B.M.S. College of Engineering

(Autonomous College Affiliated to Visvesvaraya Technological University, Belgaum)
Bull Temple Road, Basavanagudi, Bengaluru – 560019

Department of
Computer Science & Engineering (CSE)

Lab Programs Observation

Course Title: Data Structures

Course Code: 23CS3PCDST

BY
Tamanna Rukhaya(1BM22CS301)

B.M.S. College of Engineering

(Autonomous College Affiliated to Visvesvaraya Technological University, Belgaum)
Bull Temple Road, Basavanagudi, Bengaluru – 560019

Department of
Computer Science & Engineering (CSE)

CERTIFICATE

This is to certify that the report on **"DS Lab Programs"** has been carried out by **Tamanna Rukhaya** bearing USN **1BM22CS301** as a part of AAT for the course **Data Structures** with course code **23CS3PCDST**, Computer Science and Engineering from Visvesvaraya Technological University, Belgaum during the year 2023–24. It is certified that all corrections/suggestions indicated for Internal Assessments have been incorporated in the report.

Tamanna Rukhaya

1BM22CS301

Lakshmi Neelima

Assistant Professor

Department of CSE

BMSCE, Bengaluru-19

DS LAB - 1

__/__/____

Create structure : itemdetails with members itemname, quantity, price, total amount. Calculate party expenses

```
#include <stdio.h>

Struct itemdetails
{
  Char name[25];
  int quantity;
  float price;
  float total;
} item[10];

Void main()
{
int n, i;
Printf ("Enter the no. of items:");
Scanf ("%d", &n);
for(i=1; i<=n; i++)
{
Printf (" Enter the details of item %d\n", i);
Printf (" Enter item name:");
Scanf ("%s", item[i].name);
Printf (" Enter the quantity of item:");
```

```
Scanf("%d", &item[i].quantity);
Printf("enter price of item:");
Scanf("%f", &item[i].price);
item[i].total = item[i].price * item[i].
quantity;
}
float party_exp = 0;
for(i=1; i<=n; i++)
{
party_exp += item[i].total;
}
Printf("The Total party expenses is: %f",
Party_exp);
}
```

→ O/P

```
enter the number of item: -3
enter the details of item 1
enter item name: Chips
enter the quantity of item: 2
enter the price of item: 20
enter the details of item 2
enter item name: Chocos
enter the quantity of item: 1
enter the price of item: 30
enter the details of item 3
```

__/__/____

enter item name : Rice
enter the quantity of item : 3
enter the price of item : 30
the total party expenses is 140.000000

Create structure with name student with structure members : name, usn, gradelist of sem 1, grade list of sem 2. The student will be promoted to 3rd semester if he/she is not having backlog of credit count >= 16.

a. Write a C program using the concepts of functions and structure to check whether a student is promoted to 3rd semester or not based on the backlog credit count : Program logic should contain sem wise calculation for backlog credit count

b. same program using the concepts structure variable is array.

```
#include <stdio.h>
struct student {
char name[20];
char usn[20];
int gradelist1[10];
int gradelist2[10];
} abc;
int promoted (struct student abc) {
int coent = 0;
int i ;
for(i = 1; i <5; i++) {
if (abc.gradelist1[i] >0)
count = coent + abc.gradelist1[i];
if (abc.gradelist2[i] >0 )
coent= colnt + abc.gradelist2[i];
}
return count >= 16;
}
Void main() {
Printf ("Enter Name:");
Scanf ("%s", abc.name);
Printf ("Enter usn:");
Scanf ("%s", abc.usn);
int i;
```

```
For (i=0; i<5; i++){
Printf ("enter grade for subject %d in
1st sem:", i+1);
&Scanf ("%d", &abc.gradelist1[i]; }
For (i=0; i<5; i++){
Printf ("enter grade for subject %d in
2nd Sem: ", i+1);
Scanf ("%d", &abc.gradelist2[i]);
}
If [Promoted (abc)]
  Printf ("%s, Promoted to 3rd Sem", abc.name);
else
  Printf ("%s, Not Promoted to 3rd sem"); }
For (i=0; i<5; i++){
  Printf                                  abc.name
```

