```
void display() {
{
struct node *p= ~~head~~ top;
int x;
printf("The stack is :");
while (top!=NULL){
~~prin~~ x = top-> data;
  printf("%d ->", x);
  ~~x~~ top = top -> next;
 }
 printf("NULL\n");
}
```

~~1)~~ b) Queue

```
 struct node{
   int data;
   struct node *next;
 };
struct node *enqueue(struct node* head, int value){
  struct node* temp=(struct node*)malloc(sizeof
                                          (struct node));

  temp->data=value;
  if(head == NULL){
    temp->next=head;
    head=temp;
  }
  else {
  struct node * newN=head;
   while (newN->next!=NULL){
     newN=newN->next;
   }
   newN->next=temp;
```

```
    temp -> next = NULL;
  }
  return head;
}
struct node* dequeue(struct node *head){
   if (head == NULL){
      printf("Queue is empty \n");
      return head;
   }
   struct node *tp = head;
   head = tp -> next;
   free(tp);
   return head;
}

void display(struct node *head){
    struct node *d= head;
    while (d != NULL){
       printf("%d ->", d->data);
       d = d -> next;
    }
    printf("NULL\n");
}
int main(){
   struct node * head = NULL;
   int choice, value;
   do{
   printf("\n1. Enqueue\n");
   printf("2. Dequeue \n");
   printf("3. Display\n");
   printf("4. Exit\n");
   printf("Enter your choice:");
```

```
scanf("%d", &choice);
switch (choice) {
  case 1:
     printf("Enter value to enquene:");
     scanf("%d", &value);
     head = enquene(head, value);
     break;
  case 2:
     head = dequeue(head);
     break;
  case 3:
     printf("Queue:");
     display(head);
     break;
  case 4:
     printf("Exiting...\n");
     break;
  default:
     printf("Invalid choice! ~~Enter~~ Please enter a valid
       option.\n");
  }
 } while (choice != 4);
 return 0;
}
```

29/01/24

10) a) Output: ~~Enter values to be inserted~~ Enter the size of the linked list: 5

Enter values to be inserted one by one: 1 2 3 4 5

Before sorting: 5→4→3→2→1→NULL

After sorting: 1→2→3→4→5→NULL

(b) Enter the size of the linked list 5
Enter values to be inserted one by one: 1 2 3 4 5
Before reversing 5→4→3→2→1→NULL
After reversing: 1→2→3→4→5→NULL

(c) Enter the size of linked list1: 3
Enter values to be inserted one by one: 1 2 3
Enter the size of linked list2: 3
Enter values to be inserted one by one: 4 5 6
After concatenation: 1→2→3→4→5→6→NULL

Stack 1) a) Output

```
1. push
2. pop
3. display
4. exit
Enter your choice: 1
Enter value to push: 6

1. push
2. pop
3. display
4. Exit
Enter your choice 2
Enter value to push: 7

1. push
2. pop
3. display
4. Exit
Enter your choice. 3
Stack 7→6→NULL
```

```
1. push
2. pop
3. display
4. exit
Enter your choice : 2
7 has been popped

1. push
2. pop
3. display
4. exit
Enter your choice : 3
Stack : 6 → NULL

1. push
2. pop
3. display
4. exit
Enter your choice : 4
```

11) b) Queue

Output:

```
1. Enqueue
2. Dequeue
3. display
4. exit
Enter your choice : 1
Enter the value : 6
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value: 7

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue: 6 → 7 → NULL

1. Enqueue
2. Dequeue
3. Display
4. exit
Enter your choice: 2
 6 has been dequeued

1. Enqueue
2. Dequeue
3. Display
4. exit
Enter your choice: 3
 ~~7~~ Queue: 7 → NULL

1. Enqueue
2. Dequeue
3. Display
4. Exit
```

Enter your choice:4

12) Doubly linked list

```
struct node{
    int data;
    struct node* prev;
    struct node* next;
};
struct node* insertAtLeft(int x){
    struct node* p=(struct node*)malloc(sizeof(struct
                                                node));

    if(head==NULL){
        p->data=x;
        p->prev=NULL;
        p->next=NULL;
        head=p;
    }
    else{
        p->data=x;
        p->prev=NULL;
        p->next=head;
        head->prev=p;
        head=p;
    }
    return head;
}
void deleteVal(int x){
    struct node * p= head;
    int ch=0;
    while(p->next!=NULL){
            if (p->data == x){
                ch++;
                break;
            }
            p=p->next;
        }
        if (ch==0){
            printf("Value %d not found in linked list\n
        }
        else {
            p->prev->next = p->next;
            p->next->prev= p->prev;
            printf("%d has been removed \n", p->data);
            free(p);
        }
}
void print(){
    struct node* temp= head;
    while (temp! =NULL){
        printf("%d ->", temp-> data);
        temp= temp-> next;
    }
    printf("NULL \n");
}
```

