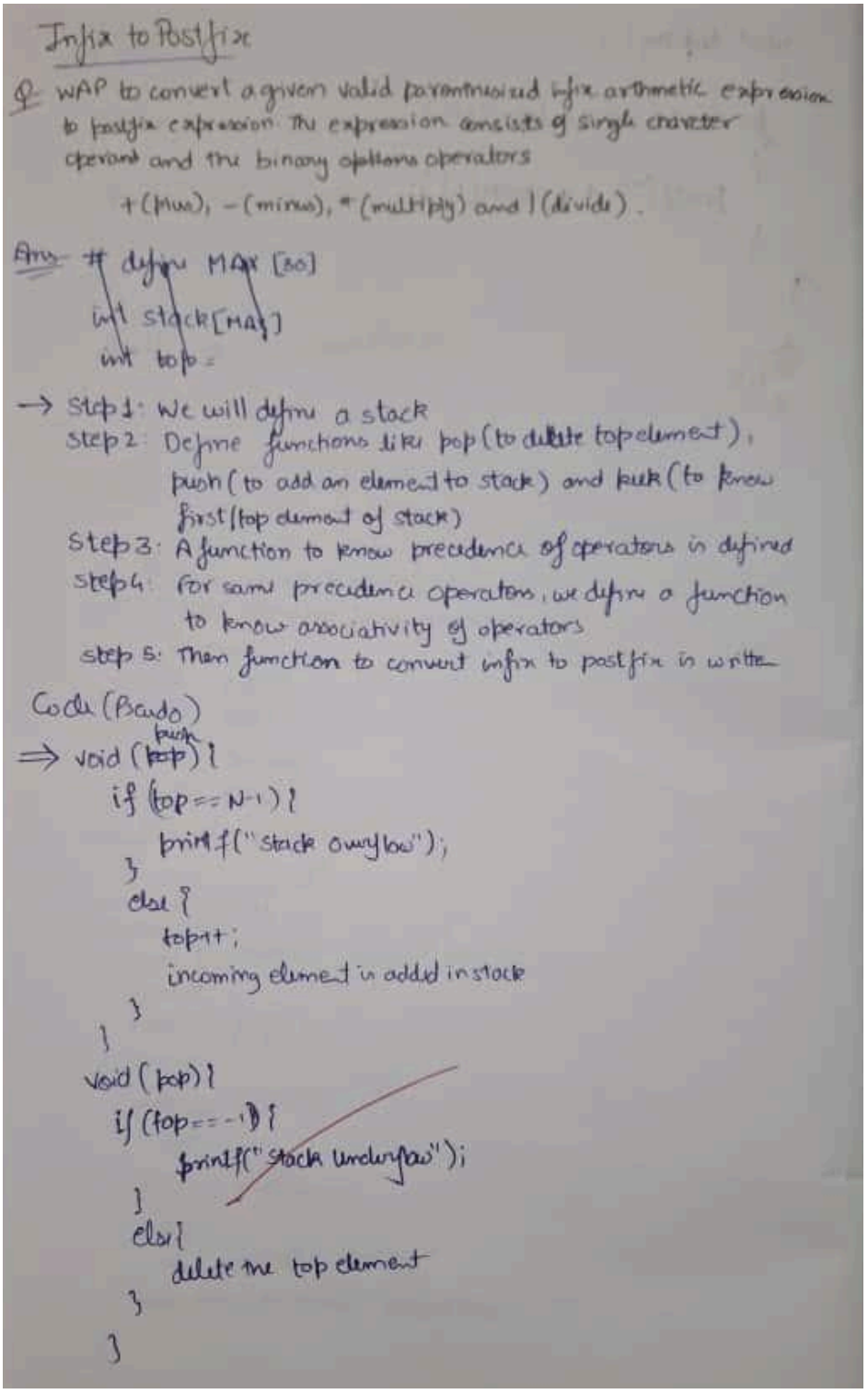
OUTPUT:

```
Enter valid parenthesized infix expression:2+4*(3-4/5)^9
Postfix Expression: 24345/-9^*+
PS C:\Users\student\Documents\1BF24CS195(DS Lab)>
```

OBSERVATION:



Infix to Postfix

Q: WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operand and the binary operators + (plus), - (minus), * (multiply) and / (divide).

