PCET’s

Pimpri Chinchwad College of Engineering

Department of Computer Engineering

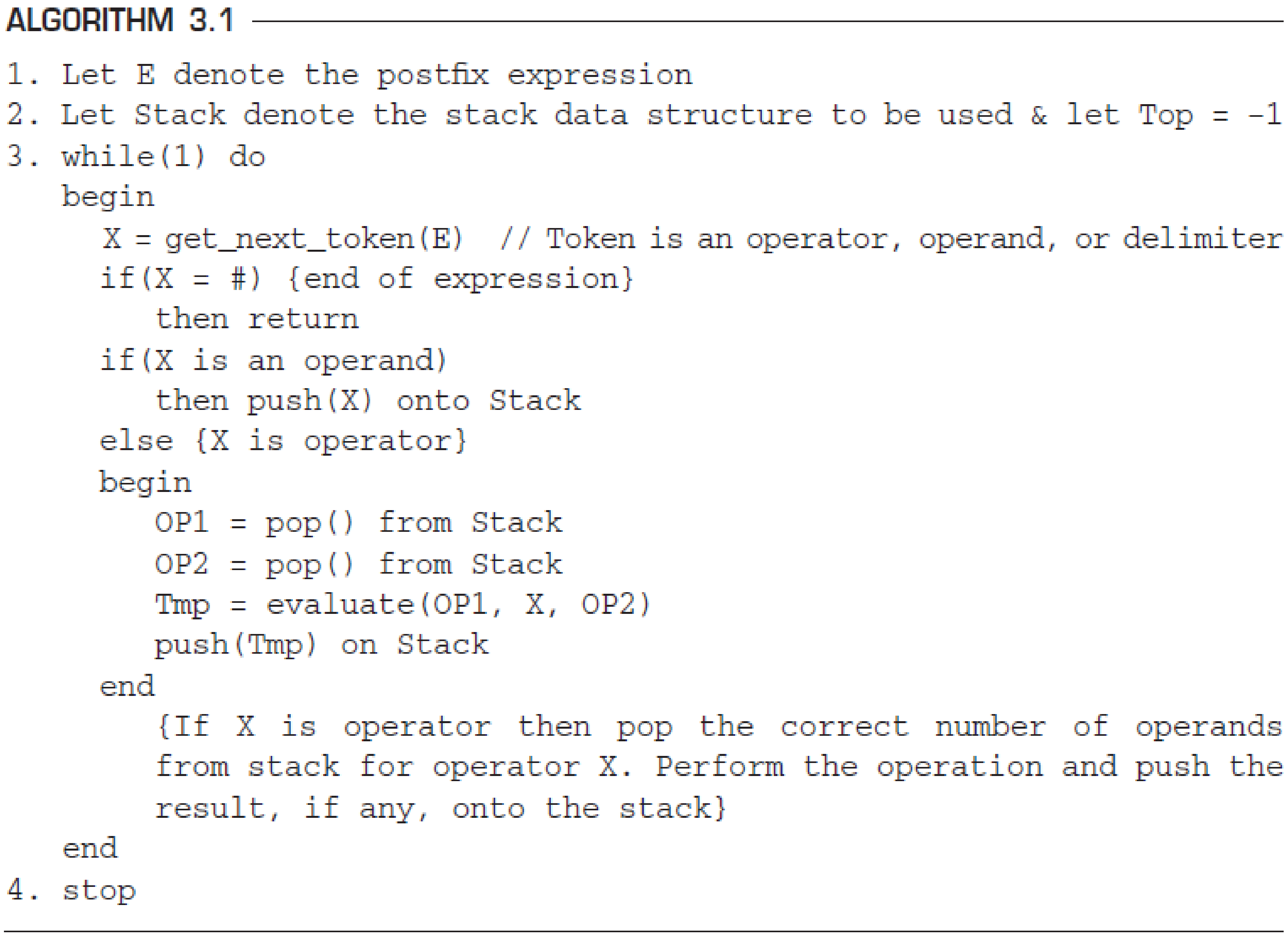
B. Tech. (Computer Engineering)

Course: Data Structures and Algorithms (BCE3401)

