6.3

STEP1[Class]: first make a class of node.

STEP2[initialize]: Set value; Set Node head; Set Node next.

STEP3[constructor]: Assign values of value and next.

STEP4[driver code]: create object of linked list, pass the arguments to node class, connect the nodes

STEP5[Traverse]: use while loop.

STEP6[increment]: linkedList.head.next++

STEP7[Test counter]: linkedList.head != null

Write: LinkedList.head.value exit.

STEP 5. [Repeat loop.] Go to Step 5

6.4

STEP1[import]: import LinkedList.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[print] print the whole list.

6.5

STEP1[import]: import LinkedList.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[initialize]: Set element as increment counter.

STEP5[increment]: element++

Write element exit.

STEP5[repeat loop]: Go to step 4

6.6

STEP1[import]: import LinkedList and Iterator.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[initialize]: initialize p as iterating variable and Set p=: 1.

STEP5[print]: use while loop to print list from 2nd position.

STEP6[increment]: p++

Write p exit.

STEP5[repeat loop]: Go to step 4

6.7

STEP1[import]: import LinkedList and Iterator.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[initialize]: initialize it as iterating variable and call method decendingIterator().

STEP5[print]: use while loop to print list in descending order.

STEP6[increment]: it++

Write it exit.

STEP5[repeat loop]: Go to step 4

6.8

STEP1[import]: import LinkedList.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[indexing]: add another element by defining its index number.

STEP5[print]: print the whole list.

6.9

STEP1[import]: import LinkedList.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[add at start]: add element at first position using .addFirst() method.

STEP5[add at last]: add element at last position using .addLast() method.

STEP6[print]: print the whole list.

6.10

STEP1[import]: import LinkedList.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[insert at start]: insert specific element at first position using .offerFirst() method.

STEP5[print]: print the whole list.

6.11

STEP1[import]: import LinkedList and Iterator.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[initialize]: initialize p as iterating variable and Set p=: 0.

STEP5[increment]: p++

STEP6[Test counter]: p < list size.

Write p and list exit.

STEP7[repeat loop]: Go to step 4

6.12

STEP1[import]: import LinkedList.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[delete first]: delete element at first position using .removeFirst() method.

STEP5[delete last]: delete element at last position using .removeLast() method.

STEP6[print]: print the whole list.

6.13

STEP1[import]: import LinkedList.

STEP2[object]: create object of LinkedList of String data type name list.

STEP3[add]: add elements to list using add method.

STEP4[swap]: swap element of index 0 with index 2 using Collections.swap().

STEP5[print]: print the whole list.

6.14

STEP1[import]: import LinkedList.

STEP2[object]: create two object of LinkedList of String data type name list1 andlist2.

STEP3[add]: add elements to list1 and list2 using add method.

STEP4[object]: create object of LinkedList of String data type name list3 which is empty.

STEP5[initialize]: initialize e as iterating variable.

STEP6[compare]: using enhanced for loop add list2.contains(e) in list3.

STEP7[condition] if contains yes otherwise no

STEP8[print]: print the whole list3.

6.15

STEP1[class]: create node class.

STEP2[initialize] initialize data of integer type head as node and next as node.

STEP3[constructor]: assign value to data.

STEP4[insert at first]: in this method we insert element at the first position of linkedlist

Create object of Node class name new\_node

Set new\_node.next as head and Set head as new\_node.

STEP5[insert after]: in this method we insert element at specific location

Pass parameter prev\_node as Node type and new\_data as integer typr

[condition] prev\_node != null

Print message and return

Create object of Node class name new\_node and pass argument to it

New\_data

Set new\_node.next as prev\_node.next

Set prev\_node as new\_node

STEP6[insert at end]: in this method we insert element at last

Create object of Node class name new\_node

[condition] head == null

Assign head as new\_node and return

Assign new\_node.next as null

Create variable name last of type Node and set as head

[while loop] run until last.next != null

Assign last as last.next

And the last.next becomes the new\_node

STEP7[delete]: in this method we delete element from linked list

[condition] head==null

Return

Create variable name temp of type Node and set as head

[condition] position == 0

Set head as temp.next and return

Now find the key to be deleted using for loop

Set i as iterating variable

Test counter is tem != null && i<position-1

Set temp as temp.next and return

If the key is not present

[condition] temp==null || temp.next == null

Return

Remove the node

Set node next as temp.next.next

Set tem.next as next

STEP8[search method]: In this method we search for the element by providing its key as

argument

create Node as current and set it as head

[while loop] run until current != null

[condition] current.data==key

Print message and return true

Set current as current.next

Otherwise print message and return false

STEP9[sort link list]: initialize current of Node type as head and index of Node type as null

Temp of integer data type

[condition] head==null

Return

Otherwise run while loop until current != null

Set index as current.next

Run another while loop until index != null

[condition] current.data > index.data

Set temp as current.data

Set current.data as index.data

Set index.data as temp

Set index as index.next

Set current as current.next

STEP11[print method]: initialize node tnode as head

[Condition] if head is null

Print message

Otherwise run while loop of until tnode != null

Assign current as tnode.next

STEP12[drive code]: create object of program5 name as list

Now start calling methods

Progam4

STEP1[class]: create node class.

