**1.implementation of stack using array in c**

#include <stdio.h>

#include <stdlib.h>

#define MAX 4

int stack\_arr[MAX];

int top = -1;

void push (int data)

{

if (top == MAX – 1){

printf (“Stack Overflow\n”);

return;

}

top = top + 1;

stack\_arr[top] = data;

}

int pop (){

int value;

if (top == -1) {

printf (“Stack underflow\n”);

exit (1);

}

value=stack\_arr[top];

top=top-1;

return value;

}

void print (){

int i;

if (top == -1){

printf (“Stack underflow\n”);

return;

}

for (i = top; i >= 0; i--)

printf (“\t%d\n”, stack\_arr[i]);

}

int main (){

int data;

push (1);

push (2);

push (3);

push (4);

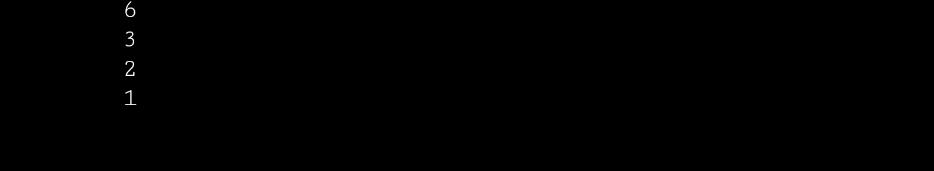
data = pop ();

push (6);

print ();

return 0;

}

**Output:**

**2.Infix to prefix conversion**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

char stack[MAX];

char infix[MAX], postfix[MAX];

int top=-1;

void push(char);

char pop();

int isEmpty();

void inToPost();

void print();

int precedence(char);

int main(){

printf(“Enter the infix expression: “);

gets(infix);

inToPost();

print();

return 0;

}

void inToPost(){

int i,j=0;

char next,symbol;

for(i=0;i<strlen(infix);i++) {

symbol=infix[i];

switch(symbol){

case’(‘:

push(symbol);

break;

case ‘)’:

while((next=pop())!=’(‘)

postfix[j++]=next;

break;

case ‘+’:

case ‘-‘:

case ‘/’:

case ‘\*’:

case ‘^’:

while(!isEmpty()&&precedence(stack[top])>=precedence(symbol))

postfix[j++]=pop();

push(symbol);

break;

default:

postfix[j++]=symbol;

}

}

while (!isEmpty())

postfix[j++]=pop();

postfix[j]=’\0’;

}

int precedence(char symbol){

Switch(symbol){

case’^’:

return 3;

case’/’:

case’\*’:

return 2;

case’+’:

case’-‘:

return 1;

default:

return 0;

}

}

void print (){

int i=0;

printf(“The equivalent postfix expression is:”);

while (postfix [i]){

printf(“%c”,postfix[i++]);

}

printf(“\n”);

}

void push(char c){

if(top==MAX-1){

printf(“stack overflow\n”);

return;

}

top++;

stack[top]=c;

}

char pop(){

char c;

if(top==-1){

printf(“Stack underflow\n”);

exit(1);

}

c=stack[top];

top=top-1;

return c;

}

int isEmpty(){

if (top==-1)

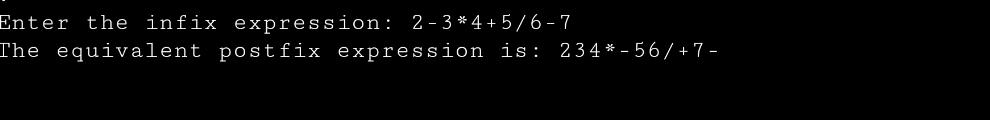
return 1;

else

return 0;

}

**Output:**



**3.Evaluation of postfix expression**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include<string.h>

#define MAX 50

int top = -1;

int stack[MAX];

int is\_operator(char c) {

if (c == ‘+’ || c == ‘-‘ || c == ‘\*’ || c == ‘/’)

return 1;

else

return 0;

}

int apply\_operator(int a, int b, char op) {

switch (op) {

case ‘+’: return a + b;

case ‘-‘: return a – b;

case ‘\*’: return a \* b;

case ‘/’: return a / b;

}

return 0;

}

int evaluate\_postfix(char \*expression) {

int I, a, b;

for (i = 0; expression[i]; i++) {

if (isdigit(expression[i])) {

stack[++top] = expression[i] – ‘0’;

} else if (is\_operator(expression[i])) {

b = stack[top--];

a = stack[top--];

Stack[++top] = apply\_operator(a, b, expression[i]);

}

}

return stack[0];

}

int main() {

char expression[MAX];

printf(“Enter postfix expression: “);

fgets(expression, MAX, stdin);

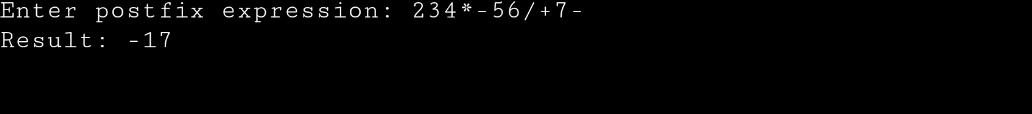
expression[strlen(expression) – 1] = ‘\0’;

printf(“Result: %d\n”, evaluate\_postfix(expression));

return 0;

}

**Output:**



**4.Tower of honai**

#include <stdio.h>

void move(int n, int source, int destination, int auxiliary) {

while (n > 0) {

if (n % 2 == 0) {

printf(“Move disk %d from source %d to destination %d\n”, n, source, destination);

n--;

}

else {

printf(“Move disk %d from source %d to destination %d\n”, n, source, auxiliary);

printf(“Move disk %d from source %d to destination %d\n”, n – 1, source, destination);

printf(“Move disk %d from source %d to destination %d\n”, n, auxiliary, source);

n -= 2;

}

}

}

int main() {

int n;

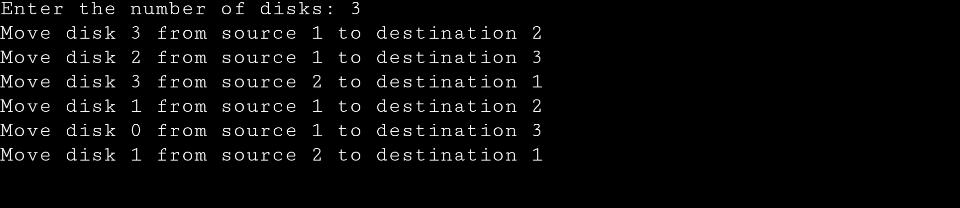
printf(“Enter the number of disks: “);

scanf(“%d”, &n);

move(n, 1, 3, 2);

return 0;

}

**Output:**

**The Time Complexity of the iterative solution is O(n), which is more suitable for large inputs.**

**The space complexity of the iterative solution is O(1).**