DAY-7

Implementation of Stack:

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int count = 0;
struct stack {
  int items[MAX];
  int top;
};
typedef struct stack st;

void createEmptyStack(st *s) {
  s->top = -1;
}

int isfull(st *s) {
  if (s->top == MAX - 1)
    return 1;
  else
    return 0;
}

int isempty(st *s) {
  if (s->top == -1)
```

```
    return 1;
  else
    return 0;
}

void push(st *s, int newitem) {
  if (isfull(s)) {
    printf("STACK FULL");
  } else {
    s->top++;
    s->items[s->top] = newitem;
  }
  count++;
}

void pop(st *s) {
  if (isempty(s)) {
    printf("\n STACK EMPTY \n");
  } else {
    printf("Item popped= %d", s->items[s->top]);
    s->top--;
  }
  count--;
  printf("\n");
}

void printStack(st *s) {
```

```
  printf("Stack: ");
  for (int i = 0; i < count; i++) {
    printf("%d ", s->items[i]);
  }
  printf("\n");
}

int main() {
  int ch;
  st *s = (st *)malloc(sizeof(st));

  createEmptyStack(s);

  printf("Enter the number of elements you want to push\n");
  int n;
  scanf("%d", &n);
  int a;

  for(int i = 0; i < n; i++){
    scanf("%d", &a);
    push(s,a);
  }
//   push(s, 1);
//   push(s, 2);
//   push(s, 3);
//   push(s, 4);
```

```
  printStack(s);

  pop(s);

  printf("\nAfter popping out\n");
  printStack(s);
}
```

Output:

Implementation of Queue:

Code:

```
#include <stdio.h>
#define SIZE 5

void enQueue(int);
void deQueue();
void display();

int items[SIZE], front = -1, rear = -1;

int main() {

  deQueue();

  enQueue(1);
```

```
  enQueue(2);
  enQueue(3);
  enQueue(4);
  enQueue(5);

  // 6th element can't be added to because the queue is full
  enQueue(6);

  display();

  //deQueue removes element entered first i.e. 1
  deQueue();

  //Now we have just 4 elements
  display();

  return 0;
}

void enQueue(int value) {
  if (rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (front == -1)
      front = 0;
    rear++;
    items[rear] = value;
```

```
    printf("\nInserted -> %d", value);
  }
}

void deQueue() {
  if (front == -1)
    printf("\nQueue is Empty!!");
  else {
    printf("\nDeleted : %d", items[front]);
    front++;
    if (front > rear)
      front = rear = -1;
  }
}

// Function to print the queue
void display() {
  if (rear == -1)
    printf("\nQueue is Empty!!!");
  else {
    int i;
    printf("\nQueue elements are:\n");
    for (i = front; i <= rear; i++)
      printf("%d  ", items[i]);
  }
  printf("\n");
}
```

Output:

Implementation of Binary Tree:

Code:

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

struct Node {
    int a;
    struct Node *left;
    struct Node *right;
};

struct Node *root = NULL;

struct Node* insert() {
    int data;
    struct Node *nn=(struct Node*)malloc(sizeof(struct Node));
    printf("Enter Data [-1 for start inserting left or right]\n");
    scanf("%d", &data);
    if(data == -1)
        return 0;
    nn->a = data;
```

```
    printf("Enter Left Node Data ");
    nn->left=insert();
    printf("Enter Right Node Data ");
    nn->right=insert();
    return nn;
}

void preorder(struct Node *root) {
    if (root == NULL) {
        return;
    }
    printf("%d ", root->a);
    preorder(root->left);
    preorder(root->right);
}
void inorder(struct Node *root) {
    if(root == NULL) {
        return ;
    }
    inorder(root->left);
    printf("%d ", root->a);
    inorder(root->right);
}
void postorder(struct Node *root) {
    if(root == NULL) {
        return ;
    }
```

```
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->a);

}
int main() {

    root = insert();
    printf("Printing the data in the list[preOrder Traversal]\n");
    preorder(root);

    printf("\nPrinting the data in the list[inOrder Traversal]\n");
    inorder(root);

    printf("\nPrinting the data in the list[postOrder Traversal]\n");
    postorder(root);

    return 0;
}
```

Output: