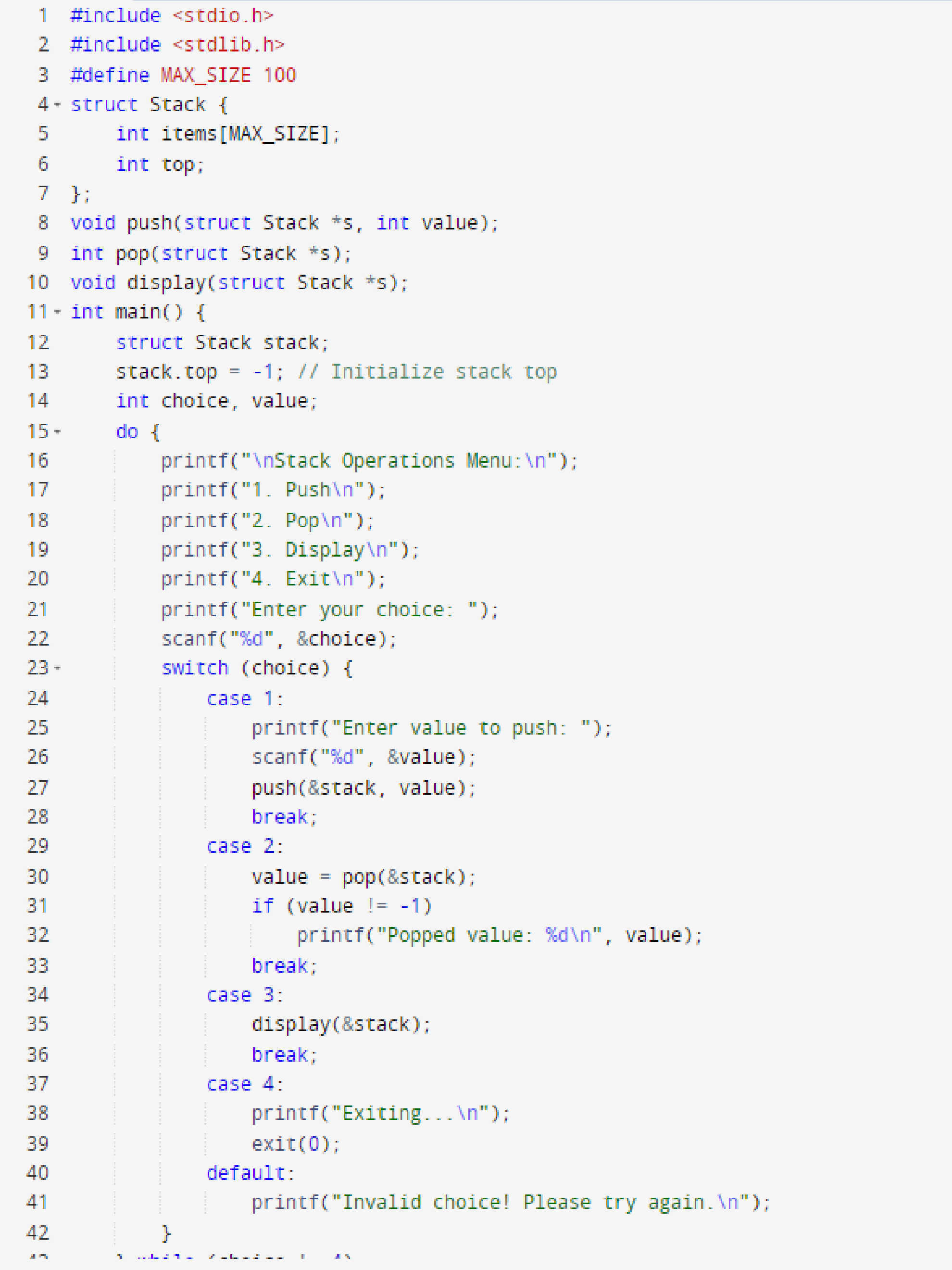
BHAVLEEN KAUR
3EIC-2
102155010

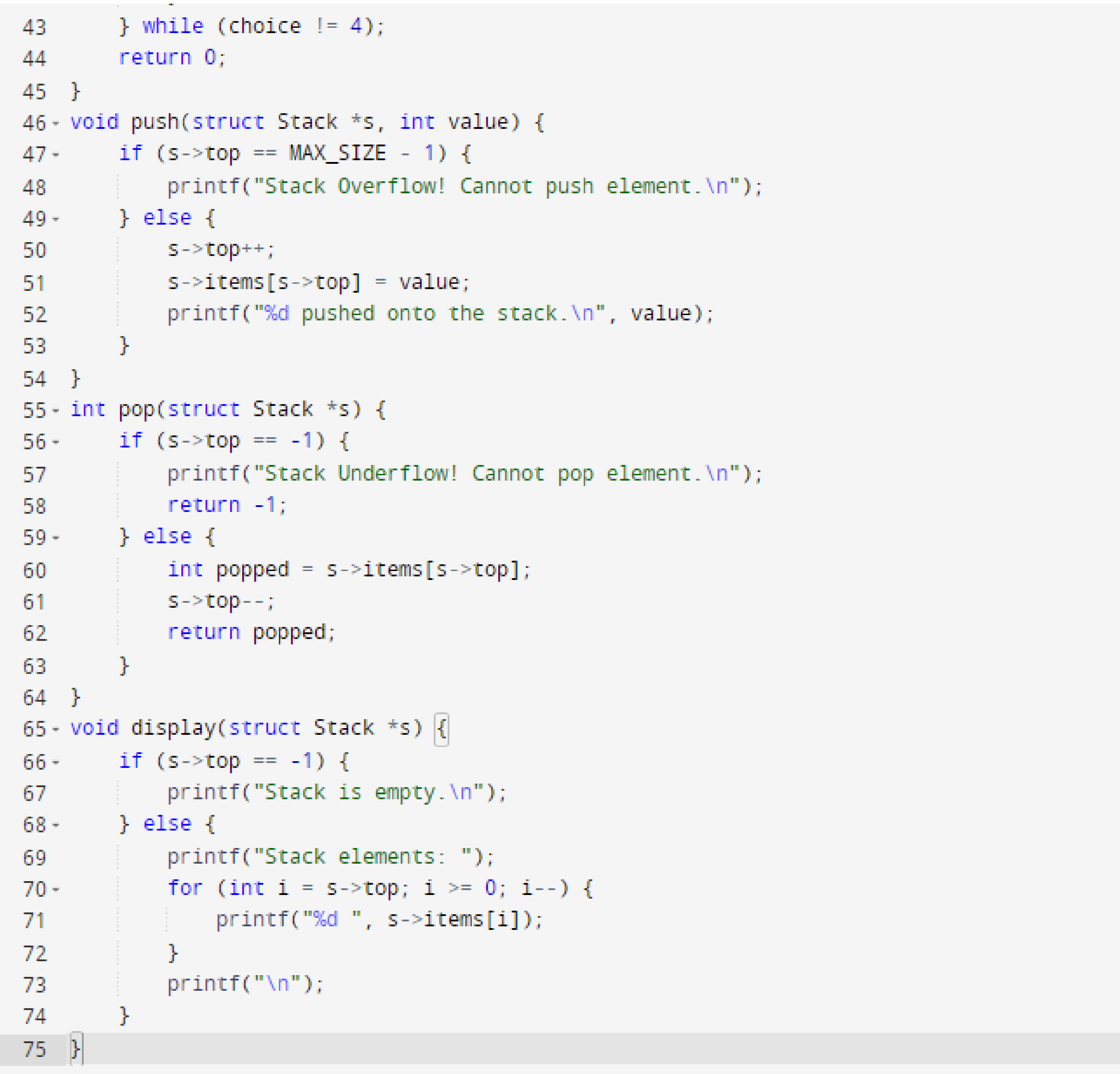
LAB ASSIGNMENT 5
Stacks and its Applications

1. Write a menu driven program with 4 options (Push, Pop, Display, and Exit) to demonstrate the working of stacks using arrays.



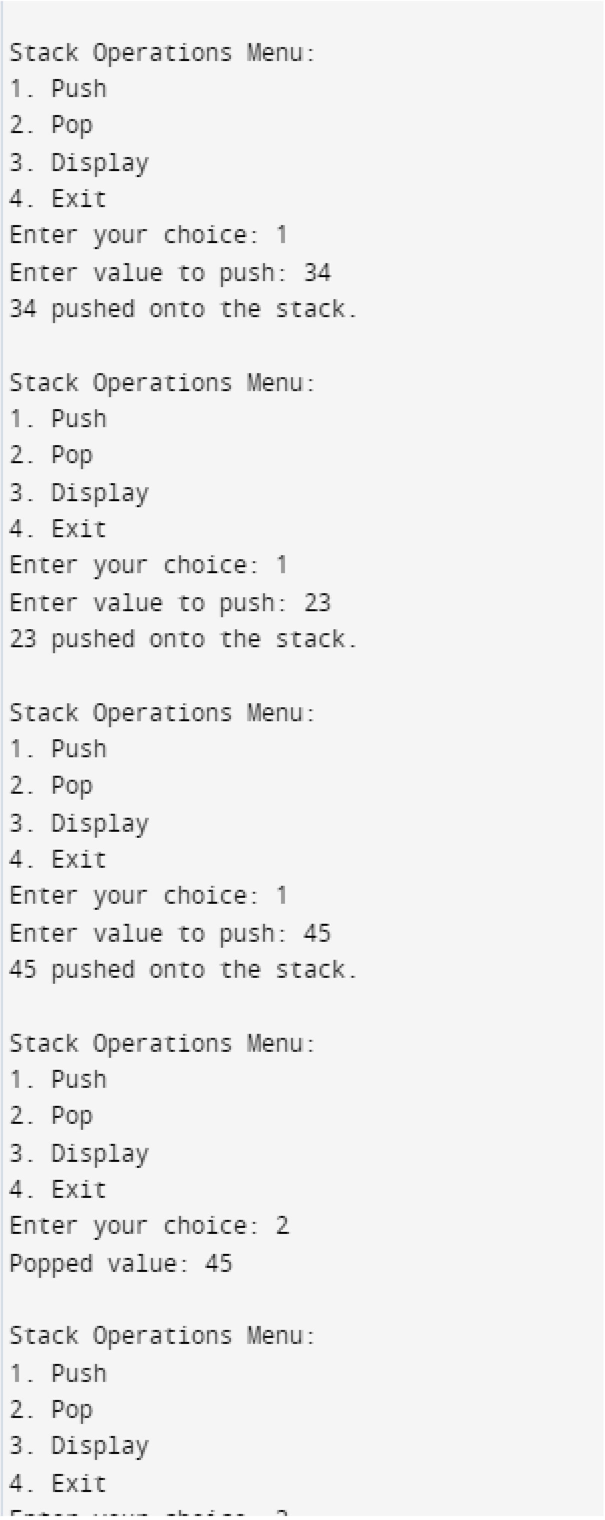
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Stack {
    int items[MAX_SIZE];
    int top;
};
void push(struct Stack *s, int value);
int pop(struct Stack *s);
void display(struct Stack *s);
int main() {
    struct Stack stack;
    stack.top = -1; // Initialize stack top
    int choice, value;
    do {
        printf("\nStack 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:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 2:
                value = pop(&stack);
                if (value != -1)
                    printf("Popped value: %d\n", value);
                break;
            case 3:
                display(&stack);
                break;
            case 4:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4)
```

BHAVLEEN KAUR
3EIC-2
102155010



