Data structures and Algorithms Lab
Assessment - 1

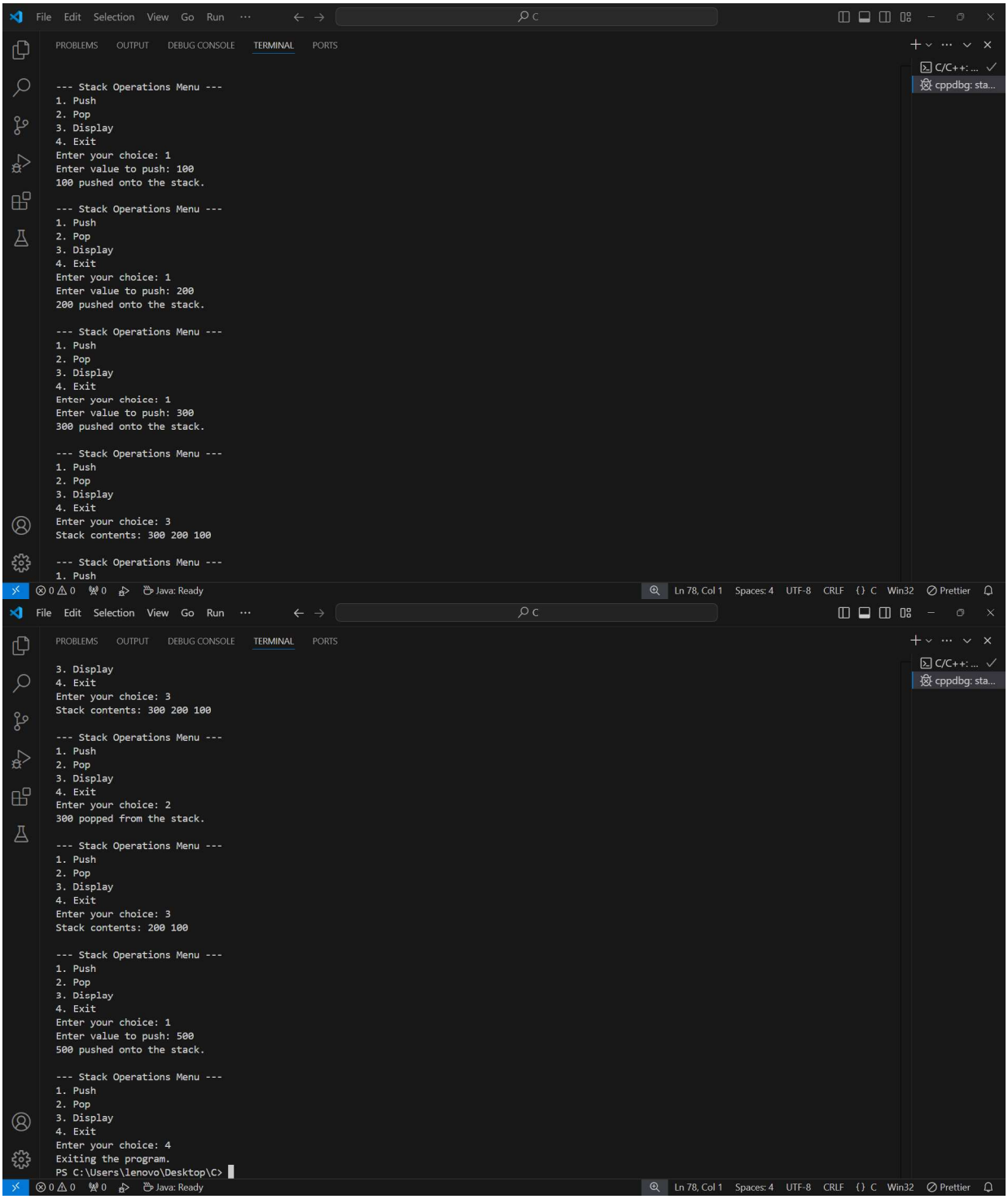
PROGRAM-1

Stack

```c
#include <stdio.h>

// Stack structure
struct stack {
    int elements[5];
    int top;
} myStack;

// Function to push an element onto the stack
void push() {
    int value;
    if (myStack.top == 4) {
        printf("Stack is full!\n");
    } else {
        printf("Enter value to push: ");
        scanf("%d", &value);
        myStack.elements[++myStack.top] = value;
        printf("%d pushed onto the stack.\n", value);
    }
}

// Function to pop an element from the stack
void pop() {
    if (myStack.top == -1) {
        printf("Stack is empty!\n");
    } else {
        int poppedValue = myStack.elements[myStack.top--];
        printf("%d popped from the stack.\n", poppedValue);
    }
```

```
}

// Function to display the stack
void display() {
    if (myStack.top == -1) {
        printf("Stack is empty!\n");
    } else {
        printf("Stack contents: ");
        for (int i = myStack.top; i >= 0; i--) {
            printf("%d ", myStack.elements[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice;
    myStack.top = -1;

    do {
        printf("\n--- Stack Operations Menu ---\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice! Please enter a number between 1 and
4.\n");
        }
```

```
    } while (choice != 4);

    return 0;
}
```

