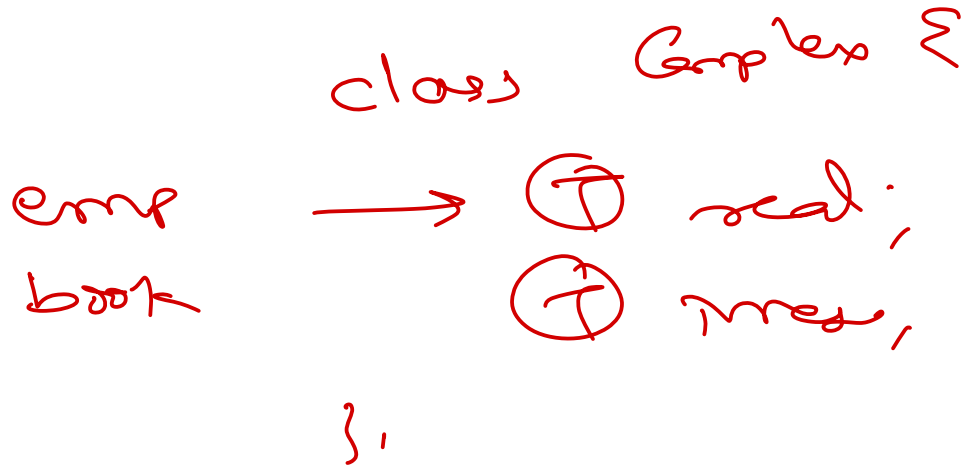
Data Structure & Algorithms

Sunbeam Infotech

Agenda

- STL: Stack, Queue
- STL: String
- Infix to Postfix
- Infix to Prefix
- Postfix Evaluation
- Prefix Evaluation
- Postfix to Infix
- Prefix to Postfix
- Parenthesis Balancing
- Stack using Queue
- Linked list introduction



STL – Stack & Queue

- STL is part of C++ standard.
- It is efficient template implementation of data structures.
- Stack & Queue are container adapters.

- http://www.cplusplus.com/reference/

manual / help

- #include <stack>
- stack<int> obj;
  - obj.push(ele)
  - obj.pop()
  - ele = obj.top() → like peek()
  - obj.empty() → return true/false.
  - obj.size() → num of eles.

LIFO → O(1)

- #include <queue>
- queue<int> obj;
  - obj.push(ele)
  - obj.pop()
  - ele = obj.front() → like peek()
  - obj.empty() → return true/false
  - obj.size() → return num of eles.

FIFO → O(1)

STL – string

- C string functions are available in C++
  - #include <cstring> → <string.h>
- C++ library provide string class to represent array of characters.
  - #include <string>
  - string str;
- String operations
  - getline(cin, str); → input string from console (cin).
  - str.length() → number of chars in string.
  - str.push_back('A'); → append a char in string.
  - str.pop_back() → remove last char.
  - str[i] or str.at(index) → access ith char of string.
  - diff = str.compare(str2) → like strcmp()
  - str.c_str() → return char*

```
#include<string>
using namespace std;
int main() {
    string str = "Sunbeam";
    cout << str.length();          // 7
    str.push_back('I');
    cout << str;                   // SunbeamI
    str.pop_back();
    cout << str;                   // Sunbeam
    string str2 = "Sundays";
    int diff = str.compare(str2);
    cout << diff;
    return 0;
}
```

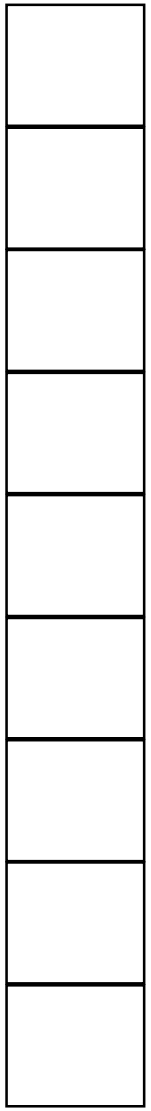
Infix to Postfix

5 9 + 4 8 6 2 / − * − 1 7 3 − $ +

- 5 + 9 − 4 * (8 − 6 / 2) + 1 $ (7 − 3)

