

DAY 3

C PROGRAMMING – DATA STRUCTURES

1. Write a c program to implement the SINGLY LINKED LIST with the following operations:
 - a. Insert an element into the list [beginning, middle, last].

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Structure for a node in the linked list
5  struct Node {
6      int data;
7      struct Node *next;
8  };
9
10 // Function to insert a new node at the beginning of the linked list
11 struct Node *insertAtBeginning(struct Node *head, int value) {
12     struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
13     newNode->data = value;
14     newNode->next = head;
15     return newNode;
16 }
17
18 // Function to insert a new node at the middle of the linked list
19 struct Node *insertAtMiddle(struct Node *head, int value, int position) {
20     struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
21     newNode->data = value;
22
23     struct Node *current = head;
24     for (int i = 1; i < position - 1 && current != NULL; ++i) {
25         current = current->next;
26     }
27
28     if (current == NULL) {
29         printf("Invalid position for insertion.\n");
30         free(newNode);
31         return head;
32     }
33
34     newNode->next = current->next;
35     current->next = newNode;
36
37     return head;
38 }
39
40 // Function to insert a new node at the end of the linked list
41 struct Node *insertAtEnd(struct Node *head, int value) {
42     struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
```

```

43     newNode->data = value;
44     newNode->next = NULL;
45
46     if (head == NULL) {
47         return newNode;
48     }
49
50     struct Node *current = head;
51     while (current->next != NULL) {
52         current = current->next;
53     }
54
55     current->next = newNode;
56     return head;
57 }
58
59 // Function to display the linked list
60 void displayList(struct Node *head) {
61     struct Node *current = head;
62     while (current != NULL) {
63         printf("%d -> ", current->data);
64         current = current->next;
65     }
66     printf("NULL\n");
67 }
68
69 int main() {
70     struct Node *head = NULL;
71
72     // Insert at the beginning
73     head = insertAtBeginning(head, 20);
74     head = insertAtBeginning(head, 10);
75     printf("Linked List after inserting at the beginning: ");
76     displayList(head);
77
78     // Insert at the middle
79     head = insertAtMiddle(head, 15, 2);
80     printf("Linked List after inserting at the middle: ");
81     displayList(head);
82
83     // Insert at the end
84     head = insertAtEnd(head, 30);
85     // Insert at the end
86     head = insertAtEnd(head, 30);
87     printf("Linked List after inserting at the end: ");
88     displayList(head);
89     return 0;
90 }
91

```

input

```

Linked List after inserting at the beginning: 10 -> 20 -> NULL
Linked List after inserting at the middle: 10 -> 15 -> 20 -> NULL
Linked List after inserting at the end: 10 -> 15 -> 20 -> 30 -> NULL

```

- b. Delete an element into the list [beginning, middle, last].

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Structure for a node in the linked list
5  struct Node {
6      int data;
7      struct Node *next;
8  };
9
10 // Function to insert a new node at the beginning of the linked list
11 struct Node *insertAtBeginning(struct Node *head, int value) {
12     struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
13     newNode->data = value;
14     newNode->next = head;
15     return newNode;
16 }
17
18 // Function to delete a node from the beginning of the linked list
19 struct Node *deleteFromBeginning(struct Node *head) {
20     if (head == NULL) {
21         printf("List is already empty.\n");
22         return NULL;
23     }
24
25     struct Node *temp = head;
26     head = head->next;
27     free(temp);
28
29     return head;
30 }
31
32 // Function to delete a node from the middle of the linked list
33 struct Node *deleteFromMiddle(struct Node *head, int position) {
34     if (head == NULL) {
35         printf("List is empty.\n");
36         return NULL;
37     }
38
39     struct Node *prev = NULL;
40     struct Node *current = head;
41
42     for (int i = 1; i < position && current != NULL; ++i) {
```

```

42 for (int i = 1; i < position && current != NULL; ++i) {
43     prev = current;
44     current = current->next;
45 }
46
47 if (current == NULL) {
48     printf("Invalid position for deletion.\n");
49     return head;
50 }
51
52 if (prev != NULL) {
53     prev->next = current->next;
54     free(current);
55 } else {
56     head = current->next;
57     free(current);
58 }
59
60 return head;
61 }
62