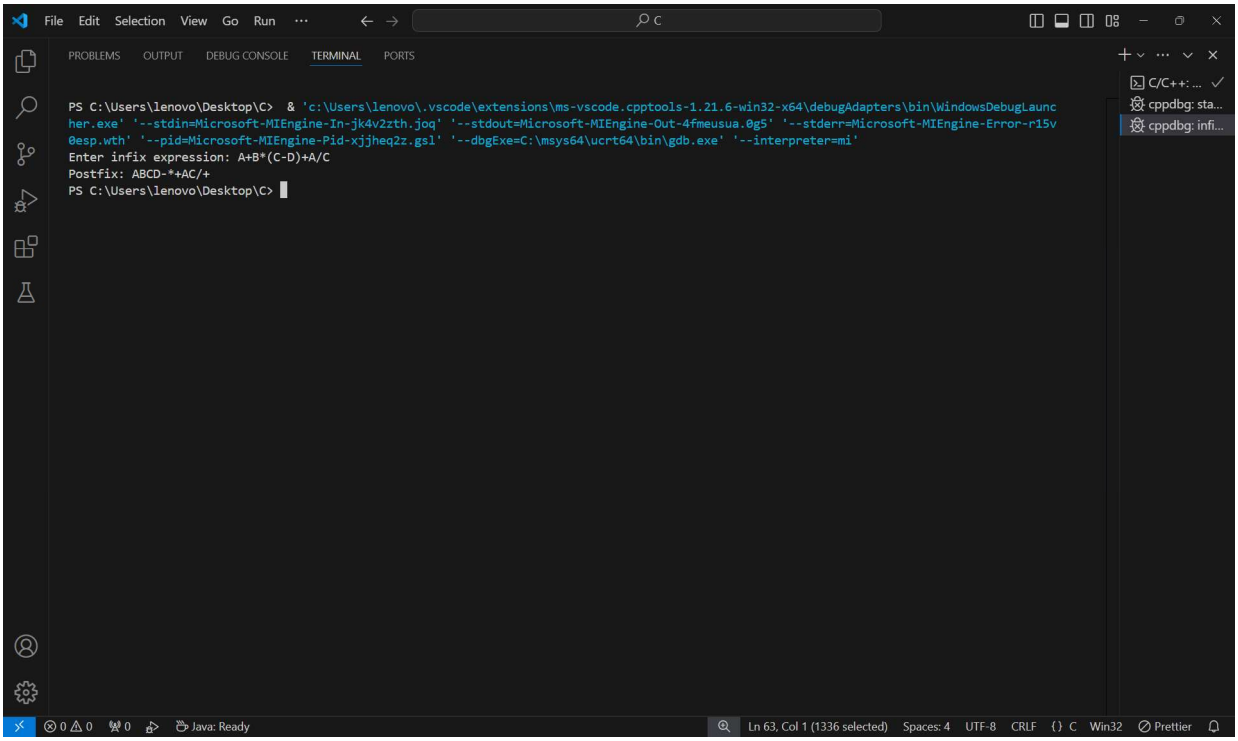


PROGRAM-2

Infix to Postfix

```c
#include <stdio.h>
struct Stack {
    char a[20];
    int top;
} st;

char postfix[20];
char infix[20];

void push(char val) {
    st.a[++st.top] = val;
}

char pop(){
    return st.a[st.top--];
}

int isOperand(char c){
    if(c == '/' || c == '*') return 2;
    if(c == '+' || c == '-') return 1;
    return 0;
}

void main(){
    int k = -1;
    st.top = -1;
    int i, op;

    printf("Enter infix expression: ");
    scanf("%s", infix);

    for(i = 0; infix[i] != '\0'; i++){
        op = isOperand(infix[i]);

        if(op != 0){
            while(st.top != -1 && op <= isOperand(st.a[st.top])){
                postfix[++k] = pop();
            }
            push(infix[i]);
        }
```

```
        else if(infix[i] == '('){
            push(infix[i]);
        }
        else if(infix[i] == ')'){
            while(st.top != -1 && st.a[st.top] != '('){
                postfix[++k] = pop();
            }
            pop(); // Remove '(' from stack
        }
        else{
            postfix[++k] = infix[i];
        }
    }

    // Pop all the remaining operators from the stack
    while(st.top != -1){
        postfix[++k] = pop();
    }

    postfix[++k] = '\0'; // Null-terminate the postfix expression
    printf("Postfix: %s", postfix);
}
```

