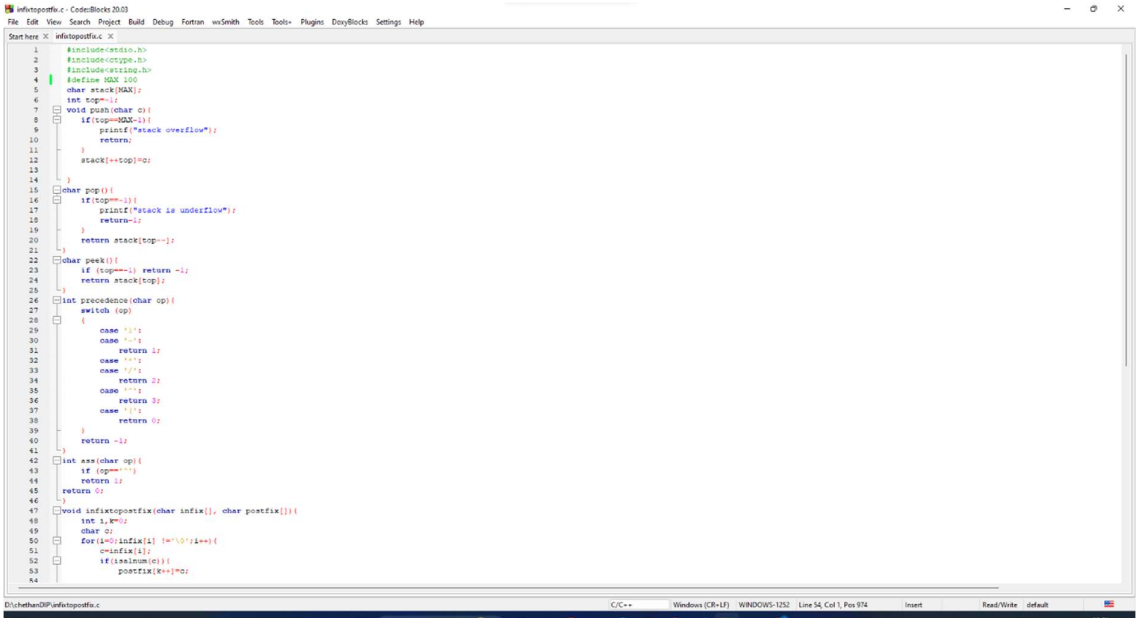
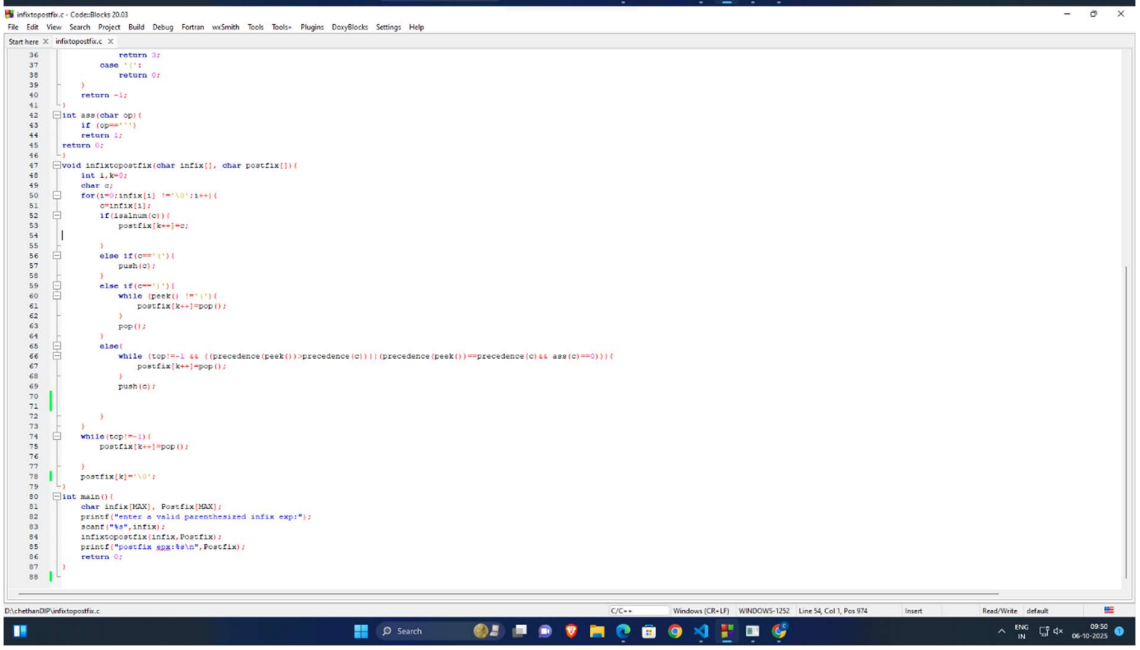
