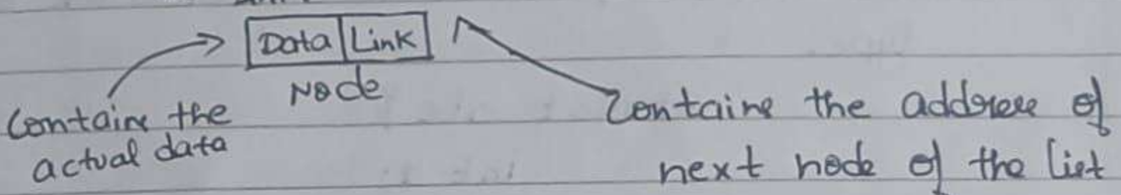
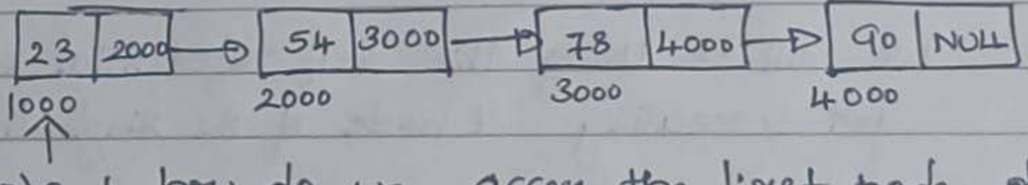
Skill Lab

1) Linked List :

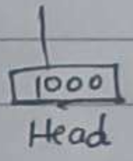
* Array has certain limitations in the way it operates, linked list provides us certain implementations to overcome this
* Types of linked list :
  (i) Single linked list
  (ii) Doubly linked list
  (iii) Circular linked list
* (i) Single linked list :
  - A single linked list is a list made up of nodes that consists of two parts:
    (i) Data
    (ii) Link



→ Representation of the single linked list :
  * Suppose we want to store a list of numbers:
    23, 54, 78, 90



Q) But how do we access the first node of the list?
Ans⇒ We need a pointer to access the first node of the list



→ If we want to store a list of numbers: 23, 54, 78, 90

Array

23	54	78	90
1000	1004	1008	1012

* In an array, elements are stored in consecutive memory locations
* Array is a sequential representation of a list

→ In linked list, the data is stored randomly in the memory locations but still we will be able to access all the data.

→ Creating the Node of a Single Linked List :

* Self Referential Structure is a structure which contains a pointer to a structure of the same type.

```
struct abc {
        int a;
        char b;
        struct abc * self;
    };
```

We will be using this self referential structure for creating a node of the single linked list

→ <u>Node representation in C</u> :

```
struct node {
    int data;
    struct node * link;
};
```

In general,

```
struct node{
    data_type member1;
    data_type member2;
        ,
        ,
        ,
    struct node *link;
};
```

* There can be multiple datatype but only one link in the node.

Some basic understanding of a pointers in c:

* pointer is a variable that stores the memo address of another variable
* It is declared using * operator

```
int * ptr;
```

Assigning a pointer:

* A pointer stores the address of a variable using (&) operator.
* `int a=10;`
  `int *ptr= &a;`

eg:

```
int main(){
        int a=10;
        int * ptr= &a;
        printf("Value of a:%d\n", a);
        printf("Address of a: %p\n", &a);
        printf(" Pointer stores: %p\n", ptr);
        printf("Value at pointer address: %d\n",
                    *ptr);
}   return 0;
```

* Arrow operator (->) in C:

* The arrow operator(->) is used to access member of a structure when working with pointers.

* It is a shorthand for dereferencing a pointer and accessing a struct member.

-> <u>Creating a Node in C</u>:

```
# include <stdio.h>
#include <stdlib.h>
struct node{
      int data;
      struct node *link;
     };
int main(){
      struct node *head = NULL;
       head =(struct node *) malloc (sizeof (struct
                                             node));
        head -> data = 45;
          head -> link = NULL;

         printf(" %d", head ->data);
         return 0;
 }
```

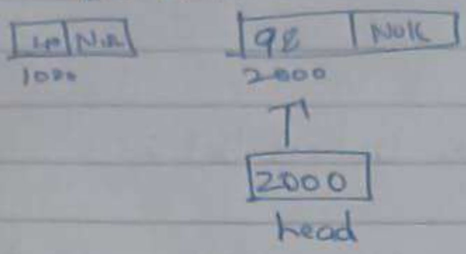
-> Now adding a node to the current ~~str~~ list :

same code till <u>head -> link = NULL</u>;

```
~~str~~ head = ~~(~~ Malloc (sizeof(struct node));
        head -> data = 98;
        head -> link = NULL;
```

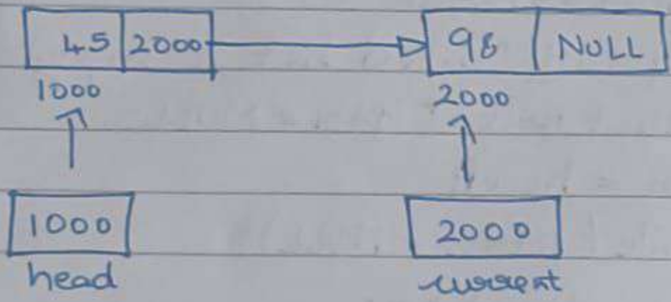
Now, we have two issue:

* head is now pointing to the second node
* Now, there is no way to access the first node of the list.



so, we need to create another pointer as current and assign this current to link part of the first nod

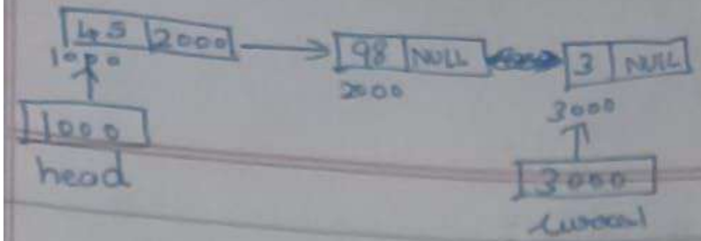
```
same first node code
  struct node *current = malloc(sizeof(struct node)
   current -> data = 98;
   current -> link = NULL;
    head -> link = current;
    return 0;
```



→ Adding three nodes :

```
int main()
{

   same till second node;
   // now we will reuse the current pointer
    current = malloc(sizeof(struct node));
    current ~~=~~ -> data = 3;
    current -> link = NULL;
```



now, Use this to link all the nodes
head → link → link = current;
→ It will link all the data

Traversing a singe linked list:

* Traversing means visiting each node of a single linked list until the endnode is reached.
* We will have to first count the nodes present in the linked list,

```
int main(){
        count_of_nodes(head);
    }
```

so function definition of it :

```
void count_of_nodes (struct node * head){
        int count =0;
        if (head == NULL)
            printf("Linked list is empty");
        struct node * ptr = NULL;
        ptr = head;
        while (ptr != NULL){
            count++;
            ptr = ptr -> link;
        }
        printf("%d", count);
}
```

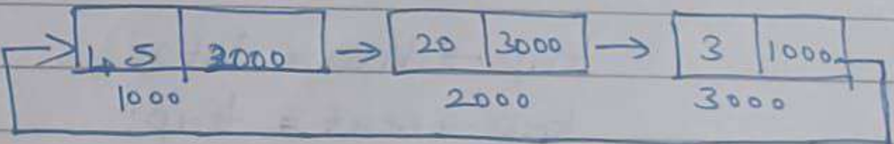
→ Printing the data:

```
void print_data(struct node * head){
        if (head == NULL)
            printf("Linked List is empty");
        struct node * ptr = NULL;
        ptr = head;
         while ( ptr ! = NULL){
               Printf("%d", ptr -> data);
               ptr = ptr -> link;
         }
}
```

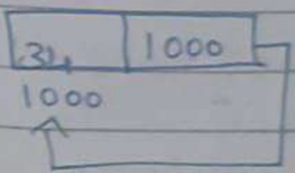
* Intro to Circular linked list:

* There are two types of Circular linked list:
  (i) Circular Singly Linked List
  (ii) Circular Doubly Linked list

* Circular singly linked list is similar to the singly linked list except that the last node of the circular singly linked list points to the first node
  Representation:



⇒ Circular Singly linked list node

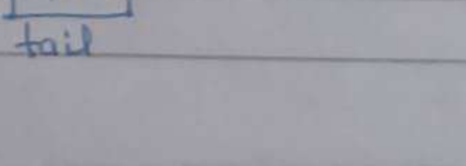


* Creating a node of the circular singly list

```
struct node {
        int data;
        struct node* next;
      };
  int main(){
       int data=34;
       struct node* tail;
       tail = circularSingly (data);

       printf("%d\n", tail->data);
       return 0;
  }

struct node* circularSingly (int data)
   {
        struct node* temp = malloc(sizeof(struct
                                        node));
         temp->data = data;
         temp->next = temp;
         return temp;
  }
```


34
1000
1000
1000
tail