Practical No : 07 (C)

Title: Write C++ program to maintain club member‘s information using singly linked list.

Aim: The Department of Computer Engineering has a student's 1/8 club named 'Pinnacle Club'. Students of Second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of a club. First node is reserved for the president of the club and the last node is reserved for the secretary of the club. Write C++ program to maintain club member‘s information using singly linked lists. Store student PRN and Name. Write functions to

a) Add and delete the members as well as president or even secretary.
b)Compute total number of members of club
c) Display members
d) Display list in reverse order using recursion
e) Two linked lists exists for two divisions. Concatenate two lists

Input: Individual details

Output: Maintain information of the Club member's

Objectives: To maintain club member's information by performing different operations like add, delete, count, concatenate on singly linked list.

Theory:

Linked list :

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers. In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

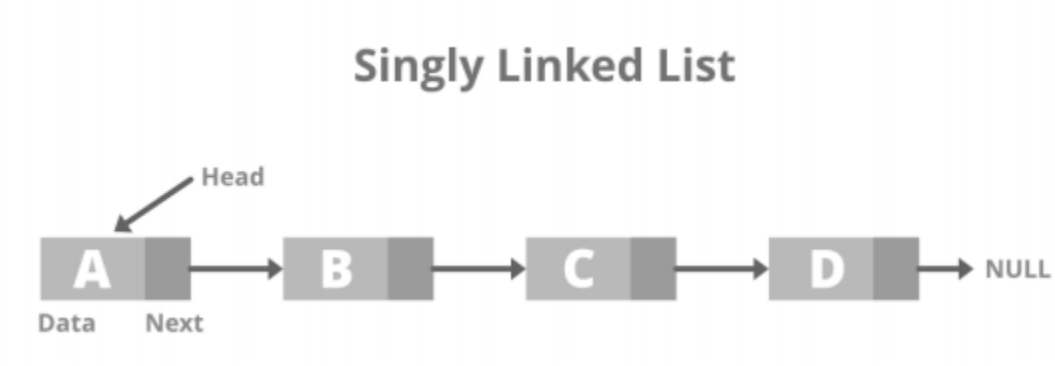
Types of linked lists:

- Singly linked lists
- Doubly linked lists
- Circular linked lists
- Circular doubly linked lists

Singly Linked list:

It 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. The node contains a pointer to the

next node means that the node stores the address of the next node in the sequence. A single linked list allows traversal of data only in one way. Below is the diagram for the same



structure of Singly Linked List:

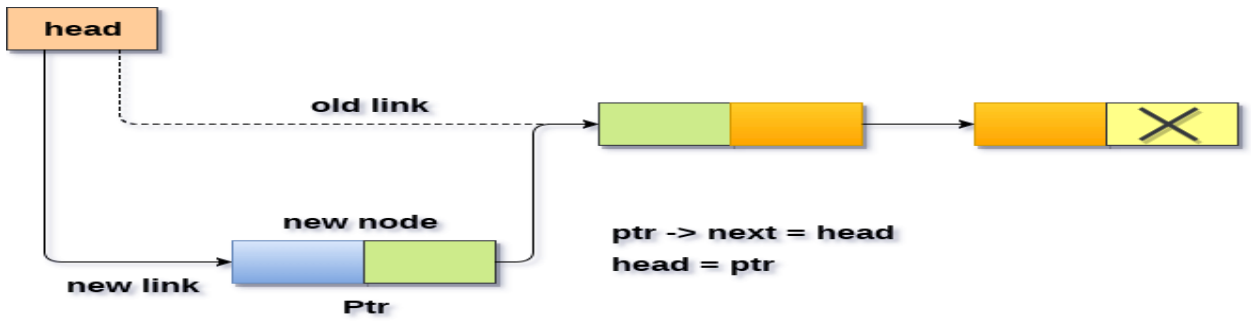
```
Struct node {
    int data;
  struct node *next;
}
```

Operation on Singly Linked list:

1.Insertion

The insertion into a singly linked list can be performed at different positions. Based on the position of the new node being inserted, the insertion is categorized into the following categories.