```
    } while (choice != 4);
    return 0;
}
void push(struct Stack *s, int value) {
    if (s->top == MAX_SIZE - 1) {
        printf("Stack Overflow! Cannot push element.\n");
    } else {
        s->top++;
        s->items[s->top] = value;
        printf("%d pushed onto the stack.\n", value);
    }
}
int pop(struct Stack *s) {
    if (s->top == -1) {
        printf("Stack Underflow! Cannot pop element.\n");
        return -1;
    } else {
        int popped = s->items[s->top];
        s->top--;
        return popped;
    }
}
void display(struct Stack *s) {
    if (s->top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements: ");
        for (int i = s->top; i >= 0; i--) {
            printf("%d ", s->items[i]);
        }
        printf("\n");
    }
}
```

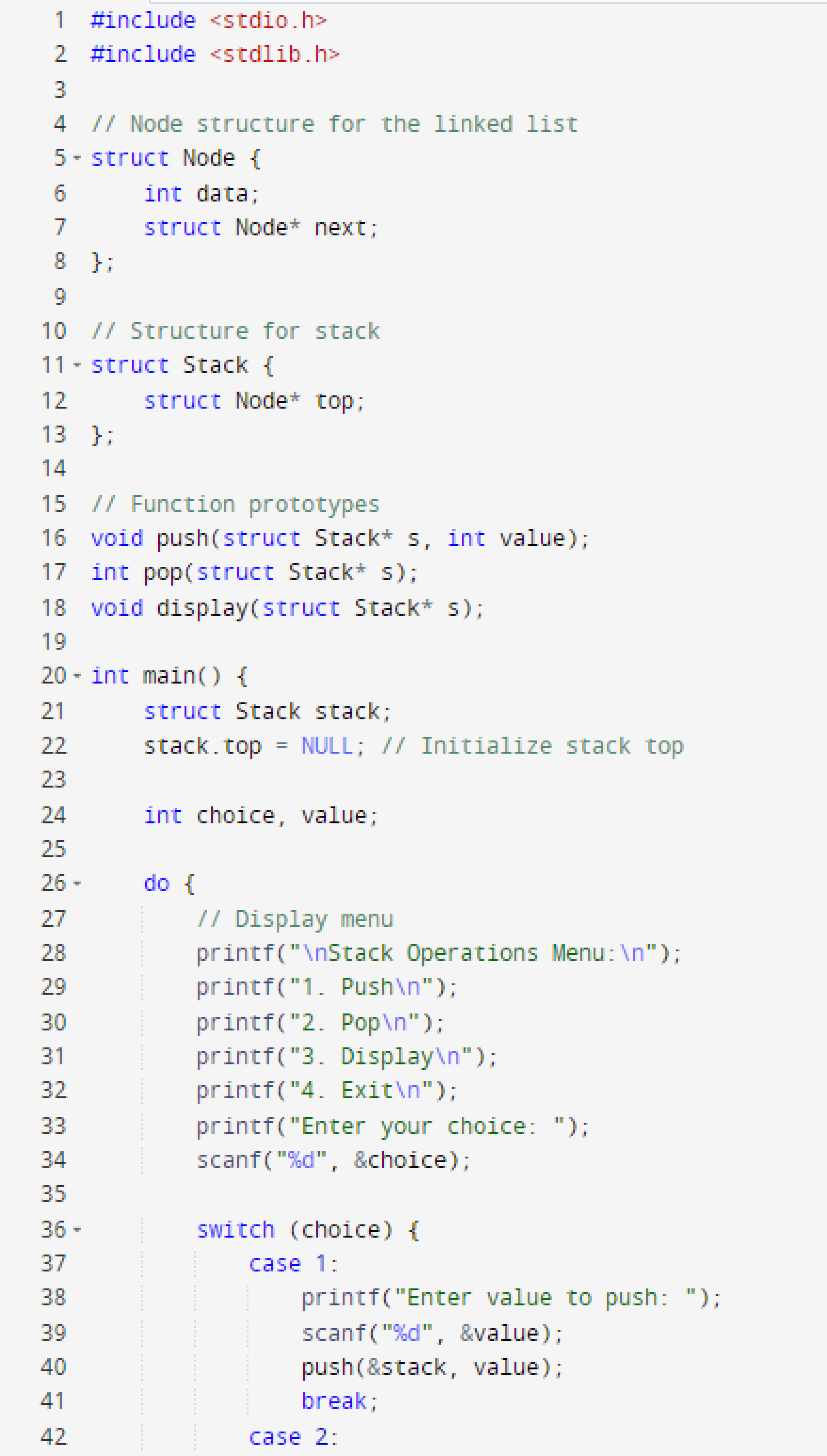
BHAVLEEN KAUR
3EIC-2
102155010



```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 34
34 pushed onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 23
23 pushed onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 45
45 pushed onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped value: 45

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
```

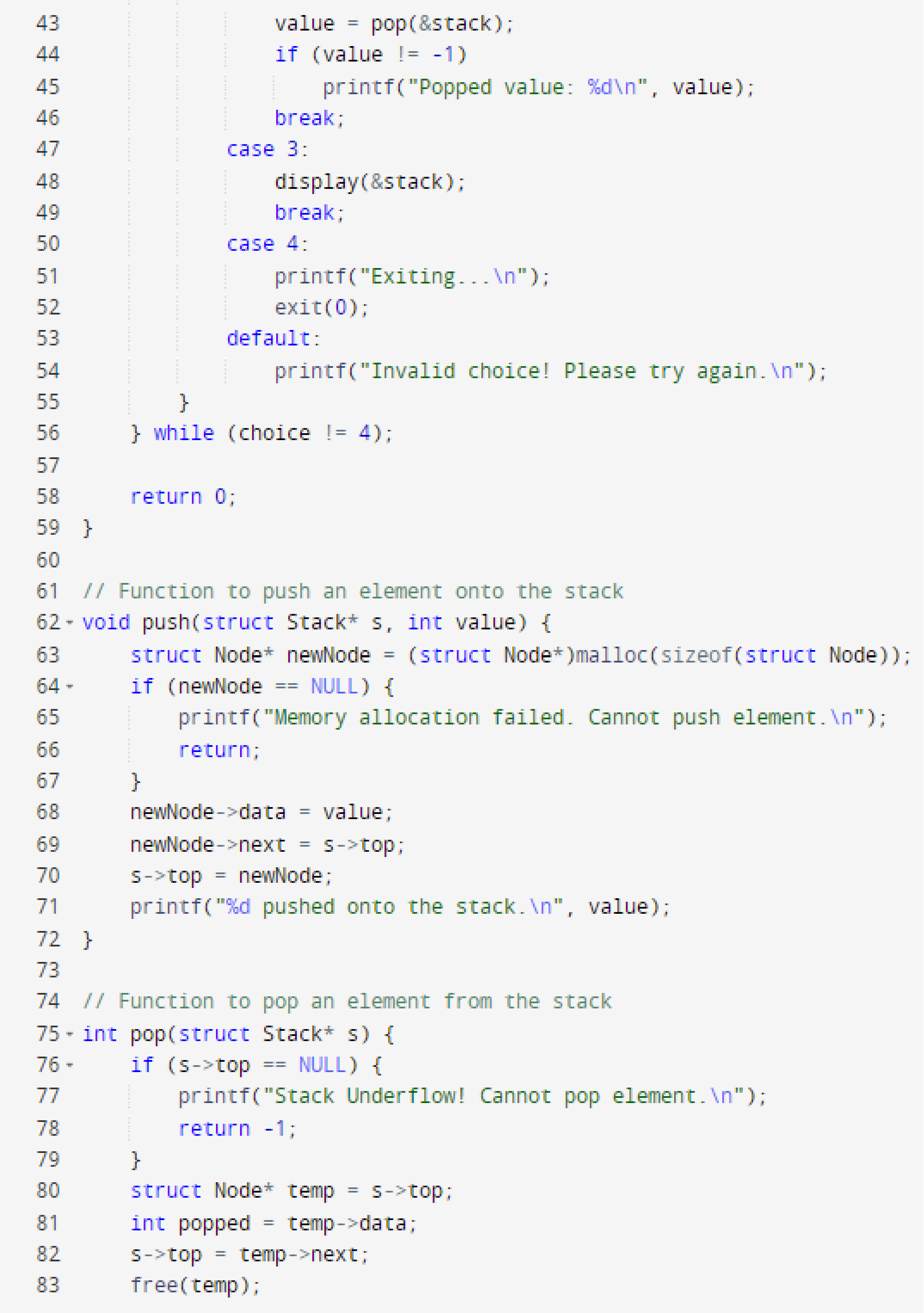
BHAVLEEN KAUR
3EIC-2
102155010

2. Write a menu driven program with 4 options (Push, Pop, Display, and Exit) to demonstrate the working of stacks using linked-list.



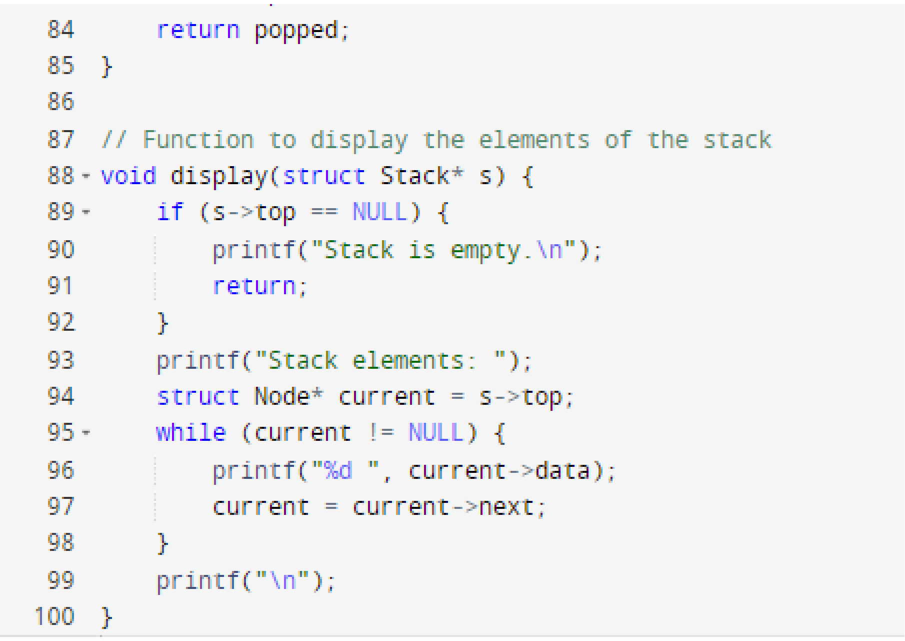
```c
#include <stdio.h>
#include <stdlib.h>

