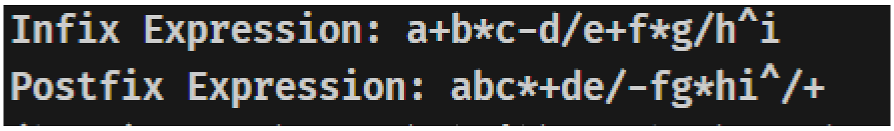
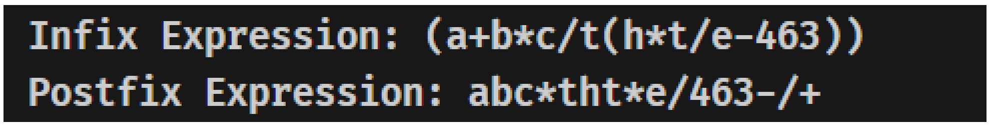
Lab 02

TASK: WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + , - , *, /, ^

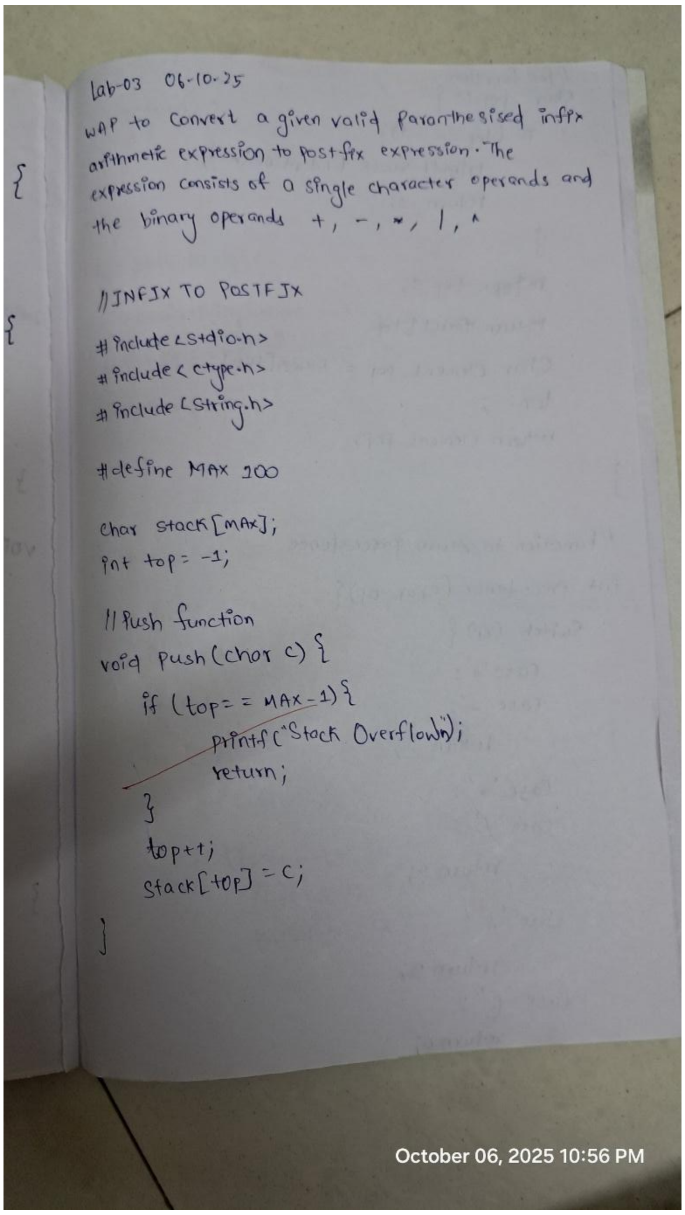
DATE: 06-10-25



```
Infix Expression: a+b*c-d/e+f*g/h^i
Postfix Expression: abc*+de/-fg*hi^/+
```



```
Infix Expression: (a+b*c/t(h*t/e-463))
Postfix Expression: abc*tht*e/463-/+
```

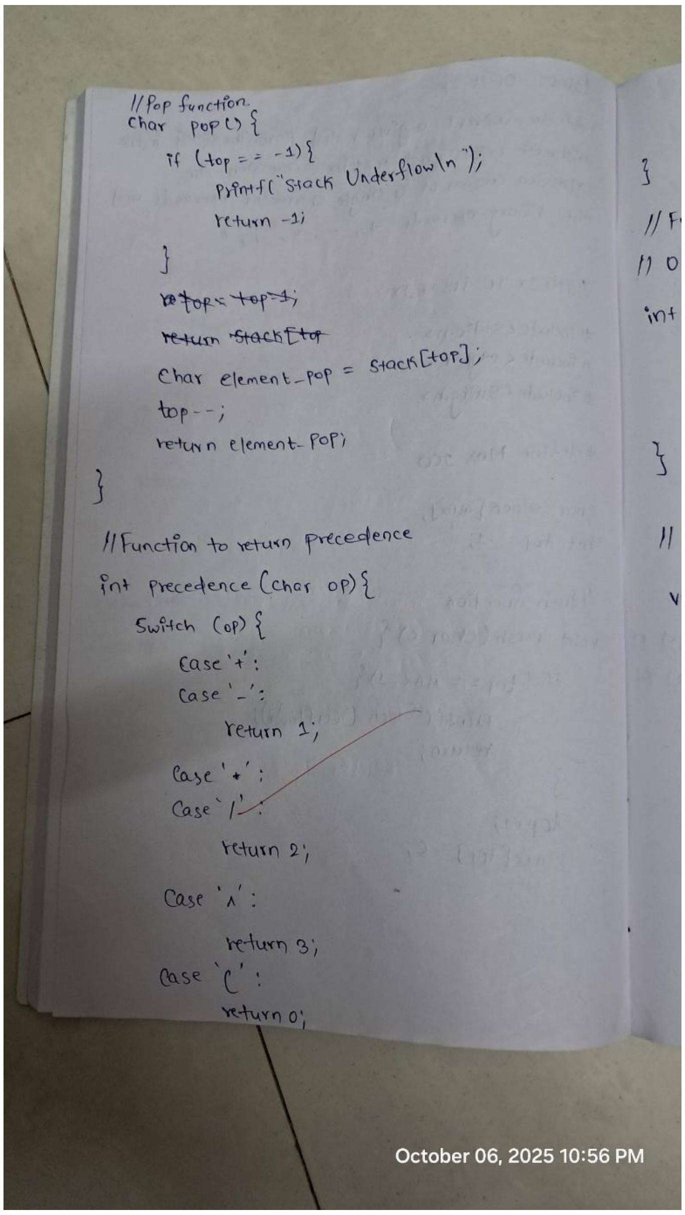


lab-03 06-10-25

WAP to convert a given valid parenthesised infix arithmetic expression to post-fix expression. The expression consists of a single character operands and the binary operands +, -, *, /, ^