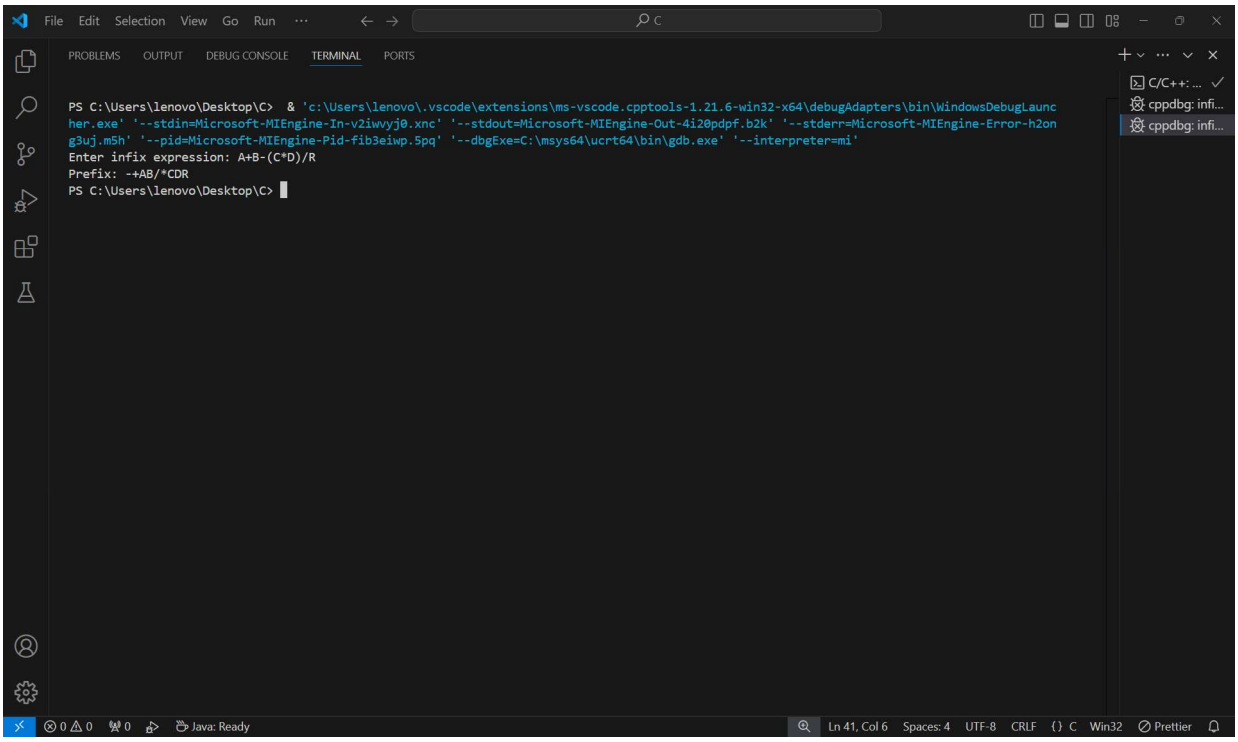


PROGRAM-3

Infix to Prefix

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

struct Stack {
    char a[15];
    int top;
} st;

char prefix[20];
char infix[20];

void push(char val) {
    st.a[++st.top] = val;
}

char pop() {
    return st.a[st.top--];
}

int isOperand(char c) {
    if (c == '/' || c == '*') return 2;
    if (c == '+' || c == '-') return 1;
    return 0;
}

void reverse(char *exp) {
    int len = strlen(exp);
    for (int i = 0; i < len / 2; i++) {
        char temp = exp[i];
        exp[i] = exp[len - i - 1];
        exp[len - i - 1] = temp;
    }
}

void replaceParentheses(char *exp) {
    for (int i = 0; exp[i] != '\0'; i++) {
        if (exp[i] == '(')
            exp[i] = ')';
        else if (exp[i] == ')')
            exp[i] = '(';
    }
}
```

```
void infixToPrefix() {
    int k = -1;
    st.top = -1;
    int i, op;

    reverse(infix);
    replaceParentheses(infix);

    for (i = 0; infix[i] != '\0'; i++) {
        op = isOperand(infix[i]);

        if (op != 0) {
            while (st.top != -1 && op < isOperand(st.a[st.top])) {
                prefix[++k] = pop();
            }
            push(infix[i]);
        } else if (infix[i] == '(') {
            push(infix[i]);
        } else if (infix[i] == ')') {
            while (st.top != -1 && st.a[st.top] != '(') {
                prefix[++k] = pop();
            }
            pop(); // Remove '(' from stack
        } else {
            prefix[++k] = infix[i];
        }
    }
    while (st.top != -1) {
        prefix[++k] = pop();
    }

