**Aim:**

Write a program to implement a Singly linked list.

**Algorithm:**

Step 1 - Start.

Step 2 - Declare functions, empty\_message, memory\_message, getData, getPosition, create\_Node, insert\_at\_beginning, insert\_at\_end, insert\_at\_position, delete\_at\_beginning, delete\_at\_end, delete\_at\_position and display.

Step 3 - Define a structure 'Node' with two members, 'data' and 'next'.

Step 4 - Define a Node pointer 'head' and set it to NULL.

Step 5 – Display the list of operations, and get the choices from the user.

Step 6 – While the choice is less than or equal to 8, continue. Else, go to step 62

Step 7 – If the choice is 1, call getData and insert\_at\_beginning functions. Then go to step 6.

Step 8 – if choice is 2, call getData and insert\_at\_end functions. Then go to step 6.

Step 9 - if choice is 3, call getData, getPosition, and insert\_at\_position functions. Then go to step 6.

Step 10 – if the choice is 4, call the delete\_at\_beginning function. Then go to step 6.

Step 11 – if choice is 5, call delete\_at\_end function. Then go to step 6.

Step 12 – If choice is 6, call getPosition and delete\_at\_position functions. Then go to step 6.

Step 13 – if the choice is 7, call the display function. Then go to step 6.

Step 14 – if the choice is 8, go to step 6.

Step 15 – Under the empty\_message function, display "List is Empty!".

Step 16 – Under the memory\_message function, display "Memory can't be allocated!".

Step 17 – Under the getData function, get data from the user and return it.

Step 18 – Under the getPosition function, get the position from the user and return it.

Step 19 – Under the create\_Node function, get data from the user and dynamically allocate the Node pointer 'new\_Node'.

Step 20 – if new\_Node is NULL, call the memory\_message function and return NULL. Else continue.

Step 21 – Set new\_Node's 'data' as data and new\_Node's 'next' as NULL.

Step 22 – Return new\_Node.

Step 23 – Under the insert\_at\_beginning function, get data from the user.

Step 24 - Define the Node pointer 'new\_Node' and set it as NULL.

Step 25 – Call the create\_Node function and store the return value in new\_Node.

Step 26 - If new\_Node is not NULL, Set new\_Node's 'next' as head and set head as 'new\_Node'. Else, call the memory\_message function.

Step 27 – Under the insert\_at\_end function, get data from the user.

Step 28 – Do Step 24 and 25.

Step 29 – If new\_Node is not NULL, continue. Else, call the memory\_message function.

Step 30 – If the head is NULL, then set head as new\_Node. Else continue.

Step 31 – Define the Node pointer' last' and set it as the head.

Step 32 – While last's 'next' is not NULL, set last as last's next.

Step 33 – Set last's 'next' as new\_Node.

Step 34 – Under insert\_at\_position, get data and position from the user.

Step 35 – Do step 25.

Step 36 - if the list is empty and the position is greater than 1 or less than 0, display "Invalid position!". Else continue.

Step 37 – Do step 29

Step 38 – define Node pointer 'temp' and set it as the head.

Step 39 – Initialize count = 1.

Step 40 – While the count is less than position – 1, set temp as temp's next and increment count by 1.

Step 41 – if the position is 1, Set new\_Node's 'next' as head and head as new\_Node.

Else, continue.

Step 42 - set new\_Node's 'next' as temp's 'next' and temp's 'next' as new\_Node.

Step 43 – under delete\_at\_beginning, if the head is NULL, call the empty\_message function. Else, continue.

Step 44 – define Node pointer 'temp' and set it as the head.

Step 45 - Initialize data and set it as the head's 'data'.

Step 46 – set head as head's 'next' and free temp.

Step 47 – under the delete\_at\_end function, if head is NULL, Call empty\_message function. Else continue.

Step 48 – define Node pointer temp and prev.

Step 49 – set temp as head and prev as NULL.

Step 50 – while temp's 'next' is not NULL, set prev as temp and temp as temp's 'next'.

Step 51 – Set data as temp's 'data'

Step 52 – if the temp is head, free temp and set head as NULL. Else, free temp and set prev's 'next' as NULL.

Step 53 – under the delete\_at\_position, get the position from the user.

Step 54 – Do step 36.

Step 55 – Do step 48 and 49.

Step 56 – Do Step 39.

Step 57 – while count less than position, set prev as a temp, temp as temp's 'next' and increment count by 1.

Step 58 – initialize data as temp's 'data'.

Step 59 – if the temp is head, set head as head's 'next' and free temp. Else, continue.

Step 60 – set prev's 'next' as temp's 'next' and free temp.

Step 61 - If the head is NULL, call the empty\_message function under the display function. Else, continue.

Step 62 - define Node pointer temp and set it as the head.

Step 63 – while temp is not NULL, display temp's 'data'.

Step 64 – set temp as temp's 'next'.

Step 65 – Stop.

**Result:**

The program was successfully executed and achieved the aim of the program.

**Output:**





