Sunbeam Institute of Information Technology
Pune and Karad

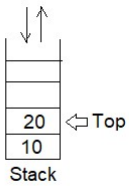
Module – Data Structures and Algorithms

Email – devendra.dhande@sunbeaminfo.com

Stack

Stack

- Stack is Last-In-First-Out structure.
  - Stack Operations:
    - push()
    - pop()
    - peek()
    - is_empty()
    - is_full()



Stack

- Parenthesis balancing
- Expression conversion and evaluation
- Function calls
- Used in advanced data structures for traversing

- **Expression conversion and evaluation:**
  - Infix to postfix
  - Infix to prefix
  - Postfix evaluation
  - Prefix evaluation
  - Prefix to postfix
  - Postfix to infix

Infix to Postfix Conversion

- Process each element of infix expression from left to right
- If element is Operand
  - Append it to the postfix expression
- If element is Operator
  - If priority of topmost element (Operator) of stack is greater or equal to current element (Operator), pop topmost element from stack and append it to postfix expression
  - Repeat above step if required
  - Push element on stack
- Pop all remaining elements (Operators) from stack one by one and append them into the postfix expression
- e.g. a * b / c * d + e – f * h + i

Infix to Prefix Conversion

- Process each element of infix expression from right to left
- If element is Operand
  - Append it to the prefix expression
- If element is Operator
  - If priority of topmost element of stack is greater than current element (Operator), pop topmost element from stack and append it to prefix expression
  - Repeat above step if required
  - Push element on stack
- Pop all remaining elements (Operators) from stack one by one and append them into the prefix expression
- Reverse prefix expression
- e.g. a * b / c * d + e – f * h + i

Postfix Evaluation

- Process each element of postfix expression from left to right
- If element is operand
  - Push it on a stack
- If element is operator
  - Pop two elements (Operands) from stack, in such a way that
    - Op2 – first popped element
    - Op1 – second popped element
  - Perform current element (Operator) operation between Op1 and Op2
  - Again push back result onto the stack
- When single value will remain on stack, it is final result
- e.g. 4 5 6 * 3 / + 9 + 7 -

Prefix Evaluation

- Process each element of prefix expression from right to left
- If element is operand
  - Push it on a stack
- If element is operator
  - Pop two elements (Operands) from stack, in such a way that
    - Op1 – first popped element
    - Op2 – second popped element
  - Perform current element (Operator) operation between Op1 and Op2
  - Again push back result onto the stack
- When single value will remain on stack, it is final result
- e.g. - + + 4 / * 5 6 3 9 7

Prefix to Postfix

- Process each element of prefix expression from right to left
- If element is an Operand
  - Push it on to the stack
- If element is an Operator
  - Pop two elements (Operands) from stack, in such a way that
    - Op1 – first popped element
    - Op2 – second popped element
  - Form a string by concatenating Op1, Op2 and Opr (element)
  - String = “Op1+Op2+Opr”, push back on to the stack
- Repeat above two steps until end of prefix expression.
- Last remaining on the stack is postfix expression
- e.g. * + a b – c d

Postfix to infix

- Process each element of postfix expression from left to right
- If element is an Operand
  - Push it on to the stack
- If element is an Operator
  - Pop two elements (Operands) from stack, in such a way that
    - Op2 – first popped element
    - Op1 – second popped element
  - Form a string by concatenating Op1, Opr (element) and Op2
  - String = “Op1+Opr+Op2”, push back on to the stack
- Repeat above two steps until end of postfix expression.
- Last remaining on the stack is infix expression
- E.g. a b c - + d e – f g – h + / *

Thank you!

Devendra Dhande
<devendra.dhande@sunbeaminfo.com>