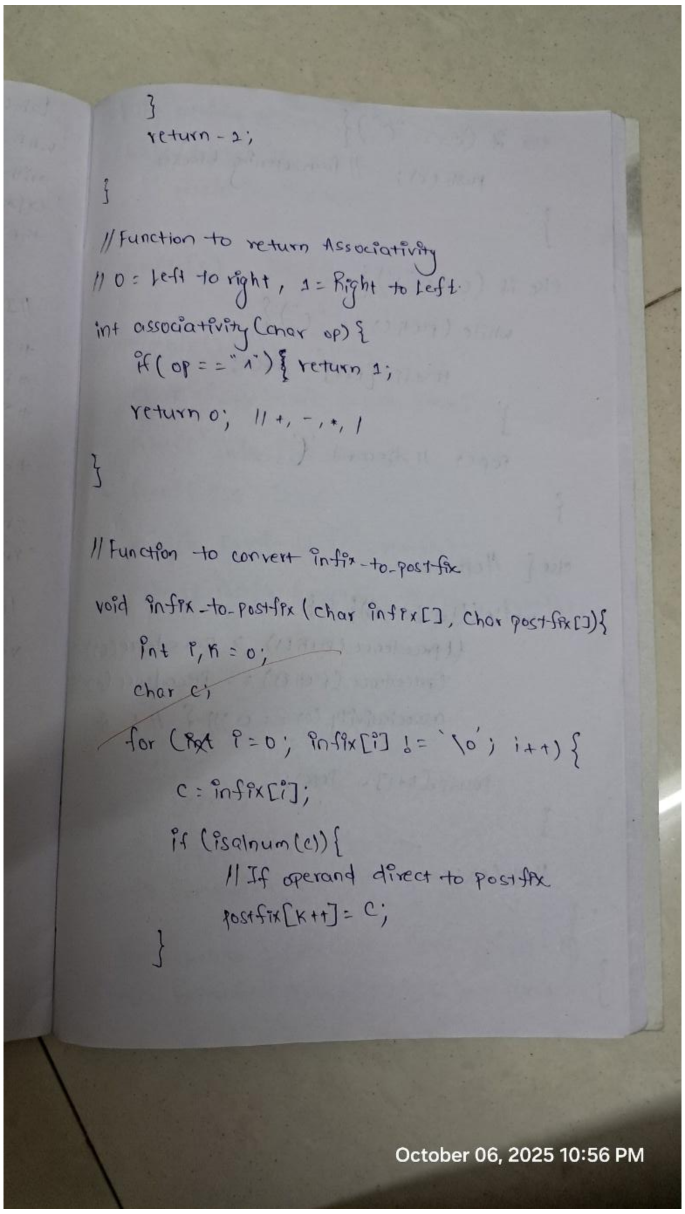
```
//INFIX TO POSTFIX

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

#define MAX 100

char stack[MAX];
int top = -1;

//Push function
void push(char c) {
    if (top= = MAX-1) {
        printf("Stack Overflow\n");
        return;
    }
    top++;
    stack[top] = c;

}
```

October 06, 2025 10:56 PM



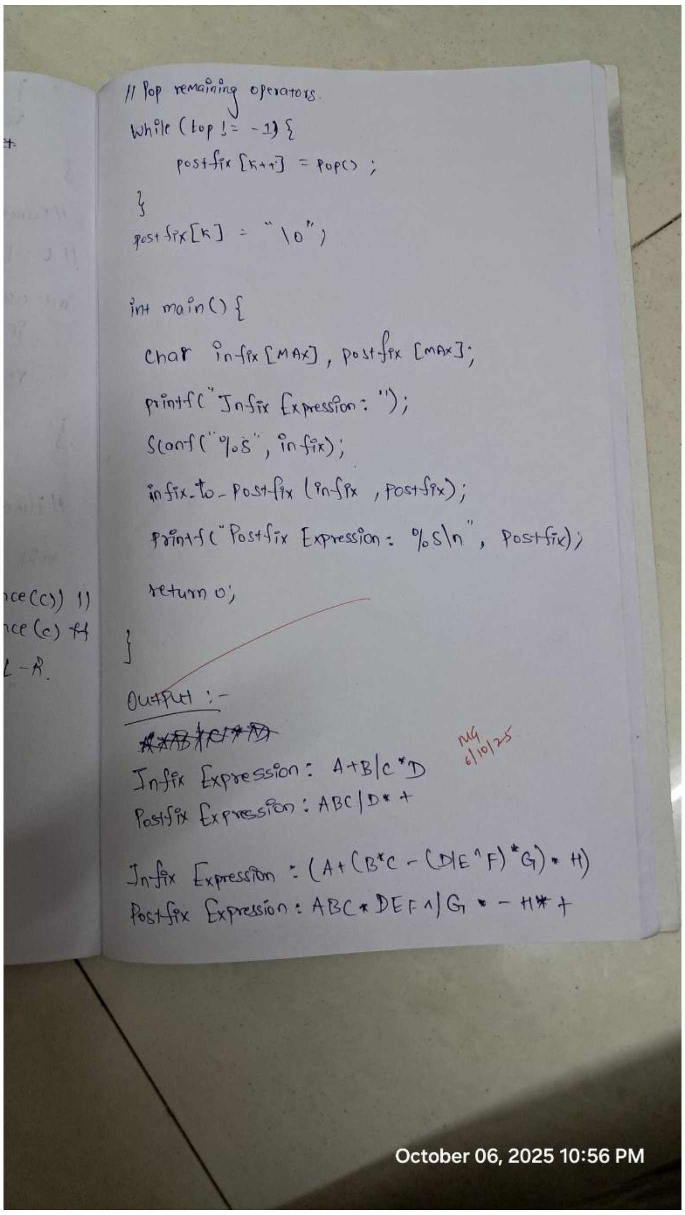
```
//Pop function
Char pop(){
    if (top == -1){
        Printf("Stack Underflow\n");
        return -1;
    }
    ~~top = top-1;~~
    ~~return stack[top~~
    Char element_pop = stack[top];
    top--;
    return element_pop;
}

//Function to return precedence
int precedence(Char op){
    Switch (op){
        Case '+':
        Case '-':
            return 1;
        Case '*':
        Case '/':
            return 2;
        Case '^':
            return 3;
        Case '(':
            return 0;
```

October 06, 2025 10:56 PM



```
    }
    return -1;

}

//Function to return Associativity
// 0 = Left to right , 1 = Right to Left.
int associativity(char op){
    if( op == '^'){ return 1;
    return 0;   // +, -, *, /
}

//Function to convert infix-to-postfix
void infix_to_postfix (char infix[], Char postfix[]){
    int i, k = 0;
    char c;
    for (~~int~~ i = 0; infix[i] != '\0'; i++) {
        c = infix[i];
        if (isalnum(c)){
            // If operand direct to postfix
            postfix[k++] = c;
        }
```

October 06, 2025 10:56 PM



```
ce(c)) ||
ce (c) &&
L-R.
```

```
// Pop remaining operators.
While (top != -1) {
     postfix [k++] = pop() ;
}
postfix[k] = "\0";

int main() {
  char infix [MAX], postfix [MAX];
  printf("Infix Expression: ");
  scanf("%s", infix);
  infix_to_postfix (infix , postfix);
  printf("Postfix Expression: %s\n", postfix);
  return 0;
}
```

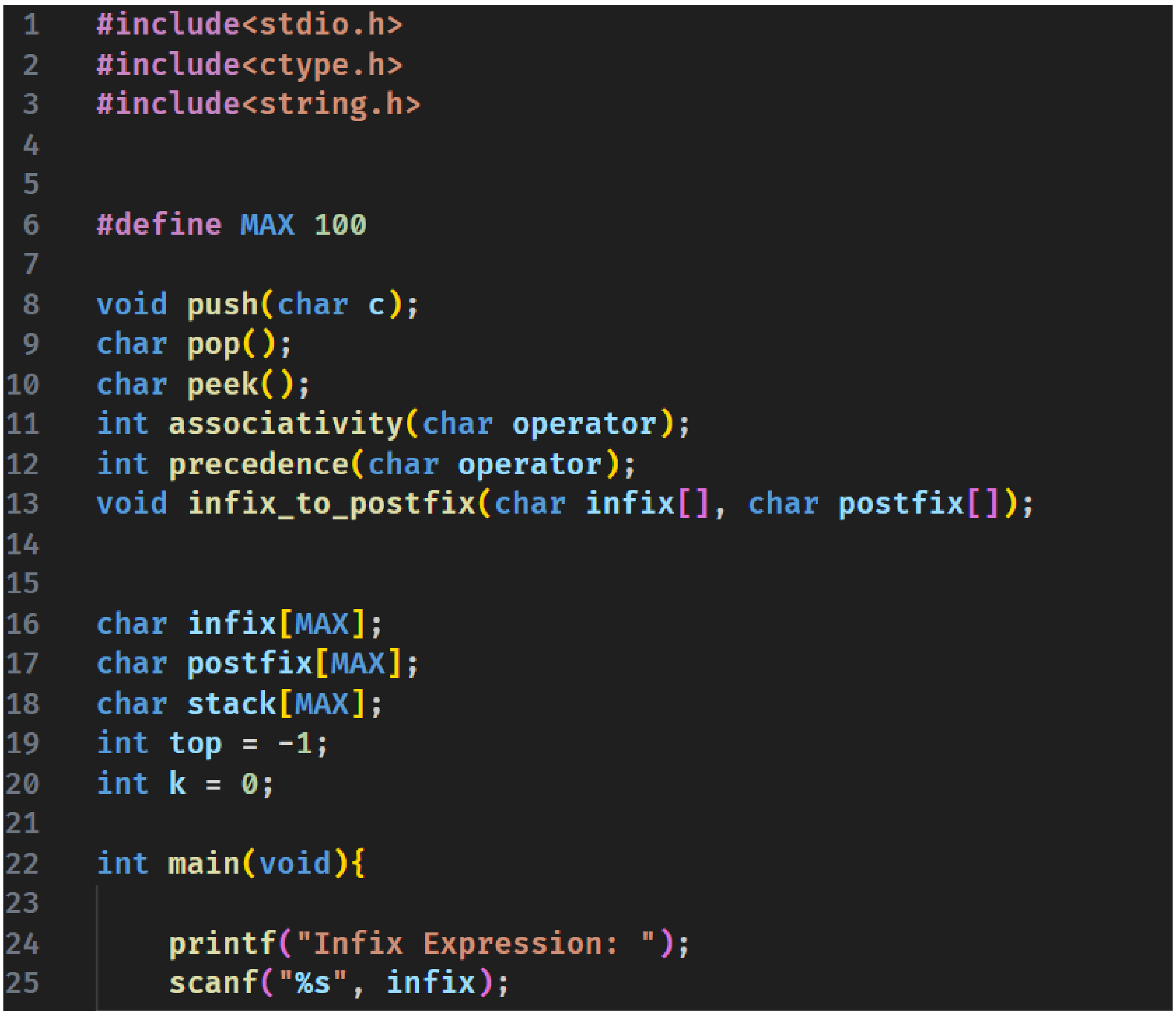
Output :-

~~A*AB*C**D~~

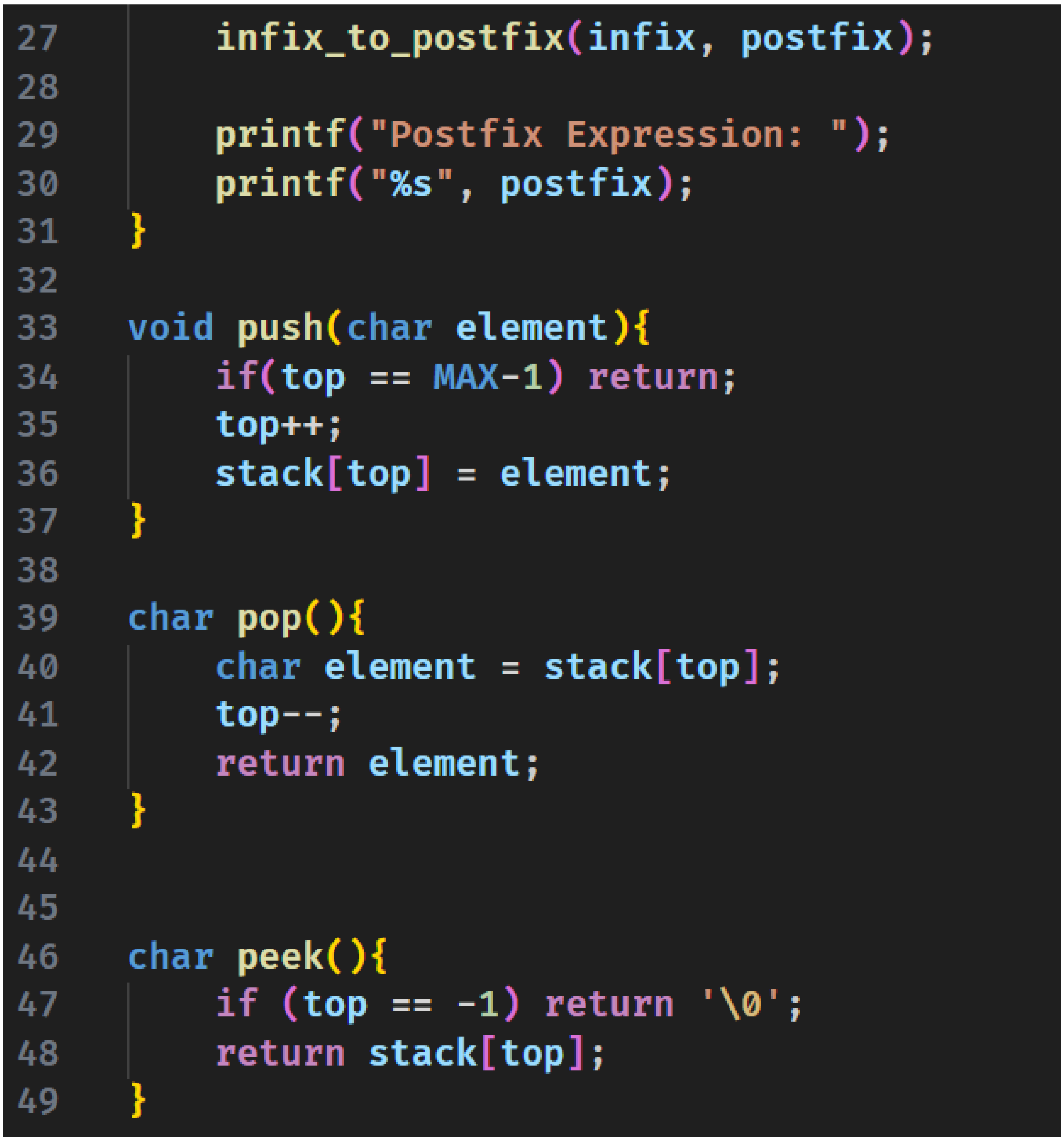
```
Infix Expression: A+B/C*D
Postfix Expression: ABC/D*+

Infix Expression : (A+(B*C-(D/E^F)*G)*H)
Postfix Expression: ABC*DEF^/G*-H*+
```

NG
6/10/25.