→ O/P

```
enter name : Tamanna
enter usn : 1BM22CS301
enter grade for subject 1 in 1st sem: 3
enter grade for subject 2 in 1st sem: 3
enter grade for subject 3 in 1st sem: 4
enter grade for subject 4 in 1st sem: 3
enter grade for subject 5 in 1st sem: 3
```

enter grade for subject 1 in 2nd sem: 3
enter grade for subject 2 in 2nd sem: 3
enter grade for subject 3 in 2nd sem: 4
enter grade for subject 4 in 2nd sem: 3
enter grade for subject 5 in 2nd sem: 3
tamanna, promoted to 3rd sem

Given an array arr[] containing (N) distances of the inch-feet system such that each element of the array represents a distance in the form of {inch, feet}. The task is to add all the N-inch-feet distance using structu

input arr[] = {{10, 3.7}{10, 5.5}, {6, 8.0}};
Output : Feet sum : 27 inch sum 5.20

input arr[] = {{1, 1.7}, {1.1.5}, {6, 8}};
Output : Feet Sum : 8 Inch sum : 11.20

```
#include <stdio.h>

struct distances{
  float arr[10][10];
} dist;

Void main()
{
int a, i, j, feetsum = 0, Inch sum = 0;
Printf("enter the number of inch-feet
  pairs to be added : \n");
```

```
scanf ("%d", &a);
Printf ("Enter the pairs : ");
for(i=0; i<a; i++)
 {
   for (j=0; j<2; j++){
    scanf ("%f", &dist.arr[i][j]);
    }
 }
for (i=0; i<a; i++){
   Feetsum += dist.arr[i][0];
   Inch sum += dist.arr[i][1];
}
Printf ("total feet sum = %d \n", Feetsum);
Printf ("total inch sum = %d \n", Inchsum);
}
```

→ enter the number of inch-feet pairs to be added:
2
enter the pairs: 25
30
45
50

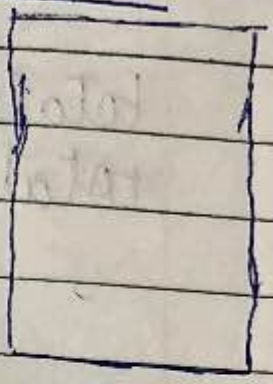
[25 30] [45 50]

25 + 45 = 70

totalfeetsum = 70
total inch sum = 80

DS LAB -2

PUSH POP AND DISPLAY OPERATOR/__/___



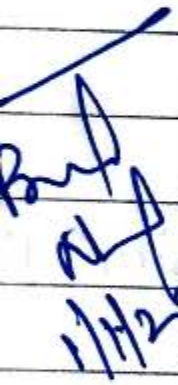
```
#include <stdio.h>
int stack [5], top = = 1;
Void main ()
{
Push ();
Pop ();
display ();
}

Void Push()
{
int a;
Printf ("Enter the Value of a\n");
scanf ("%d", &a);
top++;
Stack [top] = a;
}

Void pop()
{
int temp;
temp = stack [top];
top --;
}
Void display()
```

```
{
int i;
for (i = top; i > 0; i--)
{
Printf ("%d\n", stack[top]);
}
}
```

Output:



INFIX TO POSTFIX

```
# include <stdio.h>
# include <string.h>

int temp, index1 = 0, pos = 0, length;
char symbol, stack[20], infin[20],
postfin[20];
void push ( char symbol);
char pop()
int preced (char symbol);

void main()
{
printf ("Enter the infin expression");
scanf ("%s", infin);
infin to postfin();
printf ("Enter the postfin expression");
}
void infin to postfin()
{
length = strlen (infin);
while (index1 <= length)
{
symbol = infin [index1];
switch (symbol)
{
```

```
case '(' : push (Symbol);
           break;
case ')' : temp = pop();
           while (temp != '(')
           {
           postfix [pos] = temp;
           pos ++;
           temp = pop();
           }
           push (Symbol);

case '+';
case '-';
case '*';
case '/';
case '^';
while (preced (stack [top]) >= preced
(Symbol))
{
  temp = pop();
 postfix [pos] = temp;
   pos ++;
}
push (Symbol);
default: postfix [pos++] = Symbol;
}
```

```
index 1++;
}
while (top >=0)
{
temp= pop()
postfix [pos] = temp;
pos++;
}
}
void push (char symbol)
{
 top ++;
stack [top] = symbol;
}
char pop()
{
 char symbol;
 symbol = stack [top];
 top --;
 return (symbol);
}

int preced (char symbol)
{
int n;
```

```
switch (symbol)
{
 case '^' : n=3 ;
  break;
  case '*' :
  case '/' :    n = 2;
  case '+' :
  case '-' :    n=1;
  Case 'c' :   n =0;
}
return (n);
}
```