    prefix[++k] = '\0';
    reverse(prefix); // Get the final prefix expression
}

void main() {
    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixToPrefix();

    printf("Prefix: %s", prefix);
}
```

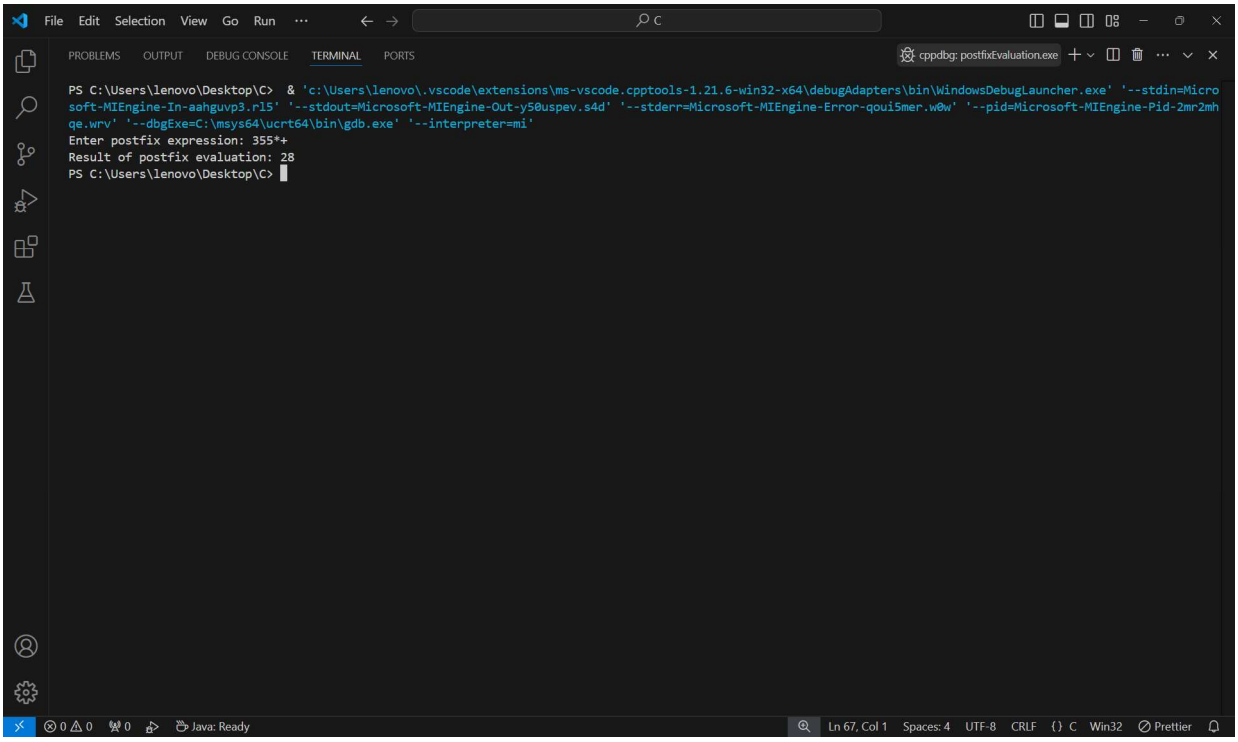


PROGRAM-4
POSTFIX EVALUATION

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

// Stack structure to hold operands
struct Stack {
    int elements[20];
    int top;
} operandStack;

// Function to push a value onto the stack
void push(int value) {
    operandStack.elements[++operandStack.top] = value;
}

// Function to pop a value from the stack
int pop() {
    return operandStack.elements[operandStack.top--];
}

// Function to evaluate a postfix expression
```

```c
int evaluatePostfix(char* postfixExpr) {
    operandStack.top = -1; // Initialize the stack

    for (int i = 0; postfixExpr[i] != '\0'; i++) {
        // If the character is an operand (number), push it onto the stack
        if (isdigit(postfixExpr[i])) {
            push(postfixExpr[i] - '0'); // Convert char to int
        }
        // If the character is an operator, pop two operands and apply the
operator
        else {
            int operand2 = pop();
            int operand1 = pop();
            int result;

            switch (postfixExpr[i]) {
                case '+':
                    result = operand1 + operand2;
                    break;
                case '-':
                    result = operand1 - operand2;
                    break;
                case '*':
                    result = operand1 * operand2;
                    break;
                case '/':
                    result = operand1 / operand2;
                    break;
            }
            // Push the result back onto the stack
            push(result);
        }
    }
    // The final result is the only value left in the stack
    return pop();
}

int main() {
    char postfixExpr[20];
    printf("Enter postfix expression: ");
    scanf("%s", postfixExpr);

    int result = evaluatePostfix(postfixExpr);
    printf("Result of postfix evaluation: %d\n", result);
```

```
    return 0;
}
```