Output:

```
1. Insert at Left
2. Delete value
3. display
4. exit
Enter your choice: 1
Enter value to be inserted : 1
```

```
1. Insert at left
2. Delete value
3. display
4. exit
Enter your choice: [redacted]
Enter value to be inserted: 2

1. Insert at left
2. Delete value
3. display
4. exit
Enter your choice: 1
Enter value to be inserted: 3

1. Insert at left
2. Delete value
3. display
4. exit
Enter your choice: 3
3->2->1->NULL

1. Insert at left
2. Delete value
3. display
4. exit
Enter your choice: 2
Enter value to be deleted: 2

1. Insert at left
2. Delete value
3. display
4. exit
```

```
Enter your choice: 3
3→1→NULL

1. Insert at Left
2. Delete value
3. display
4. exit
Enter your choice: 4
```

13) Binary Search tree

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

struct node{
    int value;
    struct node* right;
    struct node* left;
};

struct node* create(int x){
    struct node* p=(struct node*)malloc(sizeof
                                      (struct no

    p→right = NULL;
    p→left = NULL;
    p→value= x;
    return p;
}
struct node* insert(struct node* temp, int x){
    if (temp==NULL){
        return(create(x);
    }
    if (x<temp->value){
        temp->left = insert(temp->left, x);
    }
    else if (x > temp -> value){
        temp->right = insert(temp-> right, x);
    }
    return temp;
}
struct node* inorder(struct node* root){
    struct node* temp=root;
    if(temp!=NULL){
        inorder(temp-> left);
        printf("%d", temp-> value);
        inorder (temp-> right);
    }
}
struct node* postorder(struct node* root){
    struct node* temp=root;
    ~~p~~ if(temp!=NULL){
        postorder (temp-> left);
        postorder (temp-> right);
        printf("%d ", temp-> value);
    }
}
struct node* preorder(struct node* root){
    struct node* temp = root;
    if (temp!=NULL){
        printf("%d", temp->value);
        ~~pos~~ preorder(temp-> left);
        preorder(temp -> right);
    }
}
```

Output:

```
inorder : 10  20  30  50  60  70
postorder: 30  20  10  70  60  50
preorder: 50  10  20  30  60  70
```

19.02.24

14) BFS

```
#include <stdio.h>

void bfs(int a[10][10], int n, int u){
    int f, r, q[10], v;
    int s[10] = {0};
    printf("The nodes visited from %d:",u);
    f=0;
    r=-1;
    q[++r] = u;
    s[u] = 1;
    printf("%d ", u);
    while (f <= r){
        u = q[f++];
        for (v=0; v<n; v++) {
            if (a[u][v] == 1 && s[v]==0){
                printf ("%d ", v);
                s[v] = 1;
                q[++r] = v;
            }
        }
    }
printf("\n");
```

```
int main() {
    int n, a[10][10], i, j;
    printf("\nEnter no of nodes ");
    scanf("%d", &n);
    printf("\nEnter the adjacency Matrix:\n");
    For (i=0; i<n; i++){
        for(j=0; j<n; j++){
            scanf("%d", & a[i][j]);
        }
    }
    for (int source=0; source < n; source++){
        bfs(a, n, source);
    }
    return 0;
}
```

Output:

```
Enter no of nodes: 4
Enter the adjacency Matrix:
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0
The nodes visited from 0: 0 1 2 3
The nodes visited from 1: 1 0 2 3
The nodes visited from 2: 2 0 1 3
The nodes visited from 3: 3 0 1 2
```

15) DFS

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

void dfs(int a[10][10], int n, int u, int visited[]){
    int v;
    printf("%d", u);
    visited[u]=1;
    for (v=0;v<n;v++){
        if (a[u][v]==1 && !visited[v]){
            dfs(a,n,v,visited);
        }
    }
}
int main(){
    int n,a[10][10], source, i, j;
    int visited[10]={0};
    printf("Enter the number of vertices:");
    scanf("%d",&n);
    printf("Enter the adjacency matrix:");
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            scanf("%d",&a[i][j]);
        }
    }
    printf("DFS traversal:");
    for (source=0; source<n; source++){
        if (!visited[source]){
            dfs(a,n,source,visited);
        }}
    return 0;
}
```

Output:

```
Enter the number of vertices : 4
Enter the adjacency matrix : 0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0
DFS traversal : 0 1 2 3
```

26.02.24