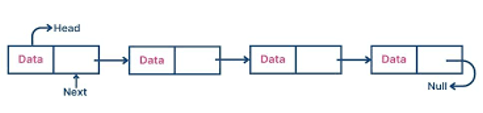
**Single Linked List**

A Single Linked List (SLL) is a linear data structure made up of nodes, where each node contains:

**Data** – the value or information.

**Pointer** (Next) – a reference (link) to the next node in the list.



**Structure of a Node**

|  |
| --- |
| struct Node {  int data; // holds the value  struct Node\* next; // points to the next node  }; |

**How a Singly Linked List Works:**

* Starts with a **head pointer** (points to the first node).
* Each node points to the **next node**.
* The **last node’s next is NULL**, indicating the end.

**Operations on Singly Linked List**

**Insertion**

At beginning → Change head to new node.

At end → Traverse till last and link new node.

At position → Traverse to desired position, insert node.

**Deletion**

At beginning → Move head to next node.

At end → Traverse to second-last and set its next to NULL.

At position → Traverse to previous node and skip one node.

**Traversal**

Start from head and keep visiting node->next till NULL.

**Search**

Traverse each node and compare node->data with target.

**Declare the head pointer and initialize it:**

**struct Node\* head = NULL; // Initially, the list is empty**

**Insert at Beginning**

We can use the following steps to insert a new node at beginning of the single linked list...

**Step 1** - Create a newNode with given value.

**Step 2** - Check whether list is Empty (head == NULL)

**Step 3** - If it is Empty then, set **newNode→next = NULL** and **head** = **newNode**.

**Step 4** - If it is Not Empty then, set **newNode→next** = **head** and **head** = **newNode**.

|  |
| --- |
| void insert\_begin (int item)  {  struct Node \*nn = (struct Node \*) malloc (sizeof(struct Node));  nn->data = item;    if(head==NULL)  {  nn->next = NULL;  }  else  {  nn->next = head;  }  head = nn;    printf("\n\n Element inserrted successfully at begin");  n++; // one element is inserted so .. increment n value by 1    } |

**Insert at End**

We can use the following steps to insert a new node at end of the single linked list...

* **Step 1 -**Create a **newNode** with given value and **newNode → next** as **NULL**.
* **Step 2 -**Check whether list is **Empty** (**head** == **NULL**).
* **Step 3 -**If it is **Empty** then, set **head** = **newNode**.
* **Step 4 -**If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
* **Step 5 -**Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).
* **Step 6 -**Set **temp → next** = **newNode**.

|  |
| --- |
| void insert\_end(int item)  {  struct Node \*nn = (struct Node \*)malloc(sizeof(struct Node));  nn->data = item;  nn->next = NULL;    if(head==NULL)  {  head=nn;  }  else  {  struct Node \*temp=head;  while(temp->next!=NULL)  {  temp = temp->next;  }  temp->next= nn;  }    printf("\n element inserrted successfully at end ");  n++; // item inserted so increment n value;    } |

**Insert at Given Position**

We can use the following steps to insert a new node at a specific position in the singly linked list:

* **Step 1:** Create a new node (newNode) and assign the given value to its data field.
* **Step 2:** Check whether the list is empty (head == NULL).
* **Step 3:**

If it is **empty**, display a message like "List is empty. Inserting at the beginning".

Set newNode → next = NULL and head = newNode.

* **Step 4:**

If the list is **not empty**, check whether the given position is **greater than the current node count (n)**.

If true, display "Invalid position to insert" and exit the function.

* **Step 5:**

Define a temporary node pointer (temp) and initialize it with head.

* **Step 6:**

Move temp to its next node repeatedly using a loop that runs from i = 1 to i < pos - 1., This brings temp to the node **just before the given position**.

* **Step 7:**

Set newNode → next = temp → next

* **Step 8:**

Set temp → next = newNode

* **Step 9:**

Increment the global node count (n++)

* **Step 10:**

Display "Inserted element at given position" to confirm the operation.

|  |
| --- |
| void insert\_givenpos(int item,int pos)  {  struct Node \*nn = (struct Node \*)malloc(sizeof(struct Node));  nn->data = item;  if(head==NULL)  {  printf("List is empty. i have inserted at begin");  nn->next = NULL;  head = nn;  }  else  {  if(pos>n)  {  printf("\n Invalid position to insert. ");  }  else  {  struct Node \*temp = head;  for(int i=1;i<pos-1;i++)  {  temp = temp->next;  }  nn->next = temp->next;  temp->next =nn;    n++;  printf("\n Inserted element at given position");  }  }  } |

**Deletion at begin**

We can use the following steps to delete a node from beginning of the single linked list...