```

Linked List after inserting at the beginning: 10 -> 20 -> 30 -> NULL
 Linked List after deleting from the beginning: 20 -> 30 -> NULL
 Linked List after deleting from the middle: 20 -> NULL
 Linked List after deleting from the end: NULL

c. Search for an element in the list.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Structure for a node in the linked list
5  struct Node {
6      int data;
7      struct Node *next;
8  };
9
10 // Function to insert a new node at the beginning of the linked list
11 struct Node *insertAtBeginning(struct Node *head, int value) {
12     struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
13     newNode->data = value;
14     newNode->next = head;
15     return newNode;
16 }
17
18 // Function to search for an element in the linked list
19 int searchElement(struct Node *head, int target) {
20     struct Node *current = head;
21     int position = 1;

```

```

23     while (current != NULL) {
24         if (current->data == target) {
25             return position;
26         }
27         current = current->next;
28         position++;
29     }
30
31     return -1; // Element not found
32 }
33
34 // Function to display the Linked List
35 void displayList(struct Node *head) {
36     struct Node *current = head;
37     while (current != NULL) {
38         printf("%d -> ", current->data);
39         current = current->next;
40     }
41     printf("NULL\n");
42 }
43

```

```

44 int main() {
45     struct Node *head = NULL;
46
47     // Insert elements at the beginning
48     head = insertAtBeginning(head, 30);
49     head = insertAtBeginning(head, 20);
50     head = insertAtBeginning(head, 10);
51
52     printf("Linked List: ");
53     displayList(head);
54
55     int target;
56
57     printf("Enter the element to search: ");
58     scanf("%d", &target);
59
60     int position = searchElement(head, target);
61
62     if (position != -1) {
63         printf("Element %d found at position %d.\n", target, position);
64     } else {
65         printf("Element %d not found in the linked list.\n", target);
66     }
67
68     return 0;
69 }
70

```

```

Linked List: 10 -> 20 -> 30 -> NULL
Enter the element to search: 30
Element 30 found at position 3.

```

d. Display the elements.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Structure for a node in the Linked List
5  struct Node {
6      int data;
7      struct Node *next;
8  };
9
10 // Function to insert a new node at the beginning of the Linked List
11 struct Node *insertAtBeginning(struct Node *head, int value) {
12     struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
13     newNode->data = value;
14     newNode->next = head;
15     return newNode;
16 }
17
18 // Function to display the elements in the Linked List
19 void displayList(struct Node *head) {
20     struct Node *current = head;
21     while (current != NULL) {
22         printf("%d -> ", current->data);
23         current = current->next;
24     }
25     printf("NULL\n");
26 }
27
28 int main() {
29     struct Node *head = NULL;
30
31     // Insert elements at the beginning
32     head = insertAtBeginning(head, 30);
33     head = insertAtBeginning(head, 20);
34     head = insertAtBeginning(head, 10);
35
36     printf("Linked List: ");
37     displayList(head);
38
39     return 0;
40 }
41
42
```

Linked List: 10 -> 20 -> 30 -> NULL

2. Write a c program to implement the stack data structure with the following.
- a. Pop an element into the list [beginning, middle, last].