Unit III: Stacks and Queues

Prepared By: Prof. Meghana P. Lokhande

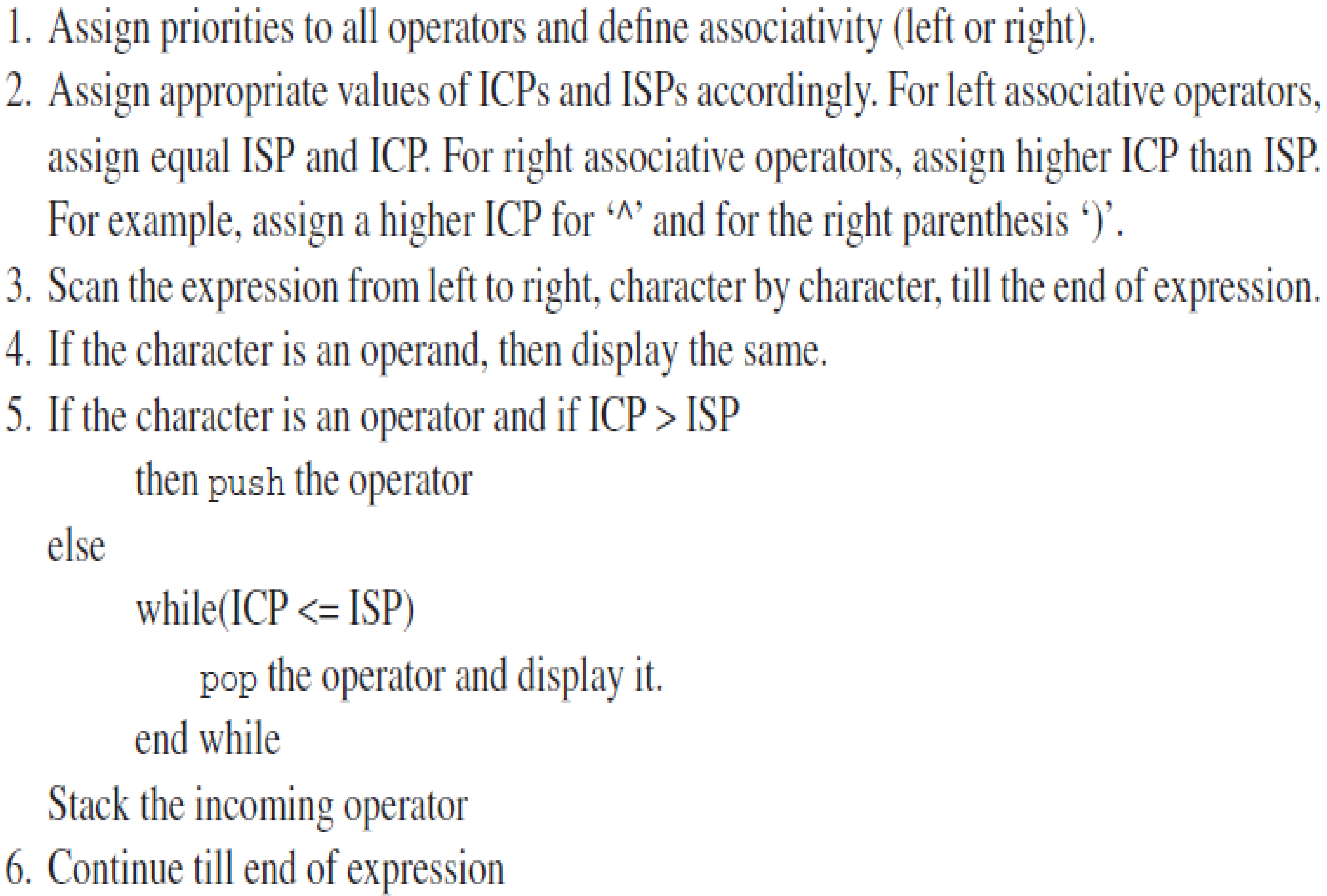
Algorithm 3.1 lists the steps involved in the evaluation of the postfix expression



ALGORITHM 3.1

```
1. Let E denote the postfix expression
2. Let Stack denote the stack data structure to be used & let Top = -1
3. while(1) do
   begin
      X = get_next_token(E)  // Token is an operator, operand, or delimiter
      if(X = #) {end of expression}
         then return
      if(X is an operand)
         then push(X) onto Stack
      else {X is operator}
      begin
         OP1 = pop() from Stack
         OP2 = pop() from Stack
         Tmp = evaluate(OP1, X, OP2)
         push(Tmp) on Stack
      end
         {If X is operator then pop the correct number of operands
         from stack for operator X. Perform the operation and push the
         result, if any, onto the stack}
   end
4. stop
```