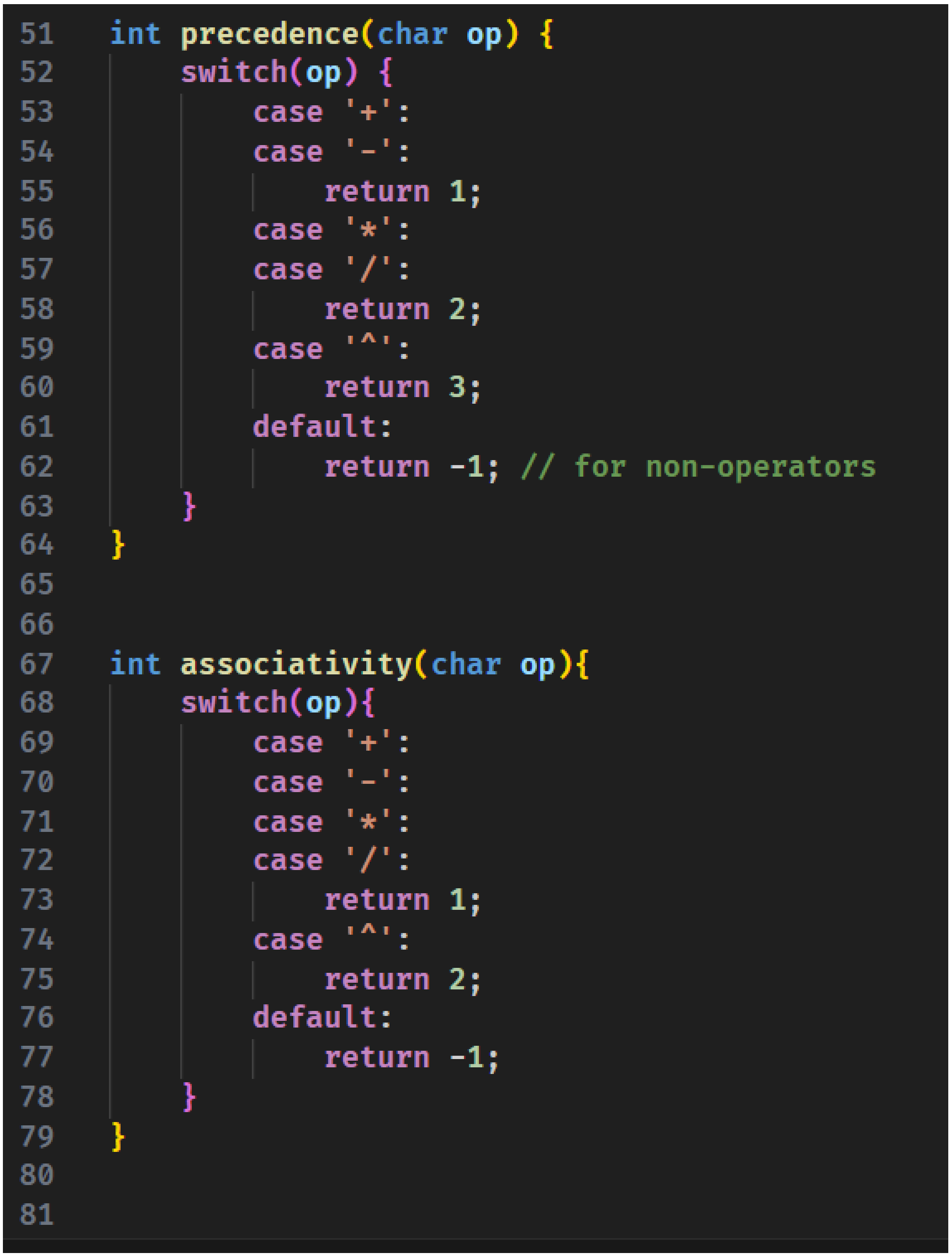
October 06, 2025 10:56 PM

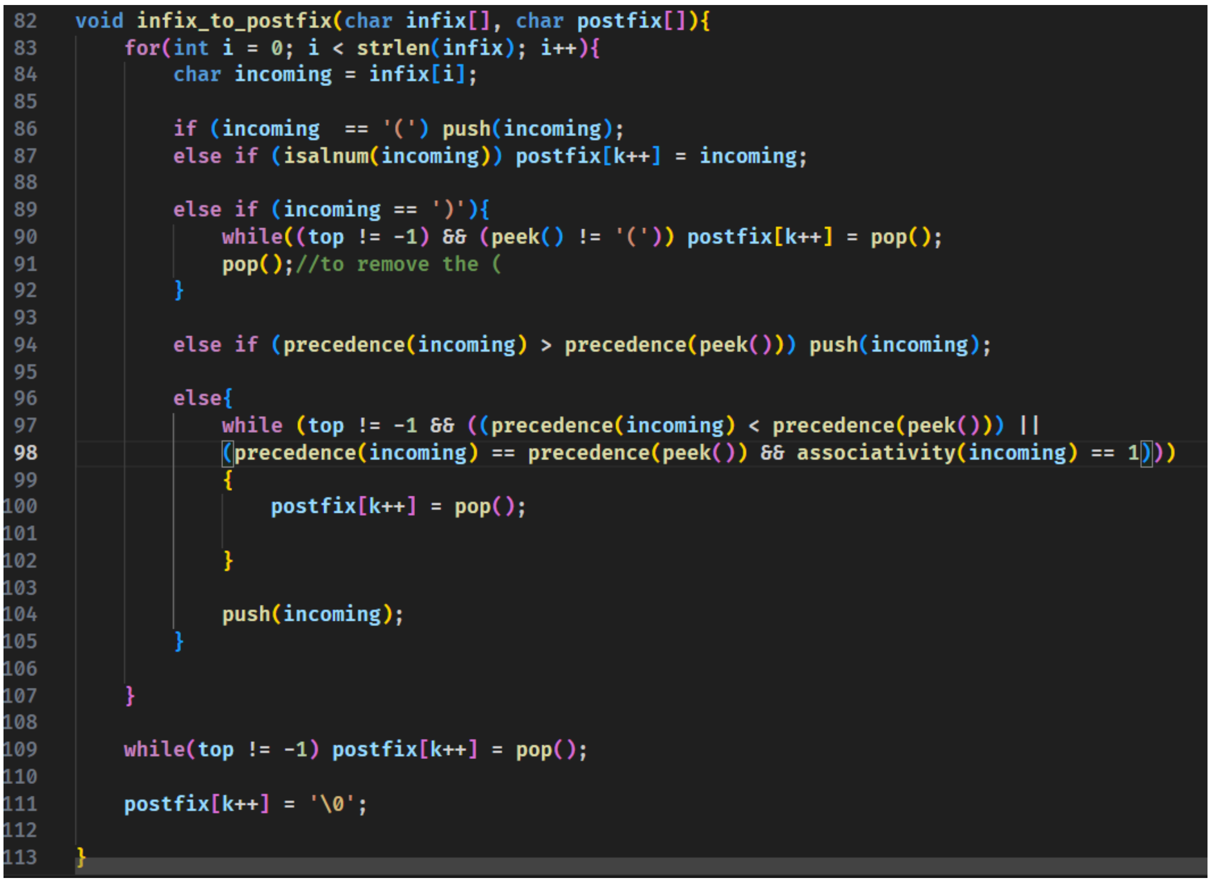


```c
#include<stdio.h>
#include<ctype.h>
#include<string.h>


#define MAX 100

void push(char c);
char pop();
char peek();
int associativity(char operator);
int precedence(char operator);
void infix_to_postfix(char infix[], char postfix[]);


char infix[MAX];
char postfix[MAX];
char stack[MAX];
int top = -1;
int k = 0;

int main(void){

    printf("Infix Expression: ");
    scanf("%s", infix);
```



```c
    infix_to_postfix(infix, postfix);

    printf("Postfix Expression: ");
    printf("%s", postfix);
}

void push(char element){
    if(top == MAX-1) return;
    top++;
    stack[top] = element;
}

char pop(){
    char element = stack[top];
    top--;
    return element;
}


char peek(){
    if (top == -1) return '\0';
    return stack[top];
}
```