~~Output~~

DS LAB-3

__/__/____

Queue

```
# include <Stdio.h>
# define MAX 5
int front = -1; rear = -1, q[MAX];
void enqueue (int value) {
if (front == -1 && rear == -1) {
    front = rear = 0;
    q[rear] = value;
}
else if (rear = MAX-1) {
printf ("Overflow");
}
else
{
q[++rear] = value;
}
}
void dequeue () {
if (front == -1) {
printf ("underflow");
}
else {
  if (front > rear) {
    front = -1;
}
else
```

```
Printf ("%d", q[front]);
    front--;
}
}
Void display()
{
if (front == -1){
Printf ("underflow");
}
 else
{
for(int i = front; i<= rear; i++){
  Printf ("%d", q[i]);
}
}
}
}
int main (){
int boolean = 1, choice, value;
 While (boolean)
 {
Printf ("1.Enqueue\n 2. Enq. Dequeue\n 3. display
        \n 4. exit \n");
Scanf ("%d", &choice);
 Switch (choice)
{
```

```
case 1: printf ("Enter a value");
        Scanf ("%d", value);
        enqueue (value);
        break;

case 2: dequeue ();
        break;

case 3: display ;
        break;

case 4: boolean = 0; break;
default: printf (" Invalid Input ");
        break;
}
return 0;
}
```

circular queue

```
#include <stdio.h>
#define N 5
int front = -1, rear = -1, q[MAX];
void enqueue(int value)
{
if(front == -1 && rear == -1){
        front=0;
        rear=0;
      q[rear]=value;
}
else if((rear+1)%N == front)
{
printf("Overflow");
}
else
{
 rear = (rear+1)%N;
  q[rear]=value;

void dequeue(){
if(front == -1){
  Printf("underflow");
if(first == rear)
{
    front = rear=-1;
}
```

```
else
{
Printf ("%d", q[front]);
front = (front + 1) % N;
}
}
void display()
{
if (front == -1)
{
Printf (" underflow");
}
else {
int i = front;
while (i != rear) {
    Printf ("%d", q[i]);
    i = (i + 1) % N;
}
}
int main () {
int boolean = 1, choice, value;
while (Boolean)
{
Printf ("1. enqueue\n
        2. Dequeue\n
```

```
3. display\n
4. exit\n");
scanf("%d", &choice);
switch(choice){
case 1: printf("Enter the value");
        scanf("%d", &value);
        enqueue(value);
        break;
case 2: dequeue(); break;
case 3: display(); break;
case 4: boolean = 0; break;
default: print("Invalid"); break;
}
}
}
```

linear

Output

→ Enter 1. insert 2. delete 3. display 4. exit
2
underflow
enter value:
3
value inserted
Enter 1. insert 2. delete 3. display 4. exit 1
enter value:
4
value inserted
Enter 1. insert 2. delete 3. display 4 exit 1
enter value:
5
value inserted
Enter 1. insert 2. delete 3. display 4. Exit 1
enter value:
6
Overflow
Enter 1. insert 2. delete 3. display 4. exit 3
3
4
5
Enter 1. insert 2. delete 3. display 4. exit 4

Circular

→ Enter 1. insert 2. Delete 3. display 4. Exit 2
Underflow
Enter 1. insert 2. delete 3. display 4. Exit 1
Enter value: 4
value inserted
Enter 1. Insert 2. delete 3. display 4. Exit 1
~~value~~ is enter value 5
value inserted
Enter 1. insert 2. delete 3. display. 4 exit 1
Enter value: 6
value inserted
Enter 1. insert 2. delete 3. display 4. Exit 1
Enter value: 7
Overflow
Enter 1. insert 2. delete 3. display 4. Exit 3
4 5 6
Enter 1. insert 2. delete 3. display. 4 exit