Steps involved in the evaluation of an expression

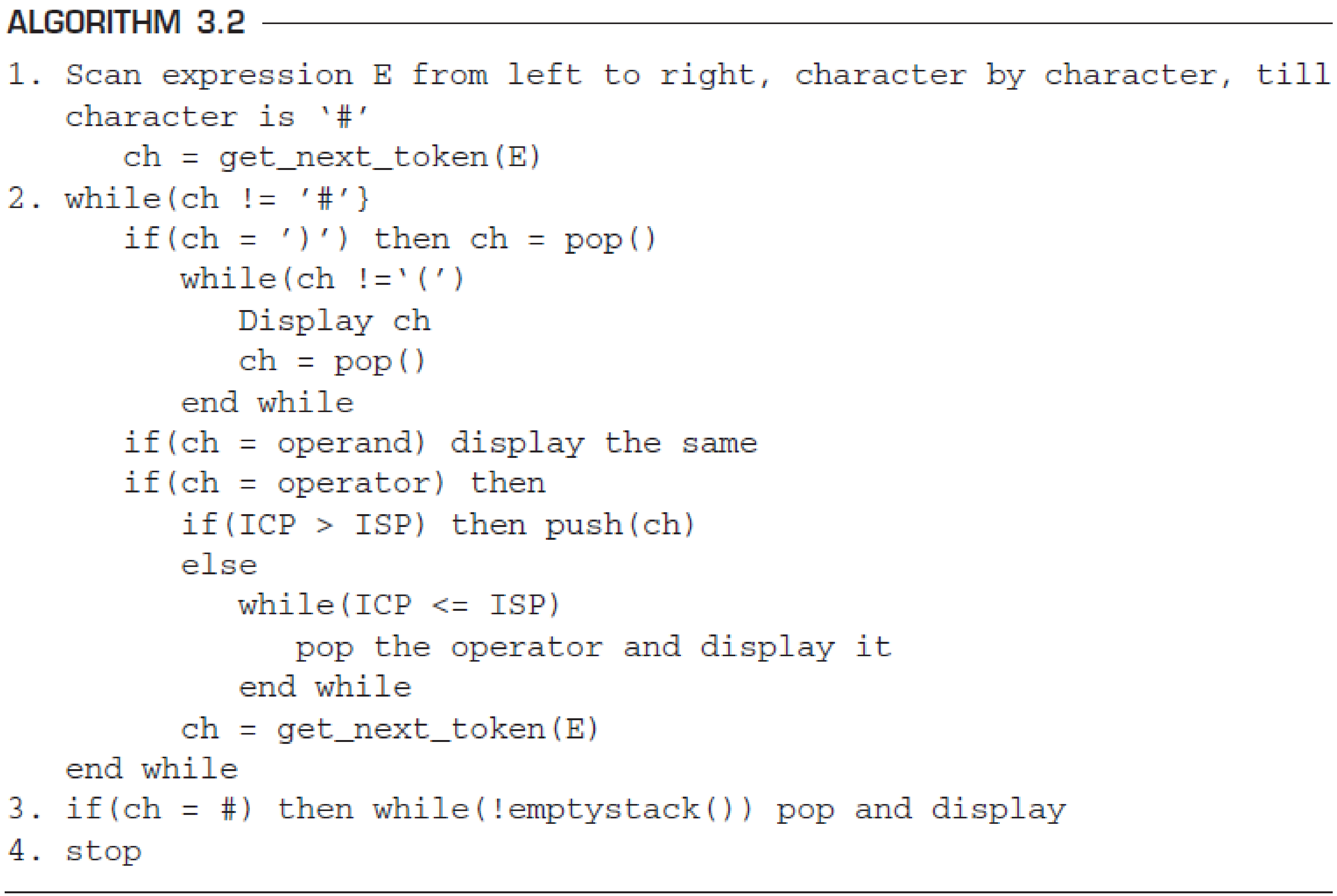


1. Assign priorities to all operators and define associativity (left or right).
2. Assign appropriate values of ICPs and ISPs accordingly. For left associative operators, assign equal ISP and ICP. For right associative operators, assign higher ICP than ISP. For example, assign a higher ICP for '^' and for the right parenthesis ')'.
3. Scan the expression from left to right, character by character, till the end of expression.
4. If the character is an operand, then display the same.
5. If the character is an operator and if ICP > ISP

```
        then push the operator
    else
        while(ICP <= ISP)
            pop the operator and display it.
        end while
    Stack the incoming operator
```