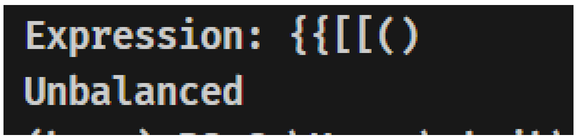

```
int precedence(char op) {
    switch(op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return -1; // for non-operators
    }
}


int associativity(char op){
    switch(op){
        case '+':
        case '-':
        case '*':
        case '/':
            return 1;
        case '^':
            return 2;
        default:
            return -1;
    }
}


```



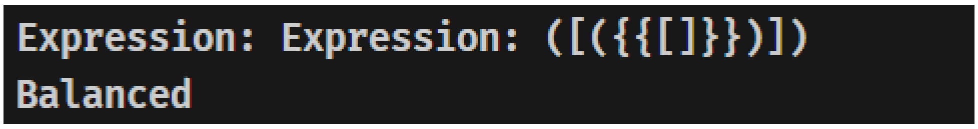
```c
void infix_to_postfix(char infix[], char postfix[]){
    for(int i = 0; i < strlen(infix); i++){
        char incoming = infix[i];

        if (incoming  == '(') push(incoming);
        else if (isalnum(incoming)) postfix[k++] = incoming;

        else if (incoming == ')'){
            while((top != -1) && (peek() != '(')) postfix[k++] = pop();
            pop();//to remove the (
        }

        else if (precedence(incoming) > precedence(peek())) push(incoming);

        else{
            while (top != -1 && ((precedence(incoming) < precedence(peek())) ||
            (precedence(incoming) == precedence(peek()) && associativity(incoming) == 1)))
            {
                postfix[k++] = pop();

            }

            push(incoming);
        }

    }

    while(top != -1) postfix[k++] = pop();

    postfix[k++] = '\0';

}
```

TASK: Check for balanced parentheses

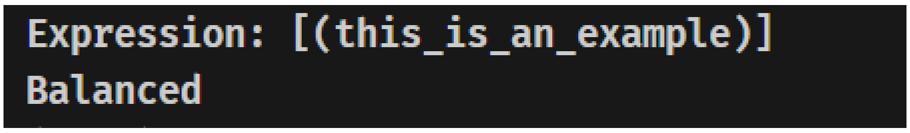
DATE: 06-09-25



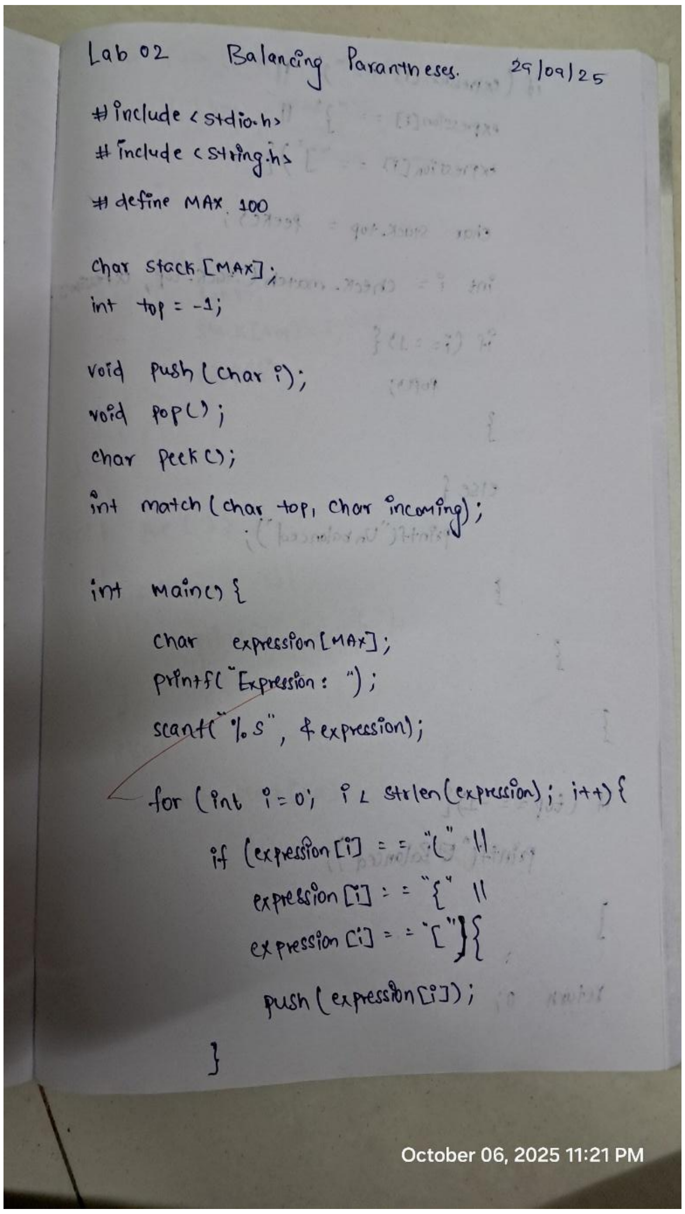
```
Expression: {{[[()
Unbalanced
```



```
Expression: Expression: ([({{[]}})])
Balanced
```