STEP2[initialize] initialize data of integer type head as node and next as node.

STEP3[constructor]: assign value to data.

STEP4[add method]: give one parameter to add method

Create object of node class name new\_node.

[Condition] if head is null

Assign head as new\_node

Otherwise create last as node assign it as head

[While loop] last.next!=null

Set last as last.next

Set last.next as new\_node

STEP5[print method]: initialize node current as head

[Condition] if head is null

Print message

Otherwise run while loop of until current != null

Assign current as current.next

STEP6[count method]: initialize node temp as head and count of integer type

Set count as count:=0

[while] temp!=null

[increment] count += 1

Assign temp as temp.next

Return count

STEP7[ToArray method] this method is to store all elements of linked list into an array

Create empty array of integer name Array

Specify length of Array by calling count method

Initialize node currents as head

Initialize index of integer type and set its value to index:=0

[while loop] run loop until current !=null

Start adding element to Array by incrementing index++

and start assigning data of currents to Array

Assign currents as currents.next

Print values of Array using for loop

Initialize i as iterating variable and set its value I:=0

Run loop utill i<Array size

Start printing Array element

STEP8[drive code]: create object of program4 name as list

Now start calling methods

Program5

STEP1[class]: create node class.

STEP2[initialize] initialize data of integer type head as node and next as node.

STEP3[constructor]: assign value to data.

STEP4[add method]: give one parameter to add method

Create object of node class name new\_node.

[Condition] if head is null

Assign head as new\_node

Otherwise create last as node assign it as head

[While loop] last.next!=null

Set last as last.next

Set last.next as new\_node

STEP5[print method]: initialize node current as head

[Condition] if head is null

Print message

Otherwise run while loop of until current != null

Assign current as current.next

STEP6[clear method]: This method is used to clear all elements in the linked list

Assign head as null

STEP7[drive code]: create object of program5 name as list

Now start calling methods

Program7

STEP1[class]: create node class.

STEP2[initialize] initialize data of integer type head as node and next as node.

STEP3[constructor]: assign value to data.

STEP4[add method]: give one parameter to add method

Create object of node class name new\_node.

[Condition] if head is null

Assign head as new\_node

Otherwise create last as node assign it as head

[While loop] last.next!=null

Set last as last.next

Set last.next as new\_node

STEP5[insert at first]: in this method we insert element at the first position of linkedlist

Create object of Node class name new\_node

Set new\_node.next as head and Set head as new\_node.

STEP6[insert]: in this method we insert element at any specific position

In this method three parameters were given two of Node type name

Prev\_node and after\_node and one of integer type data

[condition] if head is null

Print message and return

Create object of Node class name new\_node

Set prev\_node.next as new\_node and new\_node.next as after\_node

STEP7[delete]: this method is used to delete element from the linked list by key

First store head node as del

[Case1] If head node itself holds the key to be deleted

[condition] del != null && del.data == key

Then set head of list as del.next and return

[Case2] If the key is somewhere other than at head

Search for the key to be deleted using while loop

keep track of the previous node

as it is needed to change del.next

[condition] del != null && del != key

If del does not hold key

continue to next node

[Case3] The key is not present

If key was not present in linked list

del should be null

STEP8[count method]: initialize node temp as head and count of integer type

Set count as count:=0

[while] temp!=null

[increment] count += 1

Assign temp as temp.next

Return count

STEP9[search method]: In this method we search for the elment by providing its key as

argument

create Node as serc and set it as head

[while loop] run until serc != null

[condition] serc.data==key

Print message and return true

Set serc as serc.next

Otherwise print message and return false

STEP10[drive code]: create object of program5 name as list

Now start calling methods

Circular linked list

STEP1[class]: create node class.

STEP2[initialize] initialize data of integer type head as node and next as node.

STEP3[constructor]: assign value to data.

STEP4[initialize]: Set head and tail as null.

STEP4[add method]: give one parameter to add method

Create object of node class name new\_node.

[Condition] if head is null

Assign head and tail as new\_node

Set new\_node.next as head

Otherwise set tail.next as new\_node

Set tail as new\_node

Set the element after tail as head

STEP5[print method]: initialize node current as head

[Condition] if head is null

Print message

Otherwise run do while loop

Print message

Assign current as current.next

[condition] run loop until current != head

STEP6[insert method]: in this method we insert element at any specific position

In this method three parameters were given two of Node type name

Prev\_node and after\_node and one of integer type data

[condition] if head is null

Print message and return

Create object of Node class name new\_node

Set prev\_node.next as new\_node and new\_node.next as after\_node

STEP7[delete]: this method is used to delete element from the linked list by key

First store head node as del

[Case1] If head node itself holds the key to be deleted

[condition] del != null && del.data == key

Then set head of list as del.next and and del.next as head

return

[Case2] If the key is somewhere other than at head

Search for the key to be deleted using while loop

keep track of the previous node

as it is needed to change del.next

[condition] del != null && del != key

If del does not hold key

continue to next node

[Case3] The key is not present

If key was not present in linked list

del should be null

Set prev.next as del.next

STEP8[search method]: In this method we search for the elment by providing its key as

argument

create Node as serc and set it as head

[while loop] run until serc != null

[condition] serc.data==key

Print message and return true

Set serc as serc.next

Otherwise print message and return false

STEP9[drive code]: create object of program5 name as list

Now start calling methods

Doubly linked list

STEP1[class]: create node class.

STEP2[initialize] initialize data of integer type previous as node and next as node.

STEP3[constructor]: assign value to data.

STEP4[initialize]: Set head and tail as null.

STEP5[add method]: give one parameter to add method

Create object of node class name new\_node.

[Condition] if head is null

Assign head and tail as new\_node

Set head.previous as null and tail.next as null

Otherwise set tail.next as new\_node

Set new\_node.previous as tail

Set tail as new\_node

Set the element after tail as null

STEP6[print method]: initialize node current as head

[Condition] if head is null

Print message

Otherwise run do while loop

Print message

Assign current as current.next

[condition] run loop until current != head

STEP7[insert method]: in this method we insert element at any specific position

In this method three parameters were given two of Node type name

Prev\_node and after\_node and one of integer type data

[condition] if prev\_node is null

Print message and return

Create object of Node class name new\_node

Set prev\_node.next as new\_node and new\_node.previous as

Prev\_node and the node which is next to new\_node is after\_node

And the node before after\_node is the previous node

STEP8[delete]: this method is used to delete element from the linked list by key

First store head node as del

[Case1] If head node itself holds the key to be deleted

[condition] del != null && del.data == key

Then set head of list as del.next and and del.next as head

return

[Case2] If the key is somewhere other than at head

Search for the key to be deleted using while loop

keep track of the previous node

as it is needed to change del.next

[condition] del != null && del != key

If del does not hold key

continue to next node

[Case3] The key is not present

If key was not present in linked list

del should be null

Set prev.next as del.next

Set del.previous as prev

STEP9[search method]: In this method we search for the elment by providing its key as

argument

create Node as serc and set it as head

[while loop] run until serc != null

[condition] serc.data==key

Print message and return true

Set serc as serc.next

Otherwise print message and return false

STEP10[drive code]: create object of program5 name as list

Now start calling methods

**Program 6**

**Linked List Traversal**

The idea here is to step through the list from beginning to end. ***For example***, we may want to print the list or search for a specific node in the list.

The algorithm for traversing a list

* Start with the head of the list. Access the content of the head node if it is not null.
* Then go to the next node(if exists) and access the node information
* Continue until no more nodes (that is, you have reached the null node)

**Linked List node Insertion**

There can be three cases that will occur when we are inserting a node in a linked list.

* Insertion at the beginning
* Insertion at the end. (Append)
* Insertion after a given node

**Insertion at the beginning**

Since wo don’t have to find the end of the list for this method if the list is empty make the new node as the head of the node and if list is not empty place the new node to the position of head and node which is present at head becomes the second node

**Insertion at end**

We will traverse the list until we reach to the last node and then place the new node at that position. In case of a list being empty, we will return the updated head of the linked list because in this case, the inserted node is the first as well as the last node of the linked list.

**Insertion after a given node**

We are given the reference to a node, and the new node is inserted after the given node.

**Linked List node Deletion**

To delete a node from a linked list, we need to do these steps

* Find the previous node of the node to be deleted.
* Change the next pointer of the previous node to node which is present just after the node to be deleted
* Free the memory of the deleted node.

In the deletion, there is a special case in which the first node is deleted. In this, we need to update the head of the linked list.

**Linked List node Searching**

To search any value in the linked list, we can traverse the linked list and compares the value present in the node.

Program 8

1. Circular list are help full in those application when you want to go around the list several times its algorithm is that whenever you are ending your linked list connect the last node with the head the node
2. Circular Linked List can be used to implement a queue. We do not need to maintain the head and rear pointer if we use a circular linked list. We can store a pointer to the last node of the list, and the head can always be obtained as the next of the list.
3. In the browser when we want to use the back or next function to change the tab, it used the concept of a doubly-linked list here. Its algorithm is kind a like that we connect the node in both direction when we are assigning it. It has two types previous and next
4. To prevent the collision between the data in the **hash map**, we use a singly linked list. Its algorithm is that we connect nodes in a single direction and the last node contains the null value.
5. Doubly linked list help in redo and undo functioality