**I (a) ARRAY IMPLEMENTATION OF STACK**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 10**

**int stack[MAX];**

**int top = -1;**

**void push(int value) {**

**if (top == MAX - 1) {**

**printf("Error: Stack overflow!\n");**

**return;**

**}**

**stack[++top] = value;**

**}**

**int pop() {**

**if (top == -1) {**

**printf("Error: Stack underflow!\n");**

**return -1;**

**}**

**return stack[top--];**

**}**

**int main() {**

**int choice, value;**

**while (1) {**

**printf("1. Push\n2. Pop\n3. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**printf("Enter value to push: ");**

**scanf("%d", &value);**

**push(value);**

**break;**

**case 2:**

**value = pop();**

**if (value != -1) {**

**printf("Popped value: %d\n", value);**

**}**

**break;**

**case 3:**

**exit(0);**

**default:**

**printf("Invalid choice!\n");**

**}**

**}**

**return 0;**

**}**

**OUTPUT**

**1. Push**

**2. Pop**

**3. Exit**

**Enter your choice: 1**

**Enter value to push: 25**

**1. Push**

**2. Pop**

**3. Exit**

**Enter your choice: 3**

**I(b) ARRAY IMPLEMENTATION OF QUEUE**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 100 // Define the maximum size of the queue**

