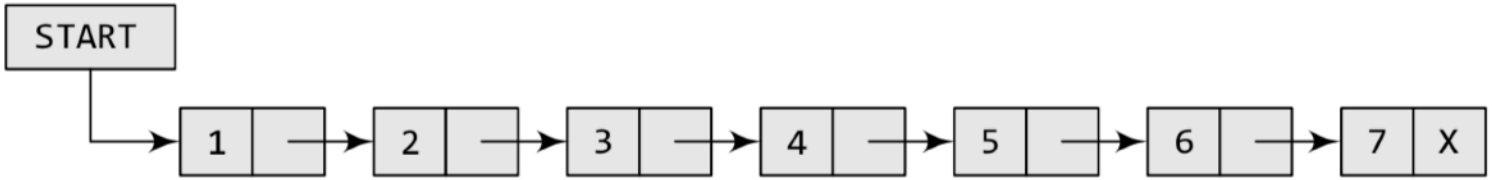


Figure 6.1 Simple linked list

Linked List

- Linked List is a linear collection of data elements.
- These data elements 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 stacks, 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.

Linked List

★ 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.

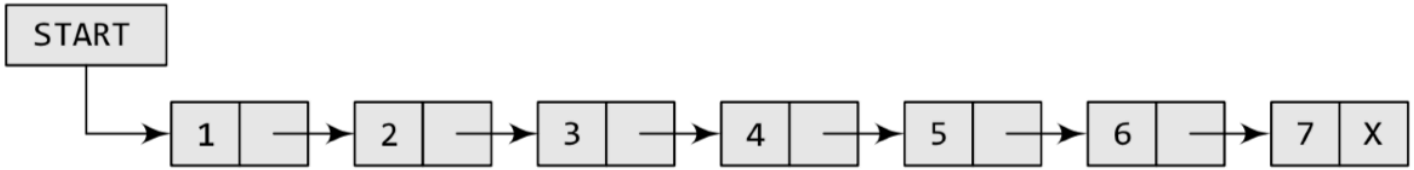


Figure 6.1 Simple linked list

★ In Fig. 6.1, we can see a linked list in which every node contains two parts, an integer and a pointer to the next node. The left part of the node which contains data may include a simple data type, an array, or a structure. The right part of the node contains a pointer to the next node (or address of the next node in sequence). The last node will have no next node connected to it, so it will store a special value called NULL.

Linked List

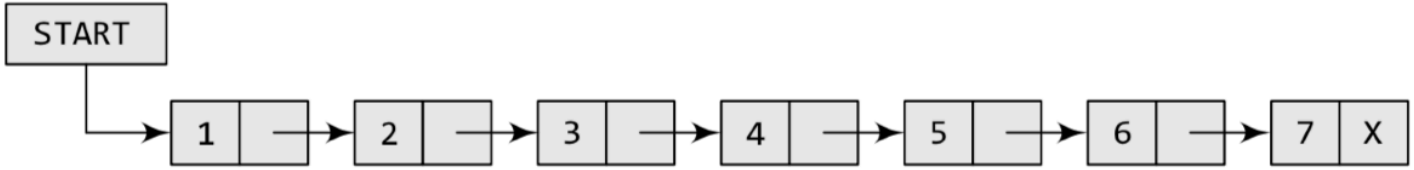


Figure 6.1 Simple linked list

In C, we can implement a linked list using the following code:

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

Linked List

In C, we can implement a linked list using the following code:

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

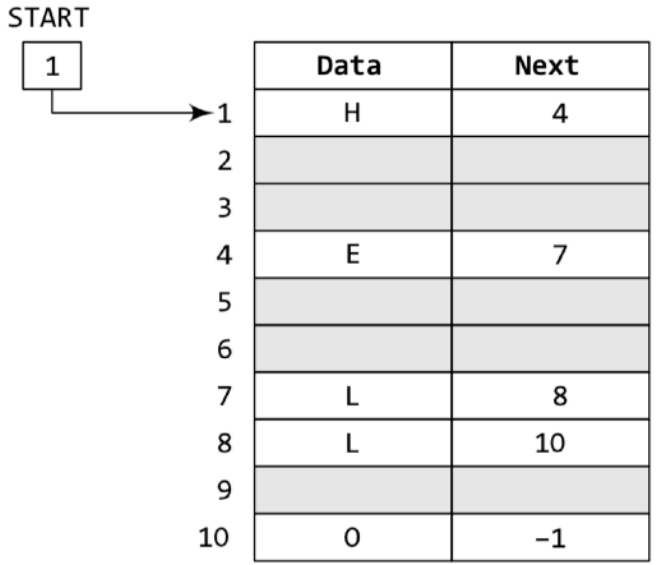
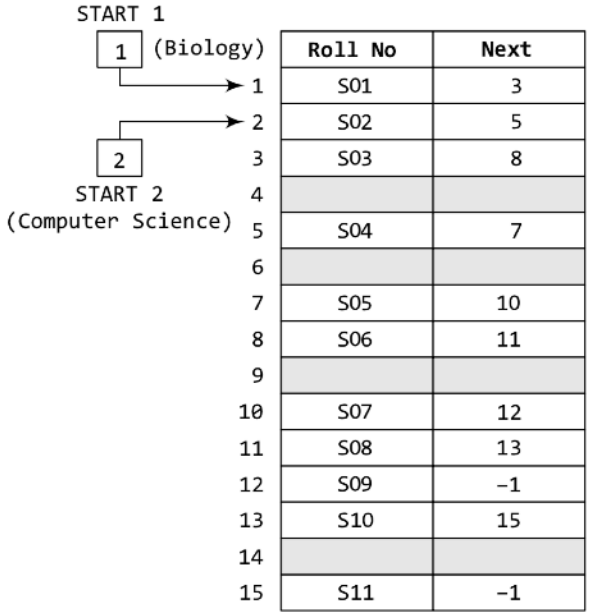


Figure 6.2 START pointing to the first element of the linked list in the memory

Linked List



	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

Figure 6.3 Two linked lists which are simultaneously maintained in the memory

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

Figure 6.4 Students' linked list

Linked List VS Array

- Both arrays and linked lists are a linear collection of data elements. But unlike an array, a linked list does not store its nodes in consecutive memory locations.
- Another point of difference between an array and a linked list is that a linked list does not allow random access of data. Nodes in a linked list can be accessed only in a sequential manner.
- But like an array, insertions and deletions can be done at any point in the list in a constant time.

Linked List VS Array

- Another advantage of a linked list over an array is that we can add any number of elements in the list. This is not possible in case of an array. For example, if we declare an array as int marks[20], then the array can store a maximum of 20 data elements only. There is no such restriction in case of a linked list.
- Thus, 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

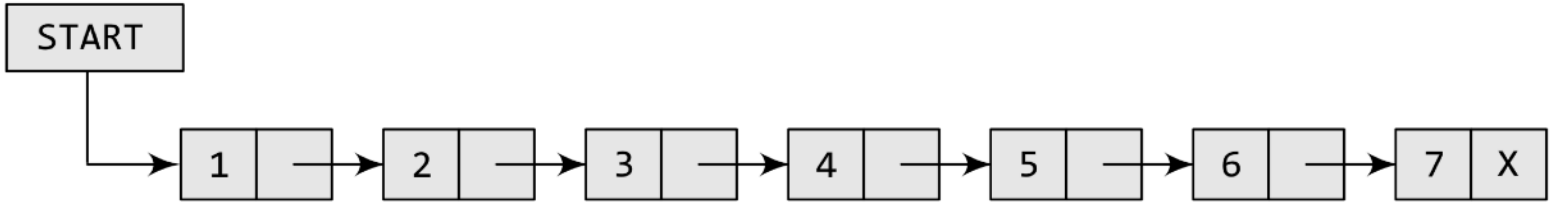


Figure 6.7 Singly linked list

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.