6. Continue till end of expression

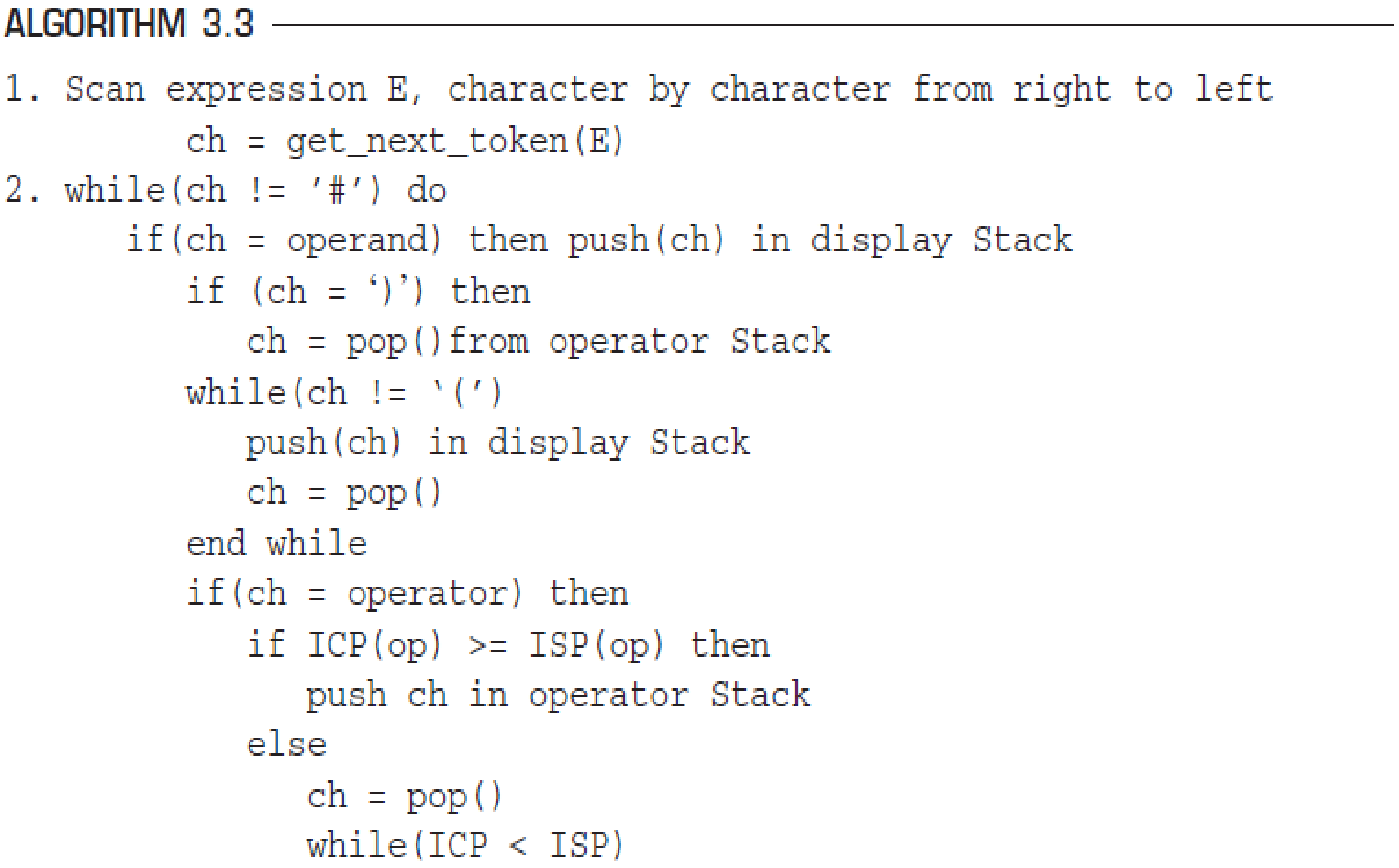
Algorithm 3.2 illustrates the infix to postfix conversion



ALGORITHM 3.2

```
1. Scan expression E from left to right, character by character, till
   character is ‘#’
      ch = get_next_token(E)
2. while(ch != ’#’}
      if(ch = ’)’) then ch = pop()
         while(ch !=‘(’)
            Display ch
            ch = pop()
         end while
      if(ch = operand) display the same
      if(ch = operator) then
         if(ICP > ISP) then push(ch)
         else
            while(ICP <= ISP)
               pop the operator and display it
            end while
         ch = get_next_token(E)
   end while
3. if(ch = #) then while(!emptystack()) pop and display
4. stop
```

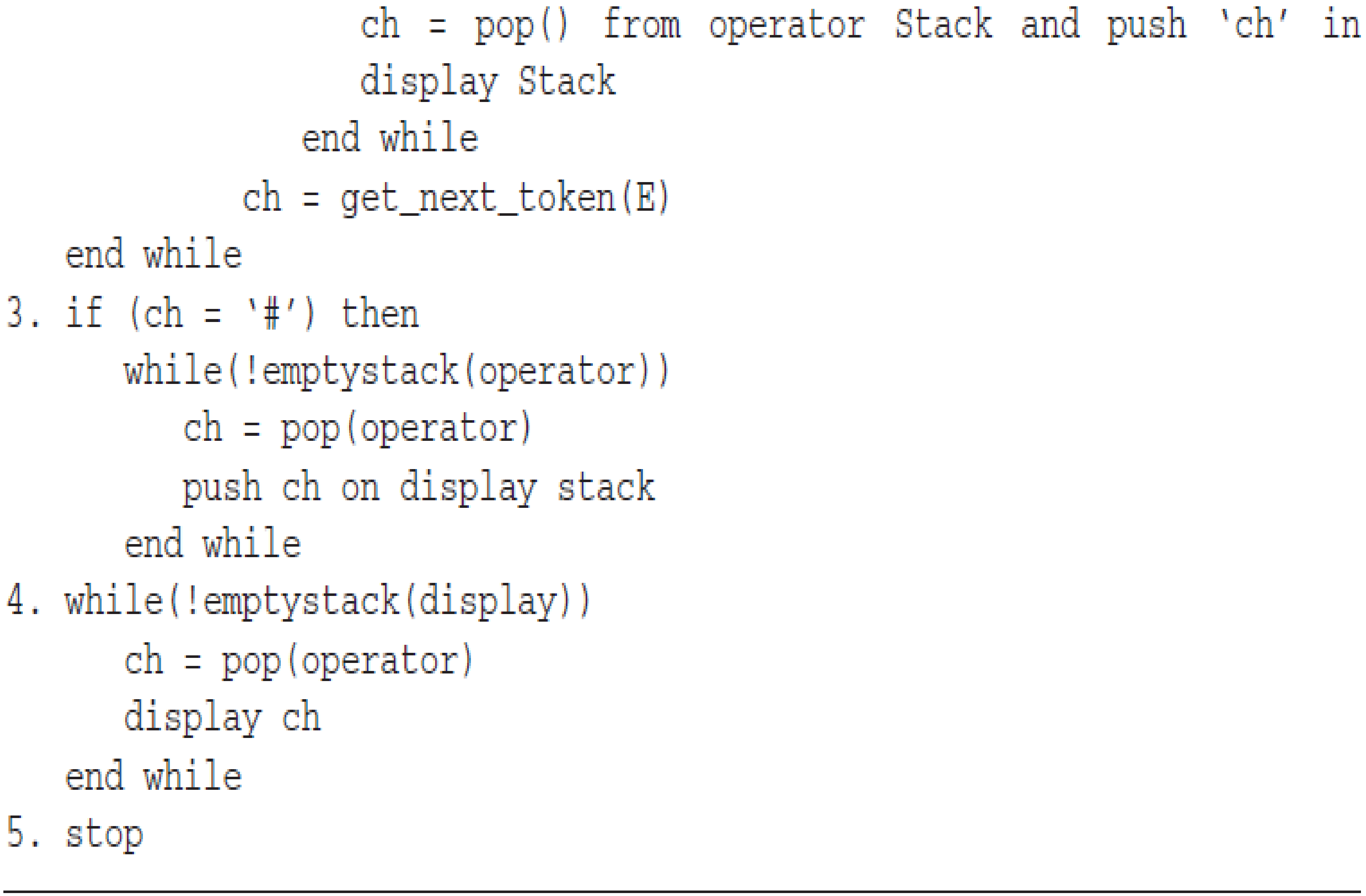
Algorithm 3.3 illustrates Infix to Prefix Conversion