**typedef struct {**

**int items[MAX];**

**int front, rear;**

**} Queue;**

**// Function to initialize the queue**

**void initializeQueue(Queue\* q) {**

**q->front = -1;**

**q->rear = -1;**

**}**

**// Function to check if the queue is empty**

**int isEmpty(Queue\* q) {**

**return q->front == -1;**

**}**

**// Function to check if the queue is full**

**int isFull(Queue\* q) {**

**return q->rear == MAX - 1;**

**}**

**// Function to add an element to the queue**

**void enqueue(Queue\* q, int value) {**

**if (isFull(q)) {**

**printf("Queue is full. Cannot enqueue %d\n", value);**

**return;**

**}**

**if (isEmpty(q)) {**

**q->front = 0; // Initialize front when the first element is enqueued**

**}**

**q->rear++;**

**q->items[q->rear] = value;**

**printf("%d enqueued to queue\n", value);**

**}**

**// Function to remove an element from the queue**

**int dequeue(Queue\* q) {**

**if (isEmpty(q)) {**

**printf("Queue is empty. Cannot dequeue\n");**

**return -1; // Indicate an error**

**}**

**int item = q->items[q->front];**

**q->front++;**

**// If the queue is empty after dequeuing, reset it**

**if (q->front > q->rear) {**

**initializeQueue(q);**

**}**

**return item;**

**}**

**// Function to get the front element of the queue without removing it**

**int peek(Queue\* q) {**

**if (isEmpty(q)) {**

**printf("Queue is empty\n");**

**return -1; // Indicate an error**

**}**

**return q->items[q->front];**

**}**

**// Function to display the contents of the queue**

**void displayQueue(Queue\* q) {**

**if (isEmpty(q)) {**

**printf("Queue is empty\n");**

**return;**

**}**

**printf("Queue elements are:\n");**

**for (int i = q->front; i <= q->rear; i++) {**

**printf("%d ", q->items[i]);**

**}**

**printf("\n");**

**}**

**// Main function to demonstrate the queue operations**

**int main() {**

**Queue q;**

**initializeQueue(&q);**

**enqueue(&q, 10);**

**enqueue(&q, 20);**

**enqueue(&q, 30);**

**displayQueue(&q);**

**printf("%d dequeued from queue\n", dequeue(&q));**

**displayQueue(&q);**

**printf("Front item is %d\n", peek(&q));**

**displayQueue(&q);**

**return 0;**

**}**

**OUTPUT**

**10 enqueued to queue**

**20 enqueued to queue**

**30 enqueued to queue**

**Queue elements are:**

**10 20 30**

**10 dequeued from queue**

**Queue elements are:**

**20 30**

**Front item is 20**

**Queue elements are:**

**20 30**

**I(c) ARRAY IMPLEMENATION OD CIRCULAR QUEUE ADT**

**#include<stdio.h>**

**#define capacity 6**

**int queue[capacity];**

**int front = -1, rear = -1;**

**// Here we check if the Circular queue is full or not**

**int checkFull ()**

**{**

**if ((front == rear + 1) || (front == 0 && rear == capacity - 1))**

**{**

**return 1;**

**}**

**return 0;**

**}**

**// Here we check if the Circular queue is empty or not**

**int checkEmpty ()**

**{**

**if (front == -1)**

**{**

**return 1;**

**}**

**return 0;**

**}**

**// Addtion in the Circular Queue**

**void enqueue (int value)**

**{**

**if (checkFull ())**

**printf ("Overflow condition\n");**

**else**

**{**

**if (front == -1)**

**front = 0;**

**rear = (rear + 1) % capacity;**

**queue[rear] = value;**

**printf ("%d was enqueued to circular queue\n", value);**

**}**

**}**

**// Removal from the Circular Queue**

**int dequeue ()**

**{**

**int variable;**

**if (checkEmpty ())**

**{**

**printf ("Underflow condition\n");**

**return -1;**

**}**

**else**

**{**

**variable = queue[front];**

**if (front == rear)**

**{**

**front = rear = -1;**

**}**

**else**

**{**

**front = (front + 1) % capacity;**

**}**

**printf ("%d was dequeued from circular queue\n", variable);**

**return 1;**

**}**

**}**

**// Display the queue**

**void print ()**

**{**

**int i;**

**if (checkEmpty ())**

**printf ("Nothing to dequeue\n");**

**else**

**{**

**printf ("\nThe queue looks like: \n");**

**for (i = front; i != rear; i = (i + 1) % capacity)**

**{**

**printf ("%d ", queue[i]);**

**}**

**printf ("%d \n\n", queue[i]);**

**}**

**}**

**int main ()**

**{**

**// Not possible as the Circular queue is empty**

**dequeue ();**

**enqueue (15);**

**enqueue (20);**

**enqueue (25);**

**enqueue (30);**

**enqueue (35);**

**print ();**

**dequeue ();**

**dequeue ();**

**print ();**

**enqueue (40);**

**enqueue (45);**

**enqueue (50);**

**enqueue (55); //Overflow condition**

**print ();**

**return 0;**

**}**

**OUTPUT:**

**Underflow condition**

**15 was enqueued to circular queue**

**20 was enqueued to circular queue**

**25 was enqueued to circular queue**

**30 was enqueued to circular queue**

**35 was enqueued to circular queue**

**The queue looks like:**

**15 20 25 30 35**

**15 was dequeued from