Creation of nodes:

Pseudocode:

Class named struct node

{

Make a variable named of type data of type int

Make a node type variable/pointer named \*next

};

Explanation:

As linked list consists of nodes, we need to declare a structure which defines a single node. Our structure should have at least one variable for data section and a pointer for the next node

Make a class named list

{

Private:

Create node type two variables \*head, \*tail

public:

calling the default constructor i.e list()

{

Put head=NULL;

Put tail=NULL;

}

};

Explanation:

Now, we need a class which will contain the functions to handle the nodes. This class should have two important pointers, i.e. head and tail. The constructer will make them NULL to avoid any garbage value.

Put value in temp->data

Set temp-> next Null

If head is equal to null

{

Put head = temp

Put tail = temp

Put temp = null

}

Else

{

set tail-> next = temp

now put tail=temp

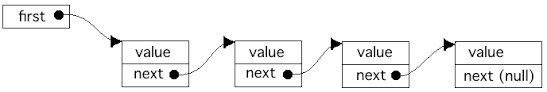
}

Explanation:

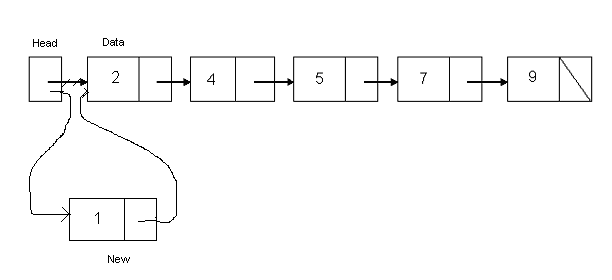
We need a pointer of a node type (which we defined) and we will insert the value in its data field. The next field of node would be declared as NULL as it would be the last node of linked list.

The creation of a new node at the end of linked list has 2 steps:

* Linking the newly created node with tail node. Means passing the address of a new node to the next pointer of a tail node.
* The tail pointer should always point to the last node. So we will make our tail pointer equal to a new node.



Insert in the beginning:



Pseudocode:

here temp is a new cell containing data 1

set temp -> data = 1

put temp -> next = head

now put head = temp

Explanation:

Insertion of a new node is quite simple. It is just a 2-step algorithm which is performed to insert a node at the start of a singly linked list.

New node should be connected to the first node, which means the head. This can be achieved by putting the address of the head in the next field of the new node.

New node should be considered as a head. It can be achieved by declaring head equals to a new node.

Insert in the end:

|  |
| --- |
| [https://1.bp.blogspot.com/-RqW5hhS4Gxw/VKr9Q50_W4I/AAAAAAAAAi0/oihIQcd1tsA/s1600/PSEUDOCODE%2BOF%2BINSERT%2BAT%2BEND%2B-.JPG](http://1.bp.blogspot.com/-RqW5hhS4Gxw/VKr9Q50_W4I/AAAAAAAAAi0/oihIQcd1tsA/s1600/PSEUDOCODE%2BOF%2BINSERT%2BAT%2BEND%2B-.JPG) |

here m is new cell containing data 9

|  |
| --- |
| [https://2.bp.blogspot.com/-01_YGzCfgIs/VKr9O0XdG_I/AAAAAAAAAiQ/4g6AdW8-TyA/s1600/Insert%2BAt%2BEnd-2%2B-.JPG](http://2.bp.blogspot.com/-01_YGzCfgIs/VKr9O0XdG_I/AAAAAAAAAiQ/4g6AdW8-TyA/s1600/Insert%2BAt%2BEnd-2%2B-.JPG) |

Pseudocode:

Loop until head-> next is not equal to null

set head = head-> next

put temp-> next = NULL

set temp->data = value

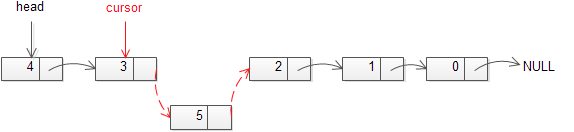
head->next = temp

Explanation:

The insertion of a node at the end of a linked list is the same as we have done in node creation function. If you noticed then, we inserted the newly created node at the end of the linked list. So this process is the same.

Insert at any position:

Pseudocode:



|  |
| --- |
|  |
|  |
|  |

Set cursor node = head

Loop from 1 to pos-1

{

Set previous node = cursor node

Set cursor=cursor->next

}

Put the data in the temp node which is to be inserted

Set previous node->next=temp node

Set node temp ->next=cursor node

Explanation:

In this case, we don’t disturb the head and tail nodes. Rather, a new node is inserted between two consecutive nodes. We call one node as current and the other as previous, and the new node is placed between them.

Now the new node can be inserted between the previous and current node by just performing two steps:

* Pass the address of the new node in the next field of the previous node.
* Pass the address of the current node in the next field of the new node.