ALGORITHM 3.3

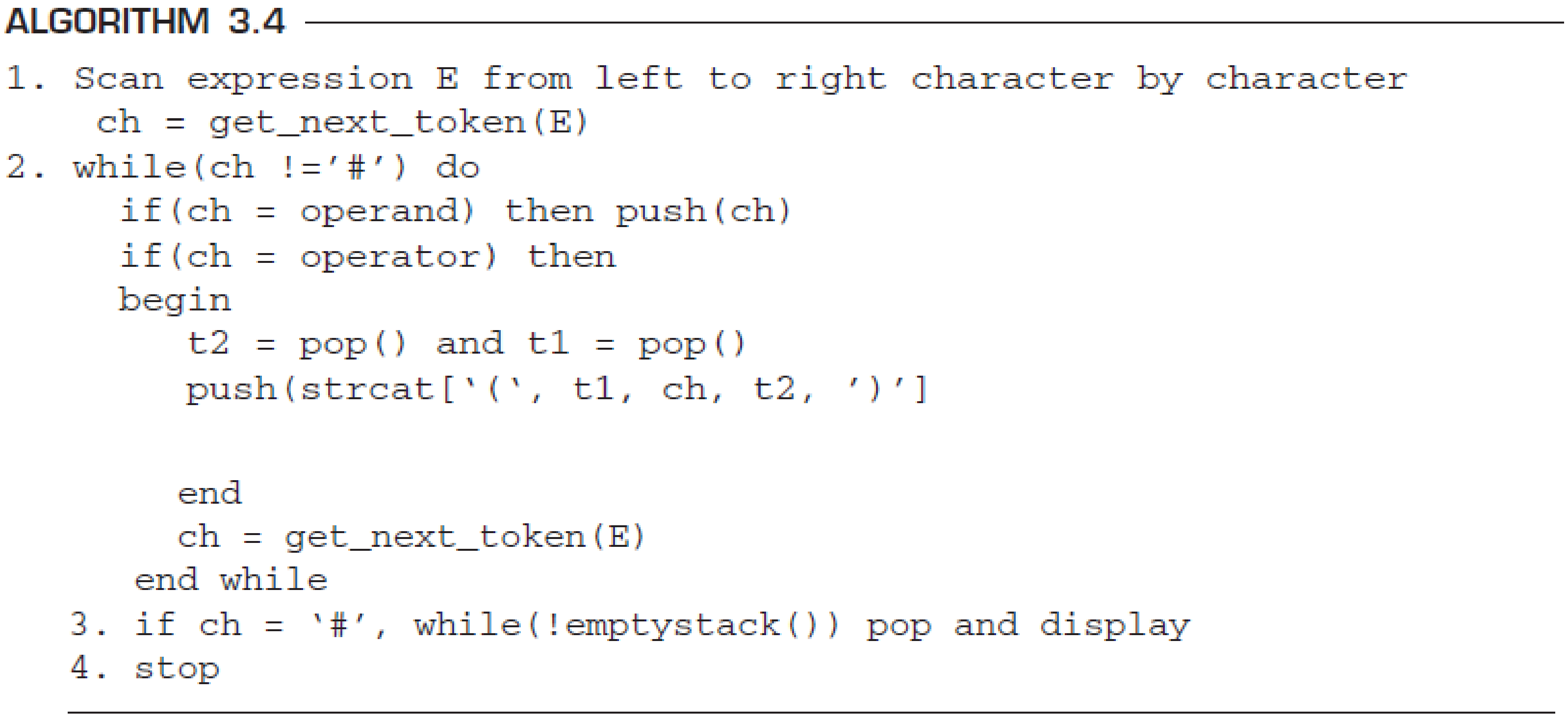
```
1. Scan expression E, character by character from right to left
         ch = get_next_token(E)
2. while(ch != ’#’) do
      if(ch = operand) then push(ch) in display Stack
         if (ch = ‘)’) then
            ch = pop()from operator Stack
         while(ch != ‘(’)
            push(ch) in display Stack
            ch = pop()
         end while
         if(ch = operator) then
            if ICP(op) >= ISP(op) then
               push ch in operator Stack
            else
               ch = pop()
               while(ICP < ISP)
```