circular queue**

**20 was dequeued from circular queue**

**The queue looks like:**

**25 30 35**

**40 was enqueued to circular queue**

**45 was enqueued to circular queue**

**50 was enqueued to circular queue**

**Overflow condition**

**The queue looks like:**

**25 30 35 40 45 50**

**(2) IMPLEMENTATION OF SINGLY LINKED LIST**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct Node**

**{**

**int data;**

**struct Node \*next;**

**};**

**void deleteStart (struct Node \*\*head)**

**{**

**struct Node \*temp = \*head;**

**// if there are no nodes in Linked List can't delete**

**if (\*head == NULL)**

**{**

**printf ("Linked List Empty, nothing to delete");**

**return;**

**}**

**// move head to next node**

**\*head = (\*head)->next;**

**printf ("\n%d deleted\n", temp->data);**

**free (temp);**

**}**

**void insertStart (struct Node \*\*head, int data)**

**{**

**// dynamically create memory for this newNode**

**struct Node \*newNode = (struct Node \*) malloc (sizeof (struct Node));**

**// assign data value**

**newNode->data = data;**

**// change the next node of this newNode**

**// to current head of Linked List**

**newNode->next = \*head;**

**//re-assign head to this newNode**

**\*head = newNode;**

**printf ("\n%d Inserted\n", newNode->data);**

**}**

**void display (struct Node \*node)**

**{**

**printf ("\nLinked List: ");**

**// as linked list will end when Node is Null**

**while (node != NULL)**

**{**

**printf ("%d ", node->data);**

**node = node->next;**

**}**

**printf ("\n");**

**}**

**int main ()**

**{**

**struct Node \*head = NULL;**

**// Need '&' i.e. address as we need to change head**

**insertStart (&head, 100);**

**insertStart (&head, 80);**

**insertStart (&head, 60);**

**insertStart (&head, 40);**

**insertStart (&head, 20);**

**// No Need for '&' as not changing head in display operation**

**display (head);**

**deleteStart (&head);**

**deleteStart (&head);**

**display (head);**

**return 0;**

**}**

**OUTPUT**

**100 Inserted**

**80 Inserted**

**60 Inserted**

**40 Inserted**

**20 Inserted**

**Linked List: 20 40 60 80 100**

**20 deleted**

**40 deleted**

**Linked List: 60 80 100**

**3(a) LINKED LIST IMPLEMENTATION OF STACK ADTs**

**#include <stdio.h>**

**#include <stdlib.h>**

**// Define the structure for a stack node**

**typedef struct Node {**

**int data;**

**struct Node\* next;**

**} Node;**

**// Define the structure for the stack**

**typedef struct {**

**Node\* top;**

**} Stack;**

**// Function to initialize the stack**

**void initializeStack(Stack\* s) {**

**s->top = NULL;**

**}**

**// Function to check if the stack is empty**

**int isStackEmpty(Stack\* s) {**

**return s->top == NULL;**

**}**

**// Function to push an element onto the stack**

**void push(Stack\* s, int value) {**

**Node\* newNode = (Node\*)malloc(sizeof(Node));**

**if (!newNode) {**

**printf("Memory allocation failed\n");**

**return;**

**}**

**newNode->data = value;**

**newNode->next = s->top;**

**s->top = newNode;**

**printf("%d pushed to stack\n", value);**

**}**

**// Function to pop an element from the stack**

**int pop(Stack\* s) {**

**if (isStackEmpty(s)) {**

**printf("Stack is empty\n");**

**return -1; // Indicate an error**

**}**

**Node\* temp = s->top;**

**int value = temp->data;**

**s->top = s->top->next;**

**free(temp);**

**return value;**

**}**

**// Function to get the top element of the stack without removing it**

**int peek(Stack\* s) {**

**if (isStackEmpty(s)) {**

**printf("Stack is empty\n");**

**return -1; // Indicate an error**

**}**

**return s->top->data;**

**}**

**// Function to display the stack elements**

**void displayStack(Stack\* s) {**

**if (isStackEmpty(s)) {**

**printf("Stack is empty\n");**

**return;**

**}**

**Node\* temp = s->top;**

**printf("Stack elements are:\n");**

**while (temp) {**

**printf("%d ", temp->data);**

**temp = temp->next;**

**}**

**printf("\n");**

**}**

**// Main function to demonstrate stack operations**

**int main() {**

**Stack s;**

**initializeStack(&s);**

**push(&s, 10);**

**push(&s, 20);**

**push(&s, 30);**

**displayStack(&s);**

**printf("%d popped from stack\n", pop(&s));**

**displayStack(&s);**

**printf("Top element is %d\n", peek(&s));**

**displayStack(&s);**

**return 0;**

**}**

**Output**

**10 pushed to stack**

**20 pushed to stack**

**30 pushed to stack**

**Stack elements are:**

**30 20 10**

**30 popped from stack**

**Stack elements are:**

**20 10**

**Top element is 20**

**Stack elements are:**

**20 10**

**3(b) LINKED LIST IMPLEMENTATION OF LINEAR QUEUE ADTs**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node**

**{**

**int data;**

**struct