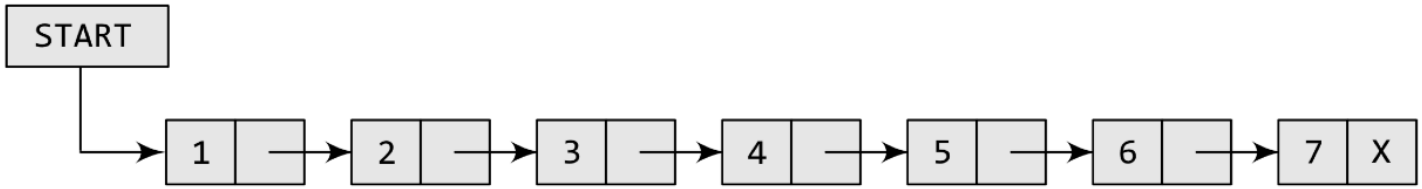


Figure 6.7 Singly linked list

Traversing a Singly Linked List

```
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
```

Figure 6.8 Algorithm for traversing a linked list

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
```

Figure 6.9 Algorithm to print the number of nodes in a linked list

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
```

Figure 6.10 Algorithm to search a linked list

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
```

Figure 6.10 Algorithm to search a linked list

Consider the linked list shown in Fig. 6.11. If we have VAL = 4, then the flow of the algorithm can be explained as shown in the figure.

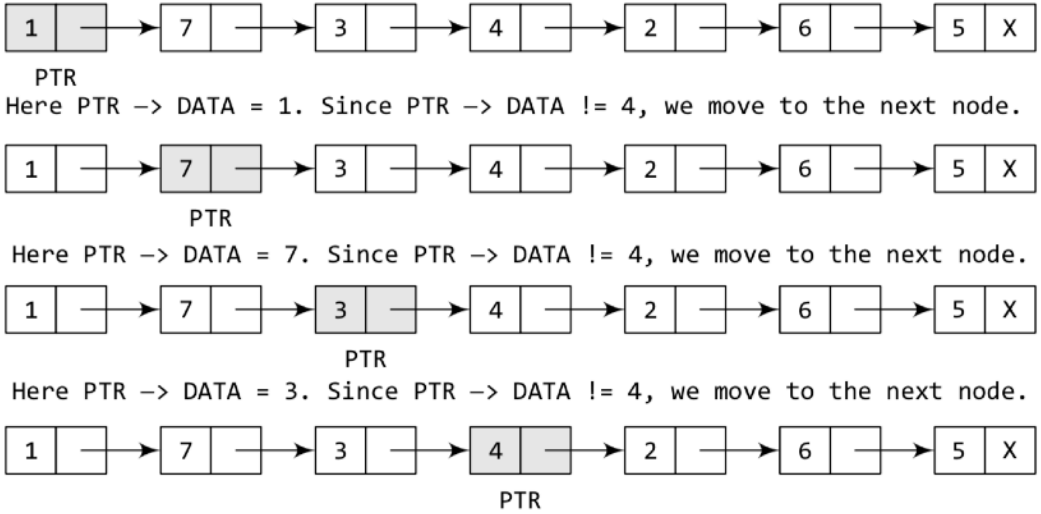


Figure 6.11 Searching a linked list

Inserting a New Node in a Linked List

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

Inserting a New Node at the Beginning

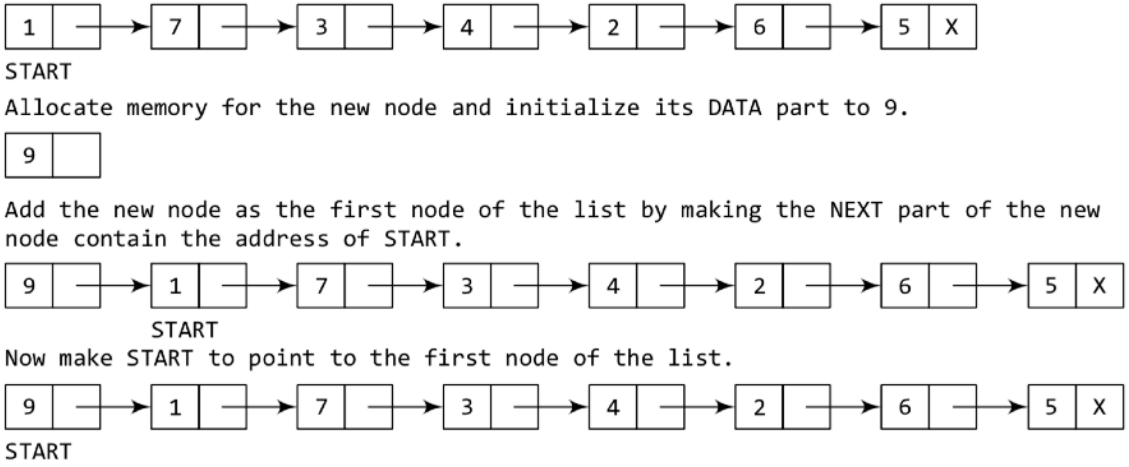
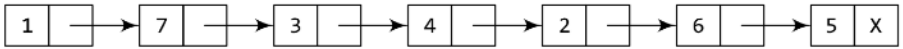


Figure 6.12 Inserting an element at the beginning of a linked list

```
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
```

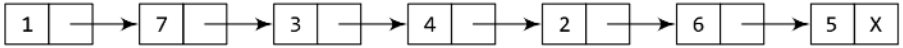
Figure 6.13 Algorithm to insert a new node at the beginning

Inserting a New Node at the End

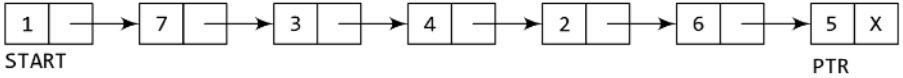


Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.

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.

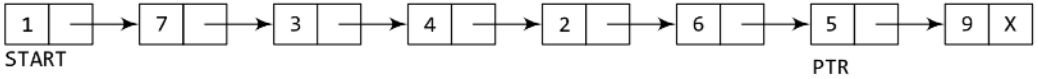


Figure 6.14 Inserting an element at the end of a linked list

```
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
```

Figure 6.15 Algorithm to insert a new node at the end

Inserting a New Node After a given Node

```
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
```

Figure 6.15 Algorithm to insert a new node at the end

```
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
```

Figure 6.16 Algorithm to insert a new node after a node that has value NUM

Inserting a New Node After a given Node

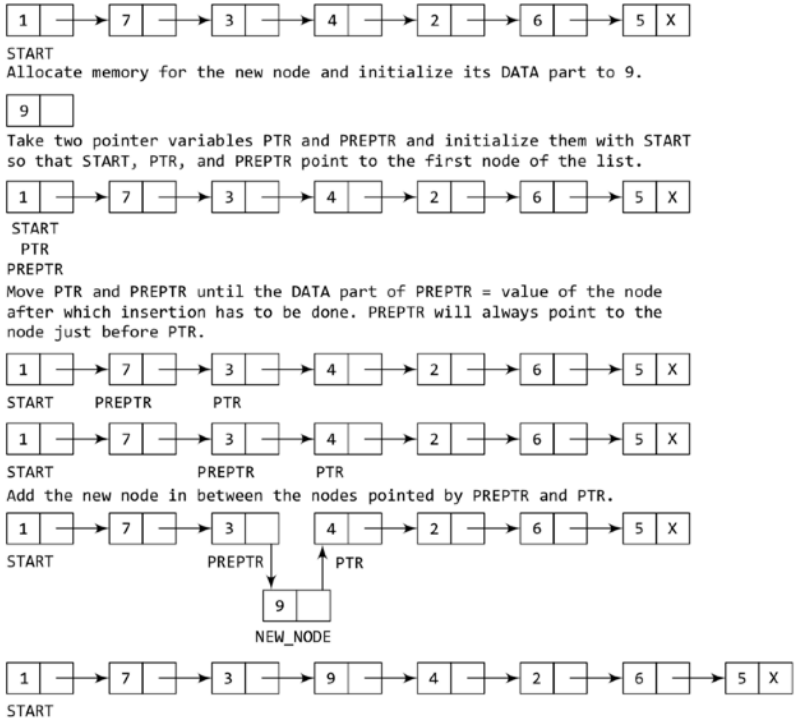


Figure 6.17 Inserting an element after a given node in a linked list