DS LAB-4

22.1.24 __/__/____

→ beg
→ end
→ At Pos
→ del at end, del beg, at pos

```
#include <stdio.h>
#include <stdlib.h>
Struct node
{
int data;
Struct node *next;
};
Void PrintData (Struct node *head)
{
 if (head == null) {
  Printf (" The list is empty ");
 }
else
{
Struct node *Ptr = head;
While (Ptr != NULL)
  {
Printf ("%d \n", Ptr → data );
    Ptr = Ptr → next;
 }
}
}
```

```
Void insertbeg (Struct node ** head, int
value)
{
 Struct node * temp = (Struct node*)
malloc ( Size of (Struct node));
    temp → data = value;
    temp → next = *head;
    *head = temp;
Void insertend (Struct node* head,
 int value)
{
   Struct node *Ptr = head;
   Struct node *temp = (Struct node*)
 malloc (Sizeof (Struct node));
    temp → data = value;
    temp → next = NULL;
   while (Ptr → next != NULL) {
     Ptr = Ptr → next;
   }
Ptr → next = temp;

Void insertatpos (Struct node* head,
 int value , int pos)
   {
```

```
Struct node *Ptr *Ptr2;
Struct node *temp = (Struct node*)
malloc (sizeof(Struct node));
    temp→ data = value;
    temp → next = NULL;
int position=pos;
    Ptr = head;
While (Pos != 1)
  {
      Ptr2 = Ptr;
    Ptr = Ptr → next;
     Pos--;
  }
temp → next = Ptr2 → next;
  Ptr2 → next = temp;
Printf ("Value %d added Sucessful
  at %d \n", value, position);

Void delet (Struct node ** head)
  {
  Struct node *Ptr;
  If (head != NULL)
{
Printf ("The list is empty");
}
```

```
else {
Ptr = *head
*head = (*head) -> next;
    free (Ptr)
    Ptr = NULL;
}
Void delend (Struct node *head)
{
Struct node *Ptr, *Ptr2;
if (head == NULL){

Printf ("The list is empty");
}
else
Ptr = head;
While (Ptr -> next = NULL){
    Ptr2 = Ptr;
    Ptr = Ptr -> next;
  }
   Ptr2 -> next = NULL;
      free (Ptr);
}
}
Void delatpos (Struct node *head, int pos)
```

```
Struct node *Ptr *Ptr2
if (head == NULL)
  {
Printf ("The list is empty");
else if (Pos == 1)
{
 Ptr = head;
   free (Ptr);
   Ptr = NULL;
}
else
 Ptr = head;
 Ptr2 = head;
  While (Pos != 1)
   {
    Ptr2 = Ptr;
    Ptr = Ptr -> next;
      Pos--;
}
Ptr2 -> next = Ptr -> next;
     free (Ptr);
        Ptr = NULL;
  }
}
```

```
int main ()
{
struct node *head = null;
insert atbeg (& head, 34);
Print Data (head);
Printf ("- - - - - - - - - \n");
insert end (head, 75);
insert End (head, 56);
insert end (head, 87);
Printdata (head);
Print ("- - - - - - - - \n");
insert At Pos (head, 89, 3);
Printdata (head);
Printf ("- - - - - - - - \n");
delbeg (&head);
Printf ("- - - - - - - - \n");
delend (&head);
PrintData (head);
Print ("- - - - - - - - \n");
detatpos (head, 2);
Print data (head);
Printf ("- - - - - - - - \n");
}
```

Value 34 added successfully at the beginning

34

- - - - - -

Value 75 added succesfully at the end

value

output

value 56 added succesfully at the end.