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_SIZE 100
5
6  // Structure for a stack
7  struct Stack {
8      int items[MAX_SIZE];
9      int top;
10 };
11
12 // Function to initialize an empty stack
13 void initialize(struct Stack *stack) {
14     stack->top = -1;
15 }
16
17 // Function to check if the stack is empty
18 int isEmpty(struct Stack *stack) {
19     return stack->top == -1;
20 }
21
22 // Function to check if the stack is full
23 int isFull(struct Stack *stack) {
24     return stack->top == MAX_SIZE - 1;
25 }
26
27 // Function to push an item onto the stack
28 void push(struct Stack *stack, int item) {
29     if (isFull(stack)) {
30         printf("Stack overflow\n");
31         exit(1);
32     }
33     stack->items[++stack->top] = item;
34 }
35
36 // Function to display the stack
37 void displayStack(struct Stack *stack) {
38     if (isEmpty(stack)) {
39         printf("Stack is empty.\n");
40         return;
41     }
42     printf("Stack contents: ");
```

```

41     }
42     printf("Stack contents: ");
43     for (int i = 0; i <= stack->top; ++i) {
44         printf("%d ", stack->items[i]);
45     }
46     printf("\n");
47 }
48
49 int main() {
50     struct Stack stack;
51     initialize(&stack);
52
53     int option, element;
54
55     do {
56         printf("\nStack Operations:\n");
57         printf("1. Push at beginning\n");
58         printf("2. Push at middle\n");
59         printf("3. Push at last\n");
60         printf("4. Display stack\n");
61         printf("5. Quit\n");
62         printf("Enter your choice: ");
63         printf("Enter your choice: ");
64         scanf("%d", &option);
65
66         switch (option) {
67             case 1:
68                 printf("Enter element to push: ");
69                 scanf("%d", &element);
70                 for (int i = stack.top; i >= 0; --i) {
71                     stack.items[i + 1] = stack.items[i];
72                 }
73                 stack.items[0] = element;
74                 stack.top++;
75                 printf("Element %d pushed at the beginning.\n", element);
76                 break;
77             case 2:
78                 if (isFull(&stack)) {
79                     printf("Stack is full.\n");
80                     break;
81                 }
82                 printf("Enter element to push: ");
83                 scanf("%d", &element);

```



```

82         scanf("%d", &element);
83         int position;
84         printf("Enter position (1 to %d) to push at: ", stack.top + 2);
85         scanf("%d", &position);
86         if (position < 1 || position > stack.top + 2) {
87             printf("Invalid position.\n");
88             break;
89         }
90         for (int i = stack.top; i >= position - 1; --i) {
91             stack.items[i + 1] = stack.items[i];
92         }
93         stack.items[position - 1] = element;
94         stack.top++;
95         printf("Element %d pushed at position %d.\n", element, position);
96         break;
97     case 3:
98         printf("Enter element to push: ");
99         scanf("%d", &element);
100        stack.items[++stack.top] = element;
101        printf("Element %d pushed at the end.\n", element);
102        break;
103    case 4:
104        displayStack(&stack);
105        break;
106    case 5:
107        printf("Exiting...\n");
108        break;
109    default:
110        printf("Invalid choice. Please try again.\n");
111    }
112    } while (option != 5);
113
114    return 0;
115 }
116
117

```

Stack Operations:

1. Push at beginning
2. Push at middle
3. Push at last
4. Display stack
5. Quit

Enter your choice: 1

Enter element to push: 2

Element 2 pushed at the beginning.

Stack Operations:

1. Push at beginning
2. Push at middle
3. Push at last
4. Display stack
5. Quit

Enter your choice: 2

Enter element to push: 3

Enter position (1 to 2) to push at: 2

Element 3 pushed at position 2.

Stack Operations:

1. Push at beginning
2. Push at middle

```
3. Push at last
4. Display stack
5. Quit
Enter your choice: 3
Enter element to push: 4
Element 4 pushed at the end.
```

```
Stack Operations:
1. Push at beginning
2. Push at middle
3. Push at last
4. Display stack
5. Quit
Enter your choice: 4
Stack contents: 2 3 4
```