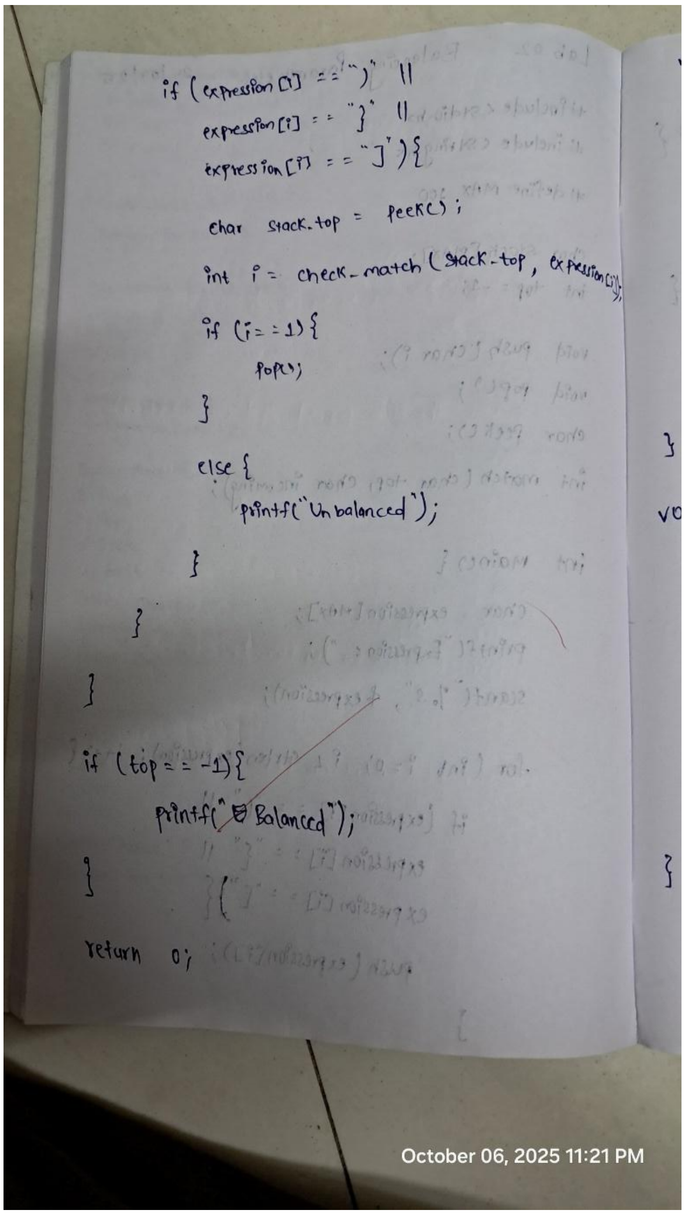
```
Expression: [(this_is_an_example)]
Balanced
```



Lab 02   Balancing Paranthesess.   29/09/25

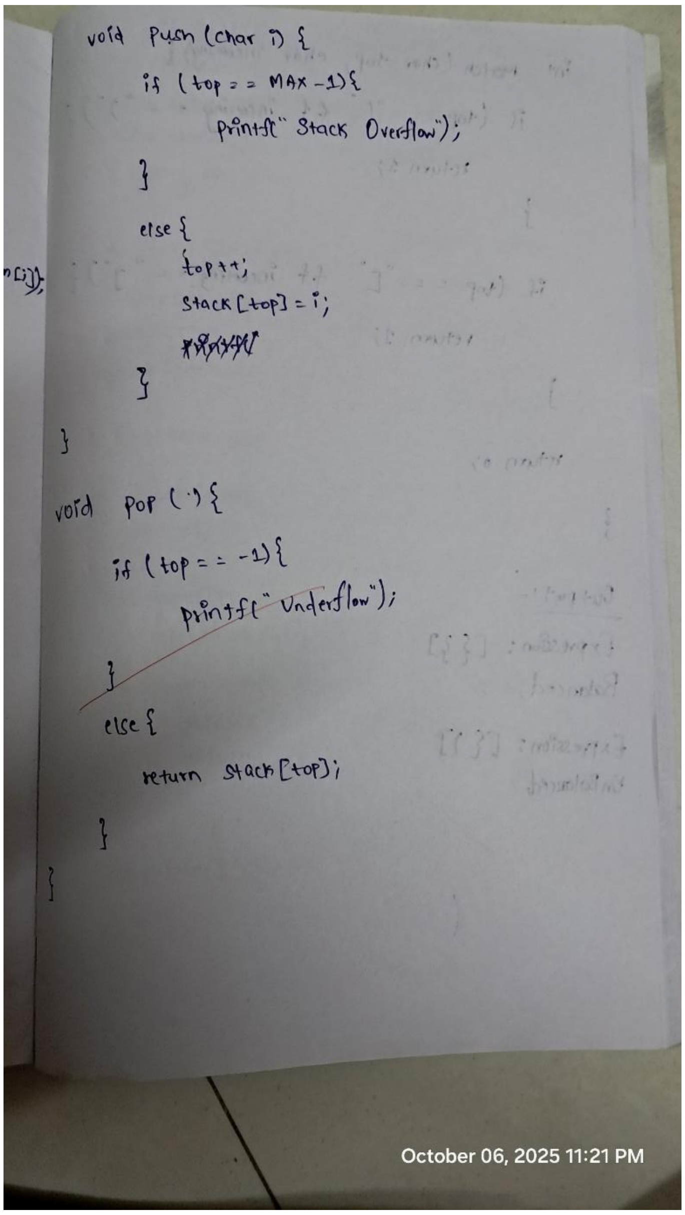
```
#include <stdio.h>
#include <string.h>

#define MAX 100

char stack [MAX];
int top = -1;

void push (char i);
void pop();
char peek();
int match (char top, char incoming);

int main() {
      char  expression[MAX];
      printf("Expression: ");
      scanf("%s", &expression);

      for (int i=0; i < strlen(expression); i++){
            if (expression[i] = = "(" ||
                  expression[i] = = "{" ||
                  expression[i] = = "["){
                  push (expression[i]);

            }
```

October 06, 2025 11:21 PM

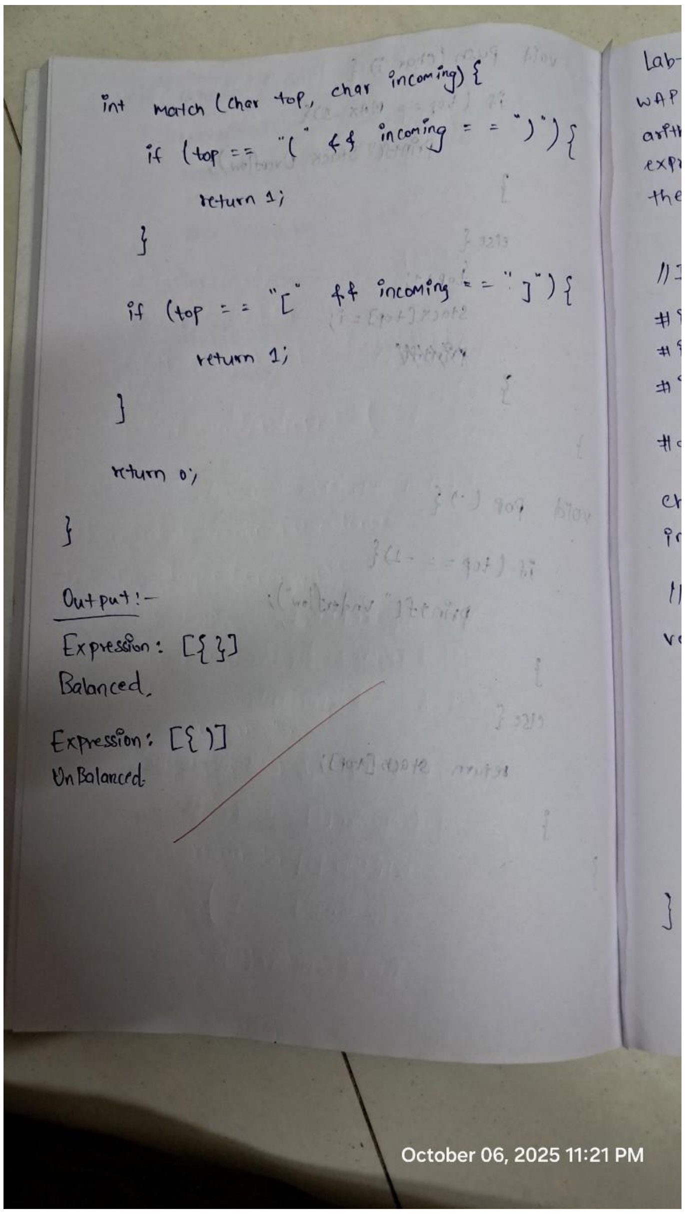


```
        if (expression[i] == ")" ||
            expression[i] == "}" ||
            expression[i] == "]"){

            char stack.top = peek();

            int i = check_match (stack_top, expression[i]);

            if (i==1){
                pop();
            }

            else {
                printf("Unbalanced");

            }

        }

    }

if (top == -1){
    printf("~~θ~~ Balanced");
}

return 0;
```

