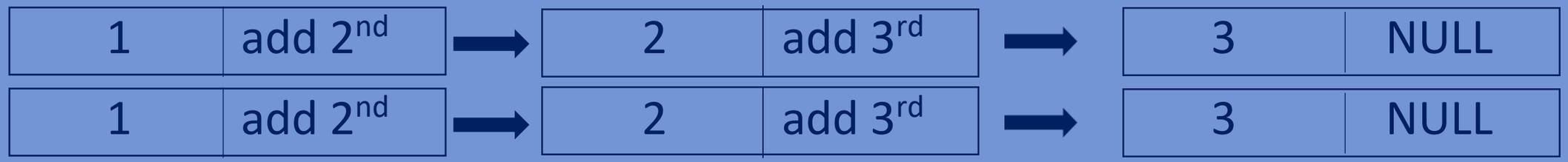
Data Structures

Comparing two linked lists

comparing of two linked lists:

- **Check whether two linked list are equal or not??**



- Both the linked lists are said to be equal if they have same no of nodes And corresponding nodes contain the same data.
- Traverse the two linked list at the same time.
- While traversing through the two linked lists simultaneously compare data between the two nodes and if data is not Equal simply return 0.
- Traverse through the linked list only if both the nodes not equal to NULL.
- After traversing if both nodes are equal to null return 1 else return 0.

Function for comparing of two linked lists(using while loop):

```
int compareLInkedListwhile(lin_list *headA,lin_list *headB){
    while(headA!=NULL && headB!=NULL){
        if(headA->data!=headB->data){
            return 0;
        }
        headA=headA->next;
        headB=headB->next;
    }
    return headA == NULL && headB == NULL ? 1 : 0;
}
```

```
if(headA==NULL && headB==NULL){
    return 1;
}
else{
    return 0;
}
```

Function for comparing of two linked lists(using recursion):

```
int compareLinkedList(lin_list *headA,lin_list *headB){
    if(headA==NULL && headB==NULL){
        return 1;
    }
    if(headA==NULL || headB==NULL || (headA->data!=headB->data)){
        return 0;
    }
    return(compareLinkedList(headA->next,headB->next));
}
```

Whole program:

```
#include<stdio.h>
#include<stdlib.h>
//creating a node.
typedef struct lin_list{
    int data;
    struct lin_list *next;
}lin_list;
//inserting nodes
```

```c
lin_list *insertnode(lin_list *head,int data) {
    lin_list *newnode=(lin_list*)malloc(sizeof(lin_list));
    newnode->data=data;
    newnode->next=head;
    head=newnode;
    return head;
}
//printing the linked list.
void PrintElements(lin_list *head){
    //base condition
    if(head==NULL){
        return;
    }
    printf("%d ",head->data);
    PrintElements(head->next);
}
//comparing two linked lists using recursion
int compareLinkedListRecursion(lin_list *headA,lin_list *headB){
    if(headA==NULL && headB==NULL){
        return 1;
    }
    if(headA==NULL || headB==NULL || (headA->data!=headB->data)){
        return 0;
    }
    return(compareLinkedListRecursion(headA->next,headB->next));
}
```

```c
//comparing two linked lists using while loop
int compareLInkedListWhile(lin_list *headA,lin_list *headB){
    while(headA!=NULL && headB!=NULL){
        if(headA->data!=headB->data){
            return 0;
        }
        headA=headA->next;
        headB=headB->next;
    }
    return headA == NULL && headB == NULL ? 1 : 0;
}
//main
int main(){
    lin_list *headA=NULL;
    lin_list *headB=NULL;
    //inserting elements into linked list A and B
    headA=insertnode(headA,1);
    headB=insertnode(headB,1);
    headA=insertnode(headA,2);
    headB=insertnode(headB,2);
    headA=insertnode(headA,3);
    headB=insertnode(headB,3);
    headA=insertnode(headA,4);
    headB=insertnode(headB,4);
    PrintElements(headA);printf("\n");
    PrintElements(headB);printf("\n");
```

```
    //comparing two linked lists
    printf("%d\n",compareLinkedListRecursion(headA,headB));
    printf("%d\n",compareLInkedListWhile(headA,headB));
    //inserting a node into linked list B
    headB=insertnode(headB,5);
    PrintElements(headA);printf("\n");
    PrintElements(headB);printf("\n");
    //comparing two linked lists
    printf("%d\n",compareLinkedListRecursion(headA,headB));
    printf("%d\n",compareLInkedListWhile(headA,headB));
    return 0;
}
```

Output:

```
4 3 2 1
4 3 2 1
1
1
4 3 2 1
5 4 3 2 1
0
0
```