A summary of the rules follows:

1. Print operands as they arrive.

2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.

3. If the incoming symbol is a left parenthesis, push it on the stack.

4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.

5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.

6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.

7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.

8. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)

int priority(char x){

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

}

main(){

char exp[20], x, ch; int n;

printf("Enter the expression :: ");

scanf("%s",exp);

n=strlen(exp);

for(int i=0;i<n;i++){

ch=exp[i];

if(isalnum(ch))

printf("%c",ch);

else if(ch == '(')

push(ch);

else if(ch == ')'){

while((x = pop()) != '(')

printf("%c", x);

}

else{

while(priority(stack[top]) >= priority(ch))

printf("%c",pop());

push(ch);

}

}

while(top != -1){

printf("%c",pop());

}

}