```
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
```

Figure 6.16 Algorithm to insert a new node after a node that has value NUM

Inserting a New Node Before a given Node

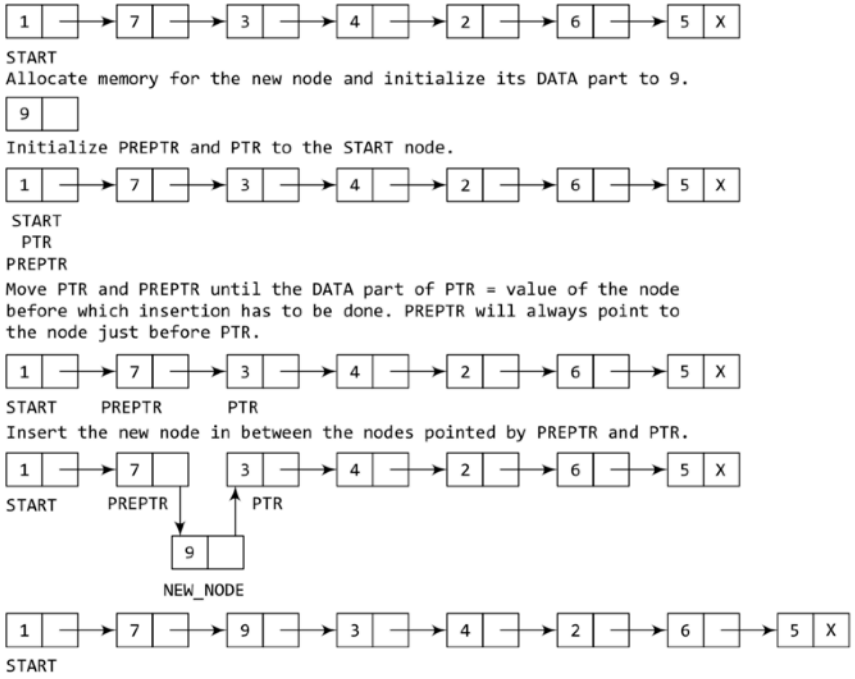


Figure 6.19 Inserting an element before a given node in a linked list

```
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
```

Figure 6.18 Algorithm to insert a new node before a node that has value NUM

Deleting a Node in a Linked List

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

Deleting the First Node in a Linked List

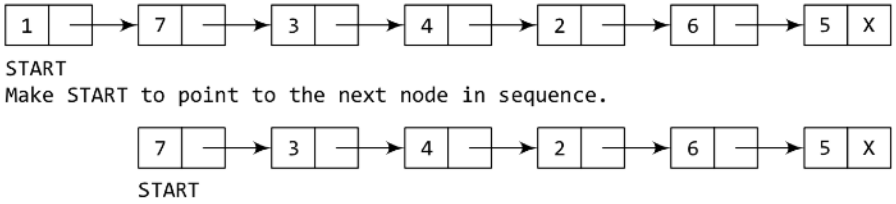


Figure 6.20 Deleting the first node of a linked list

```
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
```

Figure 6.21 Algorithm to delete the first node

Deleting the Last Node in a Linked List

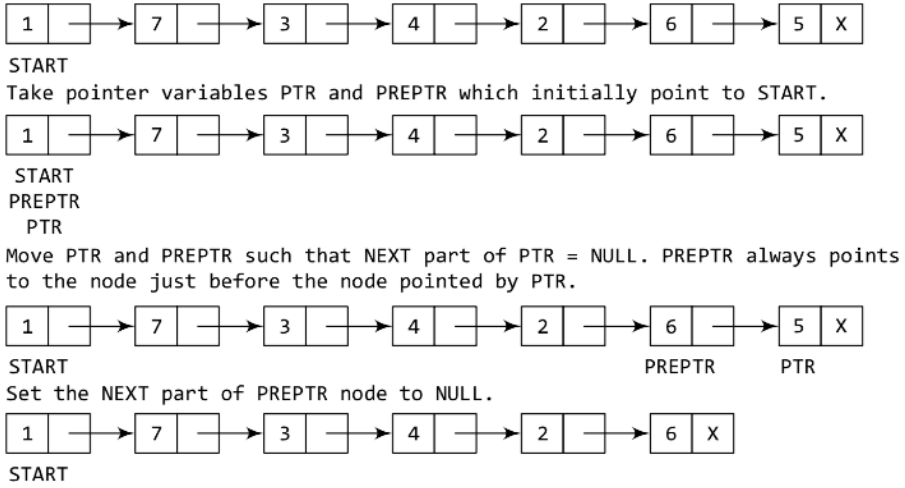


Figure 6.22 Deleting the last node of a linked list

```
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
```

Figure 6.23 Algorithm to delete the last node

Deleting the Node After a Given Node

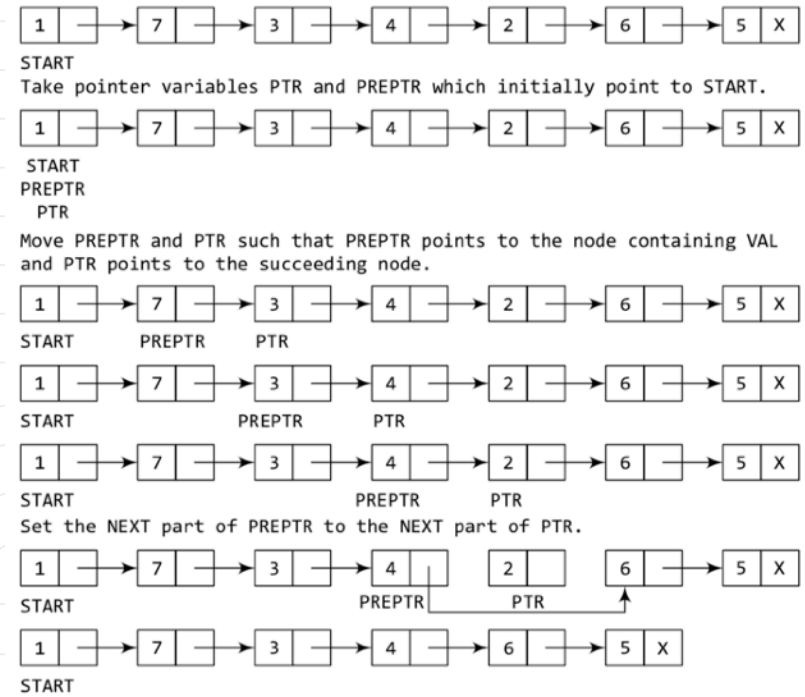


Figure 6.24 Deleting the node after a given node in a linked list

```
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
```

Figure 6.25 Algorithm to delete the node after a given node

Example

★ Write a program to create a linked list and perform insertions and deletions of all cases. Write functions to sort and finally delete the entire list at once.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>

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

struct node *start = NULL;
```

```
struct node *create_ll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *insert_before(struct node *);
struct node *insert_after(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_node(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);
struct node *sort_list(struct node *);
```

Circular Linked List

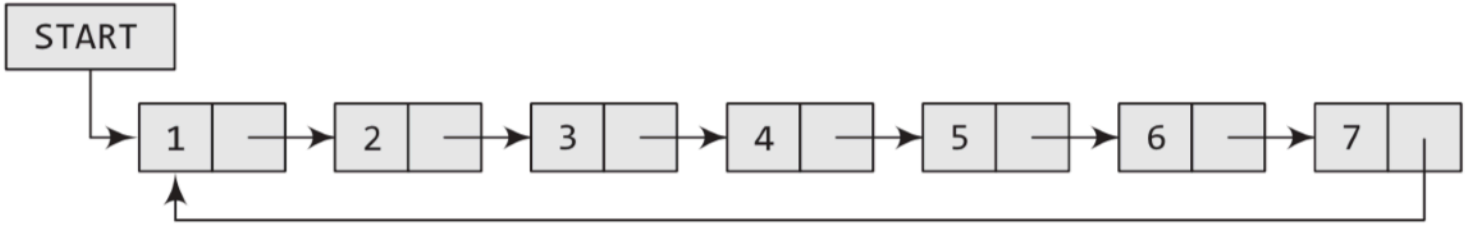


Figure 6.26 Circular linked list

Circular Linked List

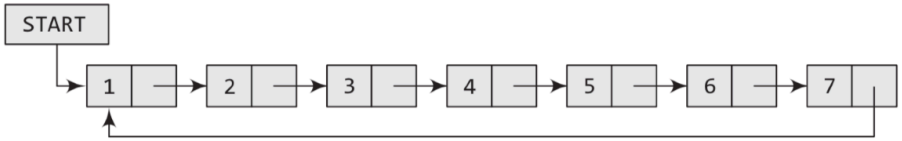
★ In a circular linked list,

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

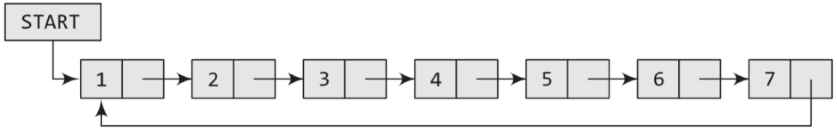


Figure 6.26 Circular linked list

START

1

	DATA	NEXT
1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	O	1

Figure 6.27 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.
- Case 1: The new node is inserted at the beginning of the circular linked list.
- Case 2: The new node is inserted at the end of the circular linked list.