5 + 9 − 4 * (8 − 6 / 2) + 1 $ (7 − 3)

① process each sym in infix from left to right.
② if sym is operand, append to postfix string.
③ if sym is operator, push on stack.
   ⓐ sym can be pushed on stack if it has higher priority than topmost ele on stack
   ⓑ if cur op priority is less or equal to topmost op on stack pop it and append to postfix string.
④ if '(' found, push on stack.
⑤ if ')' found, pop from stack and append until '(' is on stack. Also discard '('.
⑥ pop all and append to postfix

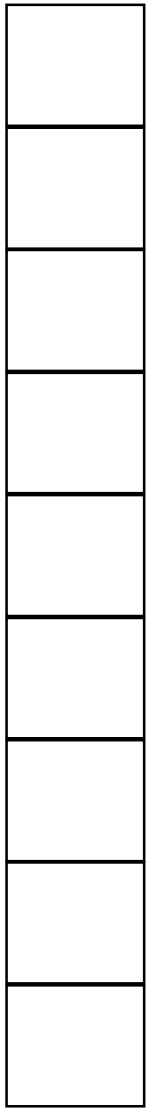


Infix to Postfix

59+4 8 62/ - * - 1 7 3 - $ +

- 5 + 9 – 4 * (8 – 6 / 2) + 1 $ (7 – 3)

- 5 9 + 4 8 6 2 / - * - 1 7 3 - $ +

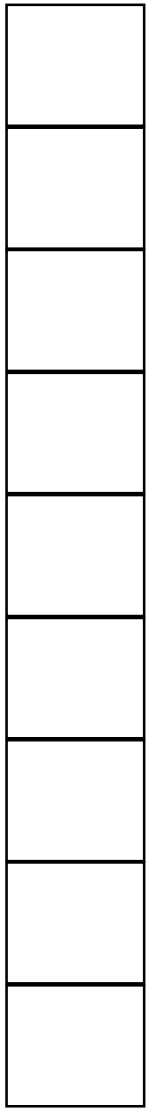


Infix to Prefix

+ - + 5 9 * 4 - 8 / 6 2 $ 1 - 7 3

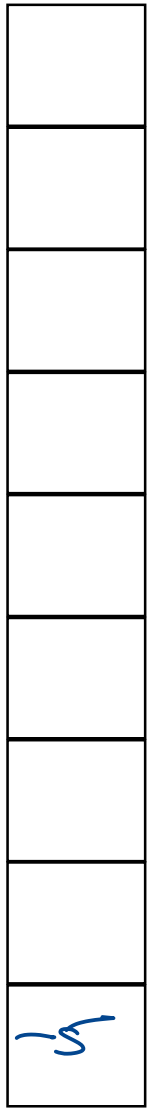
- 5 + 9 – 4 * (8 – 6 / 2) + 1 $ (7 – 3)

6 1 5 2 1 8 4 3



Postfix Evaluation

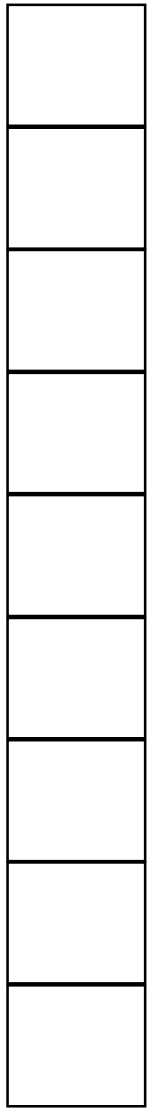
- 5 9 + 4 8 6 2 / - * - 1 7 3 - $ +



① process sym in postfix from left to right.
② if operand push on stack.
③ if operator pop 2 operands from stack, calc res -
& push result on stack.
- first poped will be second operand.
& second popped will be first operand.
④ when single value left on stack it is result