October 06, 2025 11:21 PM



```
void Push (char i) {
    if (top == MAX-1){
        printf(" Stack Overflow");
    }
    else {
        top++;
        stack[top] = i;
        ~~[illegible]~~
    }
}
void pop () {
    if (top == -1){
        printf(" Underflow");
    }
    else {
        return stack[top];
    }
}
```

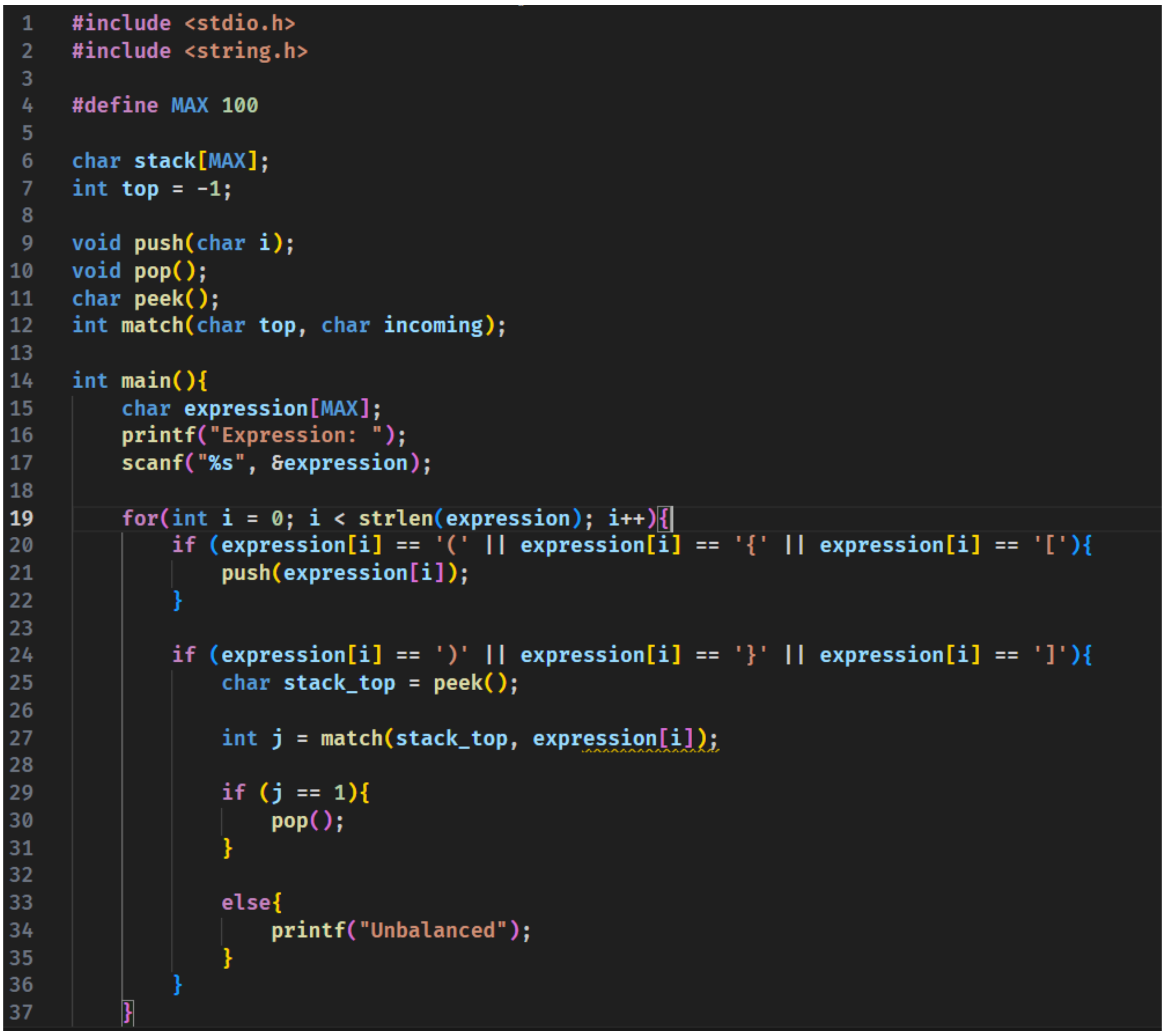


```
int  match (char top, char incoming) {
    if (top == "(" && incoming == ")") {
        return 1;
    }

    if (top == "[" && incoming == "]") {
        return 1;
    }

    return 0;

}
```

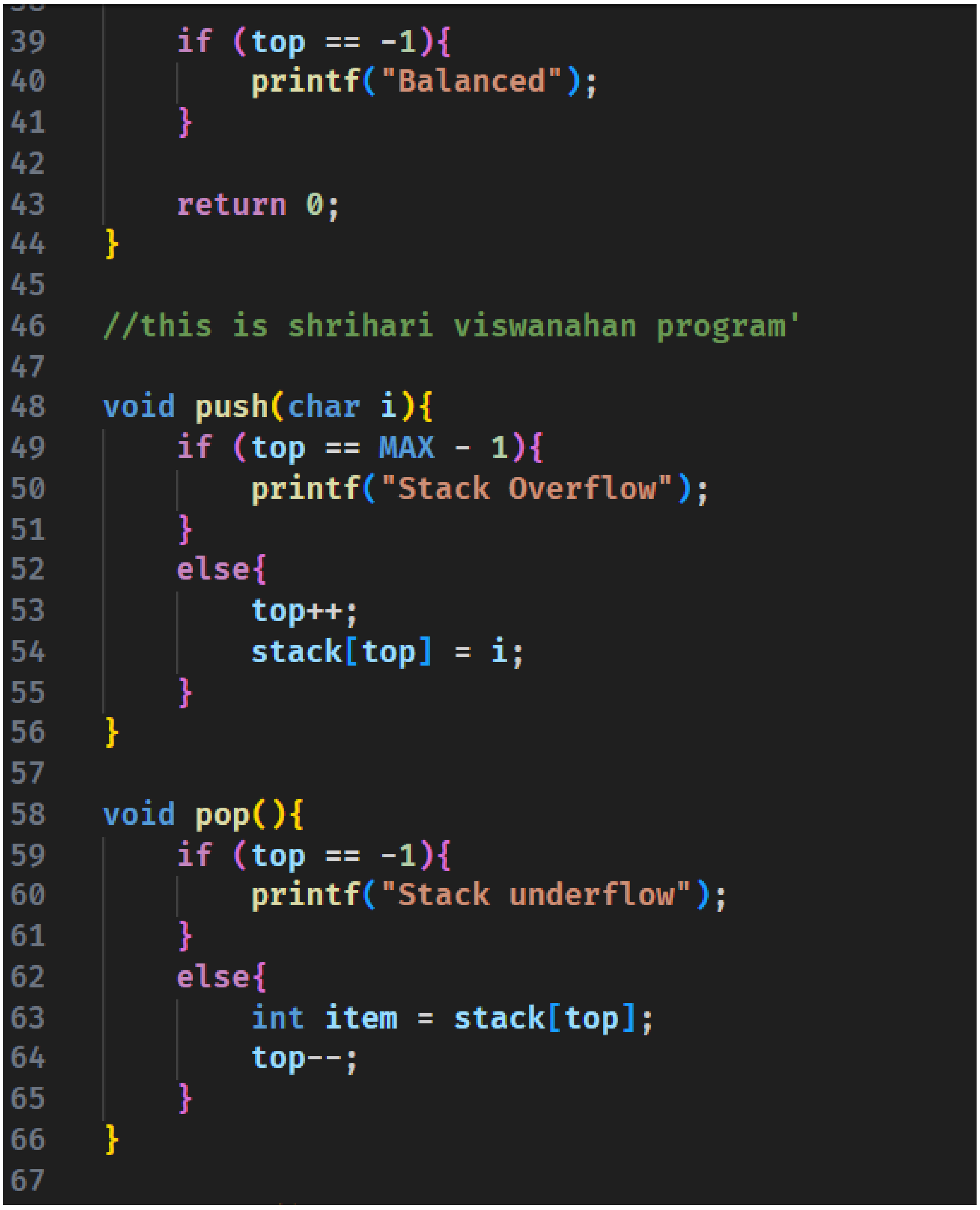
Output:-

```
Expression: [{}]
Balanced.

Expression: [{)]
UnBalanced.
```