Inserting a Node at the Beginning

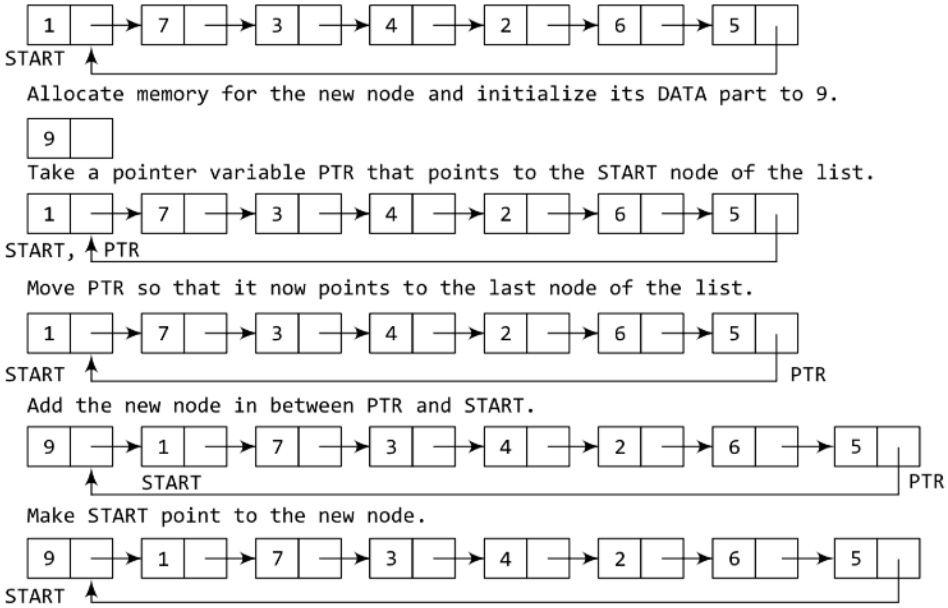


Figure 6.29 Inserting a new node at the beginning of a circular linked list

```
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
```

Figure 6.30 Algorithm to insert a new node at the beginning

Inserting a Node at the End

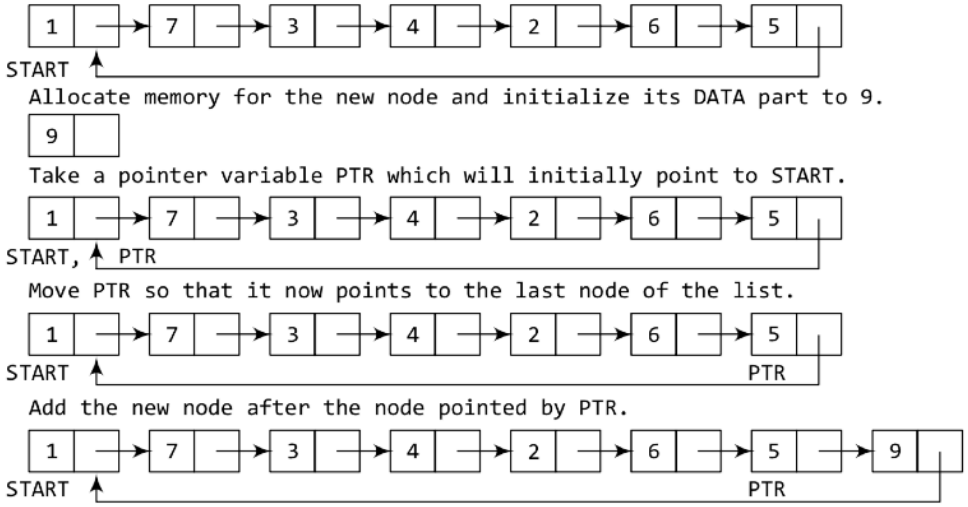


Figure 6.31 Inserting a new node at the end of a circular linked list

```
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
```

Figure 6.32 Algorithm to insert a new 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.
- Case 1: The first node is deleted.
- Case 2: The last node is deleted.

Deleting the First Node from Circular List

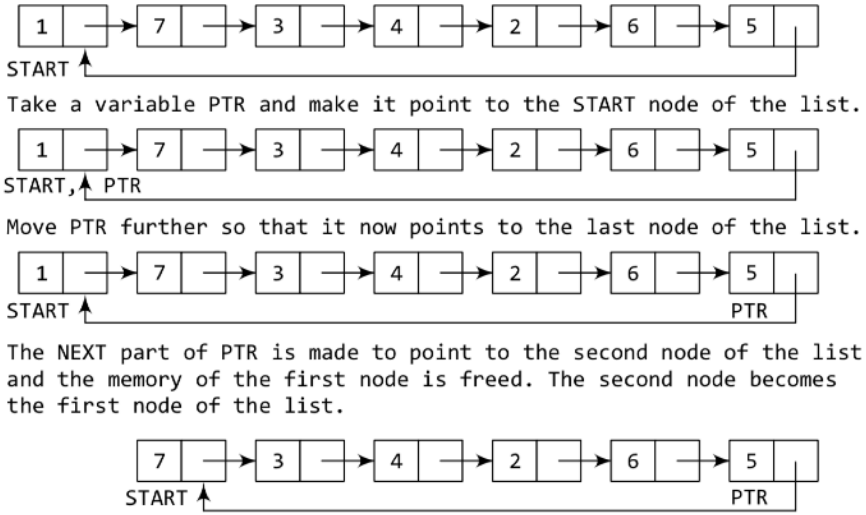


Figure 6.33 Deleting the first node from a circular linked list

```
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
```

Figure 6.34 Algorithm to delete the first node

Deleting the Last Node from Circular List

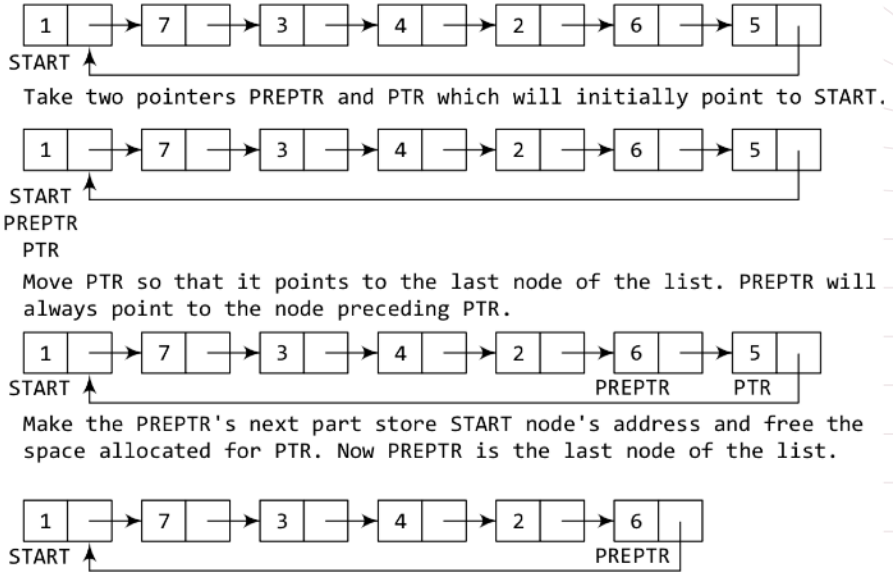


Figure 6.35 Deleting the last node from a circular linked list

```
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 != 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
```

Figure 6.36 Algorithm to delete the last node

Example

★ Write a program to create a circular linked list and perform insertions and deletions at the beginning the first and end the list.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>

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

struct node *start = NULL;
```

```
struct node *create_cll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);

struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_node(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);
```

Doubly Linked List

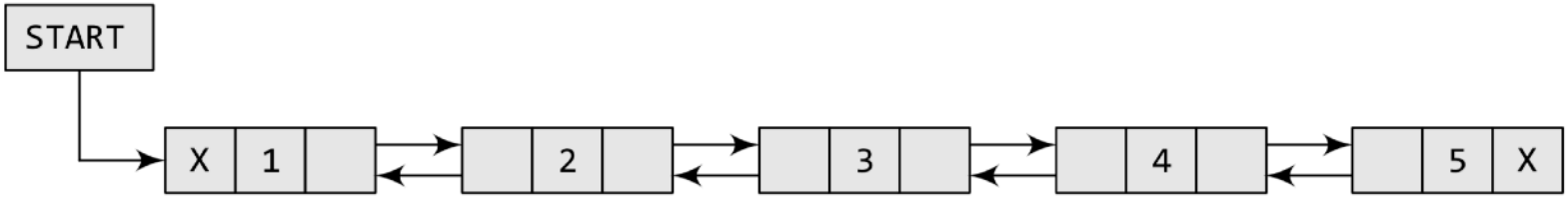


Figure 6.37 Doubly linked list

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.

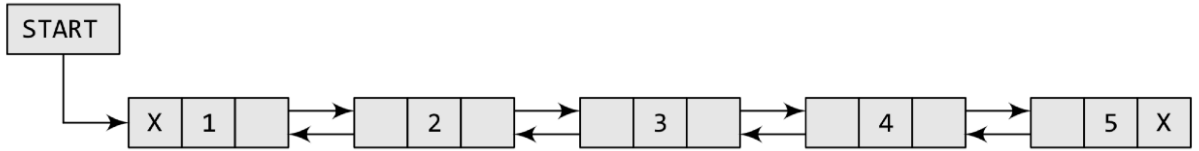


Figure 6.37 Doubly linked list

Doubly Linked List

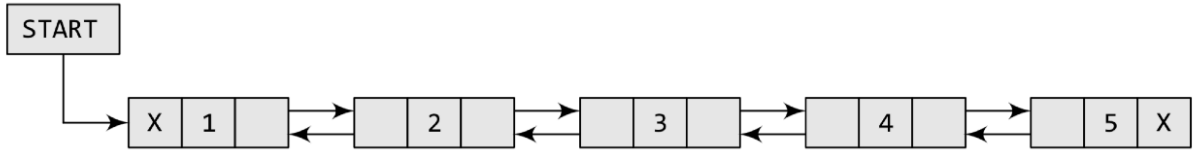


Figure 6.37 Doubly linked list

★ In C, the structure of a doubly linked list can given as,

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

Doubly Linked List

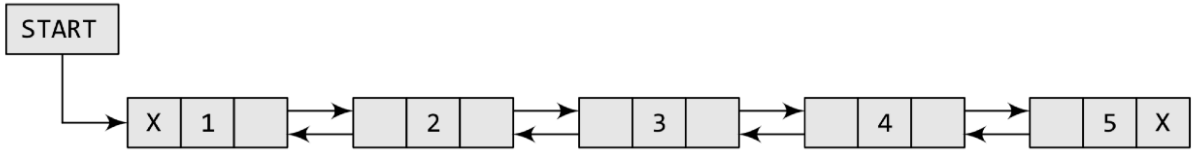


Figure 6.37 Doubly linked list

START

1

	DATA	PREV	NEXT
1	H	−1	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	O	7	−1

Figure 6.38 Memory representation of a doubly linked list

Inserting New Node to Doubly Linked List

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

Inserting New Node at the Beginning

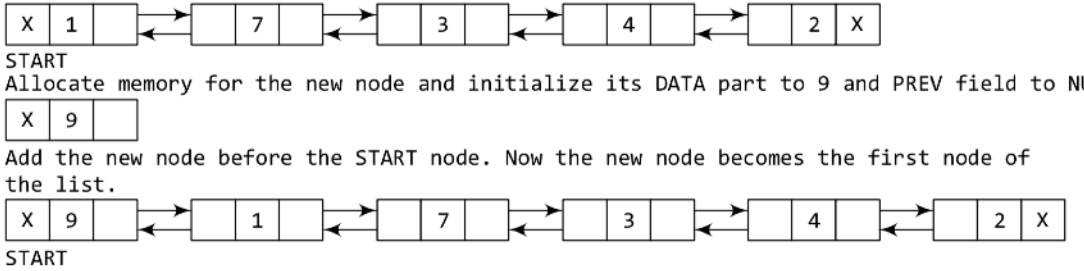


Figure 6.39 Inserting a new node at the beginning of a doubly linked list

```
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
```

Figure 6.40 Algorithm to insert a new node at the beginning

Inserting New Node at the End

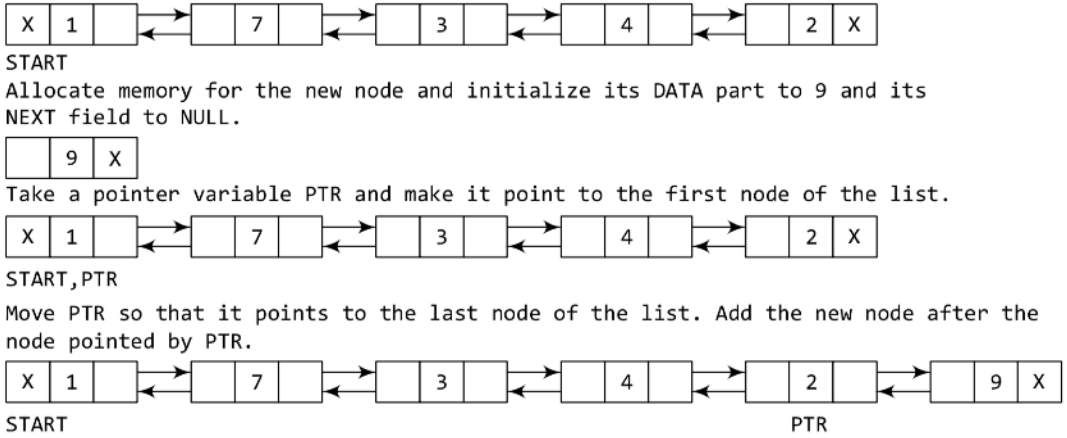


Figure 6.41 Inserting a new node at the end of a doubly linked list

```
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
```

Figure 6.42 Algorithm to insert a new node at the end

Inserting New Node after a Given Node

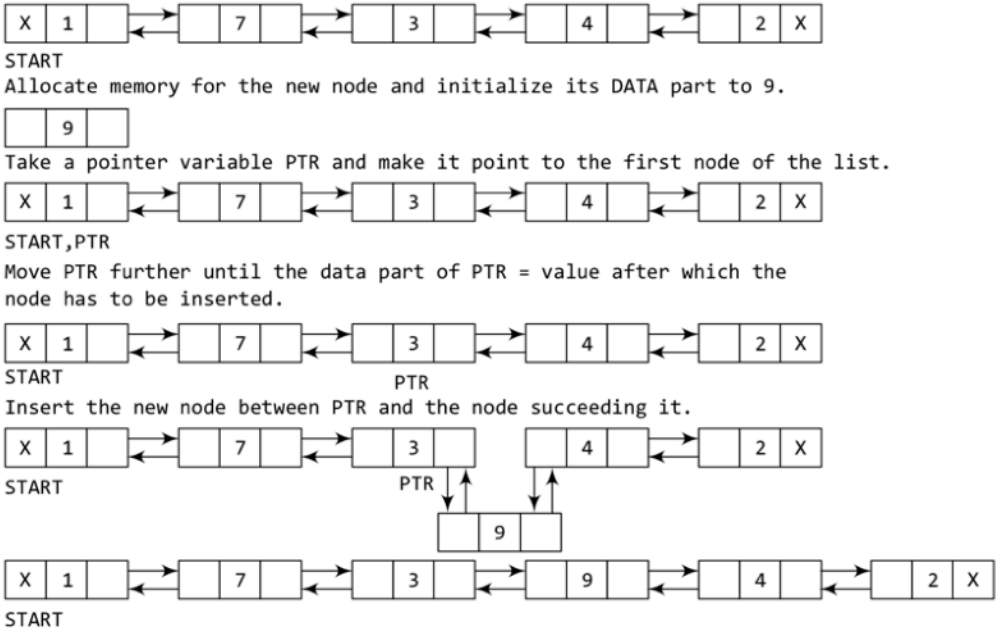


Figure 6.44 Inserting a new node after a given node in a doubly linked list

```
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
```

Figure 6.43 Algorithm to insert a new node after a given node

Inserting New Node before a Given Node

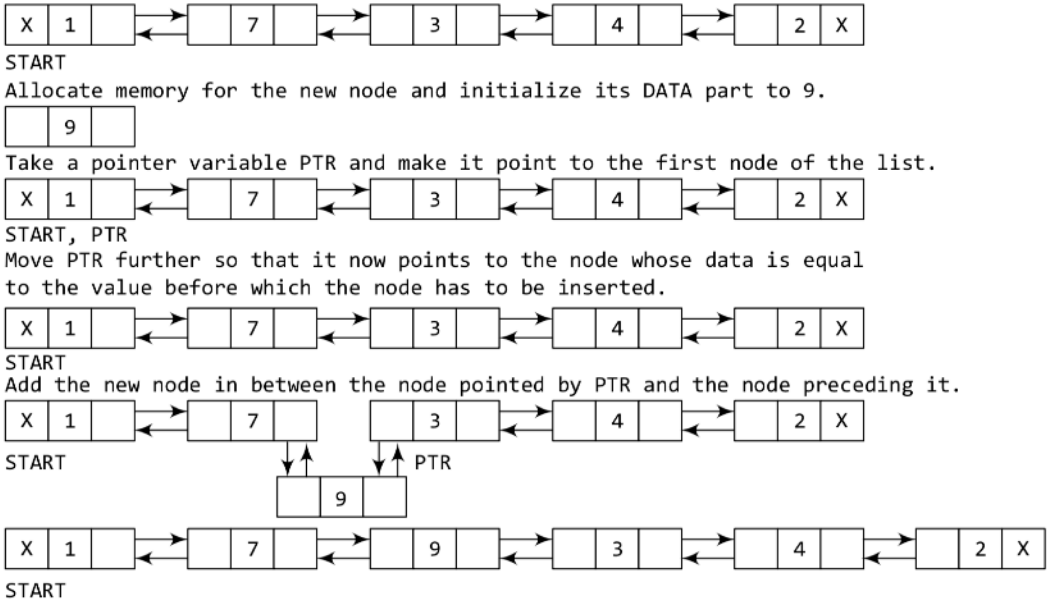


Figure 6.46 Inserting a new node before a given node in a doubly linked list