node \*next;**

**};**

**struct node \*front = NULL, \*rear = NULL;**

**void enqueue(int val)**

**{**

**struct node \*newNode = malloc(sizeof(struct node));**

**newNode->data = val;**

**newNode->next = NULL;**

**if(front == NULL && rear == NULL)**

**front = rear = newNode;**

**else**

**{**

**rear->next = newNode;**

**rear = newNode;**

**}**

**}**

**void dequeue()**

**{**

**struct node \*temp;**

**if(front == NULL)**

**printf("Queue is Empty. Unable to perform dequeue\n");**

**else**

**{**

**temp = front;**

**front = front->next;**

**if(front == NULL)**

**rear = NULL;**

**free(temp);**

**}**

**}**

**void printList()**

**{**

**struct node \*temp = front;**

**while(temp)**

**{**

**printf("%d->",temp->data);**

**temp = temp->next;**

**}**

**printf("NULL\n");**

**}**

**int main()**

**{**

**enqueue(10);**

**enqueue(20);**

**enqueue(30);**

**printf("Queue :");**

**printList();**

**dequeue();**

**printf("After dequeue the new Queue :");**

**printList();**

**dequeue();**

**printf("After dequeue the new Queue :");**

**printList();**

**return 0;**

**}**

**Output**

**Queue :10->20->30->NULL**

**After dequeue the new Queue :20->30->NULL**

**After dequeue the new Queue :30->NULL**

1. **IMPLEMENTATION OFPOLYNOMIAL MANIPULATION USING LINKED LIST**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct Node**

**{**

**int coeff;**

**int pow;**

**struct Node\* next;**

**};**

**void readPolynomial(struct Node\*\* poly)**

**{**

**int coeff, exp, cont;**

**struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));**

**\*poly = temp;**

**do{**

**printf("\n Coeffecient: ");**

**scanf("%d", &coeff);**

**printf("\n Exponent: ");**

**scanf("%d", &exp);**

**temp->coeff = coeff;**

**temp->pow = exp;**

**temp-> next = NULL;**

**printf("\nHave more terms? 1 for y and 0 for no: ");**

**scanf("%d", &cont);**

**if(cont)**

**{**

**temp->next = (struct Node\*)malloc(sizeof(struct Node));**

**temp = temp->next;**

**temp->next = NULL;**

**}**

**}while(cont);**

**}**

**void displayPolynomial(struct Node\* poly)**

**{**

**printf("\nPolynomial expression is: ");**

**while(poly != NULL)**

**{**

**printf("%dX^%d", poly->coeff, poly->pow);**

**poly = poly->next;**

**if(poly != NULL)**

**printf("+");**

**}**

**}**

**void addPolynomials(struct Node\*\* result, struct Node\* first, struct Node\* second)**

**{**

**struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));**

**temp->next = NULL;**

**\*result = temp;**

**while(first && second)**

**{**

**if(first->pow > second->pow)**

**{**

**temp->coeff = first->coeff;**

**temp->pow = first->pow;**

**first = first->next;**

**}**

**else if(first->pow < second->pow)**

**{**

**temp->coeff = second->coeff;**

**temp->pow = second->pow;**

**second = second->next;**

**}**

**else**

**{**

**temp->coeff = first->coeff + second->coeff;**

**temp->pow = first->pow;**

**first = first->next;**

**second = second->next;**

**}**

**if(first && second)**

**{**

**temp->next = (struct Node\*)malloc(sizeof(struct Node));**

**temp = temp->next;**

**temp->next = NULL;**

**}**

**}**

**while(first || second)**

**{**

**temp->next = (struct Node\*)malloc(sizeof(struct Node));**

**temp = temp->next;**

**temp->next = NULL;**

**if(first)**

**{**

**temp->coeff = first->coeff;**

**temp->pow = first->pow;**

**first = first->next;**

**}**

**else if(second)**

**{**

**temp->coeff = second->coeff;**

**temp->pow = second->pow;**

**second = second->next;**

**}**

**}**

**}**

**int main()**

**{**

**struct Node\* first = NULL;**

**struct Node\* second = NULL;**

**struct Node\* result = NULL;**

**printf("\nEnter the corresponding data:-\n");**

**printf("\nFirst polynomial:\n");**

**readPolynomial(&first);**

**displayPolynomial(first);**

**printf("\nSecond polynomial:\n");**

**readPolynomial(&second);**

**displayPolynomial(second);**

**addPolynomials(&result, first, second);**

**displayPolynomial(result);**

**return 0;**

**}**

**OUTPUT**

**Enter the corresponding data:-**

**First polynomial:**

**Coeffecient: 5**

**Exponent: 6**

**Have more terms? 1 for y and 0 for no: 1**

**Coeffecient: 565**

**Exponent: 4**

**Have more terms? 1 for y and 0 for no: 0**

**Polynomial expression is: 5X^6+565X^4**

**Second polynomial:**

**Coeffecient: 45**

**Exponent: 6**

**Have more terms? 1 for y and 0 for no: 0**

**Polynomial expression is: 45X^6**

**Polynomial expression is: 50X^6+565X^4**

1. **IMPLEMENTATION OF EVALUATING POSTFIX EXPRESSIONS**

// C program to evaluate value of a postfix expression

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Stack type

**struct** Stack {

**int** top;

    unsigned capacity;

