SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-IoT Including CS & BCT

COURSE NAME :23ITB201-DATA STRUCTURES & ALGORITHMS

II YEAR / III SEMESTER

Unit II- **STACK ADT & QUEUE ADT**

Topic :Postfix Evaluation

- ✓ Postfix evaluation is an important concept in computer science that allows us to perform arithmetic operations on postfix expressions.
- ✓ Postfix notation is also known as reverse polish notation. It is a mathematical notation in which operators come after their operands.
- ✓ For example, the infix expression 3 + 4 can be written in postfix notation as 3 4 +.

- ✓ Similarly, the infix expression (2 + 3) * 4 can be written in postfix notation as 2 3 + 4 *.
- ✓ Postfix notation has several advantages over infix notation.
- ✓ It eliminates the need for parentheses and makes parsing and evaluation of expressions easier.

Postfix Evaluation Algorithm

- ✓ Postfix evaluation algorithm is a simple algorithm that allows us to evaluate postfix expressions.
- ✓ The algorithm uses a stack to keep track of operands and performs arithmetic operations when an operator is encountered.
- ✓ The algorithm can be summarized in the following steps:

1.First of all, it will ***Create*** an ***empty stack***.

2.After that, it ***Scan*** the expression from ***left to right***.

3.If an operand is encountered, it ***push*** it onto the stack.

4.If an operator is encountered, ***pop*** the top two operands from the stack, perform the operation, and ***push*** the result back onto the stack.

5.After that, it ***Continue scanning*** the expression until all tokens have been processed.

6.When the expression has been fully scanned, the result will be the top element of the stack.

Example:

Let's consider the expression "5 6 7 + * 8 -".

We will evaluate this expression using the postfix evaluation algorithm.

1.Start scanning the expression from ***left to right***.

2.Push operand ***5*** onto the stack.

3.Push operand ***6*** onto the stack.

4.Push operand ***7*** onto the stack.

5.Pop operands ***7*** and ***6*** from the stack, perform addition, and push the result ***(13)*** back onto the stack.

6.Pop operands 13 and ***5*** from the stack, perform multiplication, and ***push the result (65)*** back onto the stack.

7.Push ***operand 8*** onto the stack.

8. Pop operands 8 and ***65*** from the stack, perform subtraction, and push the result ***(57)*** back onto the stack.

9. The final result is ***57***.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
// Stack implementation
int stack[MAX_SIZE];
int top = -1;
void push(int item) {
   if (top >= MAX_SIZE - 1) {
printf("Stack Overflow\n");
      return;
   }
   top++;
   stack[top] = item;
}
```

```
int pop() {
   if (top < 0) {
printf("Stack Underflow\n");
      return -1;
   }
   int item = stack[top];
   top--;
   return item;
}
int is_operator(char symbol) {
   if (symbol == '+' || symbol == '-' || symbol ==
'*' || symbol == '/') {
      return 1;
   }
   return 0;
}
```

```
int evaluate(char* expression) {
   int i = 0;
   char symbol = expression[i];
   int operand1, operand2, result;

   while (symbol != '\0') {
      if (symbol >= '0' && symbol
<= '9') {
         int num = symbol - '0';
         push(num);
      }
```

```
else if (is_operator(symbol)) {
         operand2 = pop();
         operand1 = pop();
         switch(symbol) {
            case '+': result = operand1 + operand2;
break;
            case '-': result = operand1 - operand2;
break;
            case '*': result = operand1 * operand2;
break;
            case '/': result = operand1 / operand2;
break;
         }
         push(result);
      }
i++;
```

```
 symbol = expression[i];
    }
    result = pop();
    return result;
}

int main() {
    char expression[] = "5 6 7 + * 8 -";
    int result = evaluate(expression);
printf("Result= %d\n", result);
return 0;
}
```

MCQ

1. What is the main advantage of using postfix notation over infix notation?

- A) It is more human-readable.
- B) It eliminates the need for parentheses in expressions.
- C) It requires fewer operations.
- D) It only works for integer expressions.

Answer: B) It eliminates the need for parentheses in expressions.

2. Which data structure is commonly used for evaluating a postfix expression?

- A) Queue
- B) Stack
- C) Linked List
- D) Tree

Answer: B) Stack

3. Evaluate the postfix expression 5 6 2 + * 3 –

A) 25

B) 27

C) 32

D) 33

Answer: A) 25

Explanation: First, evaluate 6 2 + → 8.
Then, evaluate 5 * 8 → 40.
Finally, evaluate 40 - 3 → 25.

4. Which of the following is a postfix representation of the infix expression (A + B) * (C - D)?

A) A B + C D - *

B) A + B * C – D

C) A B * C D - +

D) A B + * C D –

Answer: A) A B + C D - *

5. What will be the result of the postfix expression 7 4 5 * + 9 -?

A) 16

B) 22

C) 11

D) 12

Answer: D) 12

Any Query????

Thank you……