```
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
```

Figure 6.45 Algorithm to insert a new node before a given node

Deleting a Node in Doubly Linked List

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

Deleting the First Node

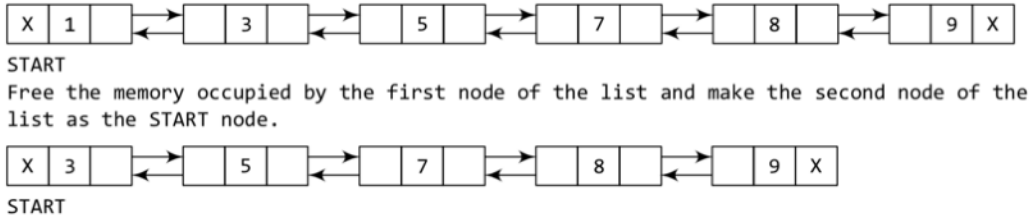


Figure 6.47 Deleting the first node from a doubly linked list

```
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
```

Figure 6.48 Algorithm to delete the first node

Deleting the End Node

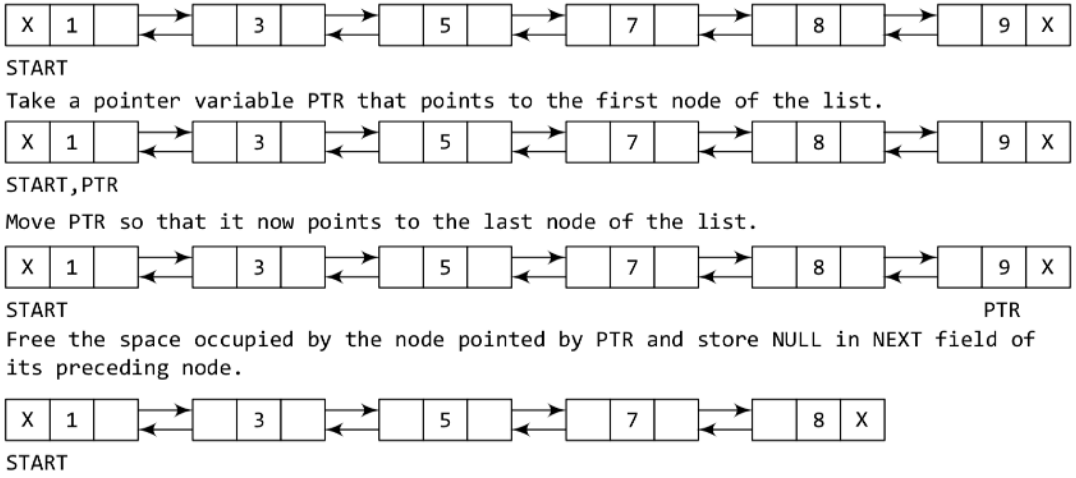


Figure 6.49 Deleting the last node from a doubly linked list

```
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
```

Figure 6.50 Algorithm to delete the last node

Deleting the Node After a Given Node

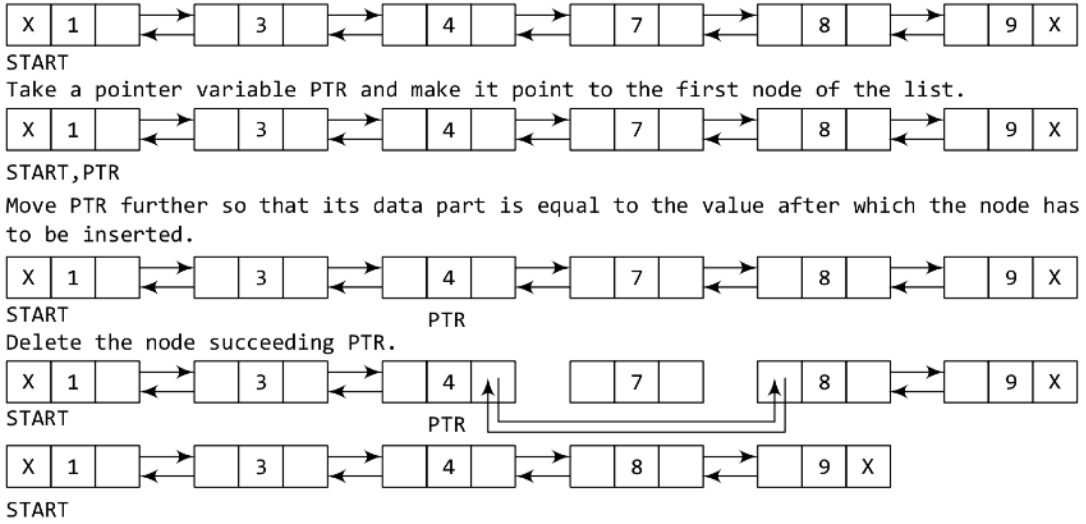


Figure 6.51 Deleting the node after a given node in a doubly linked list

```
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
```

Figure 6.52 Algorithm to delete a node after a given node

Deleting the Node Before a Given Node

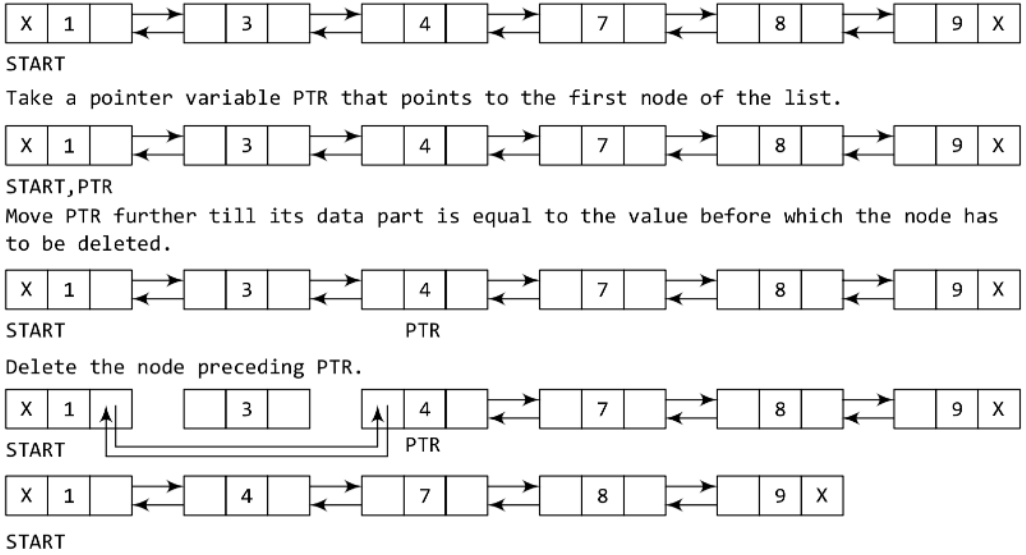


Figure 6.53 Deleting a node before a given node in a doubly linked list

```
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
```

Figure 6.54 Algorithm to delete a node before a given node

Example

★ Write a program to create a doubly linked list and perform insertions and deletions in all cases.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>

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

struct node *start = NULL;
```

```
struct node *create_ll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *insert_before(struct node *);
struct node *insert_after(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_before(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);
```

Circular Doubly Linked List

- A circular doubly linked list or a circular 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.
- The difference between a doubly linked and a circular doubly linked list is same as that exists between a singly linked list and a circular linked list.
- The circular doubly linked list does not contain NULL in the previous field of the first node and the next field of the last node.
- Rather, the next field of the last node stores the address of the first node of the list, i.e., START.
- Similarly, the previous field of the first field stores the address of the last node.

