

Linked Lists Intro

What is a linked list ?

List created by linking elements called as nodes !!

Simple Pointer

```
int main() {
    int i = 10;
    cout << i << " " << &i << endl;

    int *p = &i;
    cout << p << " " << *p << endl;
}
```

In []:

1

Static memory allocation in C++

```
struct Node {
    int data;
};

int main() {
    Node n1;
    n1.data = 10;

    cout << n1.data << endl;

    Node *p1;
    p1 = &n1;

    cout << p1 << " " << (*p1).data << " " << p1->data;
}
```

In []:

1

```
In [ ]: 1  ``C++
2  struct Node {
3      int data;
4  };
5
6  int main() {
7      Node *p1 = new Node; // dynamic memory allocation
8
9      cout << p1 << " " << p1->data;
10
11     delete p1; // deallocation
12 }
13 ``
```

```
In [ ]: 1
```

```
In [ ]: 1  struct Node {
2      int data;
3      Node *next;
4  };
5
6
7  int main() {
8      Node *p1 = new Node; // dynamic memory allocation
9      Node *p2 = new Node; // dynamic memory allocation
10     p1->data = 10;
11     p1->next = NULL;
12     p2->data = 20;
13     p2->next = NULL;
14
15     cout << p1 << " " << p1->data << " " << p1->next << endl;
16     cout << p2 << " " << p2->data << " " << p1->next << endl;
17
18     delete p1; // deallocation
19 }
```

```
In [ ]: 1
```

```

In [ ]: 1  ``
2  struct Node {
3      int data;
4      Node *next;
5  };
6
7  Node* createNode(int data) {
8      Node* p = new Node;
9      p->next = NULL;
10     p->data = data;
11     return p;
12 }
13
14 void printNode(Node* n) {
15     cout << "data:" << n->data << " ptr:" << n->next << endl;
16 }
17
18 int main() {
19     Node *n1 = createNode(10); // dynamic memory allocation
20     Node *n2 = createNode(20); // dynamic memory allocation
21
22     printNode(n1);
23     printNode(n2);
24
25 }
26 ``

```

```

In [ ]: 1

```

Node structure and self referential struct

```

// C++
struct Node {
    int data;
    Node *next;
};

// Decimal 0-9 - 10
// Hexa 0-9 ABCDEF - 16

```

Dynamic memory allocation

```
int main() {  
    Node *n1 = new Node();  
    n1->data = 10;  
    n1->next = NULL;  
  
    Node *n2 = new Node();  
    n2->data = 20;  
    n2->next = n1;  
}
```

In []:

1

JAVA

```
class Node {  
    int data;  
    Node next;  
  
    public Node(int data, Node next) {  
        this.data = data;  
        this.next = next;  
    }  
}  
  
Node n1 = new Node(10, null);  
Node n2 = new Node(20, n1);
```

Python

```
class Node:  
    def __init__(self, data: int, next: Node):  
        self.data = data  
        self.next = next  
    def __str__(self):  
        print("data", self.data, "next", self.next)  
  
n1 = Node(10, None)  
n2 = Node(20, n1)
```

In []:

1

```
In [8]: 1 class Node:
2         def __init__(self, data: int, next):
3             self.data = data
4             self.next = next
5         def __str__(self):
6             return f"(data:{self.data} next: {self.next})"
7
8         n1 = Node(10, None)
9         n1 = Node(20, n1)
10        print(n1)
11        print(n1.next)
```

```
(data:20 next: (data:10 next: None))
(data:10 next: None)
```

```
In [ ]: 1
```

Linked List fundamental

```
struct Node {
    int data;
    Node *next;
};

Node* createNode(int data, Node* next) {
    Node* p = new Node;
    p->next = next;
    p->data = data;
    return p;
}

void printNode(Node* n) {
    cout << "data:" << n->data << " next:" << n->next << endl;
}

int main() {
    Node *n1 = createNode(10, NULL); // dynamic memory allocation
    n1 = createNode(20, n1);
    n1 = createNode(30, n1);

    printNode(n1);
    printNode(n1->next);
    printNode(n1->next->next);
}
```

```
In [ ]: 1
2 Insert
3 ``C++
```

```

3  struct Node {
4      int data;
5      Node *next;
6  };
7
8  class LinkedList {
9      Node *head; // data member
10
11     Node* createNode(int data, Node* next) {
12         Node* p = new Node;
13         p->next = next;
14         p->data = data;
15         return p;
16     }
17     public:
18         LinkedList() {this->head = NULL;} // Constructor
19
20         void insertFront(int data) {
21             Node * temp = createNode(data, NULL)
22             if (head == NULL) {
23                 head = temp;
24             } else {
25                 temp->next = head;
26                 head = temp;
27             }
28         }
29
30         void print() {
31
32             cout << endl;
33             Node * temp = head;
34             if (head == NULL) cout << "list is empty" << endl;
35             while(temp != NULL) {
36                 cout << "data:" << temp->data << " next:" << temp->next
37                 << endl;
38                 temp = temp->next;
39             }
40         };
41
42     int main() {
43         LinkedList l1;
44         l1.print();
45
46         l1.insertFront(10);
47         l1.print();
48
49         l1.insertFront(20);
50         l1.print();
51     }
52     ...

```

In []:

1

Linked List implementation

```

struct Node {
    int data;
    Node *next;
};

class LinkedList {
    Node *head; // data member
    Node *tail;

    Node* createNode(int data, Node* next) {
        Node* p = new Node;
        p->next = next;
        p->data = data;
        return p;
    }
public:
    LinkedList() {this->head = this->tail = NULL;} // Constructor

    void insertFront(int data) {
        Node * temp = createNode(data, NULL)
        if (head == NULL) {
            head = temp;
            tail = temp;
        } else {
            temp->next = head;
            head = temp;
        }
    }

    void insertEnd(int data) {
        Node * temp = createNode(data, NULL)
        if (head == NULL) {
            head = tail = temp;
        } else {
            tail->next = temp;
            tail = temp;
        }
    }

    void deleteFront() {
        if (head == NULL) {
            cout << "Underflow" << endl;
        } else {
            Node *temp = head;
            head = head->next;
            delete temp; // temp.next = null
        }
    }
}

```

```

// Delete by address from middle (doesn't work for first and
Last node)
void deleteMid(Node *ptr) {
    Node *temp = ptr->next;
    ptr->data = ptr->next->data;
    ptr->next = ptr->next->next;
    delete temp;
}

// delete by value is still O(n)

// iterate and print
void print() {

    cout << endl;
    Node * temp = head;
    if (head == NULL) cout << "list is empty" << endl;
    while(temp != NULL) {
        cout << "data:" << temp->data << " next:" << temp->ne
xt << endl;
        temp = temp->next;
    }
}

};

int main() {
    LinkedList l1;
    l1.print();

    l1.insertFront(10);
    l1.print();

    l1.insertFront(20);
    l1.print();
}

```

In []:

1

In []:

```

1 ### LL vs Array
2

```

In []:

```

1 https://leetcode.com/problems/delete-node-in-a-linked-list/

```