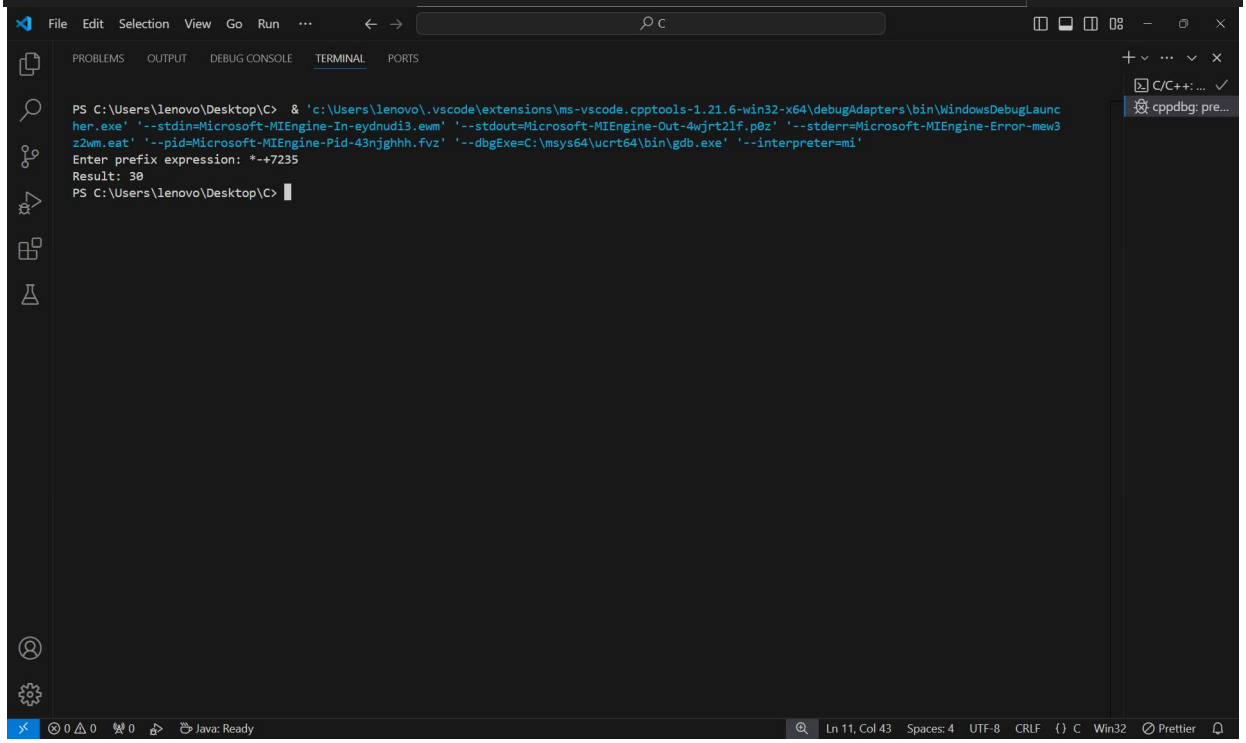


PROGRAM-5

Prefix Evaluation

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

struct Stack {
    int data[20];
    int index;
} numStack;

void push(int num) {
    numStack.data[++numStack.index] = num;
}

int pop() {
    return numStack.data[numStack.index--];
}

int evaluatePrefix(char* expression) {
    numStack.index = -1;
    int length = strlen(expression);

    for (int i = length - 1; i >= 0; i--) {
        if (isdigit(expression[i])) {
            push(expression[i] - '0');
        } else {
            int val1 = pop();
            int val2 = pop();
            int result;
```

```
            switch (expression[i]) {
                case '+':
                    result = val1 + val2;
                    break;
                case '-':
                    result = val1 - val2;
                    break;
                case '*':
                    result = val1 * val2;
                    break;
                case '/':
                    result = val1 / val2;
                    break;
            }
            push(result);
        }
    }
    return pop();
}

int main() {
    char expression[20];
    printf("Enter prefix expression: ");
    scanf("%s", expression);

    int finalResult = evaluatePrefix(expression);
    printf("Result: %d\n", finalResult);

    return 0;
}
```



PROGRAM-6

Queue

```
#include <stdio.h>
struct Queue {
    int a[5];
    int front, rear;
```

```
} q;
void enqueue(){
    int val;
    if(q.rear == 4)printf("Queue Full");
    else{
        printf("Enter value to be enqueued: ");
        scanf("%d",&val);
        q.a[++q.rear] = val;
        if(q.rear == 0) q.front = 0;
    }
}
void dequeue(){
    if(q.front == -1) printf("Queue Empty");
    if(++q.front == 5 || (q.front == q.rear+1)){
        q.front = -1;
        q.rear = -1;
    }
    printf("Dequeued");
}
void display(){
    int i;
    if(q.front == -1) printf("Queue Empty");
    else{
        for(i=q.front; i <= q.rear; i++){
            printf("%d ", q.a[i]);
        }
        printf("\n\n\n");
    }
}
void main(){
    q.front = -1;
    q.rear = -1;
    int choice;
    do{
        printf("1) Enqueue 2) Dequeue 3) Diplay 4) Quit  ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
```