// Node structure for the linked list
struct Node {
    int data;
    struct Node* next;
};

// Structure for stack
struct Stack {
    struct Node* top;
};

// Function prototypes
void push(struct Stack* s, int value);
int pop(struct Stack* s);
void display(struct Stack* s);

int main() {
    struct Stack stack;
    stack.top = NULL; // Initialize stack top

    int choice, value;

    do {
        // Display menu
        printf("\nStack 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:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 2:
```

BHAVLEEN KAUR
3EIC-2
102155010



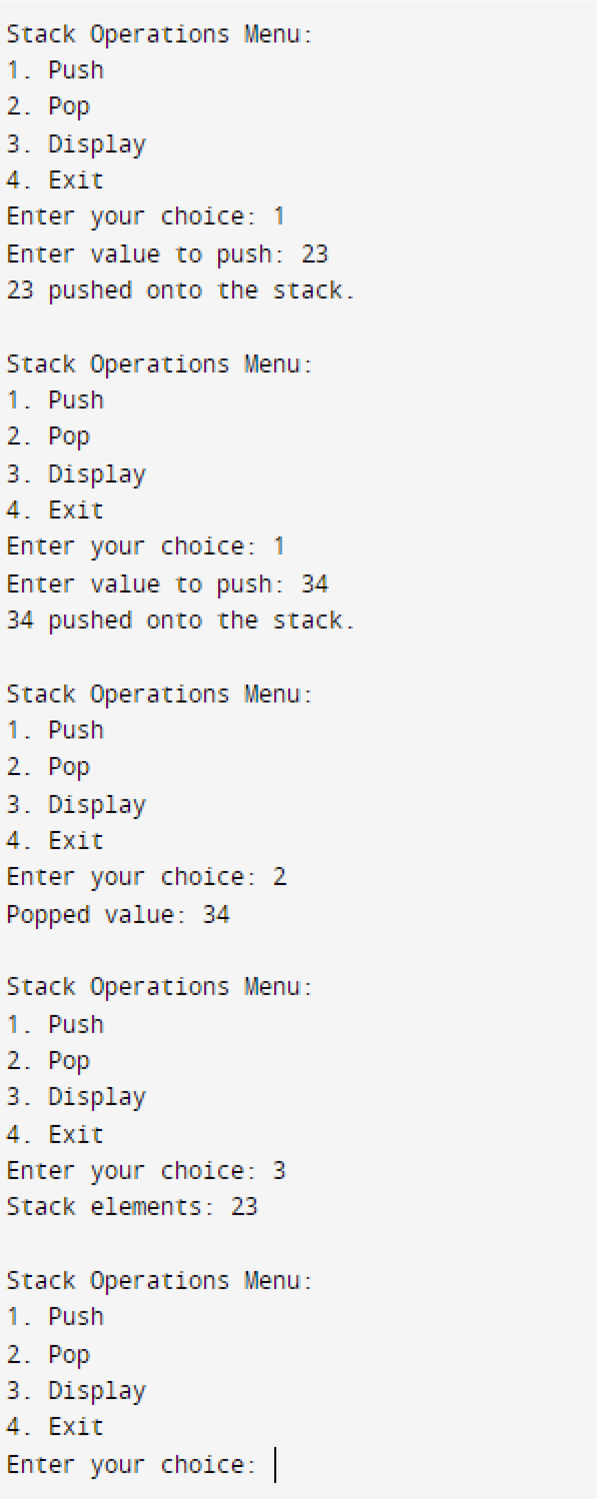
```c
                value = pop(&stack);
                if (value != -1)
                    printf("Popped value: %d\n", value);
                break;
            case 3:
                display(&stack);
                break;
            case 4:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}

// Function to push an element onto the stack
void push(struct Stack* s, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed. Cannot push element.\n");
        return;
    }
    newNode->data = value;
    newNode->next = s->top;
    s->top = newNode;
    printf("%d pushed onto the stack.\n", value);
}

// Function to pop an element from the stack
int pop(struct Stack* s) {
    if (s->top == NULL) {
        printf("Stack Underflow! Cannot pop element.\n");
        return -1;
    }
    struct Node* temp = s->top;
    int popped = temp->data;
    s->top = temp->next;
    free(temp);
```

BHAVLEEN KAUR
3EIC-2
102155010



```
    return popped;
}

// Function to display the elements of the stack
void display(struct Stack* s) {
    if (s->top == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack elements: ");
    struct Node* current = s->top;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

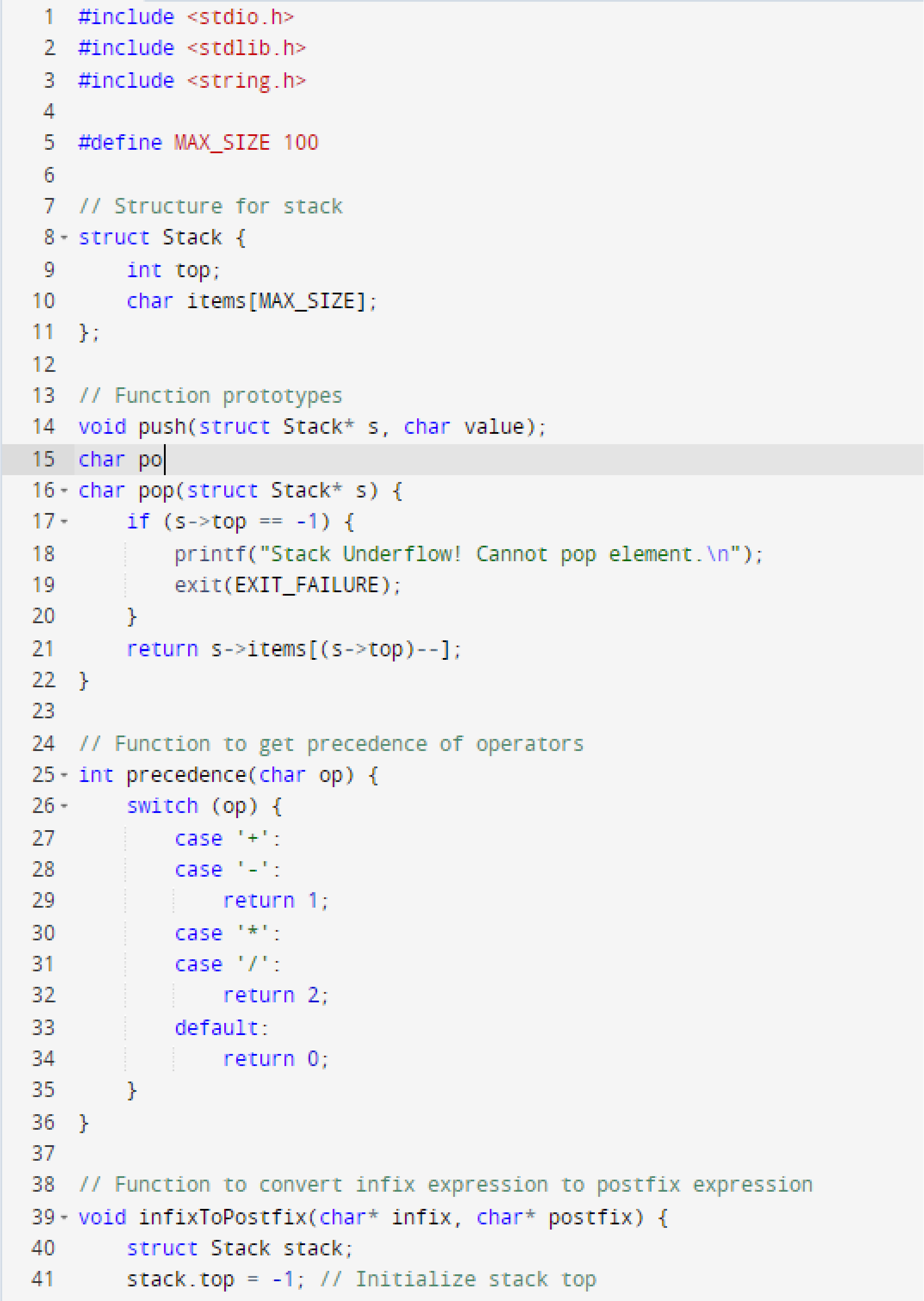
BHAVLEEN KAUR
3EIC-2
102155010



```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 23
23 pushed onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 34
34 pushed onto the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped value: 34

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 23

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
```

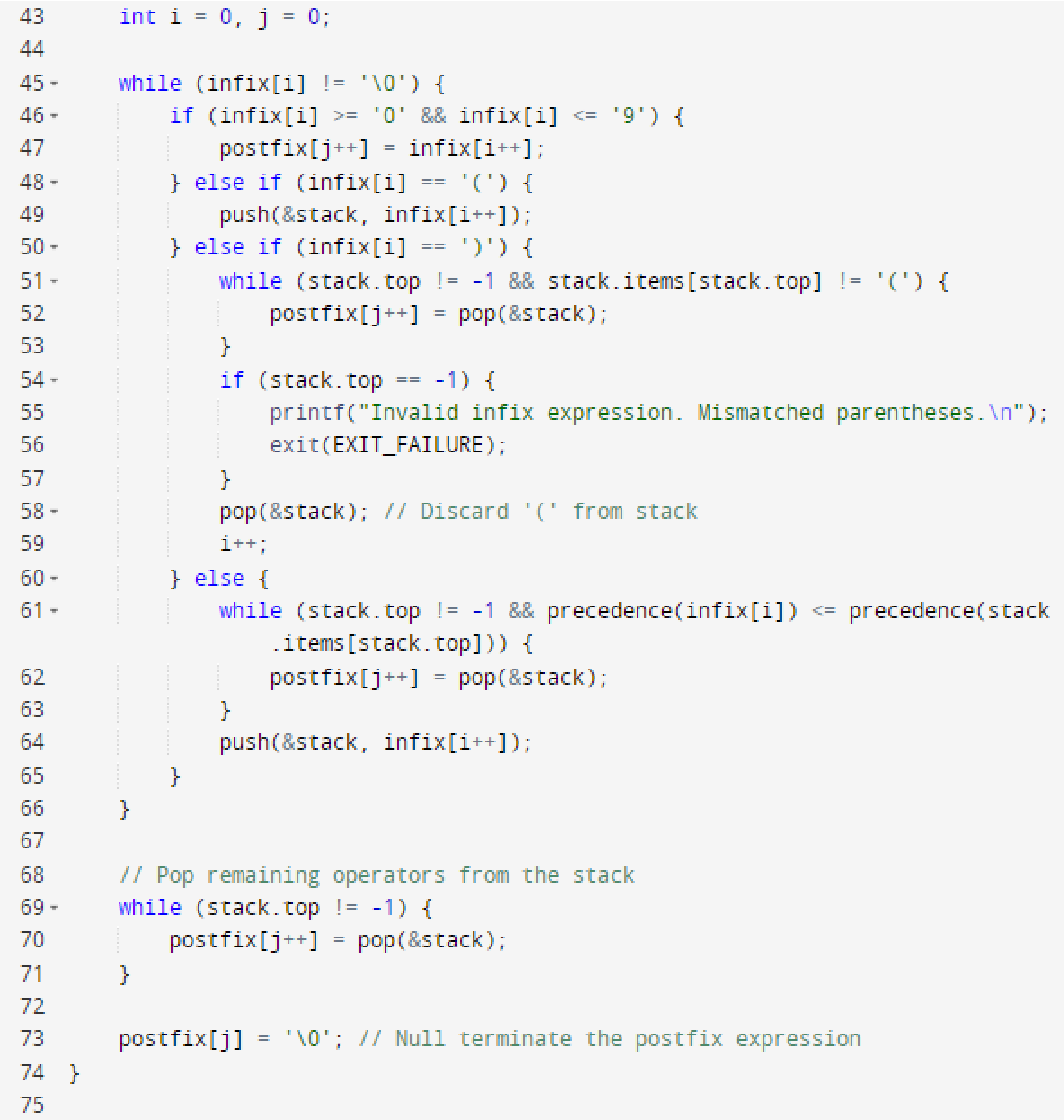
BHAVLEEN KAUR
3EIC-2
102155010

3. Write a program to convert infix expression into postfix expression using stack.

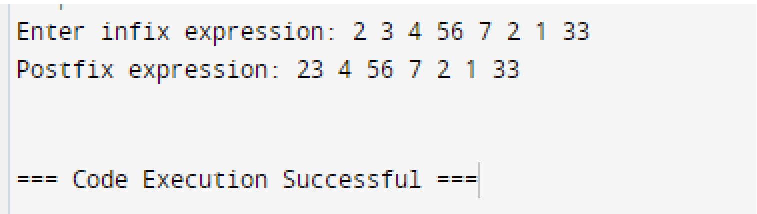


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 100

// Structure for stack
struct Stack {
    int top;
    char items[MAX_SIZE];
};

