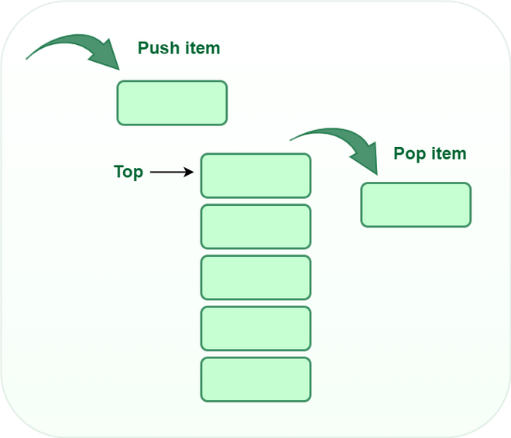
**Stacks**

A stack is a linear data structure in which the insertion of a new element and removal of an existing element takes place at the same end represented as the top of the stack.



**1. Develop a menu driven program demonstrating the following operations on a Stack:**

**push(), pop(), isEmpty(), isFull(), display(), and peek().**

Note: Use either arrays or linked list to implement stack.

**Algorithm for push:**

begin

if stack is full

return

endif

else

increment top

stack[top] assign value

end else

end procedure

**Algorithm for pop:**

begin

if stack is empty

return

endif

else

store value of stack[top]

decrement top

return value

end else

end procedure

A stack can be implemented using an array or a linked list. In an array-based implementation, the push operation is implemented by incrementing the index of the top element and storing the new element at that index. The pop operation is implemented by decrementing the index of the top element and returning the value stored at that index.

class Stack {

    int top;

public:

    int a[MAX]; // Maximum size of Stack

    Stack() { top = -1; }

    bool push(int x);

    int pop();

    int peek();

    bool isEmpty();

};

bool Stack::push(int x)

{

    if (top >= (MAX - 1)) {

        cout << "Stack Overflow";

        return false;

    }

    else {

        a[++top] = x;

        cout << x << " pushed into stack\n";

        return true;

    }

}

int Stack::pop()

{

    if (top < 0) {

        cout << "Stack Underflow";

        return 0;

    }

    else {

        int x = a[top--];

        return x;

    }

}

int Stack::peek()

{

    if (top < 0) {

        cout << "Stack is Empty";

        return 0;

    }

    else {

        int x = a[top];

        return x;

    }

}

bool Stack::isEmpty()

{

    return (top < 0);

}

int main()

{

    class Stack s;

    s.push(10);

    s.push(20);

    s.push(30);

    cout << s.pop() << " Popped from stack\n";

    cout << "Top element is : " << s.peek() << endl;

    cout <<"Elements present in stack : ";

    while(!s.isEmpty())

    {

        cout << s.peek() <<" ";

        s.pop();

    }

    return 0;

}

In a linked list-based implementation, the push operation is implemented by creating a new node with the new element and setting the next pointer of the current top node to the new node. The pop operation is implemented by setting the next pointer of the current top node to the next node and returning the value of the current top node.

class StackNode {

public:

    int data;

    StackNode\* next;

};

StackNode\* newNode(int data)

{

    StackNode\* stackNode = new StackNode();

    stackNode->data = data;

    stackNode->next = NULL;

    return stackNode;

}

int isEmpty(StackNode\* root)

{

    return !root;

}

void push(StackNode\*\* root, int data)

{

    StackNode\* stackNode = newNode(data);

    stackNode->next = \*root;

    \*root = stackNode;

    cout << data << " pushed to stack\n";

}

int pop(StackNode\*\* root)

{

    if (isEmpty(\*root))

        return INT\_MIN;

    StackNode\* temp = \*root;

    \*root = (\*root)->next;

    int popped = temp->data;

    free(temp);

    return popped;

}

int peek(StackNode\* root)

{

    if (isEmpty(root))

        return INT\_MIN;

    return root->data;

}

int main()

{

    StackNode\* root = NULL;

    push(&root, 10);

    push(&root, 20);

    push(&root, 30);

    cout << pop(&root) << " popped from stack\n";

    cout << "Top element is " << peek(root) << endl;

    cout <<"Elements present in stack : ";

    while(!isEmpty(root))

    {

        cout << peek(root) <<" ";

        pop(&root);

    }

    return 0;

}

**2. Given a String, Reverse it using STACK. For example “data structure” should be output as “erutcurtsatad.”**

Follow the steps given below:

* Create an empty stack.
* One by one push all characters of string to stack.
* One by one pop all characters from stack and put them back to string.

