WEEK-7

a.STACK IMPLEMENTATION USING ARRAYS

```
#include <stdio.h>

#define N 100
int stack[N];
int top = -1;

// Function to push an element onto the stack
void push(int data) {
    if (top == N - 1) {
        printf("Stack overflow!\n");
        return;
    }

    top++;
    stack[top] = data;
    printf("Pushed element: %d\n", data);
}

// Function to pop an element from the stack
int pop() {
    if (top == -1) {
        printf("Stack is empty!\n");
        return -1;
    }

    int item = stack[top];
    top--;
    return item;
}

// Function to peek the top element of the stack
int peek() {
    if (top == -1) {
        printf("Stack is empty!\n");
        return -1;
    }
    return stack[top];
}

// Function to display the elements of the stack
void display() {
    if (top == -1) {
        printf("Stack is empty!\n");
        return;
    }

    printf("Stack elements:\n");
    for (int i = top; i >= 0; i--) {
```

```
   printf("%d\n", stack[i]);
    }
}

int main() {
    int choice, data;
    do {
        printf("\n1. Push\n");
        printf("2. Pop\n");
        printf("3. Peek\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &data);
                push(data);
                break;
            case 2:
                printf("Popped element: %d\n", pop());
                break;
            case 3:
                printf("Top element: %d\n", peek());
                break;
            case 4:
                display();
                break;
            case 5:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 5);

    return 0;
}
```

b. Stack Using Linked List

Objective: Implement Stack using Linked List

https://www.hackerrank.com/contests/17cs1102/challenges/6a-stack-using-linkedlist

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

struct Node {
    int data;
    struct Node* prev;
};

struct Node* push(struct Node* top, int data) {
    struct Node* newNode = malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = top;
    return newNode;
}

struct Node* pop(struct Node* top) {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return NULL;
    }
    struct Node* temp = top;
    top = top->prev;
    free(temp);
    return top;
}

void display(struct Node* top) {
    if (top == NULL) {
        printf("Stack is Empty\n");
    } else {
        printf("->%d\n", top->data);
    }
}

int main() {
    struct Node* top = NULL;
    int choice, data;

    do {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf("%d", &data);
                top = push(top, data);
                break;
            case 2:
                top = pop(top);
                break;
            case 3:
                display(top);
                break;
            case 4:
                 break;
        }
    } while (choice != 4);

    return 0;
}
```

Expected Output:

Input:

```
1

10

1

20

1

30

3

2

3

2

3

2

3

4
```

Output:

```
->30

->20

->10

Stack is Empty
```

c. Stack using two Queues

Objective: Implement Stack using two Queues

https://www.hackerrank.com/contests/17cs1102/challenges/6b-implement-stackusing-two-queues

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

#define MAX_SIZE 100

// Structure to represent a queue
struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};

// Function to create a new queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = -1;
    queue->rear = -1;
    return queue;
}

// Function to check if the queue is full
int isFull(struct Queue* queue) {
    return (queue->rear == MAX_SIZE - 1);
}

// Function to check if the queue is empty
int isEmpty(struct Queue* queue) {
    return (queue->front == -1);
}

// Function to add an element to the queue
void enqueue(struct Queue* queue, int value) {
    if (isFull(queue)) {
        printf("Queue is full\n");
    } else {
        if (queue->front == -1) {
            queue->front = 0;
        }
        queue->rear++;
        queue->items[queue->rear] = value;
    }
}

// Function to remove an element from the queue
int dequeue(struct Queue* queue) {
    int item;
    if (isEmpty(queue)) {
        printf("Queue is Empty\n");
    return -1;
    } else {
        item = queue->items[queue->front];
        queue->front++;
        if (queue->front > queue->rear) {
            queue->front = queue->rear = -1;
        }
        return item;
    }
}

// Function to push an element onto the stack
void push(struct Queue* q1, struct Queue* q2, int value) {
    // Move all elements from q1 to q2
    while (!isEmpty(q1)) {
        enqueue(q2, dequeue(q1));
    }

    // Enqueue the new element into q1
    enqueue(q1, value);

    // Move all elements back to q1 from q2
    while (!isEmpty(q2)) {
        enqueue(q1, dequeue(q2));
    }
}

// Function to pop an element from the stack
int pop(struct Queue* q1) {
    if (isEmpty(q1)) {
        printf("Stack is Empty\n");
        return -1;
    }
    return dequeue(q1);
}

// Function to display the top element of the stack
void displayTop(struct Queue* q1) {
    if (isEmpty(q1)) {
        printf("Stack is Empty\n");
    } else {
        printf("->%d\n", q1->items[q1->front]);
    }
}



int main() {
    struct Queue* q1 = createQueue(); // Main queue to act as stack
    struct Queue* q2 = createQueue(); // Auxiliary queue for push operation
    int choice, value;

    do {
        scanf("%d", &choice);

        switch (choice) {
            case 1:

              scanf("%d", &value);
                push(q1, q2, value);
                break;
            case 2: {
                pop(q1);
                break;
            }
            case 3:
                displayTop(q1);
                break;
            case 4:

                break;

        }
    } while (choice != 4);

    free(q1);
    free(q2);

    return 0;
}
```