Algorithm 3.2 illustrates Infix to Prefix Conversion



```
                  ch = pop() from operator Stack and push 'ch' in
                  display Stack
               end while
            ch = get_next_token(E)
   end while
3. if (ch = '#') then
      while(!emptystack(operator))
         ch = pop(operator)
         push ch on display stack
      end while
4. while(!emptystack(display))
      ch = pop(operator)
      display ch
   end while
5. stop
```

Algorithm 3.4 illustrates Postfix to Infix Conversion

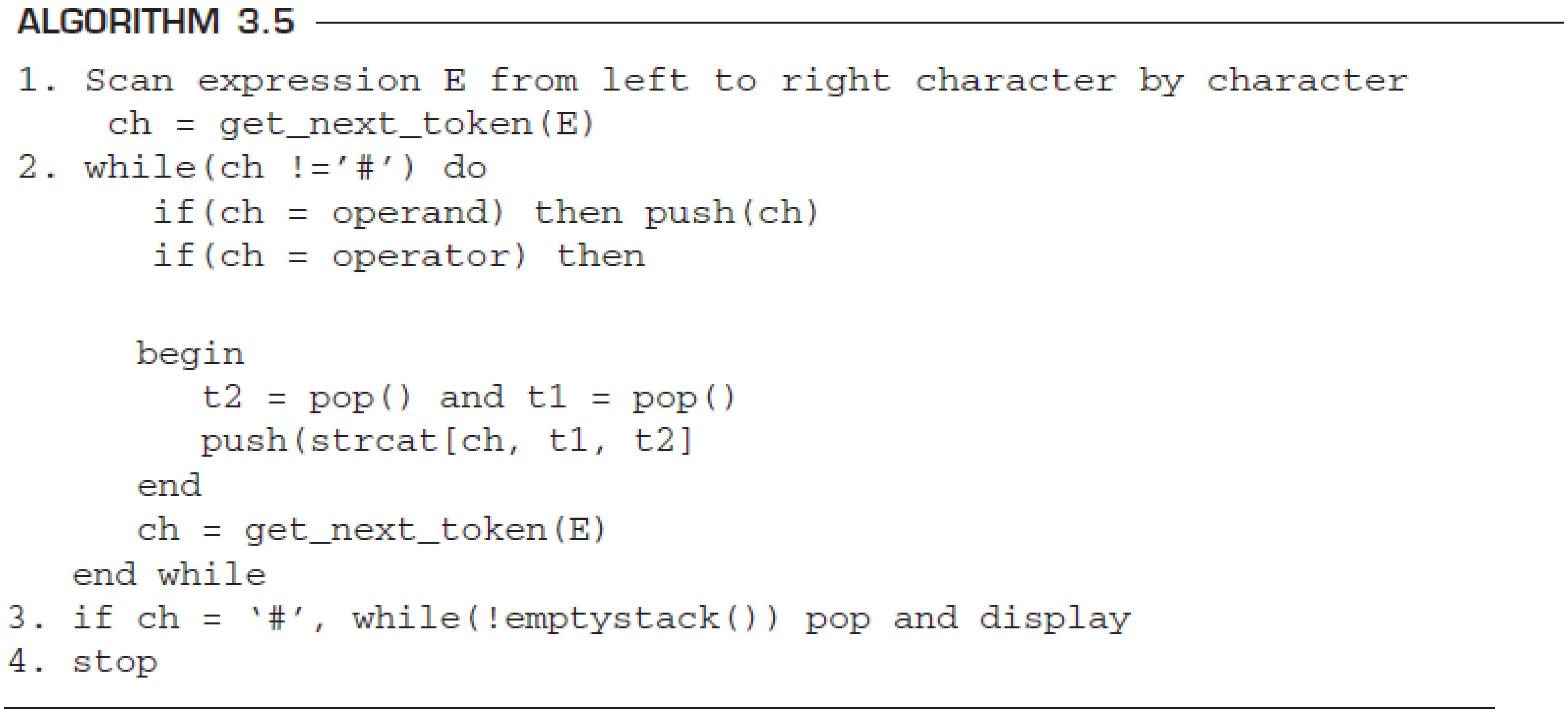


ALGORITHM 3.4

```
1. Scan expression E from left to right character by character
    ch = get_next_token(E)
2. while(ch !=’#’) do
     if(ch = operand) then push(ch)