Circular Doubly Linked List

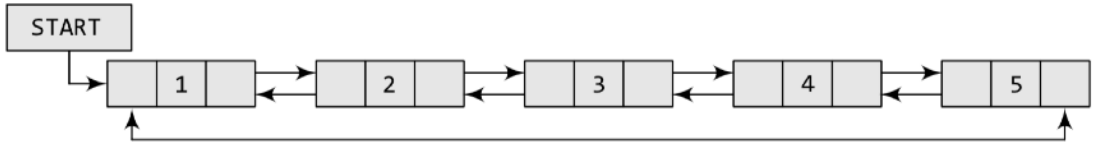


Figure 6.55 Circular doubly linked list

START = 1

	DATA	PREV	Next
1	H	9	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	O	7	1

Figure 6.56 Memory representation of a circular doubly linked list

Inserting New Node to Circular Doubly Linked List

- Case 1: The new node is inserted at the beginning.
- Case 2: The new node is inserted at the end.

Inserting New Node at the Beginning

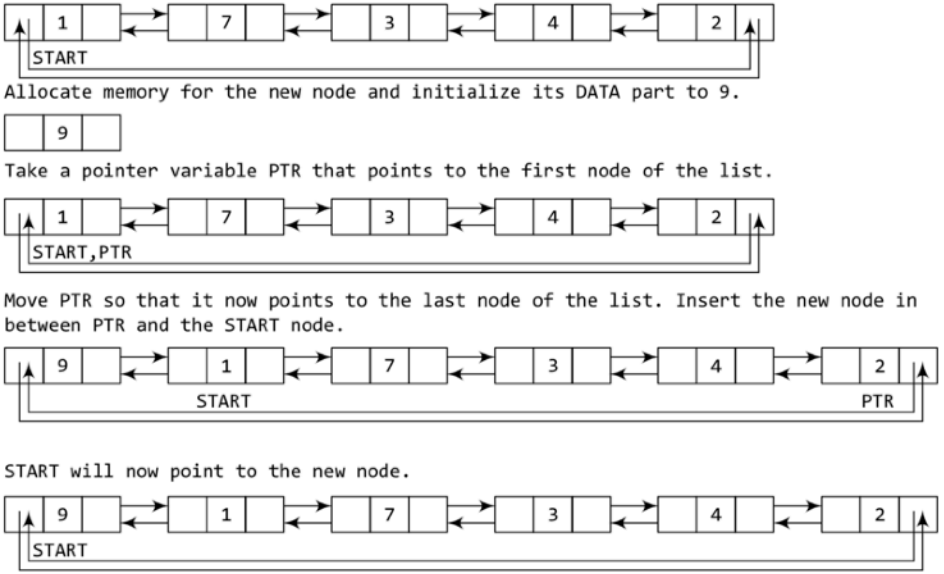


Figure 6.57 Inserting a new node at the beginning of a circular doubly linked list

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 13
        [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:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET PTR -> NEXT = NEW_NODE
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET NEW_NODE -> NEXT = START
Step 11: SET START -> PREV = NEW_NODE
Step 12: SET START = NEW_NODE
Step 13: EXIT
```

Figure 6.58 Algorithm to insert a new node at the beginning

Inserting New Node at the End

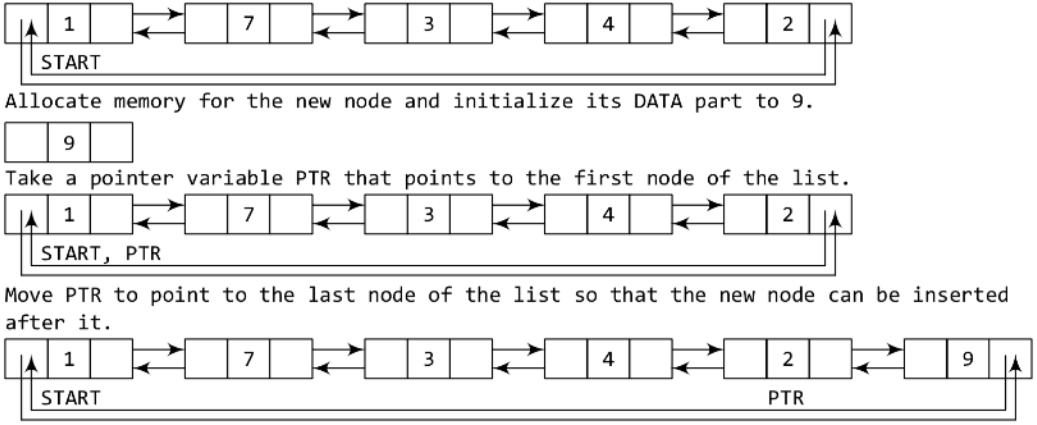


Figure 6.59 Inserting a new node at the end of a circular doubly linked list

```
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 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: SET NEW_NODE -> PREV = PTR
Step 11: SET START -> PREV = NEW_NODE
Step 12: EXIT
```

Figure 6.60 Algorithm to insert a new node at the end

Deleting New Node from Circular Doubly Linked List

- Case 1: The first node is deleted.
- Case 2: The last node is deleted.

Deleting the First Node from Circular Doubly LL

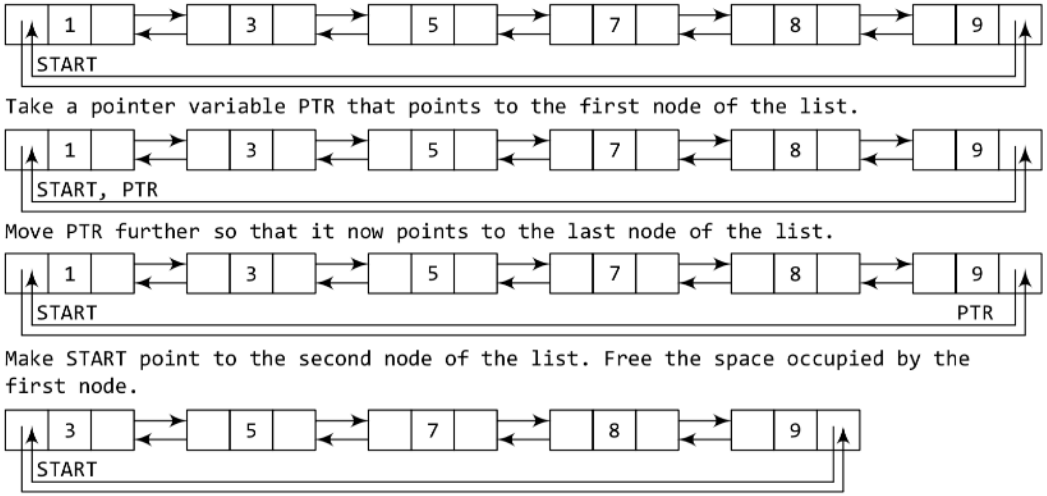


Figure 6.61 Deleting the first node from a circular doubly linked list

```
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: SET START -> NEXT -> PREV = PTR
Step 7: FREE START
Step 8: SET START = PTR -> NEXT
```

Figure 6.62 Algorithm to delete the first node

Deleting the Last Node from Circular Doubly LL

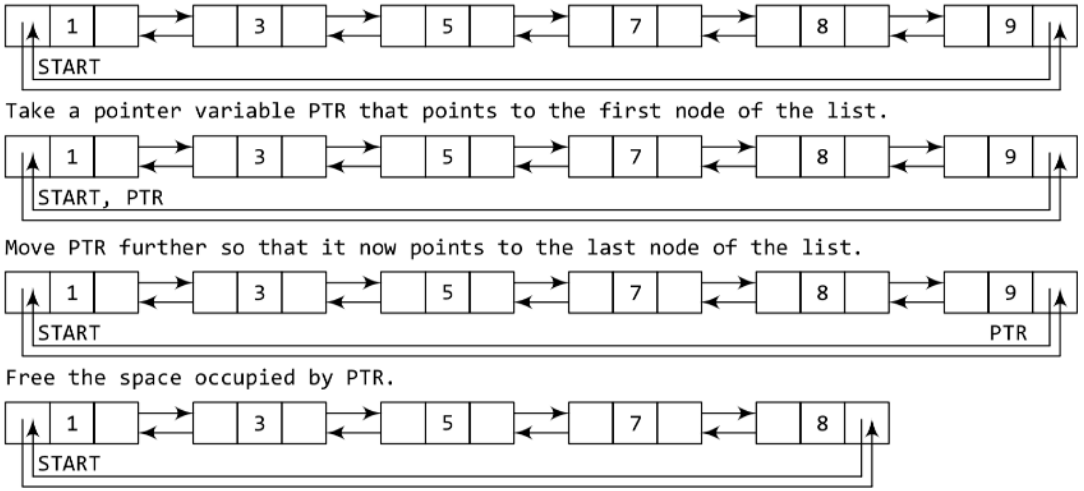


Figure 6.63 Deleting the last node from a circular doubly linked list