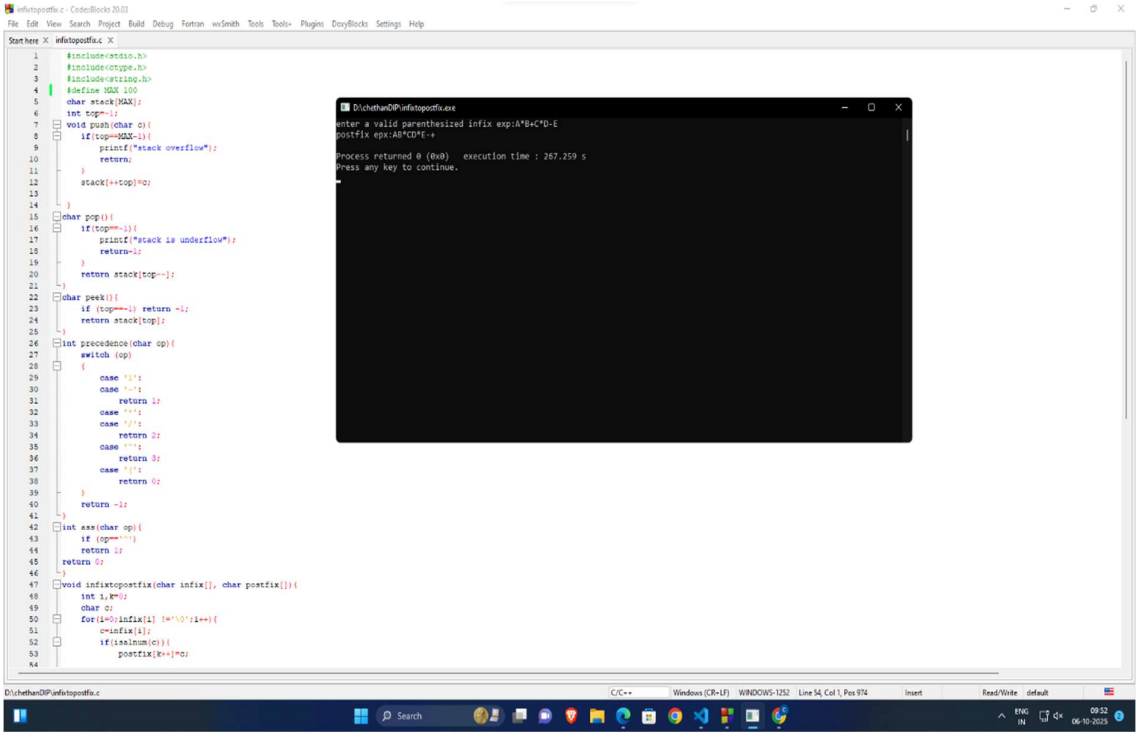