// Function prototypes
void push(struct Stack* s, char value);
char po
char pop(struct Stack* s) {
    if (s->top == -1) {
        printf("Stack Underflow! Cannot pop element.\n");
        exit(EXIT_FAILURE);
    }
    return s->items[(s->top)--];
}

// Function to get precedence of operators
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        default:
            return 0;
    }
}

// Function to convert infix expression to postfix expression
void infixToPostfix(char* infix, char* postfix) {
    struct Stack stack;
    stack.top = -1; // Initialize stack top
```

BHAVLEEN KAUR
3EIC-2
102155010



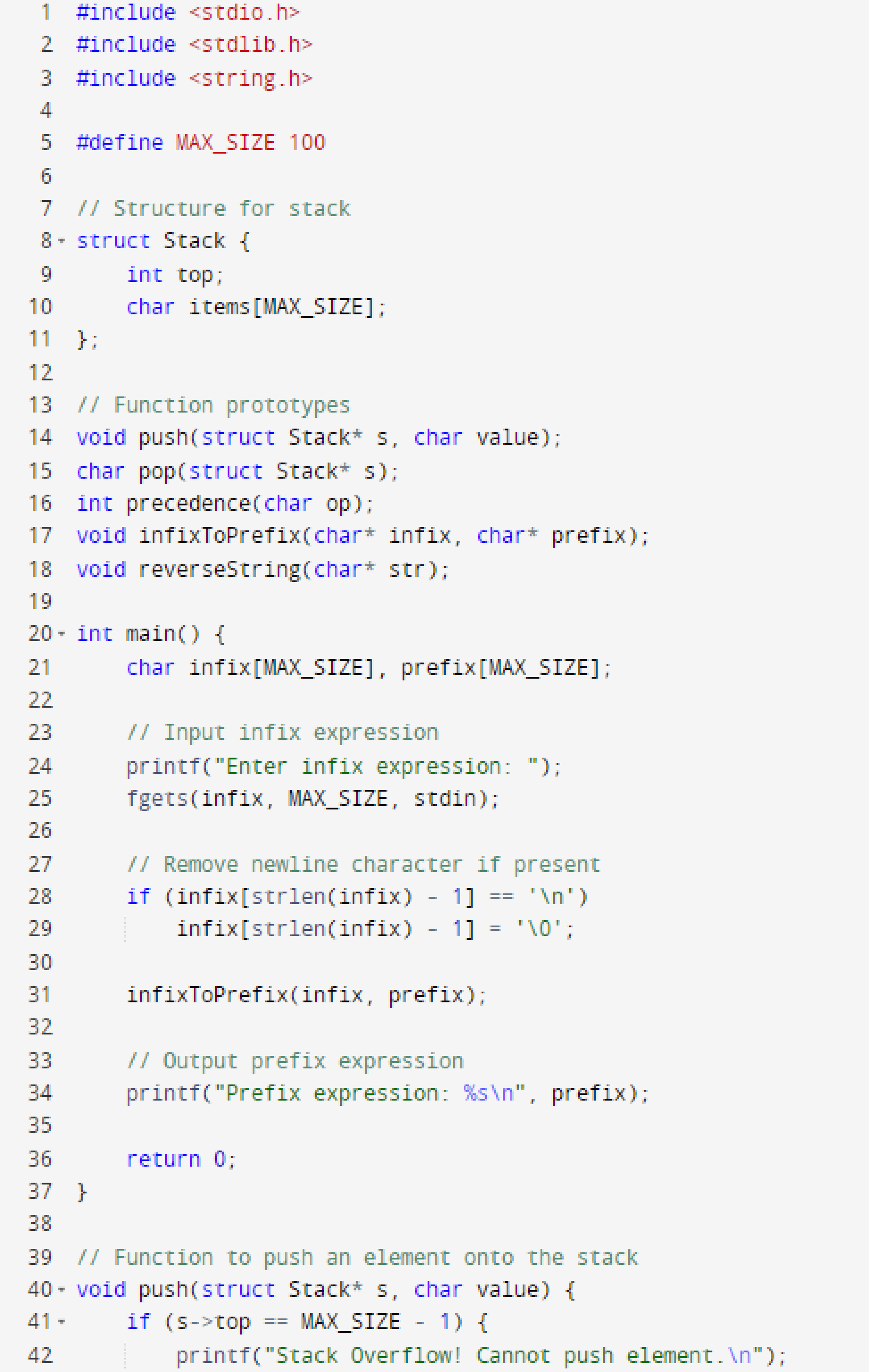
```
    int i = 0, j = 0;

    while (infix[i] != '\0') {
        if (infix[i] >= '0' && infix[i] <= '9') {
            postfix[j++] = infix[i++];
        } else if (infix[i] == '(') {
            push(&stack, infix[i++]);
        } else if (infix[i] == ')') {
            while (stack.top != -1 && stack.items[stack.top] != '(') {
                postfix[j++] = pop(&stack);
            }
            if (stack.top == -1) {
                printf("Invalid infix expression. Mismatched parentheses.\n");
                exit(EXIT_FAILURE);
            }
            pop(&stack); // Discard '(' from stack
            i++;
        } else {
            while (stack.top != -1 && precedence(infix[i]) <= precedence(stack
                .items[stack.top])) {
                postfix[j++] = pop(&stack);
            }
            push(&stack, infix[i++]);
        }
    }

