DSA_College\Singly_linked_list.cpp

```
/*
#include<iostream>
using namespace std;

class Node
{
    public:
    int data;
    Node *next ;
    Node(int d)
    {
        data = d;
        this->next= NULL ;
    }
    ~Node()
    {
        int value = this->data ;
        if(this->next != NULL)
        {
            delete next ;
            this->next= NULL;
        }
        cout<<"memory is free for node with data- "<<value<<endl;
    }
};

void insertAtHead(Node* &head, int d)
{
    Node* temp= new Node(d) ;
    temp->next = head ;
    head = temp ;
}

void insertAtTail(Node* &tail, int d)
{
    Node* temp= new Node(d) ;
    tail->next= temp ;
    tail= temp ;
}

int getLength(Node* &head)
{
    int len=0 ;
    Node* temp= head ;
    while(temp!= NULL)
    {
        len++ ;
        temp = temp->next ;
    }
    return len ;
}
```

```
52
53 void insertAtPosition(Node* &head, Node* &tail, int d, int pos)
54 {
55     int len= getLength(head) ;
56     if(pos<1 || pos>len)
57     {
58         cout<<"invalid position"<<endl;
59         return ;
60     }
61     if(pos== 1)  // insert at head
62     {
63         insertAtHead(head,d) ;
64         if(tail == NULL) // empty list
65         {
66             tail = head ;
67         }
68         return ;
69     }
70     if(pos == len)
71     {
72         insertAtTail(tail,d) ;
73         return ;
74     }
75     // insert in middle
76     Node* temp = new Node(d);
77     Node* prev = NULL;
78     Node* curr = head;
79     for (int i = 1; i < pos; i++)
80     {
81         prev = curr;
82         curr = curr->next;
83     }
84     prev->next = temp;
85     temp->next = curr;
86 }
87
88 void deleteNode(Node* &head, int pos)
89 {
90     if(pos== 1) //delete the head node
91     {
92         Node* temp= head ;
93         head= head->next ;
94         temp->next = NULL ;
95         delete temp ;  // destructor called
96     }
97     else{  // deleting any middle or last node
98         Node* curr= head ;
99         Node* prev= NULL ;
100         int cnt= 1;
101         while(cnt<pos)
102         {
103             prev= curr ;
104             curr= curr->next ;
105             cnt++ ;
```

```
        }
        prev->next = curr->next ;
        curr->next = NULL ;
        delete curr ;
    }
}

void print(Node* &head)
{
    Node* temp= head ;
    while(temp != NULL)
    {
        cout<<temp->data<<" ";
        temp= temp->next ;
    }
    cout<<endl;
}
void search(Node* head, int value)
{
     if (head == NULL)
     {
        cout << "The list is empty" << endl;
        return;
     }

    bool found = false;
    while (head != NULL)
    {
        if (head->data == value)
        {
            found = true;
            break; // Exit loop as soon as we find the value
        }
        head = head->next;
    }

    if (found)
    {
        cout << "Node with value " << value << " is present." << endl;
    }
    else
    {
        cout << "Node with value " << value << " is not found." << endl;
    }
}

int main()
{
  Node* node1= new Node(1) ;
  Node* head= node1 ;
  Node* tail= node1 ;
  insertAtPosition(head,tail,4,1); // insert 4 at position 1 (abhi jo head hai)
  print(head) ;

```

```
  insertAtPosition(head,tail,9,2); // insert 4 at position 2( abhi jo tail hai)
  print(head) ;

  insertAtPosition(head,tail,12,2); // insert 4 at position 1
  print(head) ;

  deleteNode(head,3) ;
  print(head) ;

  search(head,12) ;

  return 0;
}
*/









#include<iostream>
using namespace std;

class Node
{
    public:
    int data ;
    Node* next ;
    Node(int data) // constructor
    {
       this->data = data ;
       this->next = NULL ;
    }
    ~Node()
    {
        int value = this->data ;
        if(this->next != NULL)
        {
            delete next ;
            this->next = NULL ;
        }
        cout<<"memory is free for node with data= "<<value<<endl;
    }
};

void insertAtHead(Node* &head, int d)
{
    if(head == NULL) // empty linked list
    {
        Node* temp = new Node(d) ;
        head = temp ;
    }
    else{
        Node* temp = new Node(d) ;
```

```
214         temp->next = head ;
215         head = temp ;
216     }
217 }
218 
219 void insertAtTail(Node* &tail, int d)
220 {
221     if(tail == NULL)
222     {
223         Node* temp = new Node(d) ;
224         tail = temp ;
225     }
226     else{
227         Node* temp = new Node(d) ;
228         tail->next = temp ;
229         tail = temp ;
230     }
231 }
232 
233 int getLength(Node* &head)
234 {
235     Node* temp = head ;
236     int len = 0;
237     while(temp != NULL)
238     {
239         len++ ;
240         temp = temp->next ;
241     }
242     return len ;
243 }
244 
245 void print(Node* head)
246 {
247     Node* temp = head ;
248     while(temp != NULL)
249     {
250         cout<<temp->data<<" " ;
251         temp= temp->next ;
252     }
253     cout<<endl;
254 }
255 
256 void insertAtPosition(Node* &head, Node* &tail, int d, int pos)
257 {
258     if(pos == 1)
259     {
260         insertAtHead(head,d) ;
261         return ;
262     }
263     Node* temp = head ;
264     int cnt = 1 ;
265     while(cnt < pos-1)
266     {
267         temp = temp->next ;
```

```
268         cnt++ ;
269     }
270     if(temp->next == NULL)
271     {
272         insertAtTail(tail,d) ;
273         return ;
274     }
275     Node* newnode = new Node(d) ;
276     newnode->next=  temp->next   ;
277     temp->next = newnode ;
278 }
279 
280 void deleteNode(Node* &head, int pos)
281 {
282     if(pos == 1) // deleteing head node
283     {
284         Node* temp = head ;
285         head = head->next ;
286         temp->next = NULL ;
287         delete temp ; // destructor called
288     }
289     Node* curr = head ;
290     Node* prev = NULL ;
291     int cnt = 1 ;
292     while(cnt < pos)
293     {
294         prev= curr ;
295         curr= curr->next ;
296         cnt++ ;
297     }
298     prev->next = curr->next ;
299     curr->next = NULL ;
300     delete curr ;
301 }
302 
303 int main()
304 {
305     Node *node1= new Node(10) ; // node1 is object of class Node created using dynamic
    memory allocation
306 
307     //head and tail pointed to node1
308     Node* head= node1 ;
309     Node* tail= node1 ;
310 
311     print(head) ; // 10
312 
313     insertAtHead(head,12) ; // 12 10
314     print(head) ;
315 
316     insertAtTail(tail,4) ; // 12 10 4
317     print(head) ;
318 
319     insertAtPosition(head,tail,49,2) ; // 12 49 10 4
320     print(head) ;
```

```
321
322     deleteNode(head,3) ;
323     print(head) ; // 12 49 4
324
325     int length= getLength(head);
326     cout<<"length of linked list = "<<length<<endl; // 3
327
328     return 0;
329 }
```