Prefix Evaluation

- + - + 5 9 * 4 – 8 / 6 2 $ 1 – 7 3



(12) (+) (34) (/) (() (48) (-) (46) ()) → parsing logic.

java/python → split()
C++ → stringstream.
C → strtok()

isdigit() → ctype.h

ascii - char

(ch >= '0' && ch <= '9')

'0' → 48
'1' → 49
'2' → 50
'3' → 51
'4' → 52
:

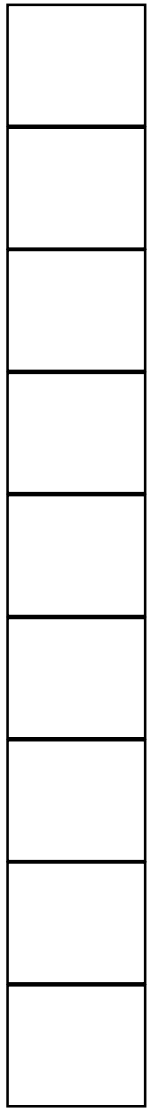
'0' - 48 → 0
'1' - 48 → 1
'2' - 48 → 2
'3' - 48 → 3
'4' - 48 → 4

Postfix to Infix

- While there are input symbol left
- Read the next symbol from input.
- If the symbol is an operand , Push it onto the stack.
- Otherwise, the symbol is an operator.
- If there are fewer than 2 values on the stack
- Show Error
- Else
- Pop the top 2 values from the stack.
- Put the operator, with the values as arguments and form a string.
- Encapsulate the resulted string with parenthesis.
- Push the resulted string back to stack.
- If there is only one value in the stack
- That value in the stack is the desired infix string.
- If there are more values in the stack
- Show Error

- a b c - + d e – f g – h + / *

stack<string> s;

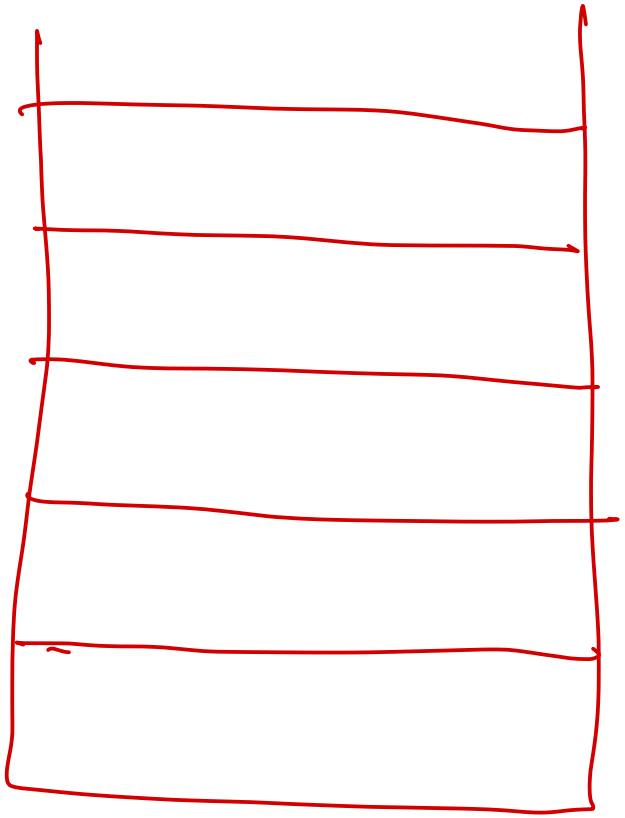


Prefix to Postfix

postfix: A B + C D − *

- Read the Prefix expression in reverse order (from right to left)
- If the symbol is an operand, then push it onto the Stack
- If the symbol is an operator, then pop two operands from the Stack
- Create a string by concatenating the two operands and the operator after them.
- string = operand1 + operand2 + operator
- And push the resultant string back to Stack
- Repeat the above steps until end of Prefix expression.

- * + A B – C D



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>