```
            display();
            break;
        default:
            printf("Wrong Choice");
        }
    }while (choice < 4);
}
```

PROGRAM-7

Circular Queue

```
#include <stdio.h>
struct CircularQueue {
    int a[5];
    int front, rear;
} q;
void enqueue(){
    int val;
    if(q.front == (q.rear+1)%5) printf("Queue Full");
    else{
        printf("Enter value to be enqueued: ");
        scanf("%d",&val);
        q.rear = (q.rear+1)%5;
        q.a[q.rear] = val;
        if(q.rear == 0){
            q.front++;
        }
        printf("Enqueued\n\n");
    }
}
void dequeue(){
    if(q.front == -1) printf("Queue Empty");
    if(q.rear == q.front){
        q.front = -1;
        q.rear = -1;
    }else q.front = (q.front+1)%5;
    printf("Dequeued");
}
void display(){
    int i;
    if(q.front == -1) printf("Queue Empty");
    else{
        i=q.front;
        while(i != q.rear){
            printf("%d ", q.a[i]);
            i = (i+1)%5;
        }
        printf("%d\n\n\n", q.a[q.rear]);
    }
}
void main(){
    q.front = -1;
    q.rear = -1;
    int choice;
    do{
        printf("1) Enqueue 2) Dequeue 3) Diplay 4) Quit  ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
```

```
        case 3:
            display();
            break;
        default:
            printf("Wrong Choice");
        }
    }while (choice < 4);
}
```

```
1) Enqueue 2) Dequeue 3) Diplay 4) Quit  1
Enter value to be enqueued: 30
Enqueued

1) Enqueue 2) Dequeue 3) Diplay 4) Quit  1
Enter value to be enqueued: 40
Enqueued

1) Enqueue 2) Dequeue 3) Diplay 4) Quit  1
Enter value to be enqueued: 50
Enqueued

1) Enqueue 2) Dequeue 3) Diplay 4) Quit  1
Queue Full1) Enqueue 2) Dequeue 3) Diplay 4) Quit  3
10 20 30 40 50


1) Enqueue 2) Dequeue 3) Diplay 4) Quit  2
Dequeued1) Enqueue 2) Dequeue 3) Diplay 4) Quit  3
20 30 40 50


1) Enqueue 2) Dequeue 3) Diplay 4) Quit  3
20 30 40 50


1) Enqueue 2) Dequeue 3) Diplay 4) Quit  1
Enter value to be enqueued: 60
Enqueued

1) Enqueue 2) Dequeue 3) Diplay 4) Quit  1
Enter value to be enqueued: 70
Enqueued

1) Enqueue 2) Dequeue 3) Diplay 4) Quit  4
Wrong Choice
PS C:\Users\lenovo\Desktop\C>
```

PROGRAM-8

Priority Queue

```
#include <stdio.h>

struct PriorityQueue {
    int a[5];
    int size;
} q;

void enqueue(){
    int val, i, j;
    if(q.size == 5) {
        printf("Queue Full\n");
    } else {
        printf("Enter value to be enqueued: ");
        scanf("%d", &val);

        // Insert while maintaining the priority (min-priority queue)
        if(q.size == 0) {
            q.a[0] = val;
        } else {
            for(i = q.size - 1; i >= 0 && q.a[i] > val; i--) {
                q.a[i + 1] = q.a[i];
            }
            q.a[i + 1] = val;
        }
        q.size++;
        printf("Enqueued\n\n");
    }
}
```

```c
void dequeue(){
    if(q.size == 0) {
        printf("Queue Empty\n");
    } else {
        printf("Dequeued: %d\n", q.a[0]);
        // Shift elements to the left after dequeuing
        for(int i = 0; i < q.size - 1; i++) {
            q.a[i] = q.a[i + 1];
        }
        q.size--;
    }
}

void display(){
    if(q.size == 0) {
        printf("Queue Empty\n");
    } else {
        for(int i = 0; i < q.size; i++) {
            printf("%d ", q.a[i]);
        }
        printf("\n\n\n");
    }
}

void main(){
    q.size = 0;
    int choice;
    do{
        printf("1) Enqueue 2) Dequeue 3) Display 4) Quit  ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        default:
            if (choice != 4) printf("Wrong Choice\n");
        }
    }while (choice != 4);
}
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\lenovo\Documents\C>  & 'c:\Users\lenovo\.vscode\extensions\ms-vscode.cpptools-1.21.6-win32-x64\debugAdapters\bin
\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-3de2m05a.4ob' '--stdout=Microsoft-MIEngine-Out-kybn3mmn.wsi' '--st
derr=Microsoft-MIEngine-Error-ulhxzglw.hdt' '--pid=Microsoft-MIEngine-Pid-1mool1qi.rpv' '--dbgExe=C:\msys64\ucrt64\bin\gdb.e
xe' '--interpreter=mi'
1) Enqueue 2) Dequeue 3) Display 4) Quit  1
Enter value to be enqueued: 10
Enqueued