     if(ch = operator) then
     begin
        t2 = pop() and t1 = pop()
        push(strcat[‘(‘, t1, ch, t2, ’)’]

        end
        ch = get_next_token(E)
      end while
   3. if ch = ‘#’, while(!emptystack()) pop and display
   4. stop
```

Algorithm 3.5 illustrates Postfix to Prefix Conversion

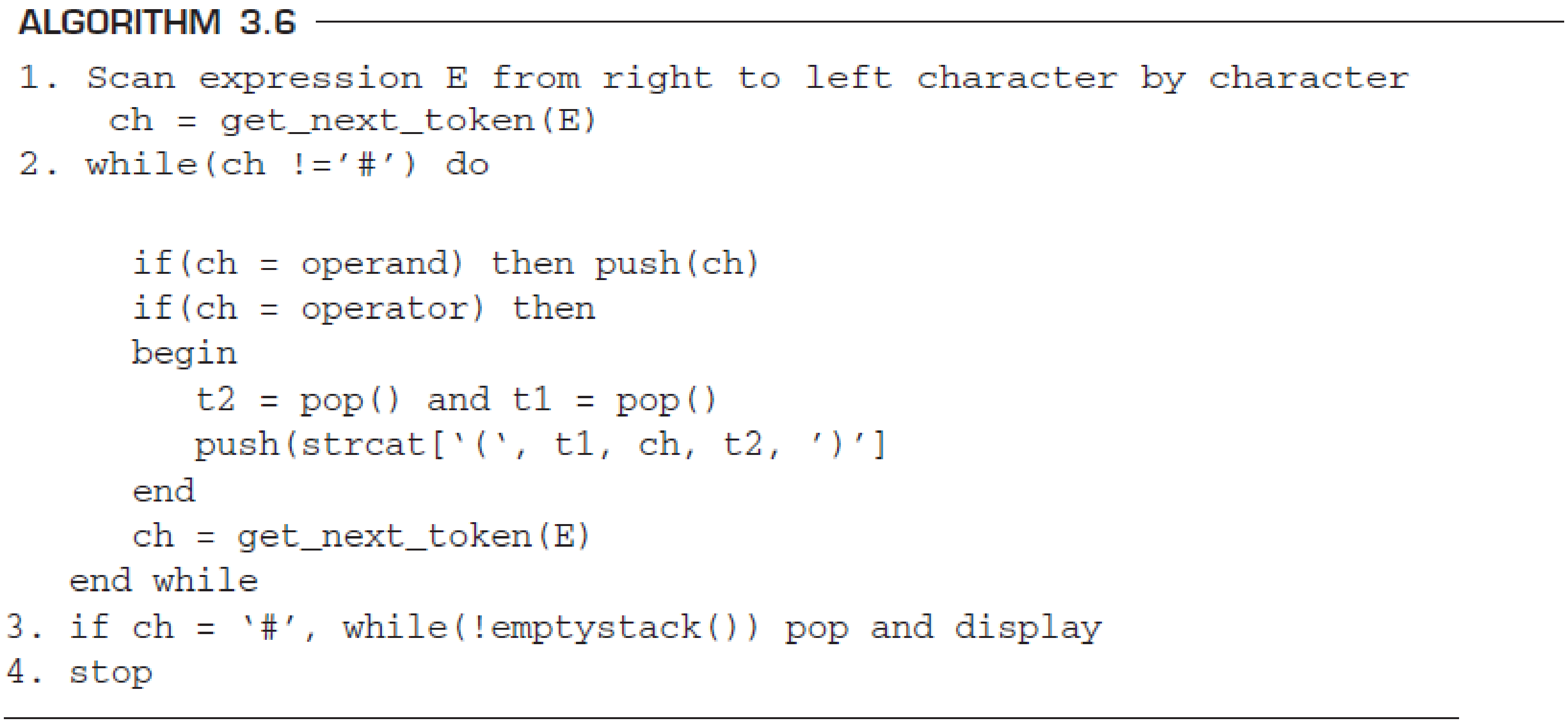


ALGORITHM 3.5

```
1. Scan expression E from left to right character by character
     ch = get_next_token(E)
2. while(ch !=’#’) do
       if(ch = operand) then push(ch)
       if(ch = operator) then

      begin
         t2 = pop() and t1 = pop()
         push(strcat[ch, t1, t2]
      end
      ch = get_next_token(E)
   end while
3. if ch = ‘#’, while(!emptystack()) pop and display
4. stop
```