#define max 100

int top, stack[max];

void push(char x){

if(top == max-1){

printf("stack overflow");

} else { stack[++top]=x; }

}

void pop(){

printf("%c",stack[top--]);

}

main()

{

char str[]="data structure";

int len = strlen(str);

int i;

for(i=0;i<len;i++)

push(str[i]);

for(i=0;i<len;i++)

pop();

}

**3. Write a program that checks if an expression has balanced parentheses.**

**2 \* ( ( 4/2 ) + 5 ), 2 \* ( ( 4/2 ) + 5**

int main()

{

char expression[50];

int x=0, i=0;

printf("\nEnter an expression");

scanf("%s", expression);

while(expression[i]!= '\0')

{

if(expression[i]=='(')

{

x++;

}

else if(expression[i]==')')

{

x--;

if(x<0)

break;

}

i++;

}

// Condition to check whether x is equal to 0 or not.

if(x==0) { printf("Expression is balanced"); }

else { printf("Expression is unbalanced"); }

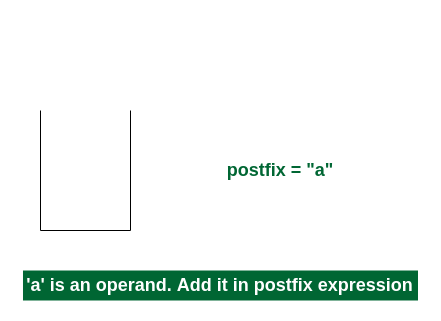
return 0;

}

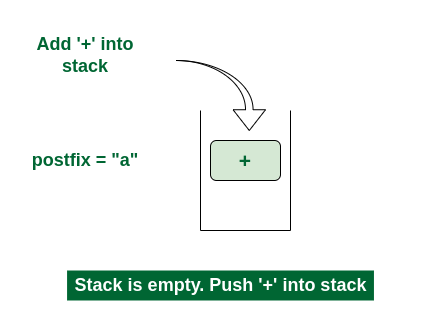
**4. Write a program to convert an Infix expression into a Postfix expression.**

Consider the infix expression exp = “a+b\*c+d” and the infix expression is scanned using the iterator i, which is initialized as i = 0.

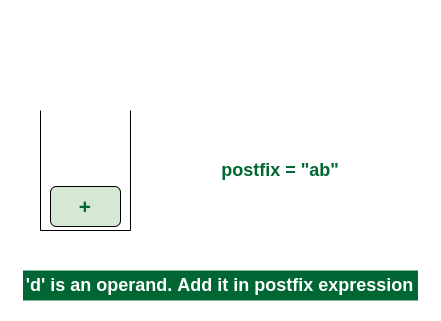
1st Step: Here i = 0 and exp[i] = ‘a’ i.e., an operand. So add this in the postfix expression. Therefore, postfix = “a”.



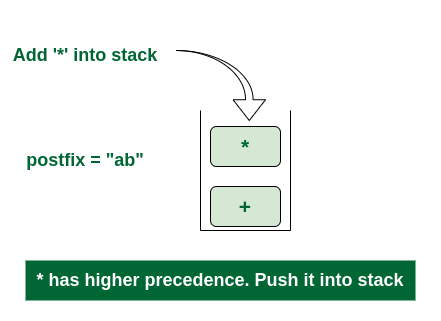
2nd Step: Here i = 1 and exp[i] = ‘+’ i.e., an operator. Push this into the stack. postfix = “a” and stack = {+}.