Ans: # define MAX [30]
int stack[MAX]
int top =

→ Step 1: We will define a stack
Step 2: Define functions like pop (to delete top element), push (to add an element to stack) and peek (to know first/top element of stack)
Step 3: A function to know precedence of operators is defined
Step 4: For same precedence operators, we define a function to know associativity of operators
Step 5: Then function to convert infix to postfix is written

Code (Pseudo)

⇒ void (push) {
if (top == N-1) {
printf("stack overflow");
}
else {
top++;
incoming element is added in stack
}
}
void (pop) {
if (top == -1) {
printf("stack underflow");
}
else {
delete the top element
}
}

```
void (peek){
   to su nu top element of stack
}
int
void precedence (char op){
      switch (op){
         case '+'
         case '-': return 1;
                   break;
         case '*':
         case '/': return 2;
         case '^': return 3;
         case '(': return 0;
         default: return -1;
      }
}
int associativity (char op){
      if (op=='^'){
         return 1;
      }
      return 0;
}
int conversion (char infix[], char postfix[]){
      int i, k=0;
      for ( [illegible] i=0; infix[i]!='\0'; i++){
        c = infix[i];
        if (isalnum(c)){
            push(c);
        }
        else
        {
          precedence(c);
          if (precedence(c)
        else if (c=='('){
            push(c);
        }
        else if (c==')'){
            while(peek()!='('){
              postfix[k++] = pop();
```

```
        pop();
      }
      else {
      while (top!=-1 && precedence(peek())>=precedence(c)||
                        precedence(peek())==precedence(c)&&
                        associativity(c)==0)){
            postfix[k++]=pop();
      }  }
       push(c);
      }
    }
    while(top!=-1){
        postfix[k++]=pop();
    }
    postfix='\0';
}
```

Code

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#define MAX 100

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

char push(char c){
    if(top==MAX-1){
        printf("stack overflow\n");
    }
    else{
        top++;
        stack[top]=c;
    }

}
```

```
char pop(){
    if(top==-1){
        printf("Stack Underflow\n");
        return -1;
    }
    return stack[top--];
}

char peek(){
    if(top==-1){
        return -1;
    }
    return stack[top];
}
int precedence(char op){
    switch(op){
        case '+':
        case '-':
            return 1;
            break;
        case '*':
        case '/':
            return 2;
            break;
        case '^':
            return 3;
            break;
        case '(':
            return 0;
            break;
    }
    return -1;
}
int associativity(char op){
    if (op=='^')
        return 1;
    return 0;
}
```

```
int conversion(char infix[], char postfix[]){
    int i, k=0;
    char c;
    for(i=0; infix[i]!='\0'; i++){
        c= infix[i];

        if (isalnum(c)){
            postfix[k++]=c;
        }
        else if(c=='('){
            push(c);
        }
        else if(c==')'){
            while(peek()!='('){
                postfix[k++]=pop();
            }
            pop();
        }
        else{
            while(top!=-1 && (precedence(peek()) > precedence(c)
                || (precedence(peek())==precedence(c) &&
                    associativity(c)==0)){
                postfix[k++]=pop();
            }
            push(c);
        }
    }
    while(top!=-1){
        postfix[k++]=pop();
    }
    postfix[k]='\0';
}
int main(){
    char infix[MAX], char postfix[MAX];

    printf("Enter valid parenthesised infix expression:");
    scanf("%s", infix);
    conversion(infix, postfix);
    printf("Postfix Expression: %s\n", postfix);
}
```

Output → Enter Valid infix Expression: 2-4*(3^6+4)/8

Postfix Expression: 2436^4+*8/-

13/10/25