```
int main(){
    Hero adi;
    adi.level = 'A';
    adi.name = "Spiderman";
    adi.sethealth(100);

    cout << adi.level << endl;
    cout << adi.name << endl;
    cout << adi.gethealth << endl;
```

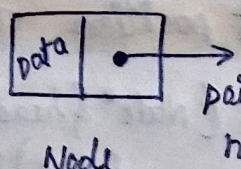
o/p : A
Spiderman
100

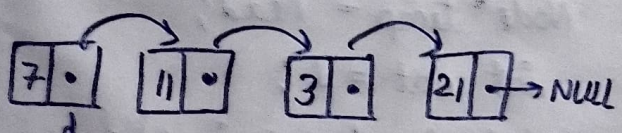
sethealth() → setter (void)
gethealth() → getter (returns a value)

lec 44

Linked list

→ Linear data structure (collection of Nodes)





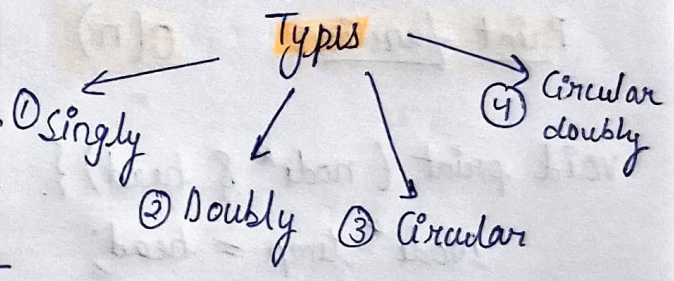
NOTE
↳ Vector is not optimal to inc↑ size during run time.

Advantages

1) Grow / Shrink optimally during runtime (Dynamic)
2) Insertion / Deletion → O(1)
3) No memory wastage.

Disadv

1) Uses more memory to store pointer.
2) Traversal → O(n)
3) Complex Implementation.
4) Not suited for small dataset.



Implimentation.

```
class Node {
    public:
        int Data;
        Node* next;
};
int main() {
    node *node1 = new node;
    node1->data = 100;
    node1->next = NULL;

    return 0;
}
```

Insertion O(1)

① Insertion at head.

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

Print function O(n)

```
void print (node* &head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp -> data << " ";
        temp = temp -> next;
    }
    cout << endl;
}
```

② Insertion at tail / NULL

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

confusion point

& → address-of / reference

* → Dereference (access value at an address via a ptr)

-> → Access member of an object through a ptr

shortcut for (*ptr).member

↓

ptr -> member.

③ Insert at position n

```
void insert (Node* &head, int posi,
             int d) {
    Node* temp = head;
    int cnt = 1;
    while (cnt < posi - 1) {
        temp = temp -> next;
        cnt++;
    }
```

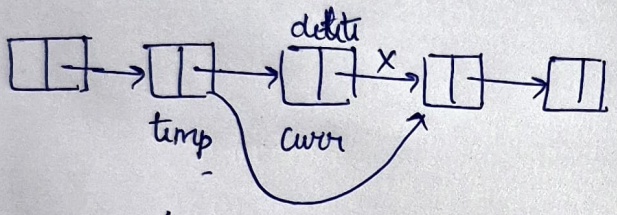
```
Node* nodeToInsert = new Node(d)
nodeToInsert->next = temp->next;
temp->next = nodeToInsert;
}
```

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

Free pointer before deleting any node.

Delition

① Delition position 'n'



temp->next = curr->next.

```
void delete (int position, Node*
                          &head){
  if (position == 1) {
      Node* temp = head;
      head = head->next;
      delete temp;
  } else {
      Node* curr = head;
      Node* prev = NULL;
      int cnt=1;
      while (cnt < position) {
          ~~curr = curr->~~
          prev = curr;
          curr = curr->next;
      }       cnt++;
```