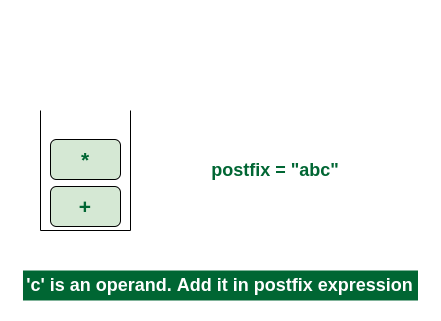
3rd Step: Now i = 2 and exp[i] = ‘b’ i.e., an operand. So add this in the postfix expression. postfix = “ab” and stack = {+}.



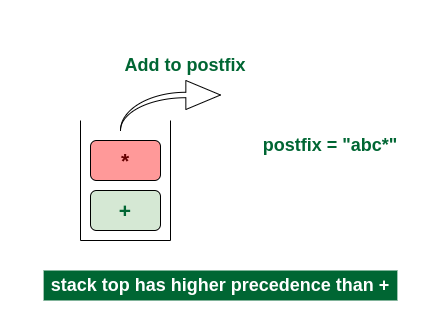
4th Step: Now i = 3 and exp[i] = ‘\*’ i.e., an operator. Push this into the stack. postfix = “ab” and stack = {+, \*}.



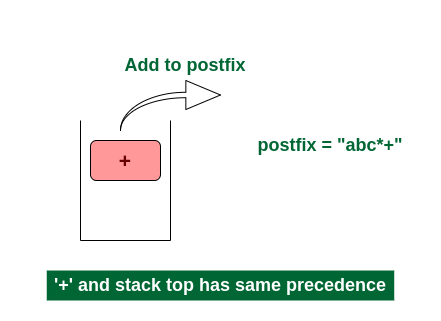
5th Step: Now i = 4 and exp[i] = ‘c’ i.e., an operand. Add this in the postfix expression. postfix = “abc” and stack = {+, \*}.



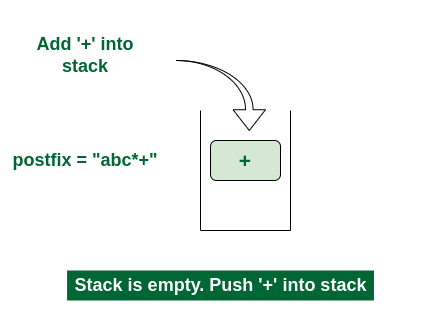
6th Step: Now i = 5 and exp[i] = ‘+’ i.e., an operator. The topmost element of the stack has higher precedence. So pop until the stack becomes empty or the top element has less precedence. ‘\*’ is popped and added in postfix. So postfix = “abc\*” and stack = {+}.



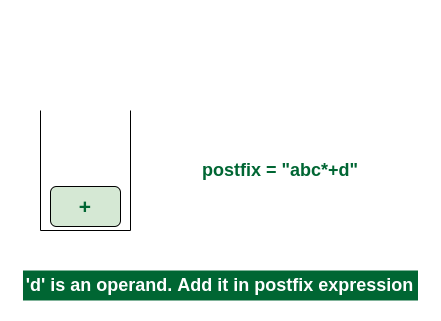
Now top element is ‘+‘ that also doesn’t have less precedence. Pop it. postfix = “abc\*+”.



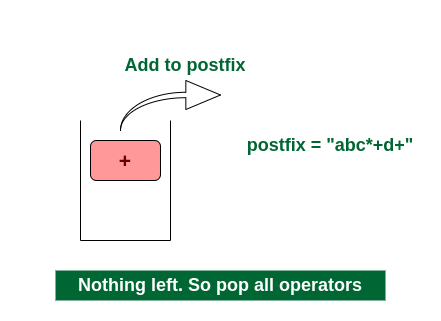
Now stack is empty. So push ‘+’ in the stack. stack = {+}.



7th Step: Now i = 6 and exp[i] = ‘d’ i.e., an operand. Add this in the postfix expression. postfix = “abc\*+d”.