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

    postfix[j] = '\0'; // Null terminate the postfix expression
}

```



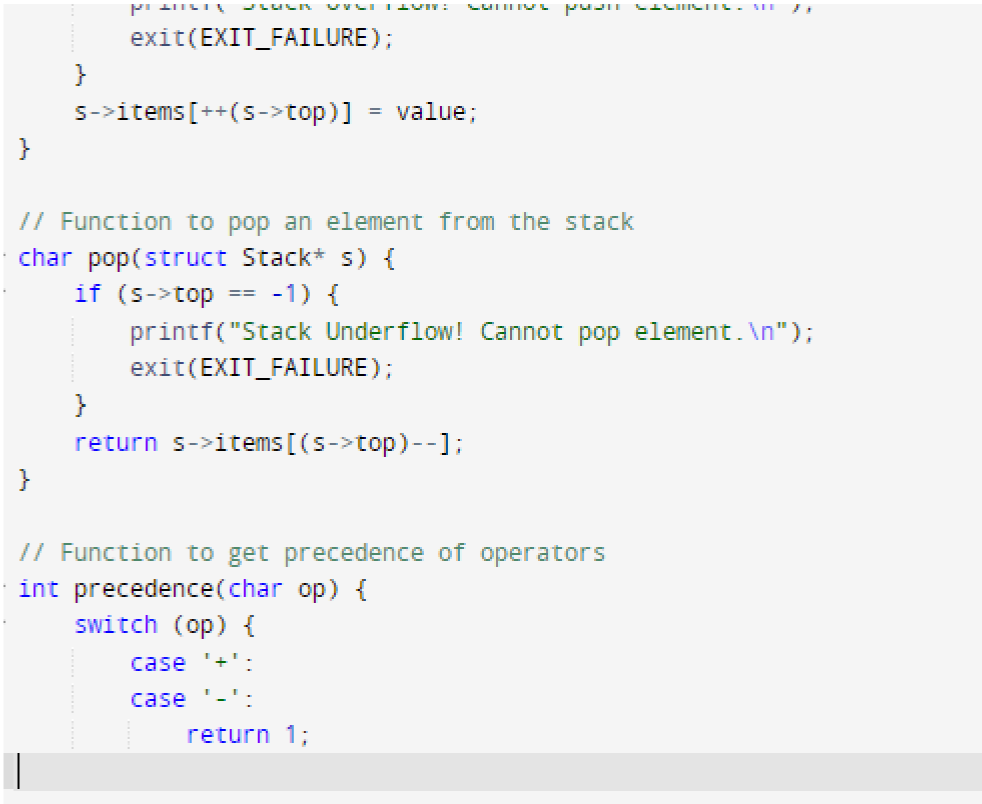
```
Enter infix expression: 2 3 4 56 7 2 1 33
Postfix expression: 23 4 56 7 2 1 33


=== Code Execution Successful ===
```

BHAVLEEN KAUR
3EIC-2
102155010

4. Write a program to convert infix expression into prefix expression using stack.



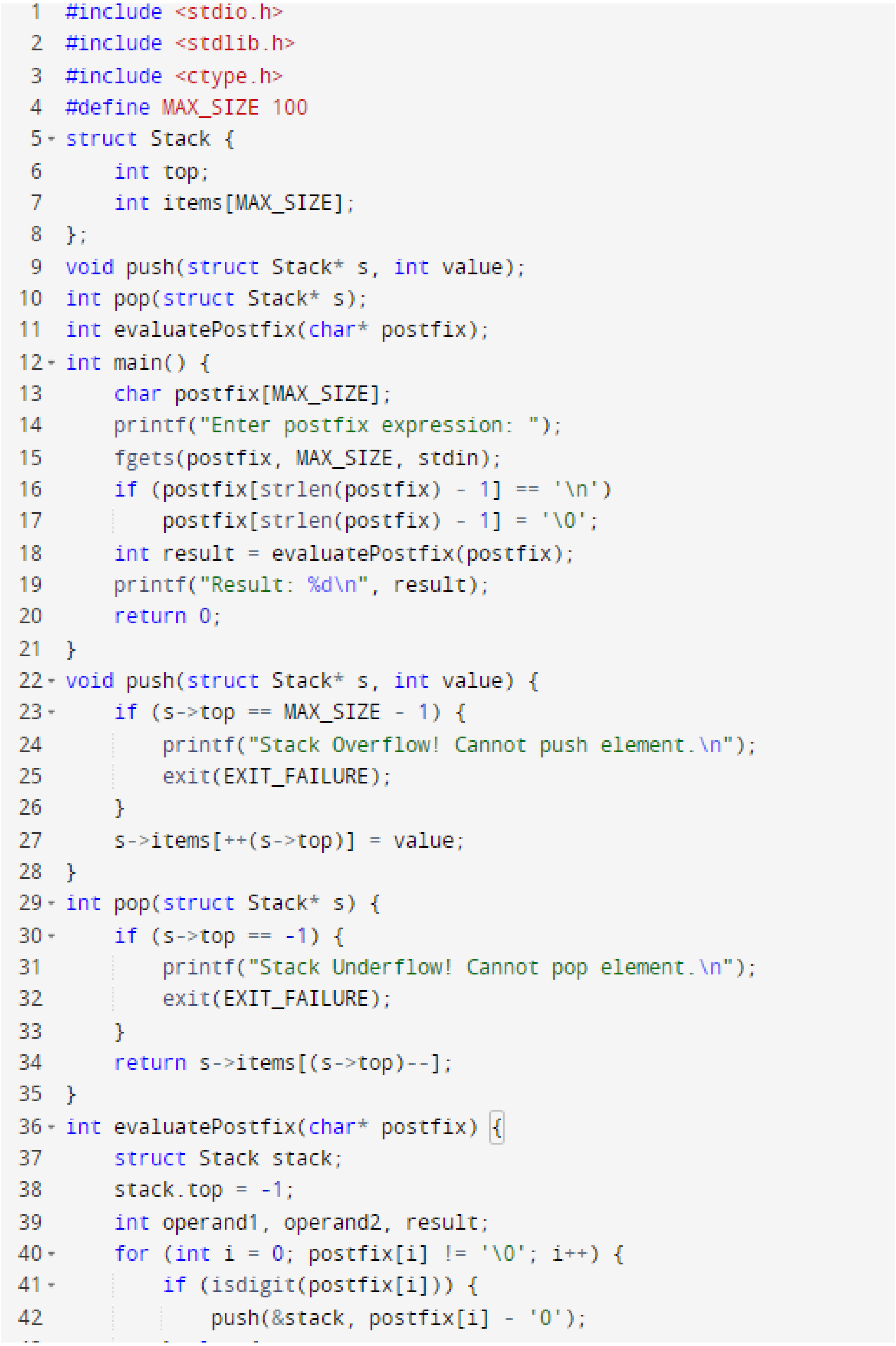
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 100

// Structure for stack
struct Stack {
    int top;
    char items[MAX_SIZE];
};

// Function prototypes
void push(struct Stack* s, char value);
char pop(struct Stack* s);
int precedence(char op);
void infixToPrefix(char* infix, char* prefix);
void reverseString(char* str);

int main() {
    char infix[MAX_SIZE], prefix[MAX_SIZE];

    // Input infix expression
    printf("Enter infix expression: ");
    fgets(infix, MAX_SIZE, stdin);

    // Remove newline character if present
    if (infix[strlen(infix) - 1] == '\n')
        infix[strlen(infix) - 1] = '\0';

    infixToPrefix(infix, prefix);

    // Output prefix expression
    printf("Prefix expression: %s\n", prefix);

    return 0;
}

// Function to push an element onto the stack
void push(struct Stack* s, char value) {
    if (s->top == MAX_SIZE - 1) {
        printf("Stack Overflow! Cannot push element.\n");
```



```
        printf("Stack Overflow! Cannot push element.\n");
        exit(EXIT_FAILURE);
    }
    s->items[++(s->top)] = value;
}

// Function to pop an element from the stack
char pop(struct Stack* s) {
    if (s->top == -1) {
        printf("Stack Underflow! Cannot pop element.\n");
        exit(EXIT_FAILURE);
    }
    return s->items[(s->top)--];
}

// Function to get precedence of operators
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
```