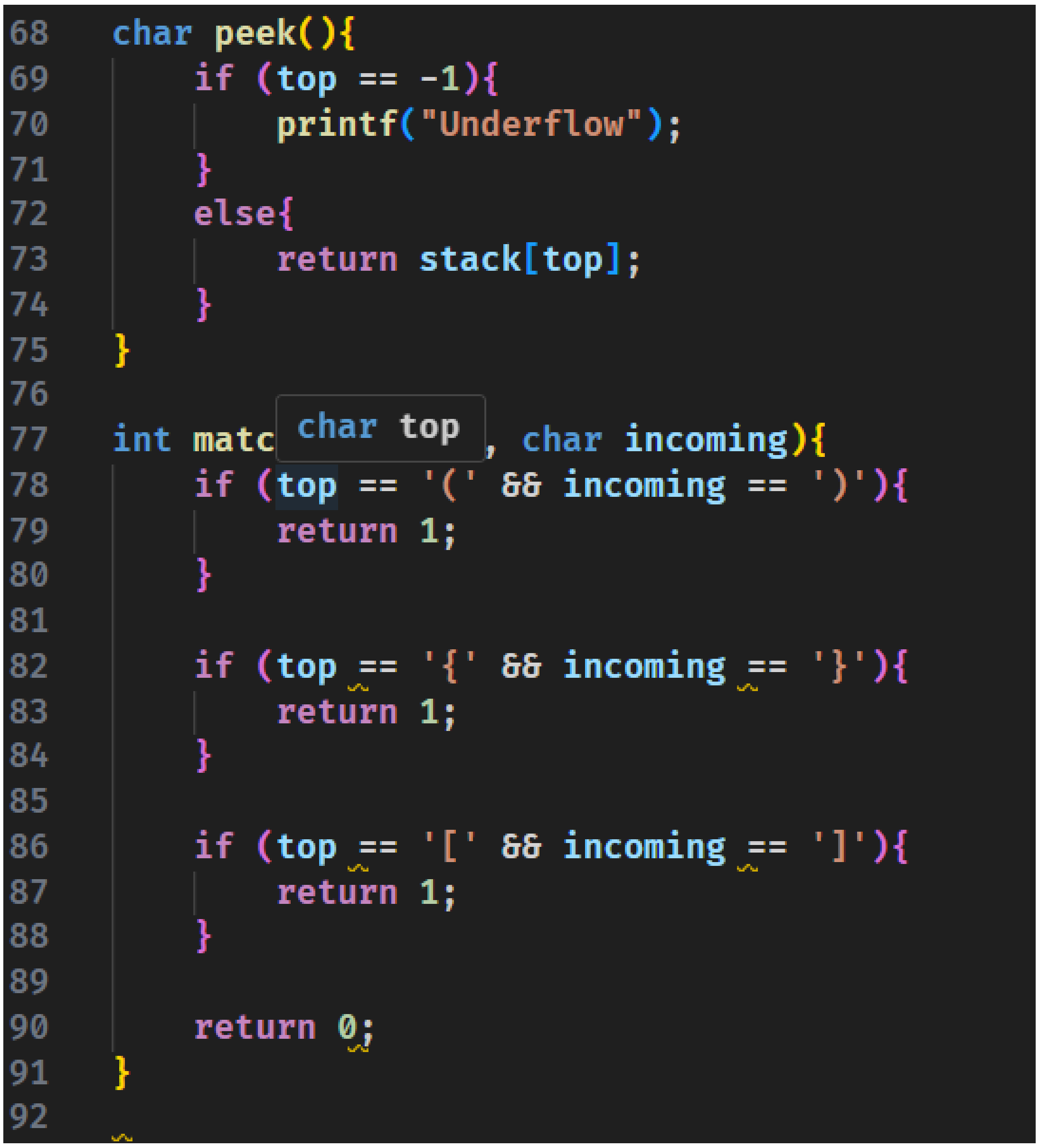
October 06, 2025 11:21 PM



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

#define MAX 100

char stack[MAX];
int top = -1;

void push(char i);
void pop();
char peek();
int match(char top, char incoming);

int main(){
    char expression[MAX];
    printf("Expression: ");
    scanf("%s", &expression);

    for(int i = 0; i < strlen(expression); i++){
        if (expression[i] == '(' || expression[i] == '{' || expression[i] == '['){
            push(expression[i]);
        }

        if (expression[i] == ')' || expression[i] == '}' || expression[i] == ']'){
            char stack_top = peek();

            int j = match(stack_top, expression[i]);

            if (j == 1){
                pop();
            }

            else{
                printf("Unbalanced");
            }
        }
    }
```



```
    if (top == -1){
        printf("Balanced");
    }

    return 0;
}

//this is shrihari viswanahan program'

void push(char i){
    if (top == MAX - 1){
        printf("Stack Overflow");
    }
    else{
        top++;
        stack[top] = i;
    }
}

void pop(){
    if (top == -1){
        printf("Stack underflow");
    }
    else{
        int item = stack[top];
        top--;
    }
}
```



```c
char peek(){
    if (top == -1){
        printf("Underflow");
    }
    else{
        return stack[top];
    }
}

int matc char top, char incoming){
    if (top == '(' && incoming == ')'){
        return 1;
    }

    if (top == '{' && incoming == '}'){
        return 1;
    }

    if (top == '[' && incoming == ']'){
        return 1;
    }

    return 0;
}

```