Expected Output:

Input:

2

4

Output:

Stack is Empty

WEEK-8

a. Queue and its operations using arrays

```
#include <stdio.h>
#define N 5
int queue[N];
int front=-1;
int rear=-1;
void enqueue(int data)
{
   if(rear == N-1)
   {
      printf("Overflow");
   }
   else if(front==-1 && rear==-1)
   {
      front=rear=0;
      queue[rear]=data;
   }
   else
   {
      rear++;
      queue[rear]=data;
   }
}
void deque()
{
   if(front==-1 && rear==-1)
   {
       printf("underflow");
   }
   else if(front==rear)
   {
      front=rear=-1;
   }
   else
   {
      printf("%d",queue[front]);
      front++;
   }
}
void peek()
{
    if(front==-1 && rear==-1)
   {
       printf("Queue is empty");
   }
   else
   {
      printf("%d",queue[front]);
   }
}
void display()
{
   if(front==-1 && rear==-1)
   {
       printf("Queue is empty");
   }
   else
   {
      int i=0;
      for(i=front;i<=rear;i++)
      {
        printf("%d ",queue[i]);
      }
   }
}
int main()
{
   int data,ch;
   do{
        printf("\n Menu");
        printf("\n 1.Enqueue");
        printf("\n 2.Dequeue");
        printf("\n 3.Display");
        printf("\n 4.peek");
        scanf("%d",&ch);
      switch(ch)
      {
       case 1: scanf("%d",&data);
               enqueue(data);
               break;
       case 2: deque();
               break;
       case 3: display();
               break;
       case 4: peek();
               break;
       default : printf("invalid choice");
                 break;
      }
   }
   while(ch!=0);
   return 0;
   }
```

b. Queue Using Linked List Objective: Implement a queue using Linked List https://www.hackerrank.com/contests/17cs1102/challenges/7b-implement-a-queueusing-linked-list

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {


  int data;
    struct node* next;
} node;

node* front, * rear = 0;

void enqueue(int data) {
    node* newnode = (node*)malloc(sizeof(node));
    newnode->data = data;
    newnode->next = 0;

    if (front == 0 && rear == 0) {
        front = rear = newnode;
    }
    else {
        rear->next = newnode;
        rear = newnode;
    }
}

void deque() {

    if (front == 0 && rear == 0) {
        printf("Queue is empty\n");
    }
    else if (front == rear) {
        front = rear = 0;
    }
    else {
        node* temp = front;
        front = front->next;
        free(temp);

    }

}

void display() {
    if (front == 0 && rear == 0) {
        printf("NULL\n");


}
    else {
        node* temp = front;
        while (temp != 0) {
            printf("->%d", temp->data);
            temp = temp->next;