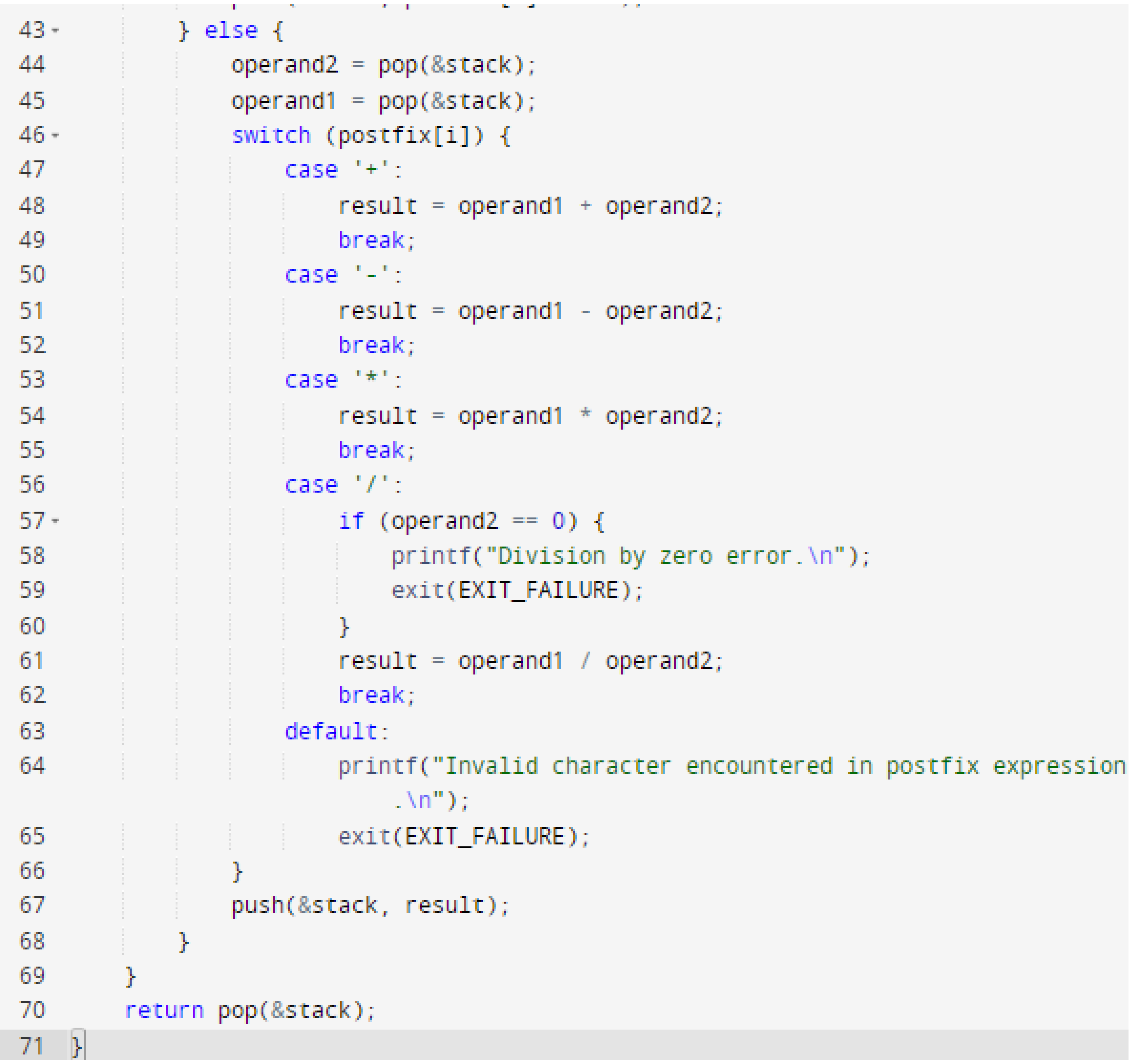
5. Write a program to evaluate a postfix expression using stack.

BHAVLEEN KAUR
3EIC-2
102155010



```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX_SIZE 100
struct Stack {
    int top;
    int items[MAX_SIZE];
};
void push(struct Stack* s, int value);
int pop(struct Stack* s);
int evaluatePostfix(char* postfix);
int main() {
    char postfix[MAX_SIZE];
    printf("Enter postfix expression: ");
    fgets(postfix, MAX_SIZE, stdin);
    if (postfix[strlen(postfix) - 1] == '\n')
        postfix[strlen(postfix) - 1] = '\0';
    int result = evaluatePostfix(postfix);
    printf("Result: %d\n", result);
    return 0;
}
void push(struct Stack* s, int value) {
    if (s->top == MAX_SIZE - 1) {
        printf("Stack Overflow! Cannot push element.\n");
        exit(EXIT_FAILURE);
    }
    s->items[++(s->top)] = value;
}
int pop(struct Stack* s) {
    if (s->top == -1) {
        printf("Stack Underflow! Cannot pop element.\n");
        exit(EXIT_FAILURE);
    }
    return s->items[(s->top)--];
}
int evaluatePostfix(char* postfix) {
    struct Stack stack;
    stack.top = -1;
    int operand1, operand2, result;
    for (int i = 0; postfix[i] != '\0'; i++) {
        if (isdigit(postfix[i])) {
            push(&stack, postfix[i] - '0');
```

BHAVLEEN KAUR
3EIC-2
102155010



```c
        } else {
            operand2 = pop(&stack);
            operand1 = pop(&stack);
            switch (postfix[i]) {
                case '+':
                    result = operand1 + operand2;
                    break;
                case '-':
                    result = operand1 - operand2;
                    break;
                case '*':
                    result = operand1 * operand2;
                    break;
                case '/':
                    if (operand2 == 0) {
                        printf("Division by zero error.\n");
                        exit(EXIT_FAILURE);
                    }
                    result = operand1 / operand2;
                    break;
                default:
                    printf("Invalid character encountered in postfix expression.\n");
                    exit(EXIT_FAILURE);
            }
            push(&stack, result);
        }
    }
    return pop(&stack);
}
```