value 87 added succesfully at the end

34

75

56

87

value 89 added succefully at 3

34

75

89

56

87

__/__/____

75
89
56
87

75
89
56

75
87

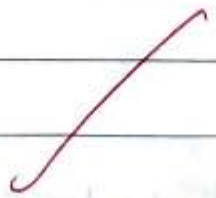
DS-LAB-5

29.04.24 LinkedList Stacks

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int data;
struct node *next; ?
  *top = NULL;
Void push (int value)
{
Struct node *newNode = (struct node*)
malloc (Size of (Struct node));
newNode → data = value;
newnode → next = top;
top= newnode;
Printf ("Succefully Added '%d', Value);
}
Struct node *temp;
 If (top == NULL)
{
Printf("underflow");
}
else {
 temp = top;
int val = temp → data;
top= top → next;
 free (temp);
```

```
temp = NULL;
Printf(" %d", val);
}
}
void display()
{
if(top == NULL)
{
Printf(" the stack is empty");
}
else {
struct node * ptr = top;
while(ptr != NULL)
{
Printf("%d\n", ptr -> data);
ptr = ptr -> next; }
}
}
void main()
{
int choice, value, boolean = 1;
while(boolean){
printf("\n 1.Push \t 2. pop \t 3. display \t
 4. Exit");
scanf("%d", &choice);
```

```
switch (choice)
{
case 1:
scanf("%d", &value);
Push(value);
break;
case 2:
Pop();
break;
case 3:
display()
break;
case 4
boolean = 0;
break;
default:
Printf("Invalid output");
}
}
}
```



op

```
1. Push 2. Pop 3. display 4. Exit :
2
succefully Added 2
1. Push 2. pop 3. display 4. Exit :
3
```

1. Push 2. pop 3. display 4. exit 3

3

2

1. Push 2. pop 3. display 4. exit 2

3. deleted.

29.01.24

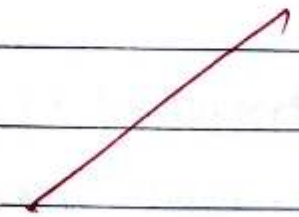
Linked list queue.

```
#include <stdio.h>
#include <stdlib.h>
struct node{
int data;
struct node *next;
};
struct node *rear = NULL;
struct node *front = NULL

void enqueue(int val){
struct node *newnode = (struct node*)
malloc(sizeof(struct node));
newnode -> data = value;
newnode -> next = NULL;
if (front==0 && rear==0)
{   front = rear = newnode;
}
```

```
else{
rear -> next = newnode;
rear = newnode

Void dequeue(){
if (front-- = NULL && rear = = NULL)
{
Printf(" empty");
}
else{
struct node *temp;
    temp = front;
 front = front ->next;
 free(temp)
 temp = NULL;
 }
}
void display()
 {
if (front = = NULL){
Printf (" empty");
 }
else
{
Struct node *ptr = front;
```



```
while(ptr!=NULL)
{
Printf("%d\n", ptr->data);
ptr = ptr->next; }
}
}
void main()
{
int choice, val, bool=1;
while(bool)
{
Printf("\n 1.Enqueue 2.Dequeue 3.display 4.Exit
Scanf("%d", &choice);
Switch (Choice)
{
case 1:
Scanf("%d", &value);
enqueue(value);
break;
case 2:
dequeue();
break;
case 3:
display();
break;
```

```
case 4:
boolean = 0;
break; }
}
}
```

o/p

```
1. Enqueue 2. Dequeue 3. display 4. Exit 1
2.
1. Enqueue 2. Dequeue 3. display 4. exit 1
3
1. Enqueue 2. Dequeue 3. display 4. exit 2
1. enqueue 2. Dequeue 3. display 4. exit 3
3
```

DS LAB - 6

__/__/____

```
#include<stdio.h>
#include<stdlib.h>
Struct node
{
int data;
Struct node* prev;
Struct node*next;
};
Struct node * insertatbeg (Struct node*head,
int val)
{
Struct node* temp;
temp = (Struct node*) malloc (sizeof (Struct node));
temp→nent = NULL;
temp→ prev = NULL;
temp → data = val;
If (head = = NULL)
{
 head = temp;
  return head;
}
else {
 temp → nent = head;
 head → Prev = temp.
    head = temp;
return head;}
}
```