```
Stack Operations:
1. Push at beginning
2. Push at middle
3. Push at last
4. Display stack
5. Quit
Enter your choice: 5
Exiting...
```

- b. Search for an element in the stack.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_SIZE 100
5
6  // Structure for a stack
7  struct Stack {
8      int items[MAX_SIZE];
9      int top;
10 };
11
12 // Function to initialize an empty stack
13 void initialize(struct Stack *stack) {
14     stack->top = -1;
15 }
16
17 // Function to check if the stack is empty
18 int isEmpty(struct Stack *stack) {
19     return stack->top == -1;
20 }
```

```
22 // Function to check if the stack is full
23 ▾ int isFull(struct Stack *stack) {
24     return stack->top == MAX_SIZE - 1;
25 }
26
27 // Function to push an item onto the stack
28 ▾ void push(struct Stack *stack, int item) {
29     if (isFull(stack)) {
30         printf("Stack overflow\n");
31         exit(1);
32     }
33     stack->items[++stack->top] = item;
34 }
35
36 // Function to pop an item from the stack
37 ▾ int pop(struct Stack *stack) {
38     if (isEmpty(stack)) {
39         printf("Stack underflow\n");
40         exit(1);
41     }
```

```
42     return stack->items[stack->top--];
43 }
44
45 // Function to search for an element in the stack
46 int search(struct Stack *stack, int target) {
47     for (int i = stack->top; i >= 0; --i) {
48         if (stack->items[i] == target) {
49             return i;
50         }
51     }
52     return -1; // Element not found
53 }
54
55 int main() {
56     struct Stack stack;
57     initialize(&stack);
58
59     int option, element, target;
60
61     do {
```

```
62     printf("\nStack Operations:\n");
63     printf("1. Push\n");
64     printf("2. Pop\n");
65     printf("3. Search\n");
66     printf("4. Display stack\n");
67     printf("5. Quit\n");
68     printf("Enter your choice: ");
69     scanf("%d", &option);
70
71     switch (option) {
72         case 1:
73             printf("Enter element to push: ");
74             scanf("%d", &element);
75             push(&stack, element);
76             printf("Element %d pushed onto the stack.\n",
                    element);
77             break;
78         case 2:
79             if (isEmpty(&stack)) {
80                 printf("Stack is empty.\n");
```

```

81         break;
82     }
83     element = pop(&stack);
84     printf("Element %d popped from the stack.\n",
            element);
85     break;
86     case 3:
87         printf("Enter element to search: ");
88         scanf("%d", &target);
89         int position = search(&stack, target);
90         if (position != -1) {
91             printf("Element %d found at position | %d
                        from top of the stack.\n", target,
                        stack.top - position + 1);
92         } else {
93             printf("Element %d not found in the stack
                        .\n", target);
94         }
95         break;
96     case 4:
97         printf("Stack contents: ");
98         for (int i = stack.top; i >= 0; --i) {
99             printf("%d ", stack.items[i]);
100         }
101         printf("\n");
102         break;
103     case 5:
104         printf("Exiting...\n");
105         break;
106     default:
107         printf("Invalid choice. Please try again.\n");
108     }
109 } while (option != 5);
110
111 return 0;
112 }

```

/tmp/1PfCkQSRW1.o

Stack Operations:

1. Push
2. Pop
3. Search
4. Display stack
5. Quit

Enter your choice: 1

Enter element to push: 3

Element 3 pushed onto the stack.

Stack Operations:

1. Push
2. Pop
3. Search
4. Display stack
5. Quit

Enter your choice: 1

Enter element to push: 4

Element 4 pushed onto the stack.

Stack Operations:

1. Push
2. Pop
3. Search
4. Display stack
5. Quit

Enter your choice: 1

Enter element to push: 5

Element 5 pushed onto the stack.

Stack Operations:

1. Push
2. Pop
3. Search
4. Display stack
5. Quit

Enter your choice: 4

Stack contents: 5 4 3

Stack Operations:

1. Push
2. Pop
3. Search
4. Display stack
5. Quit

Enter your choice: 3

Enter element to search: 4

Element 4 found at position 2 from top of the stack.

Stack Operations:

1. Push
2. Pop
3. Search
4. Display stack
5. Quit

Enter your choice: 5

Exiting...

c. Display the stack.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_SIZE 100
5
6  // Structure for a stack
7  struct Stack {
8      int items[MAX_SIZE];
9      int top;
10 };
11
12 // Function to initialize an empty stack
13 void initialize(struct Stack *stack) {
14     stack->top = -1;
15 }
16
17 // Function to check if the stack is empty
18 int isEmpty(struct Stack *stack) {
19     return stack->top == -1;
20 }
21
```

```

21
22 // Function to check if the stack is full
23 int isFull(struct Stack *stack) {
24     return stack->top == MAX_SIZE - 1;
25 }
26
27 // Function to push an item onto the stack
28 void push(struct Stack *stack, int item) {
29     if (isFull(stack)) {
30         printf("Stack overflow\n");
31         exit(1);
32     }
33     stack->items[++stack->top] = item;
34 }
35
36 // Function to pop an item from the stack
37 int pop(struct Stack *stack) {
38     if (isEmpty(stack)) {
39         printf("Stack underflow\n");
40         exit(1);
41     }
42     return stack->items[stack->top--];
43 }
44
45 // Function to display the stack
46 void displayStack(struct Stack *stack) {
47     if (isEmpty(stack)) {
48         printf("Stack is empty.\n");
49         return;
50     }
51     printf("Stack contents: ");
52     for (int i = stack->top; i >= 0; --i) {
53         printf("%d ", stack->items[i]);
54     }
55     printf("\n");
56 }
57
58 int main() {
59     struct Stack stack;
60     initialize(&stack);
61
62     int option, element;

```

```

63
64 do {
65     printf("\nStack Operations:\n");
66     printf("1. Push\n");
67     printf("2. Pop\n");
68     printf("3. Display stack\n");
69     printf("4. Quit\n");
70     printf("Enter your choice: ");
71     scanf("%d", &option);
72
73     switch (option) {
74         case 1:
75             printf("Enter element to push: ");
76             scanf("%d", &element);
77             push(&stack, element);
78             printf("Element %d pushed onto the stack.\n", element);
79             break;
80         case 2:
81             if (isEmpty(&stack)) {
82                 printf("Stack is empty.\n");
83                 break;
84             }
85             element = pop(&stack);
86             printf("Element %d popped from the stack.\n", element);
87             break;
88         case 3:
89             displayStack(&stack);
90             break;
91         case 4:
92             printf("Exiting...\n");
93             break;
94         default:
95             printf("Invalid choice. Please try again.\n");
96     }
97 } while (option != 4);
98
99 return 0;
100 }
101
102
103

```

Stack Operations:

1. Push
2. Pop
3. Display stack
4. Quit

Enter your choice: 1

Enter element to push: 2

Element 2 pushed onto the stack.

Stack Operations:

1. Push
2. Pop
3. Display stack
4. Quit

Enter your choice: 1

Enter element to push: 3

Element 3 pushed onto the stack.

Stack Operations:

1. Push
2. Pop
3. Display stack
4. Quit

Enter your choice: 1

Enter your choice: 1

Enter element to push: 4

Element 4 pushed onto the stack.

Stack Operations:

1. Push
2. Pop
3. Display stack
4. Quit

Enter your choice: 2

Element 4 popped from the stack.

Stack Operations:

1. Push
2. Pop
3. Display stack
4. Quit

Enter your choice: 3

Stack contents: 3 2

Stack Operations:

1. Push
2. Pop
3. Display stack
4. Quit

Enter your choice: 4

Exiting...

3. Write a c program to implement queue data structure with the following operations.
- a. Enqueue
 - b. Dequeue
 - c. Display

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_SIZE 100
5
6  // Structure for the queue
7  struct Queue {
8      int items[MAX_SIZE];
9      int front;
10     int rear;
11 };
12
13 // Function to initialize a new queue
14 void initialize(struct Queue* queue) {
15     queue->front = -1;
16     queue->rear = -1;
17 }
18
19 // Function to check if the queue is empty
20 int isEmpty(struct Queue* queue) {
21     return (queue->front == -1);
22 }
23
24 // Function to check if the queue is full
25 int isFull(struct Queue* queue) {
26     return (queue->rear == MAX_SIZE - 1);
27 }
28
29 // Function to enqueue an element into the queue
30 void enqueue(struct Queue* queue, int value) {
31     if (isFull(queue)) {
32         printf("Queue is full. Cannot enqueue.\n");
33         return;
34     } else if (isEmpty(queue)) {
35         queue->front = queue->rear = 0;
36     } else {
37         queue->rear++;
38     }
39
40     queue->items[queue->rear] = value;
41     printf("%d enqueued to the queue.\n", value);
42 }
```

```

43
44 // Function to dequeue an element from the queue
45 int dequeue(struct Queue* queue) {
46     int dequeuedValue;
47
48     if (isEmpty(queue)) {
49         printf("Queue is empty. Cannot dequeue.\n");
50         return -1;
51     } else {
52         dequeuedValue = queue->items[queue->front];
53
54         if (queue->front == queue->rear) {
55             queue->front = queue->rear = -1;
56         } else {
57             queue->front++;
58         }
59     }
60
61     return dequeuedValue;
62 }
63

```

```

64 // Function to display the contents of the queue
65 void display(struct Queue* queue) {
66     if (isEmpty(queue)) {
67         printf("Queue is empty.\n");
68         return;
69     }
70
71     printf("Queue elements: ");
72     for (int i = queue->front; i <= queue->rear; i++) {
73         printf("%d ", queue->items[i]);
74     }
75     printf("\n");
76 }
77
78 int main() {
79     struct Queue queue;
80     initialize(&queue);
81
82     // Enqueue some elements
83     enqueue(&queue, 10);
84     enqueue(&queue, 20);

```

```

84     enqueue(&queue, 20);
85     enqueue(&queue, 30);
86
87     // Display the queue
88     display(&queue);
89
90     // Dequeue an element
91     int dequeuedValue = dequeue(&queue);
92     if (dequeuedValue != -1) {
93         printf("Dequeued: %d\n", dequeuedValue);
94     }
95
96     // Display the queue again
97     display(&queue);
98
99     return 0;
100 }
101

```

```

10 enqueued to the queue.
20 enqueued to the queue.
30 enqueued to the queue.
Queue elements: 10 20 30
Dequeued: 10
Queue elements: 20 30

```

4. To convert infix to postfix using stack

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX_SIZE 100
7
8  // Structure for a stack
9  struct Stack {
10     char items[MAX_SIZE];
11     int top;
12 };
13
14 // Function to initialize an empty stack
15 void initialize(struct Stack *stack) {
16     stack->top = -1;
17 }
18
19 // Function to check if the stack is empty
20 int isEmpty(struct Stack *stack) {
21     return stack->top == -1;

```

```

22 }
23
24 // Function to push an item onto the stack
25 void push(struct Stack *stack, char item) {
26     if (stack->top >= MAX_SIZE - 1) {
27         printf("Stack overflow\n");
28         exit(1);
29     }
30     stack->items[++stack->top] = item;
31 }
32
33 // Function to pop an item from the stack
34 char pop(struct Stack *stack) {
35     if (isEmpty(stack)) {
36         printf("Stack underflow\n");
37         exit(1);
38     }
39     return stack->items[stack->top--];
40 }
41
42 // Function to get the precedence of an operator

```

```

43 int getPrecedence(char op) {
44     switch (op) {
45         case '+':
46         case '-':
47             return 1;
48         case '*':
49         case '/':
50             return 2;
51         case '^':
52             return 3;
53     }
54     return 0;
55 }
56
57 // Function to convert infix to postfix expression
58 void infixToPostfix(char infix[], char postfix[]) {
59     struct Stack stack;
60     initialize(&stack);
61
62     int i, j;
63     i = j = 0;

```

```

64     while (infix[i] != '\0') {
65         if (isalnum(infix[i])) {
66             postfix[j++] = infix[i];
67         } else if (infix[i] == '(') {
68             push(&stack, '(');
69         } else if (infix[i] == ')') {
70             while (!isEmpty(&stack) && stack.items[stack.top] != '(') {
71                 postfix[j++] = pop(&stack);
72             }
73             pop(&stack); // Pop '('
74         } else {
75             while (!isEmpty(&stack) && getPrecedence(infix[i]) <= getPrecedence(stack.items[stack.top])) {
76                 postfix[j++] = pop(&stack);
77             }
78             push(&stack, infix[i]);
79         }
80         i++;
81     }
82
83     while (!isEmpty(&stack)) {

```



```

83
84 while (!isEmpty(&stack)) {
85     postfix[j++] = pop(&stack);
86 }
87
88 postfix[j] = '\0';
89 }
90
91 int main() {
92     char infix[MAX_SIZE], postfix[MAX_SIZE];
93
94     printf("Enter an infix expression: ");
95     scanf("%s", infix);
96
97     infixToPostfix(infix, postfix);
98
99     printf("Postfix expression: %s\n", postfix);
100
101     return 0;
102 }
103

```

```

Enter an infix expression: ((a*b)/(c-d))
Postfix expression: ab*cd-/

```

5. To evaluate the given expression using stack.

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 #define MAX_SIZE 100
6
7 typedef struct{
8     int items[MAX_SIZE];
9     int top;
10 }Stack;
11
12 void push(Stack*stack,int value){
13     if(stack->top == MAX_SIZE-1){
14         printf("Stack overflow\n");
15         exit(1);
16     }
17     stack->top++;
18     stack->items[stack->top]=value;
19 }
20
21 int pop(Stack*stack){

```

```

22 ~ pop(&stack->stack);
23 ~ if(stack->top== -1){
24 ~     printf("Stack underflow\n");
25 ~     exit(1);
26 ~ }
27 ~ int value=stack->items[stack->top];
28 ~ stack->top--;
29 ~ return value;
30 ~ }
31 ~
32 ~ int evaluateExpression(char*expression){
33 ~     Stack stack;
34 ~     stack.top=-1;
35 ~
36 ~     for(int i=0;i<strlen(expression);i++){
37 ~         if(expression[i]>='0'&& expression[i]<='9'){
38 ~             push(&stack,expression[i]-'0');
39 ~         }else{
40 ~             int operand2=pop(&stack);
41 ~             int operand1=pop(&stack);
42 ~             switch(expression[i]){

```

```

43 ~                 case '+':
44 ~                     push(&stack,operand1+operand2);
45 ~                     break;
46 ~                 case '-':
47 ~                     push(&stack,operand1-operand2);
48 ~                     break;
49 ~                 case '*':
50 ~                     push(&stack,operand1*operand2);
51 ~                     break;
52 ~                 case '/':
53 ~                     push(&stack,operand1/operand2);
54 ~                     break;
55 ~             }
56 ~         }
57 ~     }
58 ~     return pop(&stack);
59 ~ }
60 ~
61 ~ int main(){
62 ~     char expression[MAX_SIZE];
63 ~     printf("enter an arithmetic expression:");

```

```

63 ~     printf("enter an arithmetic expression:");
64 ~     scanf("%s",expression);
65 ~
66 ~     int result= evaluateExpression(expression);
67 ~     printf("result: %d\n",result);
68 ~     return 0;
69 ~ }
70 ~
71 ~

```

```

enter an arithmetic expression: (a+b/c*(d-e))
Stack underflow

```