**int**\* array;

};

// Stack Operations

**struct** Stack\* createStack(unsigned capacity)

{

**struct** Stack\* stack

        = (**struct** Stack\*)**malloc**(**sizeof**(**struct** Stack));

**if** (!stack)

**return** NULL;

    stack->top = -1;

    stack->capacity = capacity;

    stack->array

        = (**int**\*)**malloc**(stack->capacity \* **sizeof**(**int**));

**if** (!stack->array)

**return** NULL;

**return** stack;

}

**int** isEmpty(**struct** Stack\* stack)

{

**return** stack->top == -1;

}

**char** peek(**struct** Stack\* stack)

{

**return** stack->array[stack->top];

}

**char** pop(**struct** Stack\* stack)

{

**if** (!isEmpty(stack))

**return** stack->array[stack->top--];

**return** '$';

}

**void** push(**struct** Stack\* stack, **char** op)

{

    stack->array[++stack->top] = op;

}

// The main function that returns value

// of a given postfix expression

**int** evaluatePostfix(**char**\* **exp**)

{

    // Create a stack of capacity equal to expression size

**struct** Stack\* stack = createStack(**strlen**(**exp**));

**int** i;

    // See if stack was created successfully

**if** (!stack)

**return** -1;

    // Scan all characters one by one

**for** (i = 0; **exp**[i]; ++i) {

        // If the scanned character is an operand

        // (number here), push it to the stack.

**if** (**isdigit**(**exp**[i]))

            push(stack, **exp**[i] - '0');

        // If the scanned character is an operator,

        // pop two elements from stack apply the operator

**else** {

**int** val1 = pop(stack);

**int** val2 = pop(stack);

**switch** (**exp**[i]) {

**case** '+':

                push(stack, val2 + val1);

**break**;

**case** '-':

                push(stack, val2 - val1);

**break**;

**case** '\*':

                push(stack, val2 \* val1);

**break**;

**case** '/':

                push(stack, val2 / val1);

**break**;

            }

        }

    }

**return** pop(stack);

}

// Driver code

**int** main()

{

**char** **exp**[] = "231\*+9-";

    // Function call

**printf**("postfix evaluation: %d", evaluatePostfix(**exp**));

**return** 0;

}

**OUTPUT**

**postfix evaluation: -4**

**5(b) IMPLEMENTATION OF INFIX TO POSTFIX CONVERSION**

**#include<stdio.h>**

**#include<ctype.h>**

**char stack[100];**

**int top = -1;**

**void push(char x)**

**{**

**stack[++top] = x;**

**}**

**char pop()**

**{**

**if(top == -1)**

**return -1;**

**else**

**return stack[top--];**

**}**

**int priority(char x)**

**{**

**if(x == '(')**

**return 0;**

**if(x == '+' || x == '-')**

**return 1;**

**if(x == '\*' || x == '/')**

**return 2;**

**return 0;**

**}**

**int main()**

**{**

**char exp[100];**

**char \*e, x;**

**printf("Enter the expression : ");**

**scanf("%s",exp);**

**printf("\n");**

**e = exp;**

**while(\*e != '\0')**

**{**

**if(isalnum(\*e))**

**printf("%c ",\*e);**

**else if(\*e == '(')**

**push(\*e);**

**else if(\*e == ')')**

**{**

**while((x = pop()) != '(')**

**printf("%c ", x);**

**}**

**else**

**{**

**while(priority(stack[top]) >= priority(\*e))**

**printf("%c ",pop());**

**push(\*e);**

**}**

**e++;**

**}**

**while(top != -1)**

**{**

**printf("%c ",pop());**

**}return 0;**

**}**

**OUTPUT**

**Enter the expression : (a+b)\*c/d+e/f**

**a b + c \* d / e f / +**