We will access these nodes by asking the user at what position he wants to insert the new node. Now, we will start a loop to reach those specific nodes. We initialized our current node by the head and move through the linked list. At the end, we would find two consecutive nodes.

Delete from the beginning:

|  |
| --- |
| [https://2.bp.blogspot.com/-XOzifldnxIs/VKr9Lp9mC2I/AAAAAAAAAhg/Nhl3GrleTXU/s1600/Delete%2Bfrom%2BBegaining%2B-.JPG](http://2.bp.blogspot.com/-XOzifldnxIs/VKr9Lp9mC2I/AAAAAAAAAhg/Nhl3GrleTXU/s1600/Delete%2Bfrom%2BBegaining%2B-.JPG) |

Pseudocode:

Set temp = head

Put head = head->next

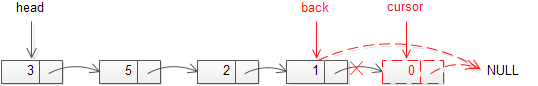
Delete the temp

Explanation:

In this case, the first node of the linked list is deleted. The first node is called the head. So, we are going to delete the head node. The process of deletion includes:

* Declare a temp pointer and pass the address of the first node, i.e. head to this pointer.
* Declare the second node of the list as head as it will be the first node of linked list after deletion.
* Delete the temp node.

Delete from End:



Pseudocode:

Set current node equal to head

Loop until current-> is not equal to null

{

Set previous node = current node;

Set node current = current->next;

}

set tail equal to previous node

set previous-> next equals null

delete current node

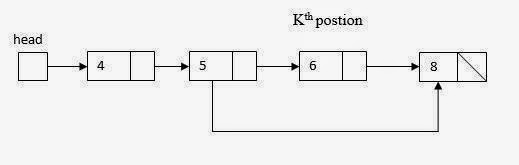
Explanation:

You need to access the last node and also need access to the second to the last node of the linked list as you will delete the last node and make the previous node as the tail of linked list.

This can be achieved by traversing the linked list. We would make two temporary pointers and let them move through the whole linked list. At the end, the previous node will point to the second to the last node and the current node will point to the last node, i.e. node to be deleted. We would delete this node and make the previous node as the tail.

Delete at any position:

Pseudocode:

**[](http://4.bp.blogspot.com/-3mSWaMdJXHw/VKr9Mhk-ErI/AAAAAAAAAh8/XF9U6lw77js/s1600/Delete%2Bfrom%2BKth%2Bposition-1%2B-.JPG)**

Loop from 1 to pos-2

put head = head->next

put temp = head->next

|  |
| --- |
| [https://1.bp.blogspot.com/-d8_owgpHkMs/VKr9M3OW0zI/AAAAAAAAAhs/6xRpnCj4tbs/s1600/Delete%2Bfrom%2BKth%2Bposition-2%2B-.JPG](http://1.bp.blogspot.com/-d8_owgpHkMs/VKr9M3OW0zI/AAAAAAAAAhs/6xRpnCj4tbs/s1600/Delete%2Bfrom%2BKth%2Bposition-2%2B-.JPG) |

head->next = temp->next

delete the temp

|  |
| --- |
| **[https://3.bp.blogspot.com/--D04EH1zBtk/VKr9R-ApJTI/AAAAAAAAAjM/-NCh2usEJy8/s1600/after%2Bdelete%2Bfrom%2BKth%2Bposition%2B-.JPG](http://3.bp.blogspot.com/--D04EH1zBtk/VKr9R-ApJTI/AAAAAAAAAjM/-NCh2usEJy8/s1600/after%2Bdelete%2Bfrom%2BKth%2Bposition%2B-.JPG)** |

Explanation:

In linked list, we can delete a specific node. The process of deletion is simple. Here we don’t use the head and tail nodes. We ask the user to input the position of the node to be deleted. After that, we just move two temporary pointers through the linked list until we reach our specific node. Now, we delete our current node and pass the address of the node after it to the previous pointer. This way, the current node is removed from the linked list and the link is established between its previous and next node.

Count the elements of the linked list:

We can use the same traversing technique to count the number of elements in a linked list. See the following count() function:

Pseudocode:

Set node \*cursor = head

make a node named cursor and equals it to head

Put a variable c = 0

Loop until cursor-> next is not equal to null

{

Increment c

Set cursor = cursor->next

}

Return value of c

Explanation:

First we make a node type pointer and put it equal to head. We also initialize a variable of int type to count. Make a loop which runs until the pointer -> next is not null and in that loop, increment the variable. At the end, return the value.

Display:

Pseudocode:

Put temp=head;

Loop until temp -> next is not equal to null

{

Print temp->data

Put temp=temp->next

}

Explanation:

Display function is very easy to implement. What we have to do is to declare a node type pointer and put it equal to head. Next we want a loop which runs till temp->next is not equal to null. In that loop we print the data present in that node and in the next step, we put the pointer equal to pointer-> next.