```
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 -> PREV -> NEXT = START
Step 6: SET START -> PREV = PTR -> PREV
Step 7: FREE PTR
Step 8: EXIT
```

Figure 6.64 Algorithm to delete the last node

Header Linked List

★ A header linked list is a special type of linked list which contains a header node at the beginning of the list.

★ So, in a header linked list, START will not point to the first node of the list but START will contain the address of the header node. The following are the two variants of a header linked list:

- *Grounded header linked list* which stores NULL in the next field of the last node.
- *Circular header linked list* which stores the address of the header node in the next field of the last node. Here, the header node will denote the end of the list.

Header Linked List

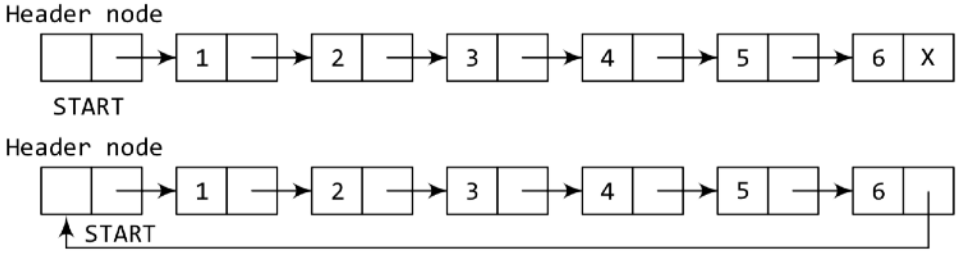


Figure 6.65 Header linked list

START
5

	DATA	NEXT
1	H	3
2		
3	E	6
4		
5	1234	1
6	L	7
7	L	9
8		
9	0	-1

Figure 6.66 Memory representation of a header linked list

START
5

	DATA	NEXT
1	H	3
2		
3	E	6
4		
5	1234	1
6	L	7
7	L	9
8		
9	0	5

Figure 6.67 Memory representation of a circular header linked list

Traverse a Circular Header Linked List

START
5

	DATA	NEXT
1	H	3
2		
3	E	6
4		
5	1234	1
6	L	7
7	L	9
8		
9	O	5

Figure 6.67 Memory representation of a circular header linked list

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

Figure 6.68 Algorithm to traverse a circular header linked list

Inserting and Deleting a Node in CHLL

```
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 PTR = START -> NEXT
Step 5: SET NEW_NODE -> DATA = VAL
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: NEW_NODE -> NEXT = PTR -> NEXT
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

Figure 6.69 Algorithm to insert a new node in a circular header linked list

```
Step 1: SET PTR = START->NEXT
Step 2: Repeat Steps 3 and 4 while
        PTR -> DATA != VAL
Step 3:     SET PREPTR = PTR
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PREPTR -> NEXT = PTR -> NEXT
Step 6: FREE PTR
Step 7: EXIT
```

Figure 6.70 Algorithm to delete a node from a circular header linked list

Multi-Linked Matrix

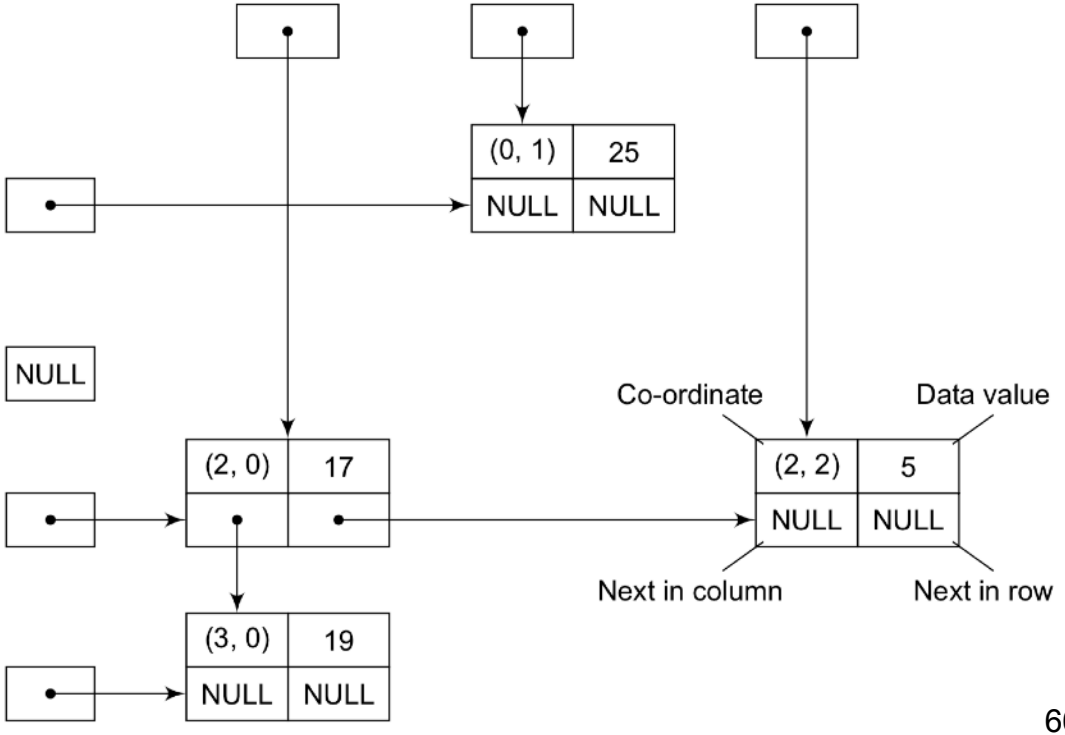
y \ x	0	1	2
0	0	25	0
1	0	0	0
2	17	0	5
3	19	0	0

Figure 6.72 Sparse matrix

Multi-Linked Matrix

y \ x	0	1	2
0	0	25	0
1	0	0	0
2	17	0	5
3	19	0	0

Figure 6.72 Sparse matrix



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 6.74 shows the linked representation of the terms of the above polynomial.

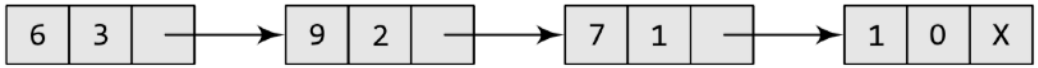


Figure 6.74 Linked representation of a polynomial

Questions and Answers

Assignment Set 4

ส่งภายในวันจันทร์ที่ 29 มกราคม 2567

*ให้เอาไปหารสมี

ระวังโดนหลอก!

Temperature

1. วัสดุที่ใช้เป็นส่วนประกอบของเลเซอร์ทำจากแก้วแท่งยาว $30.0\ cm$ และเส้นผ่านศูนย์กลาง $1.50\ cm$ สมมติค่าสัมประสิทธิ์ของการขยายตัวเชิงเส้นของแก้วเท่ากับ $9.00\times10^{-6}(^{\circ}C)^{-1}$ ถ้าอุณหภูมิของแท่งแก้วเพิ่มขึ้นจากเดิม $65.0^{\circ}C$ ปริมาณต่อไปนี้เพิ่มขึ้นเท่าใด

1) ความยาวของแท่งแก้ว (ตอบ $\Delta L = 0.176\ mm$)
2) เส้นผ่านศูนย์กลางของแท่งแก้ว (ตอบ $\Delta L_{\phi} = 8.78\ \mu m$)
3) ปริมาตรของแท่งแก้ว (ตอบ $\Delta V = 0.093\ cm^3$)

→ ดูหน่วยตอนตอบดีๆ

L, Δt, α, r = 1.5 cm / 2 = 0.75 cm

(1.) $\Delta L = \alpha \times L \times \Delta t$

$\Delta L = 9\times10^{-6} \times 30\times10^{-2} \times 65$

$\Delta L = 17{,}550 \times 10^{-8}$

$\Delta L = 0.175\ mm$ #

(2.) $\frac{\Delta A}{A_0} = 2\alpha\Delta T$

(3.) $\frac{\Delta V}{V_0} = 3\alpha\Delta T$

สู้ๆนะลิง

2. ภาชนะรูปทรงลูกบาศก์มีความกว้างด้านละ $10.0\ cm$ ภายในมีอากาศบรรจุอยู่ (เทียบเท่ากับมวลโมเลกุล $28.0\ g/mol$) ที่ความดันบรรยากาศ $1.013\times10^5\ Pa$ และอุณหภูมิ $300\ K$ ภาชนะ