1) Enqueue 2) Dequeue 3) Display 4) Quit  1
Enter value to be enqueued: 50
Enqueued

1) Enqueue 2) Dequeue 3) Display 4) Quit  1
Enter value to be enqueued: 20
Enqueued

1) Enqueue 2) Dequeue 3) Display 4) Quit  3
10 20 50


1) Enqueue 2) Dequeue 3) Display 4) Quit  2
Dequeued: 10
1) Enqueue 2) Dequeue 3) Display 4) Quit  3
20 50


1) Enqueue 2) Dequeue 3) Display 4) Quit  4
PS C:\Users\lenovo\Documents\C>
```

PROGRAM-9

Linked List

```c
#include <stdio.h>
struct node{
    int data;
    struct node *next;
};
struct node *head = NULL, *tail = NULL, *temp;
int len = 0;
void insert(){
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    printf("Enter integer to be inserted: ");
    scanf("%d", &newNode->data);
    newNode->next = NULL;

    if(head == NULL){   //if its first node
        head = newNode;
    }else{              //if list already exists
        tail->next = newNode;
    }
    tail = newNode;
    len++;
    printf("\nNode Inserted\n");
}
```

```
void insertAtBeginning(){
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    printf("Enter integer to be inserted: ");
    scanf("%d", &newNode->data);
    if(head == NULL){
        tail = newNode;
    }
    newNode->next = head;
    head = newNode;
    len++;
    printf("\nNode Inserted\n");
}
void insertAtPos(){
    int pos;
    printf("Enter position: ");
    scanf("%d", &pos);
    if(pos > len+1 || pos < 1){
        printf("Wrong position entered.\n");
        return;
    }
    if(pos == 1) {
        insertAtBeginning();
        return;
    }
    if(pos == len+1){
        insert();
        return;
    }
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    printf("Enter integer to be inserted: ");
    scanf("%d", &newNode->data);
    temp = head;
    while (pos-- != 2)
    {
        temp = temp->next;
    }
    newNode->next = temp->next;
    temp->next = newNode;
    len++;
    printf("\nNode Inserted\n");
}
void display(){
    if(head == NULL){
        printf("Empty List\n\n");
        return;
```

```
    }
    temp = head;
    while (temp != NULL)
    {
        printf("%d   ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void deleteAtBeginning(){
    if(head == NULL){
        printf("List does not exists\n"); return;
    }
    if(head == tail) {
            free(head);
            head = tail = NULL;
    }else{
        temp = head;
        head = head->next;
        free(temp);
    }
    len--;
    printf("\nNode Deleted\n");
}
void deleteAtEnd(){
    if(head == NULL){
        printf("List does not exists\n"); return;
    }
    if(tail == head){
        free(head);
        head = tail = NULL;
    }else{
        temp = head;
        while(temp->next != tail)
            temp = temp->next;
        temp->next = NULL;
        free(tail);
        tail = temp;
    }
    len--;
    printf("\nNode Deleted\n");
}
void deleteAtPos(){
    int pos;
    printf("Enter position: ");
```

```
    scanf("%d", &pos);
    if(pos > len || pos < 1){
        printf("Wrong position entered.\n");
        return;
    }
    if(pos == 1) {
        deleteAtBeginning();
        return;
    }
    if(pos == len){
        deleteAtEnd();
        return;
    }
    temp = head;
    pos--;
    while (pos-- != 1)        temp = temp->next;
    struct node *del = temp->next;
    temp->next = temp->next->next;
    free(del);
    len--;
    printf("\nNode Deleted\n");
}
void main(){
    int c;
    do{
        printf("MENU\n1) Insert At End \n2) Insert At Beginning \n3) Insert At
Position \n4) Display\n5) Delete Head\n6) Delete Tail\n7) Delete At Pos  \nEnter
your Choice: ");
        scanf("%d", &c);
        switch (c)
        {
            case 1: insert();break;
            case 2: insertAtBeginning(); break;
            case 3: insertAtPos(); break;
            case 4: display(); break;
            case 5: deleteAtBeginning(); break;
            case 6: deleteAtEnd(); break;
            case 7: deleteAtPos(); break;
            default: break;
        }
    }while (c != 0);
    printf("\n\n");
}
```

```
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 1
Enter integer to be inserted: 10