Final Step: Now no element is left. So empty the stack and add it in the postfix expression. postfix = “abc\*+d+”.



// Function to return precedence of operators

int precedence(char operator)

{

    switch (operator) {

    case '+':

    case '-':

        return 1;

    case '\*':

    case '/':

        return 2;

    case '^':

        return 3;

    default:

        return -1;

    }

}

// Function to check if the scanned character is an operator

int isOperator(char ch)

{

    return (ch == '+' || ch == '-' || ch == '\*' || ch == '/'

            || ch == '^');

}

// Main function to convert infix expression to postfix expression

char\* infixToPostfix(char\* infix)

{

    int i, j;

    int len = strlen(infix);

    char\* postfix = (char\*)malloc(sizeof(char) \* (len + 2));

    char stack[MAX\_EXPR\_SIZE];

    int top = -1;

    for (i = 0, j = 0; i < len; i++) {

        if (infix[i] == ' ' || infix[i] == '\t')

            continue;

// If the scanned character is operand add it to the postfix expression

        if (isalnum(infix[i])) {

            postfix[j++] = infix[i];

        }

// if the scanned character is '(' push it in the stack

        else if (infix[i] == '(') {

            stack[++top] = infix[i];

        }

// if the scanned character is ')', pop the stack and add it to the output string until empty or '(' found

        else if (infix[i] == ')') {

            while (top > -1 && stack[top] != '(')

                postfix[j++] = stack[top--];

            if (top > -1 && stack[top] != '(')

                return "Invalid Expression";

            else

                top--;

        }

// If the scanned character is an operator push it in the stack

        else if (isOperator(infix[i])) {

            while (top > -1

                   && precedence(stack[top])

                          >= precedence(infix[i]))

                postfix[j++] = stack[top--];

            stack[++top] = infix[i];

        }

    }

    // Pop all remaining elements from the stack

    while (top > -1) {

        if (stack[top] == '(') {

            return "Invalid Expression";

        }

        postfix[j++] = stack[top--];

    }

    postfix[j] = '\0';

    return postfix;

}

int main()

{

    char infix[MAX\_EXPR\_SIZE] = "a+b\*(c^d-e)^(f+g\*h)-i";

    char\* postfix = infixToPostfix(infix);

    printf("%s\n", postfix);

    free(postfix);

    return 0;

}

**5. Write a program for the evaluation of a Postfix expression.**

If the expression is “21+3\*”, then the answer will be 9.

**Algorithm**

* for each character ch in the postfix expression, do
  + if ch is an operator ⊙ , then
    - a := pop first element from stack,
    - b := pop second element from the stack
    - res := b ⊙ a
    - push res into the stack
  + else if ch is an operand, then
    - add ch into the stack
* return element of stack top

float scanNum(char ch){

int value;

value = ch;

return float(value-'0');//return float from character

}

int isOperator(char ch){

if(ch == '+'|| ch == '-'|| ch == '\*'|| ch == '/' || ch == '^')

return 1;//character is an operator

return -1;//not an operator

}

int isOperand(char ch){

if(ch >= '0' && ch <= '9')

return 1;//character is an operand

return -1;//not an operand

}

float operation(int a, int b, char op){

if(op == '+')

return b+a;

else if(op == '-')

return b-a;

else if(op == '\*')

return b\*a;

else if(op == '/')

return b/a;

else if(op == '^')

return pow(b,a); //find b^a

else

return INT\_MIN; //return negative infinity

}

float postfixEval(string postfix){

int a, b;

stack<float> stk;

string::iterator it;

for(it=postfix.begin(); it!=postfix.end(); it++){

if(isOperator(\*it) != -1){

a = stk.top();

stk.pop();

b = stk.top();

stk.pop();

stk.push(operation(a, b, \*it));

}else if(isOperand(\*it) > 0){

stk.push(scanNum(\*it));

}

}

return stk.top();

}

main(){

string post = "21+3\*";

cout <<postfixEval(post);

}