```
struct node * insertatend ( struct node *head,
int value) {
 struct node * temp;
temp = ( struct node *) malloc(sizeof(struct
node));
temp -> prev = NULL;
temp -> data = value;
temp -> next = NULL;
if (head == NULL){
 head = temp;
}
else {
struct node * ptr = head;
while (ptr -> next != NULL)
 {
 ptr = ptr -> next;
}
temp -> prev = ptr;
ptr -> next = temp;
}
 return head;
}

void printdata ( struct node *head)
{
```

```
struct node * ptr = head;
while (ptr != NULL)
{
printf ("%d \n", ptr -> data);
ptr = ptr -> next;
}
}
printf (" - - - - - - - - - - ");
}
struct node * temp;
temp = (struct node *) malloc (sizeof (struct node));
temp -> prev = NULL;
temp -> data = value;
temp -> next = NULL;
struct node * ptr1, * ptr2;
ptr1 = head;
pos--;
while (pos != 0){
ptr1 = ptr1 -> next;
pos--;
}
ptr2 = ptr1 -> next;
temp -> prev = ptr1;
temp -> next = ptr2;
ptr1 -> next = temp;
ptr2 -> prev = temp;
return head;
}
```

```
void deval (Struct node * head, int value)
{
If (head == NULL)
{
Printf ("empty\n");
}
else
{
struct node *Ptr1 = head.
while (Ptr1 != NULL && Ptr1 -> data != value)
{
 Ptr1 = Ptr1 -> next;
}
If (Ptr1 != NULL)
{
struct node* Ptr2 = Ptr1 -> next.
Ptr2 -> Prev = Ptr1 -> Prev.
Ptr1 -> Prev -> next = Ptr2;
 free (Ptr1);
}
else
{
Printf ("Not found"); }
}
}
```

```
int main()
{
struct node * head = NULL;
head = insertatbeg (head, 30);
head = insert at key (head, 45);
Print data (head);
head = insertatbeg (head, 56);
head = insertatend (head, 75);
Printdata (head);
head = insertbefore (head, 11, 2);
Printdata(head);
deval(head, 45);
Printdata (head);
}
```

45
30
— — — —
56
45
30
75
— — — —
56
45
11
30
75
— — — —
56
11
30
75
— — — —

19/2/24

Binary search tree

DS LAB - 7

__/__/____

```
#include<stdio.h>
#include<stdlib.h>

Struct treenode
{
int val;
Struct treenode* left;
Struct treenode* right;
};

Struct treenode* create node (int val)
{
Struct treenode* newnode = (struct tree node*)
malloc(Sizeof(Struct treenode));
newnode → val = val;
newnode → left = left;
newnode → right = right;
return newnode;
};
Struct treenode* insert (Struct treenode *root,
int val)
{
if (root == NULL)
{
return (createnode(val));
}
```

```
if (val < root -> val)
  {
    root -> left = insert (root -> left, val);
  }
else if ( val > root -> val)
  {
   root -> right = insert (root -> right, val);
  }
return root;
};
Void inorder (Struct treenode*root)
{
if ( root != NULL)
{
 inorder ( root -> left);
 printf ("%d", root -> val);
 inorder ( root -> right);
}
}
Void postorder (Struct treenode* root)
{
 if (root != NULL)
    {
      postorder (root -> left)
      postorder (root -> right);
```

```
        Printf ("%d", root->Val);
    }
}
void preorder (Struct treenode* root)
  {
if (root != NULL)
{
 printf ("%d", root->Val);
  preorder ( root->left);
   preorder ( root->right);
 }
}
Void display (Struct treenode* root)
{
 printf ("Inorder traversal");
   inorder (root);
 printf ("\n");
}
   printf (" postorder traversal");
     postorder (root);
     Printf ("\n");

   printf (" preorder Traversal");
        preorder (root);
              printf ("\n");
}
```

__/__/____

```
int main()
{
Struct treenode* root = NULL;
  root = insert(root, 50);
  insert(root, 30);
  insert(root 20);
  insert(root 40);
  insert(root, 70);
   insert(root, 60);
  insert(root, 80);
  display(root);
}
```

→ inorder traversal

20 30 40 50 60 70 80

postorder traversal

20 40 30 60 80 70 50

preorder traversal

50 30 20 40 70 60 80

Paranthesis Question

19/2/24

Leet Code - 1

```
#include <stdio.h>
#include <stdlib.h>

int scoreofParanthesis (char *S)
  {
   int Stack[50];
   int top = -1;
   int score = 0;

for(int i=0; S[i] != '\0'; i++)
   {
    if (S[i] == '(')
      {
       Stack[++top] = score;
        score = 0;
   }
  else
   {
    score = Stack[top--] + score == 0 ? 1 : 2*
    score);
  }
}
return score;
}
```

__/__/__

Case 1	Case 2	Case 3
"()"	"(())"	"()()"
1	2	2
1	2	2

19/2/24

Deletion of middle node

Leet code -2

__/__/____

```
Struct listnode* deletemiddle(Struct listnode* head)
{
If (head == NULL || head->next == NULL)
  {
     return NULL;
  }
Struct listnode* slow_ptr = head;
Struct listnode* fast_ptr = head;
Struct listnode* prev = NULL;

While (fast_ptr != NULL && fast_ptr->next != NULL)
{
 fast_ptr = fast_ptr -> next -> next;
       prev = slow_ptr;
     slow_ptr = slow_ptr -> next;
}
If (prev != NULL)
{
 Prev -> next = slow_ptr -> next;
    free (slow_ptr);
}
else
{
```

```
    head = head->next;
    free(slow->p1);
}
return head;
}
```

case 1

head =
[1, 3, 4, 7, 1, 2, 6]

output

[1, 3, 4, 1, 2, 6]

expected

[1, 3, 4, 1, 2, 6]

case 2

head =
[1, 2, 3, 4]

output =

[1, 2, 4]

expected

[1, 2, 4]

case 3

head	output	expected
[2, 1]	[2]	[2]

Even - Odd linked list

19/2/24 Leet Code - 3

```
#include <stdio.h>
#include <stdlib.h>

struct listnode{
    int val;
    struct listnode* next;
};

struct listnode* oddevenlist ( struct listnode*
head)
{
if (head == NULL || head -> next = NULL ||
    head -> next -> next == NULL)
        return head;

struct listnode *odd = head;
struct listnode* even = head -> next;
struct listnode * evenhead = even;

while(even != NULL && even -> next != NULL)
{
    odd -> next = even -> next;
    even -> next = odd -> next;
        even = even -> next;
}
```

```
odd->next = evenhead;
  return head;
}
struct listnode* newnode(int val)
{
struct listnode* node =
(struct listnode*) malloc(sizeof
(struct listnode));

 node->val = val;
 node->next= NULL;
  return node;
}
void printlist(struct listnode*head)
 {
 while(head!=NULL)
 {
printf("%d", head->val);
    head = head->next;
}
Printf("\n");
}
```

26/2/24

__/__/____

Case 1

Input

head =
[1, 2, 3, 4, 5]

output =

[1, 3, 5, 2, 4]

expected

[1, 3, 5, 2, 4]

Case 2

Input

head =
[2, 1, 3, 5, 6, 4, 7]

output

[2, 3, 6, 7, 1, 5, 4]

Expected

[2, 3, 6, 7, 1, 5, 4]

DS LAB

__/__/____

BFS AND DFS

```
#include <stdio.h>
#include <stdlib.h>
Struct Node
{
 int data;
 Struct node*next;
};

Struct graph
{
int numVertices;
Struct node**adjlists;
int visited;
};

Struct Node* create node (int data)
{
 Struct node * newnode =(Struct node*)
 malloc (sizeof (struct node));
  newnode -> data = data;
  newnode->next = NULL;
  return newnode;
}
```

__/__/__

```
Struct graph * ~~creategraph~~ creategraph
(int numVertices)
{
Struct graph* graph = (Struct graph*)
malloc (sizeof (Struct graph));

graph → numVertices = numVertices.
graph → adjlists = (Struct node**)
malloc (numVertices * sizeof (Struct node*));

For (int i = 0; i < numVertices; i++)
{
graph → adjlists [i] = NULL;
graph → visited [i] = 0;
}
return graph;
}
Void addEdge (Struct graph * graph, int src, int
dest)
{
Struct node* newnode = createnode (dest).
newnode → next = graph → adjlists [src];
graph → adjlists [src] = newnode;
```