infixtopostfix.c - CodeBlocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Start here
infixtopostfix.c
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#define MAX 100
char stack[MAX];
int top=-1;
void push(char c){
if(top==MAX-1){
printf("stack overflow");
return;
}
stack[++top]=c;
}
char pop(){
if(top==-1){
printf("stack is underflow");
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;
case '*':
case '/':
return 2;
case '^':
return 3;
case '(':
return 0;
}
return -1;
}
int ass(char op){
if (op=='^')
return 1;
return 0;
}
void infixtopostfix(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;
D:\chethanDIP\infixtopostfix.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 54, Col 1, Pos 974
Insert
Read/Write
default
Search
ENG
IN
08:51
06-10-2025


infixtopostfix.c - CodeBlocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Start here
infixtopostfix.c
return 3;
case '(':
return 0;
}
return -1;
}
int ass(char op){
if (op=='^')
return 1;
return 0;
}
void infixtopostfix(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)&& ass(c)==0))){
postfix[k++]=pop();
}
push(c);
}
}
while(top!=-1){
postfix[k++]=pop();
}
postfix[k]='\0';
}
int main(){
char infix[MAX], Postfix[MAX];
printf("enter a valid parenthesized infix exp:");
scanf("%s",infix);
infixtopostfix(infix,Postfix);
printf("postfix exp:%s\n",Postfix);
return 0;
}
D:\chethanDIP\infixtopostfix.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 54, Col 1, Pos 974
Insert
Read/Write
default
Search
ENG
IN
09:50
06-10-2025


infixtopostfix.c - CodeBlocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Start here
infixtopostfix.c
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#define MAX 100
char stack[MAX];
int top=-1;
void push(char c){
if(top==MAX-1){
printf("stack overflow");
return;
}
stack[++top]=c;
}
char pop(){
if(top==-1){
printf("stack is underflow");
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;
case '*':
case '/':
return 2;
case '^':
return 3;
case '(':
return 0;
}
return -1;
}
int ass(char op){
if (op=='^')
return 1;
return 0;
}
void infixtopostfix(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;
D:\chethanDIP\infixtopostfix.exe
enter a valid parenthesized infix exp:A*B+C*D-E
postfix epx:AB*CD*E-+
Process returned 0 (0x0) execution time : 267.259 s
Press any key to continue.
D:\chethanDIP\infixtopostfix.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 54, Col 1, Pos 974
Insert
Read/Write
default
Search
ENG
IN
09:52
06-10-2025

DATE: PAGE:

Infix to postfix conversion

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
char stack[MAX];
int top = -1;

void push(char c) {
    if (top == MAX-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];
}
```

DATE: PAGE:

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators.

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

Algorithm :-

1) Start
2) Initialize an empty stack for operators
3) Initialize an empty string postfix
4) Scan the infix expression from left to right, one symbol at a time
5) If operand is '(' push to stack.
6) If operand is ')' pop from stack and append the postfix until '(' open bracket
7) If operator (+, -, *, /)
   - If operator have high precedency push to stack
   - If operator have lower precedency append it to postfix. pop and print the top
   - If incoming operator is equal precedency, then go to associativity laws.
8) At the end of the expression, pop and print all operators.
9) Stop

```
        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], postfix[MAX];
    printf("Enter a valid parenthesized infix expression:\n");
    scanf("%s", infix);
    infixtopostfix(infix, postfix);
    printf("postfix expression: %s\n", postfix);
    return 0;
}
```

```
int precedence(char op){
    switch(op){
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        case '(':
            return 0;
    }
    return -1;
}
int associativity(char op){
    if(op=='^')
        return 1;
    return 0;
}
void infixtopostfix(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);
        }
```

DATE: PAGE:

Output:

Enter a valid parenthesized infix exp: A*B-(C+D*E

postfix expr: AB*C-DE*+

Queue:

1) Define an array queue[MAX] to hold elements.
2) set two integer variables
   Front = -1
   rear = -1 to indicate the queue is empty
3) check if rear == MAX-1
   print queue overflow
4) If front == -1, set front=0
5) Increment rear by 1 -> rear = rear+1
6) Insert the new element -> queue[rear] = item
7) stop

Dequeue

1) start
2) check if front == -1 or front > rear
   print queue underflow
3) Retrive the element -> item = queue[front]
4) Front = front+1 Increment
5) stop