Node Inserted
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 2
Enter integer to be inserted: 30

MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 3
Enter position: 2
Enter integer to be inserted: 50

Node Inserted
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 4
30   50   10
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 7
Enter position: 1

Node Deleted
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 4
Empty List
```

PROGRAM-10

Doubly Queue

```c
#include <stdio.h>
struct node{
    int data;
    struct node *next, *prev;
};
struct node *head = NULL, *tail = NULL, *temp;
int len = 0;
void insert(){
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    printf("Enter integer to be inserted: ");
    scanf("%d", &newNode->data);
    newNode->next = NULL;

    if(head == NULL){   //if its first node
        head = newNode;
        newNode->prev = NULL;
    }else{              //if list already exists
        tail->next = newNode;
        newNode->prev = tail;
    }
    tail = newNode;
    len++;
    printf("\nNode Inserted\n");
}
void insertAtBeginning(){
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    printf("Enter integer to be inserted: ");
    scanf("%d", &newNode->data);
    newNode->prev = NULL;
    if(head == NULL){
        tail = newNode;
    }else{
        head->prev = newNode;
    }
    newNode->next = head;
    head = newNode;
    len++;
    printf("\nNode Inserted\n");
}
void insertAtPos(){
    int pos;
    printf("Enter position: ");
    scanf("%d", &pos);
    if(pos > len+1 || pos < 1){
        printf("Wrong position entered.\n");
        return;
    }
    if(pos == 1) {
        insertAtBeginning();
        return;
    }
    if(pos == len+1){
        insert();
        return;
    }
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    printf("Enter integer to be inserted: ");
    scanf("%d", &newNode->data);
    temp = head;
    pos--;
    while (pos-- != 1)
    {
        temp = temp->next;
    }
    newNode->next = temp->next;
    temp->next = newNode;
    newNode->prev = temp;
    newNode->next->prev = newNode;
```

```
    len++;
    printf("\nNode Inserted\n");
}
void display(){
    if(head == NULL){
        printf("Empty List\n\n");
        return;
    }
    temp = head;
    while (temp != NULL)
    {
        printf("%d   ", temp->data);
        temp = temp->next;
    }
    printf("\n");
    displayBack();
}
void displayBack(){
    if(tail == NULL){
        printf("Empty List\n\n");
        return;
    }
    temp = tail;
    while (temp != NULL)
    {
        printf("%d   ", temp->data);
        temp = temp->prev;
    }
    printf("\n");
}
void deleteAtBeginning(){
    if(head == NULL){
        printf("List does not exists\n"); return;
    }
    if(head == tail) {
            free(head);
            head = tail = NULL;
    }else{
        head = head->next;
        free(head->prev);
        head->prev = NULL;
    }
    len--;
    printf("\nNode Deleted\n");
}
void deleteAtEnd(){
    if(head == NULL){
        printf("List does not exists\n"); return;
    }
    if(tail == head){
        free(head);
        head = tail = NULL;
    }else{
        tail = tail->prev;
        free(tail->next);
        tail->next = NULL;
    }
    len--;
    printf("\nNode Deleted\n");
}
void deleteAtPos(){
    int pos;
    printf("Enter position: ");
    scanf("%d", &pos);
    if(pos > len+1 || pos < 1){
        printf("Wrong position entered.\n");
        return;
    }
    if(pos == 1) {
        deleteAtBeginning();
        return;
    }
```

```c
    if(pos == len+1){
        deleteAtEnd();
        return;
    }
    temp = head;
    pos--;
    while (pos-- != 1)  temp = temp->next;
    struct node *del = temp->next;
    temp->next = temp->next->next;
    temp->next->prev = temp;
    free(del);
    len--;
    printf("\nNode Deleted\n");
}
void main(){
    int c;
    do{
        printf("MENU\n1) Insert At End \n2) Insert At Beginning \n3) Insert At Position \n4) Display\n5) Delete Head\n6) Delete
Tail\n7) Delete At Pos  \nEnter your Choice: ");
        scanf("%d", &c);
        switch (c)
        {
            case 1: insert();break;
            case 2: insertAtBeginning(); break;
            case 3: insertAtPos(); break;
            case 4: display(); break;
            case 5: deleteAtBeginning(); break;
            case 6: deleteAtEnd(); break;
            case 7: deleteAtPos(); break;
            default: break;
        }
    }while (c != 0);
    printf("\n\n");
}
```

```
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 1
Enter integer to be inserted: 10

Node Inserted
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 2
Enter integer to be inserted: 30
```

```
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 4
30   40   10
10   40   30
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 7
Enter position: 1
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 4
10
10
MENU
1) Insert At End
2) Insert At Beginning
3) Insert At Position
4) Display
5) Delete Head
6) Delete Tail
7) Delete At Pos
Enter your Choice: 6
```