14/10/25

~~Q) WAP to convert second lab program description to be executed for Todays DS lab~~

Q) WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operand and binary operator

+, -, *, /

Pseudo Code

```
#Define N 100
int stack[N];
int top = -1;

void push (char c) {
    stack[++top] = c;
}

~~char pop(char c)~~
char pop () {
    return stack[top--];
}

char peek () {
    return stack[top];
}
```

```
int precedence(char o){
    switch(o){
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '(':
            return 0;
    }
    return -1;
}

int assosivity(char o){
    if(o == '^')
        return 1;
    return 0;
}

void Conversion(char infix[], char postfix[]){
    int k=0;
    char c;
    for(int i=0; infix[i] != '\0'; i++){
        if(c == isnum(c)){
            postfix[++k] = c;
        }
        else if(c == '('){
            push(c);
        }
```

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

int main() {
    char infix[N];
    char postfix[N];
    // Get infix input expression
    Conversion(infix[N], postfix[N]);
    // Print postfix expression
}
```

Code:

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

#define N 100

int stack[N];
int top = -1;

void push(char c) {
    if (top == N-1) {
        printf("Stack Overflow\n");
        return;
    }
    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 o) {
    switch (o) {
        Case '+':
        Case '-':
            return 1;
        Case '*':
        Case '/':
            return 2;
        Case '^':
            return 3;
    }
    return -1;
}

int assosivity (char o){
    if(o=='^')
        return 1;
    return 0;
}

void convesion (char infix[], char postfix[]){
    int k=0;
    char c;
    for(int 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[N], postfix[N];
    printf("Enter a postfix expression: ");
    scanf("%s", infix);
    conversion(infix, postfix);
    printf("Postfix expression: %s", postfix);
    return 0;
}
```

Output ⇒ Enter infix Expression:
(3+2/3+(9-4))

Postfix Expression: 323/+94-+