* **Step 1 -**Check whether list is **Empty** (**head** == **NULL**)
* **Step 2 -**If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* **Step 3 -**If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
* **Step 4 -**Check whether list is having only one node (**temp → next** == **NULL**)
* **Step 5 -**If it is **TRUE** then set **head** = **NULL** and delete **temp** (Setting **Empty** list conditions)
* **Step 6 -**If it is **FALSE** then set **head** = **temp → next**, and delete **temp**.

|  |
| --- |
| void deleteAtBegin()  {  if(head==NULL)  {  printf("\n List is Empty- Deletion is not possible");  }  else  {  printf("\n deleted at element %d",head->data);  head=head->next;  }  n--;  printf("\n Element deleted successfully");  } |

**Delete at end**

We can use the following steps to delete a node from end of the single linked list...

* **Step 1 -**Check whether list is **Empty** (**head** == **NULL**)
* **Step 2 -**If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* **Step 3 -**If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2'** and initialize '**temp1**' with **head**.
* **Step 4 -**Check whether list has only one Node (**temp1 → next** == **NULL**)
* **Step 5 -**If it is **TRUE**. Then, set **head** = **NULL** and delete **temp1**. And terminate the function. (Setting **Empty** list condition)
* **Step 6 -**If it is **FALSE**. Then, set '**temp2 = temp1**' and move **temp1** to its next node. Repeat the same until it reaches to the last node in the list. (until **temp1 → next** == **NULL**)
* **Step 7 -**Finally, Set **temp2 → next**= **NULL** and delete **temp1**.

|  |
| --- |
| void deleteAtEnd()  {    if(head==NULL)  {  printf("List is empty- deletion is not possible");  }  else  {  struct Node \*t1 =head;  struct Node \*t2;    while(t1->next!=NULL)  {  t2=t1;  t1 = t1->next;  }  printf("\n deleteing element is %d",t1->data);  t2->next = NULL;  }  n--;  printf("\n Deleted successfully");  } |

**Delete at given Position**

We can use the following steps to delete a node at a specific position in a singly linked list:

1. **Check if the list is empty** (head == NULL). If true, display a message saying "List is Empty - Deletion not possible" and exit the function.
2. **Check if the given position is greater than the total number of nodes** (pos > n). If true, display a message saying "It exceeds the length of SLL - no deletion" and exit the function.
3. **Define two pointer variables**:
   * t1, initialized to head.
   * t2, which will track the node just before t1.
4. **Traverse the list** to reach the node at the given position:
   * Start from t1 = head.
   * Move t1 and t2 to the next node (t1 = t1->next and t2 = t1) for each iteration until t1 points to the node to be deleted.
5. **After reaching the position**, t1 points to the node to be deleted and t2 points to the node just before it.
6. **Print the deleted element's data**: Display the message "Deleted element is [value]".
7. **Remove the node** by setting t2->next = t1->next to bypass the node pointed to by t1.
8. **Decrement the node count** by reducing n by 1 (n--).

|  |
| --- |
| void deleteAtGivenPosition(int pos)  {  if(head == NULL)  {  printf("List is Empty - Deletion not possible");  }  else  {  if(pos>n)  {  printf("it exceeds the length of SLL-no deletion");  }  else  {  struct Node \*t1=head;  struct Node \*t2;    for(int i=1;i<pos;i++)  {  t2 = t1;  t1=t1->next;  }  printf("Deleted element is %d",t1->data);  t2->next = t1->next;  n--;  }  }    } |

**Displaying elements in SLL**

We can use the following steps to display the elements of a single linked list...

* **Step 1 -**Check whether list is **Empty** (**head** == **NULL**)
* **Step 2 -**If it is **Empty** then, display **'List is Empty!!!'** and terminate the function.
* **Step 3 -**If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
* **Step 4 -**Keep displaying **temp → data** with an arrow (**--->**) until **temp** reaches to the last node
* **Step 5 -**Finally display **temp → data** with arrow pointing to **NULL** (**temp → data ---> NULL**).

|  |
| --- |
| void display() // traversing  {  if(head==NULL)  {  printf("\n\n List is empty");  }  else  {  struct Node \*temp=head;    while(temp!=NULL)  {    printf("%3d",temp->data);  temp=temp->next;  }  }  } |

Program : <https://onlinegdb.com/qM-EujvSS>