 }
          printf("\n");
    }

}

int main() {
    int data, ch;
    do {
        scanf("%d", &ch);

        switch (ch) {
        case 1:
            scanf("%d", &data);
            enqueue(data);
            break;

        case 2:
            deque();
            break;

        case 3:
            display();
            break;

        case 4:
            exit(0);
            break;
        }
    } while (ch != 4);

    return 0;
}
```

Expected Output:

Input:

```
1

10

1

20

1

30

3

2

3

4
```

Output:

```
->10->20->30
 ->20->30
```

c. Queue using two Stacks

Objective: Implement Queue using two Stacks

https://www.hackerrank.com/contests/17cs1102/challenges/queue-using-two-stacks

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

// Structure to represent a stack node
struct StackNode {
   int data;
   struct StackNode* next;
};

// Function to create a new stack node
struct StackNode* newNode(int data) {
   struct StackNode* stackNode = (struct StackNode*)malloc(sizeof(struct StackNode));
   stackNode->data = data;
   stackNode->next = NULL;
   return stackNode;
}

// Function to push an element onto the stack
void push(struct StackNode** top, int data) {
   struct StackNode* stackNode = newNode(data);
   stackNode->next = *top;
   *top = stackNode;
}

// Function to check if the stack is empty
int isEmpty(struct StackNode* top) {
   return top == NULL;
}

// Function to pop an element from the stack
int pop(struct StackNode** top) {
   if (isEmpty(*top))
      return -1;
   struct StackNode* temp = *top;
   *top = (*top)->next;
   int popped = temp->data;
   free(temp);
   return popped;
}

// Structure to represent a queue
struct Queue {
   struct StackNode* stack1;
   struct StackNode* stack2;
};

// Function to create a new queue
struct Queue* createQueue() {
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
```

```
  queue->stack1 = NULL;
    queue->stack2 = NULL;
    return queue;
}

// Function to enqueue an element into the queue
void enqueue(struct Queue* queue, int x) {
    push(&queue->stack1, x);
}

// Function to dequeue an element from the queue
int dequeue(struct Queue* queue) {
    if (isEmpty(queue->stack1) && isEmpty(queue->stack2)) {
        return -1;
    }
    if (isEmpty(queue->stack2)) {
        while (!isEmpty(queue->stack1)) {
            push(&queue->stack2, pop(&queue->stack1));
        }
    }
    return pop(&queue->stack2);
}

// Function to print the front element of the queue
int front(struct Queue* queue) {
    if (isEmpty(queue->stack1) && isEmpty(queue->stack2)) {
        return -1;
    }
    if (isEmpty(queue->stack2)) {
        while (!isEmpty(queue->stack1)) {
            push(&queue->stack2, pop(&queue->stack1));
        }
    }
    return queue->stack2->data;
}

int main() {
    int queries;
    scanf("%d", &queries);

    struct Queue* queue = createQueue();

    while (queries--) {
        int query, x;
        scanf("%d", &query);
        switch (query) {
            case 1:
                scanf("%d", &x);
                enqueue(queue, x);
                break;
            case 2:
                dequeue(queue);
                break;
```

```
          case 3:
              printf("%d\n", front(queue));
              break;


              default:
              break;
        }
    }

    return 0;
}
```

Expected Output:

Input:

```
STDIN   Function
-----   --------
10      q = 10 (number of queries)
1 42    1st query, enqueue 42
2       dequeue front element
1 14    enqueue 42
3       print the front element
1 28    enqueue 28
3       print the front element
1 60    enqueue 60
1 78    enqueue 78
2       dequeue front element
2       dequeue front element
```