```
    newnode = createnode(src);
    newnode -> next = graph -> adjlists
    [dest];
    graph -> adjlists [dest] = newnode;
}

void BFS (Struct graph* graph, int
startVertex)
{
    int queue [MAXSIZE];
    int front = -1, rear = -1;
    graph -> visited [StartVertex] = 1;
    queue [++rear] = StartVertex;
    while ( front != rear)
    {
    int currentVertex = queue [++front];
    Printf ("%d", currentVertex);
    Struct node *temp = graph -> adjlists
    [current vertex];
    while (temp)
    {
    int adjVertex = temp -> data;
    if ( graph -> visited [adjVertex == 0)
    {
    graph -> visited [adjVertex] = 1;
```

```
queue[++rear] = adjVertex;
}
temp = temp->next;
}
}
}
void DFS ( Struct graph* graph, int vertex)
{
graph -> visited [vertex] = 1;
printf("%d", vertex);

Struct node* temp = graph -> adjLists
[vertex];

while (temp)
{
int adjVertex = temp -> data;
if ( graph -> visited [adjVertex] == 0)
{
DFS(graph, adjVertex);
}
temp = temp -> next;
}
}
```

```
int main()
{
Struct graph * graph = create graph(4);
 add edge (graph, 0, 1);
 add edge (graph, 0, 2);
 add edge (graph, 1, 2);
 add edge (graph, 2, 3);

printf ("BFS traversal starting from
 vertex 0:");
  BFS (graph, 0);

For(int i=0; i< graph -> numVertices; i++);
{
 graph -> visited[i]=0;
}
printf ("DFS traversal starting from
 vertex 0");
DFS (graph, 0);
return 0;
}
```

output

BFS Traversal Starting from vertex 0: 0 2 1 3
DFS Traversal Starting from vertex 0: 0 2 3 1

Leet Code - 4

__/__/____

```
Struct treenode* minvaluenode
(Struct treenode* node)
{
Struct treenode *current = node;

while (current && current->left != NULL)
 current = current->left;

return current;
}

Struct treenode* deletenode
 (Struct treenode* root, int key)

{
If (root == NULL)
return root;

If (key < root->val)
   root->left = delete node
   (root->left, key);

else If (key > root->val)
   root->right = deletenode
   (root->right, Key);
```

```
else
{
if (root->left == NULL)
{
Struct Treenode *temp = root->right;
free(root);
return temp;
}
else if (root->right == NULL)
{
Struct Treenode* temp = root->left;
free(root);
return temp;
}

Struct treenode* temp = minvaluenode
(root->right);

root->val = temp->val;
root->right = deletenode
(root->right, temp->val);
}
return root;
}
```

Leet Code - 5

```
int findbottomleftvalue (struct Treenode*
root)
{
if (root == NULL)
  {
    return = -1;
}
struct Treenode ** queue = ( struct Tree
node **) malloc (pow(10,4) * sizeof
(Structtreenode*));
int front=0,rear=0;
int leftmostvalue=0;

queue[rear++] = root;

while (front < rear)
{
int levelsize = rear - front;

for (int i=0; i <levelsize; i++)
{
struct treenode *currentnode = queue
[front++];
```

```
if (i == 0)
{
 leftmostvalue = currentnode->val;
}
if (currentnode->left)
{
 queue[rear++] = currentnode->left;
}
if (currentnode->right)
{
 queue[rear++] = currentnode->right;
}
}

free(queue);
return leftmostvalue;
}
```

Case 1

root =
[2,1,3]

output
1

expected
1

Case 2

root =
[1,2,3,4, null, 5, 6, null, null, null, 7]

output
7

Expected
7

Leet Code - 4

Output

Case 1

root=
[5,3,6,2,4, null, 7]

Key =
3

output

[5,4,6,2, null, null, 7]

Expected

[5,4,6,2, null, null, 7]

Case 2

root =

[5,3,6,2,4, null, 7]

Key=
0

output

[5,3,6,2,4,null,7]

expected

[5,3,6,2,4,null,7]

case 3

root =
[]

key =
0

output
[]

expected
[]