Algorithm 3.6 illustrates Prefix to Infix Conversion

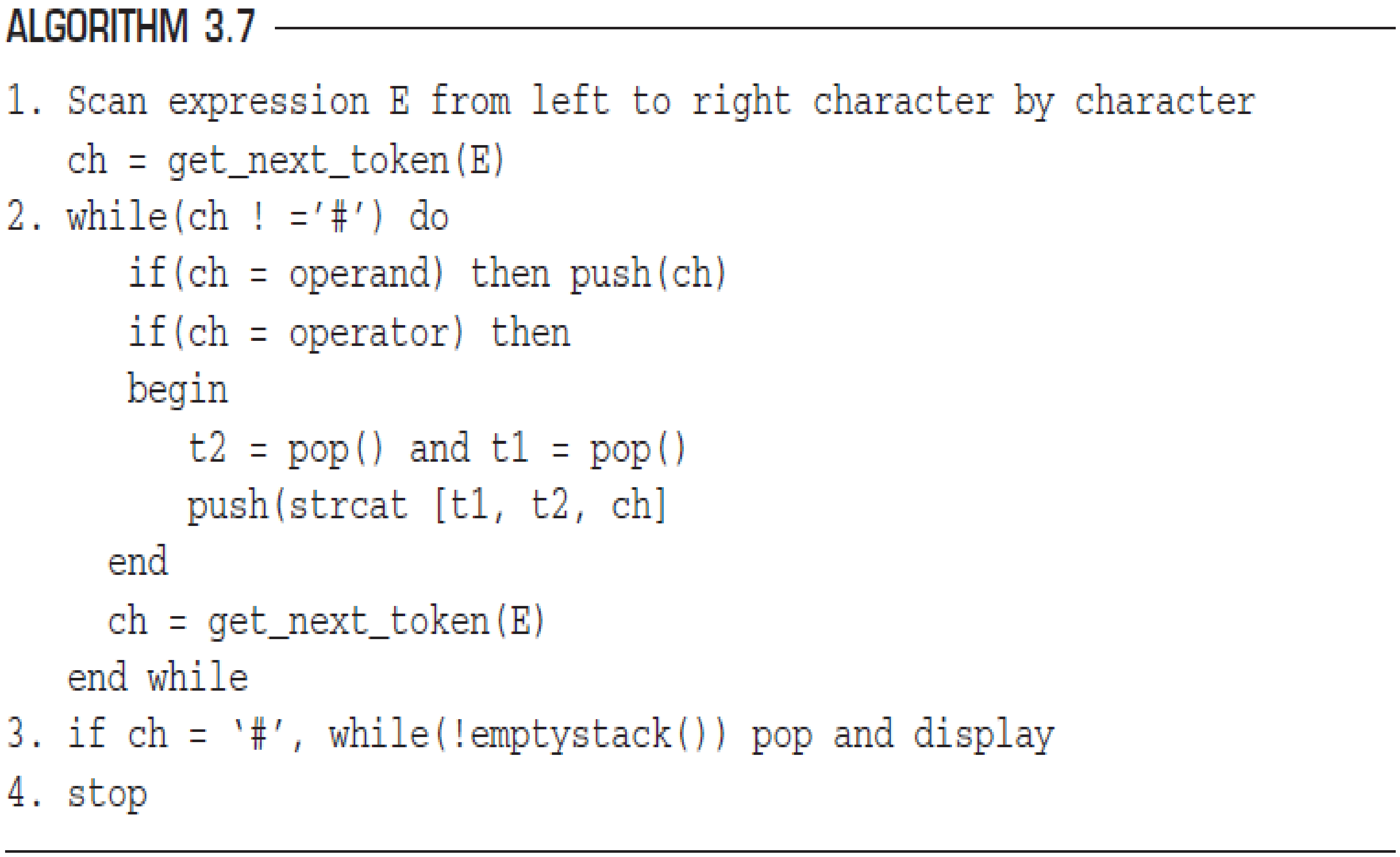


ALGORITHM 3.6

```
 1. Scan expression E from right to left character by character
      ch = get_next_token(E)
 2. while(ch !=’#’) do

       if(ch = operand) then push(ch)
       if(ch = operator) then
       begin
          t2 = pop() and t1 = pop()
          push(strcat[‘(‘, t1, ch, t2, ’)’]
       end
       ch = get_next_token(E)
   end while
3. if ch = ‘#’, while(!emptystack()) pop and display
4. stop
```

Algorithm 3.7 illustrates Prefix to Postfix Conversion



ALGORITHM 3.7

```
1. Scan expression E from left to right character by character
   ch = get_next_token(E)
2. while(ch ! ='#') do
      if(ch = operand) then push(ch)
      if(ch = operator) then
      begin
         t2 = pop() and t1 = pop()
         push(strcat [t1, t2, ch]
     end
     ch = get_next_token(E)
   end while
3. if ch = '#', while(!emptystack()) pop and display
4. stop
```