Output:

```
14
14
```

d. Circular Queues
Objective: Implement Circular Queue using Arrays
https://www.hackerrank.com/contests/17cs1102/challenges/7a-circular-queueusing-arryas

```
#include <stdio.h>
#include <stdlib.h>
#define N 5
int queue[N];
int front=-1;
int rear=-1;
void enqueue(int data)
{
   if((rear+1)%N==front)
   {
      printf("Queue Overflow\n");
   }
```

```
    else if(front==-1 && rear==-1)
    {
        front=rear=0;
        queue[rear]=data;
    }
    else
    {
       rear=(rear+1)%N;
        queue[rear]=data;
    }
}
void deque()
{
    if(front==-1 && rear==-1)
    {
         printf("Queue Underflow\n");
    }
    else if(front==rear)
    {
        front=rear=-1;
    }
    else
    {

        front=(front+1)%N;
    }
}
void display()
{
    if(front==-1 && rear==-1)
    {
         printf("NULL");
    }
    else
    {
        int i=front;
        while(i!=rear)
        {
          printf("%d ",queue[i]);
            i=(i+1)%N;
        }
        printf("%d ",queue[rear]);
    }
    printf("\n");
}
int main()
{
    int data,ch;
    do{
          scanf("%d",&ch);
        switch(ch)
        {
```

```
  case 1 :
            scanf("%d",&data);
                 enqueue(data);
                 break;
         case 2: deque();
                 break;
         case 3: display();
                 break;
         case 4: exit(0);
                 break;
        }
    }while(ch!=0);
    return 0;
}
```

Expected Output:

Input:

```
1

10

1

20

1

30

3

4
```

Output:

```
10 20 30
```

WEEK-9

a.Towers of Hanoi Using Stack Objective: Implement Towers of Hanoi using Stack https://www.hackerrank.com/contests/17cs1102/challenges/6c-towers-of-hanoiusing-stack

```
#include <stdio.h>
void towerOfHanoi(int N, char from_rod, char to_rod, char aux_rod) {
   if (N == 1) {
      printf("MOVE T%c T%c\n", from_rod, to_rod);
      return;
   }
   towerOfHanoi(N-1, from_rod, aux_rod, to_rod);
   printf("MOVE T%c T%c\n", from_rod, to_rod);
   towerOfHanoi(N-1, aux_rod, to_rod, from_rod);
}

int main() {
   int N;
   scanf("%d", &N);
   towerOfHanoi(N, 'T1', 'T3', 'T2');
   return 0;
}
```

Expected Output:

Input:

2

Output:

```
MOVE T1 T2
MOVE T1 T3
MOVE T2 T3
```

b. Balanced Brackets Objective: Given strings of brackets, determine whether each sequence of brackets is balanced.

https://www.hackerrank.com/contests/17cs1102/challenges/balanced-brackets

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

#define MAX_SIZE 10000

bool areBracketsBalanced(char expr[ ]) {
    char stack[MAX_SIZE];
    int top = -1;
    for (int i = 0; expr[i] != '\0'; i++) {
        if (expr[i] == '(' || expr[i] == '[' || expr[i] == '{') {
            stack[++top] = expr[i];
        } else if (expr[i] == ')' || expr[i] == ']' || expr[i] == '}') {
            if (top == -1 ||
                (expr[i] == ')' && stack[top] != '(') ||
                (expr[i] == ']' && stack[top] != '[') ||
 (expr[i] == '}' && stack[top] != '{')) {
                return false;
            }
            top--;
        }
    }
    return (top == -1);
}

int main() {
    int testCases;
    scanf("%d", &testCases);
    getchar(); // Consume newline character left in buffer

    for (int t = 1; t <= testCases; t++) {
        char expr[MAX_SIZE];
        fgets(expr, sizeof(expr), stdin);

        if (areBracketsBalanced(expr)) {
            printf("YES\n");
        } else {
            printf("NO\n");
        }
    }

    return 0;
}
```

Expected Output:

Input:

```
STDIN          Function
-----          --------
3              n = 3
{[()]}         first s = '{[()]}'
{[(])}         second s = '{[(])}'
{{[[(())]]}}   third s ='{{[[(())]]}}'
```

Output:

```
YES
NO
YES
```

WEEK-10

a. Infix to Postfix

Objective: Convert an infix expression into postfix expression.

https://www.hackerrank.com/contests/17cs1102/challenges/8b-infix-to-postfix

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

#define MAX_SIZE 100

// Function to return the precedence of operators
int precedence(char op) {
    if (op == '^')
        return 3;
    else if (op == '*' || op == '/')
        return 2;
    else if (op == '+' || op == '-')
        return 1;
    else
        return -1;
}

// Function to convert infix to postfix
void infixToPostfix(char *infix, char *postfix) {
    char stack[MAX_SIZE];
    int top = -1;
    int i, j;

    for (i = 0, j = 0; infix[i] != '?'; i++) {
        if (infix[i] == ' ')
            continue;

        if (infix[i] >= '0' && infix[i] <= '9') {
            while (infix[i] >= '0' && infix[i] <= '9') {
                postfix[j++] = infix[i++];
            }
            postfix[j++] = ' ';
            i--; // Move back one position to process the operator or parenthesis
        } else if (infix[i] == '(') {
            stack[++top] = infix[i];
        } else if (infix[i] == ')') {
            while (top != -1 && stack[top] != '(') {
                postfix[j++] = stack[top--];
                postfix[j++] = ' ';
            }
            if (top != -1 && stack[top] == '(') {
                top--; // Discard '('
            }
} else {
            while (top != -1 && precedence(stack[top]) >= precedence(infix[i])) {
                postfix[j++] = stack[top--];
                postfix[j++] = ' ';
            }
            stack[++top] = infix[i];
        }
    }

    while (top != -1) {
        postfix[j++] = stack[top--];
        postfix[j++] = ' ';
    }
    postfix[j] = '\0'; // Add null terminator
}

int main() {
    int N;
    scanf("%d", &N);
    getchar(); // Consume newline character

    for (int t = 0; t < N; t++) {
        char infix[MAX_SIZE], postfix[MAX_SIZE];
        fgets(infix, MAX_SIZE, stdin);

        infixToPostfix(infix, postfix);

        printf("%s\n", postfix);
    }

    return 0;
}
```

Expected Output:

Input:

```
1

31 * ( 4 + 50 ) ?
```

Output:

```
31 4 50 + *
```

b. Postfix Expression Evaluation

Objective: Implement a program to evaluate a postfix expression.

https://www.hackerrank.com/contests/17cs1102/challenges/8-c-postfix-expressionevaluation

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h> // Include ctype.h for isdigit function

#define MAX_SIZE 100

// Function to evaluate postfix expression
int evaluatePostfix(char *postfix) {
    int stack[MAX_SIZE];
    int top = -1;
    char *token = strtok(postfix, " ");

    while (token != NULL) {
        if (isdigit((unsigned char)token[0])) { // Use isdigit properly
            stack[++top] = atoi(token);
        } else {
            int operand2 = stack[top--];
            int operand1 = stack[top--];
            switch (token[0]) {
                case '+':
                    stack[++top] = operand1 + operand2;
                    break;
                case '-':
                    stack[++top] = operand1 - operand2;
                    break;
                case '*':
                    stack[++top] = operand1 * operand2;
                    break;
                case '/':
                    stack[++top] = operand1 / operand2;
                    break;
                default:
                    printf("Invalid operator\n");
                    exit(1);
            }
        }
        token = strtok(NULL, " ");
    }

    return stack[top];
}

int main() {
    int N;
    scanf("%d", &N);
    getchar(); // Consume newline character
for (int t = 0; t < N; t++) {
        char postfix[MAX_SIZE];
        fgets(postfix, MAX_SIZE, stdin);
        *strchr(postfix, '?') = '\0'; // Replace '?' with '\0'
        printf("%d\n", evaluatePostfix(postfix));
    }

    return 0;
}
```

Expected Output:

Input:

```
1

31 * ( 4 + 50 ) ?
```

Output:

```
1674
```