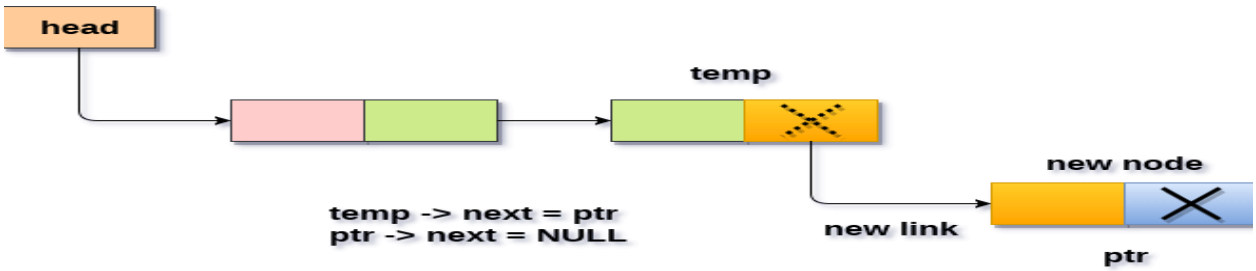
Insertion at beginning:



Algorithm

- Step 1: IF PTR = NULL
- Write OVERFLOW
  Go to Step 7
  [END OF IF]Step 2: SET NEW_NODE = PTR
- Step 3: SET PTR = PTR → NEXT
- Step 4: SET NEW_NODE → DATA = VAL
- Step 5: SET NEW_NODE → NEXT = HEAD
- Step 6: SET HEAD = NEW_NODE
- Step 7: EXIT

Insertion at end:

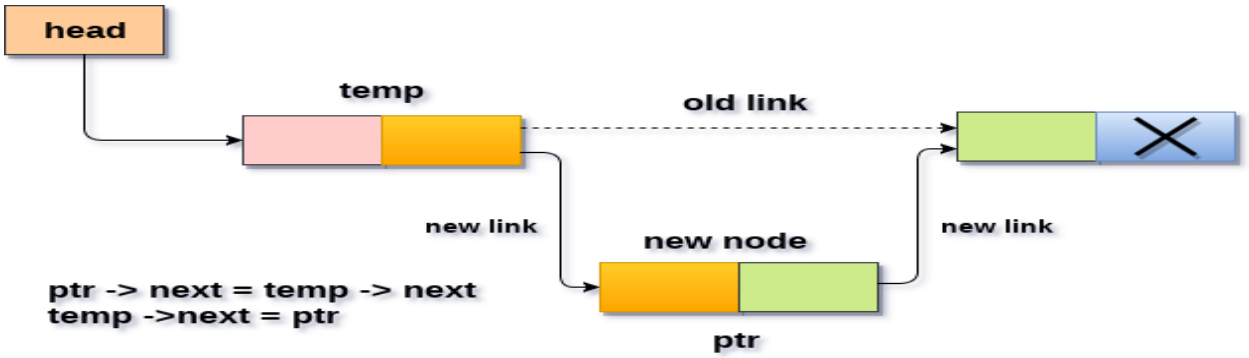


Algorithm

- Step 1: IF PTR = NULL Write OVERFLOW
  Go to Step 1
  [END OF IF]
- Step 2: SET NEW_NODE = PTR
- Step 3: SET PTR = PTR - > NEXT
- Step 4: SET NEW_NODE - > DATA = VAL
- Step 5: SET NEW_NODE - > NEXT = NULL
- Step 6: SET PTR = HEAD
- 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

Insertion after specified node:



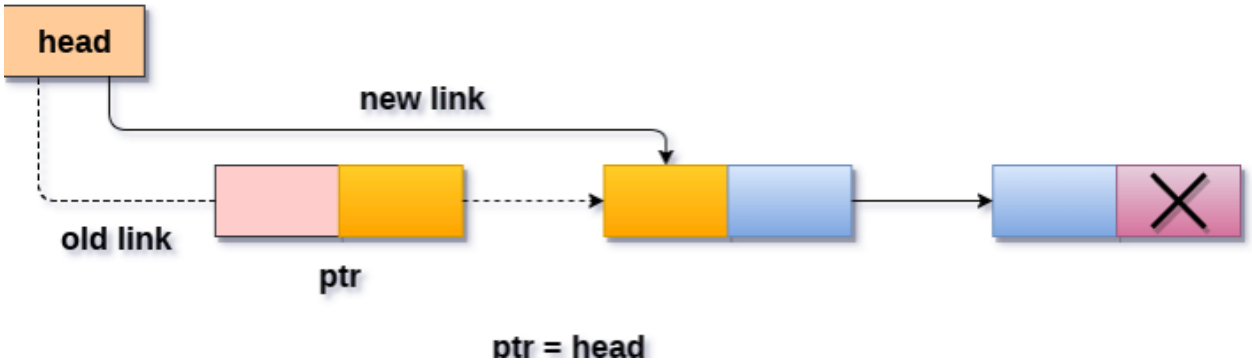
Algorithm

- STEP 1: IF PTR = NULL
- WRITE OVERFLOW
  GOTO STEP 12
  END OF IF STEP 2: SET NEW_NODE = PTR
- STEP 3: NEW_NODE → DATA = VAL
- STEP 4: SET TEMP = HEAD
- STEP 5: SET I = 0
- STEP 6: REPEAT STEP 5 AND 6 UNTIL I
- STEP 7: TEMP = TEMP → NEXT
- STEP 8: IF TEMP = NULL
- WRITE "DESIRED NODE NOT PRESENT"
  GOTO STEP 12
  END OF IF
  END OF LOOP STEP 9: PTR → NEXT = TEMP → NEXT
- STEP 10: TEMP → NEXT = PTR
- STEP 11: SET PTR = NEW_NODE
- STEP 12: EXIT

Deletion:

The Deletion of a node from a singly linked list can be performed at different positions. Based on the position of the node being deleted, the operation is categorized into the following categories.

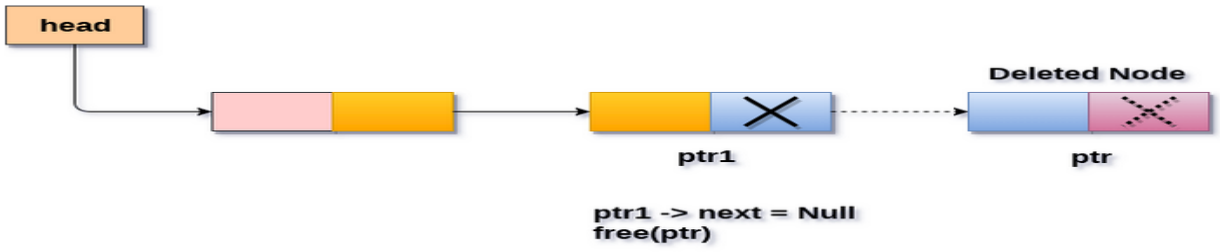
Deletion at beginning:



Algorithm:

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

Deletion at the end of the list



Algorithm:

- Step 1: IF HEAD = NULL
- Write UNDERFLOW
  Go to Step 8
  [END OF IF]Step 2: SET PTR = HEAD

- 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

Traversing:

In traversing, we simply visit each node of the list at least once in order to perform some specific operation on it, for example, printing data part of each node present in the list.

Algorithm

- **STEP 1:** SET PTR = HEAD
- **STEP 2:** IF PTR = NULL
- WRITE "EMPTY LIST"
  GOTO STEP 7
  END OF IF **STEP 4:** REPEAT STEP 5 AND 6 UNTIL PTR != NULL
- **STEP 5:** PRINT PTR→ DATA
- **STEP 6:** PTR = PTR → NEXT
- [END OF LOOP]**STEP 7:** EXIT

Count:

In count function we simple visit each node and count he total members present in linked list

Algorithm:

- **STEP 1:**initialize count as 0
- **STEP 2:**Initialize a node pointer, current = head.
- **STEP 3:** Do following while current is not NULL
  - current = current -> next
  - Increment count by 1.
- **STEP4:**Return count

Concatenate:

In concatenate function we simple append second linkedlist to first linked list

Algorithm:

- **STEP 1:**create linkedlist1 by calling insert()
- **STEP 2:**create linkedlist1 by calling insert()
- **STEP 3:** create temp pointer and assign it to head1
- **STEP4:**traverse temp until temp->next!=NULL
- **STEP5:**else temp->next=head2

Conclusion:

By this way, we can maintain club member‘s information using singly linked list.