SCJ2013 Data Structure & Algorithms

Stack Application

Nor Bahiah Hj Ahmad & Dayang
Norhayati A. Jawawi

Stack Application Examples

- Check whether parantheses are balanced (open and closed parantheses are properly paired)
- Evaluate Algebraic expressions.
- Creating simple Calculator
- Backtracking (example. Find the way out when lost in a place)

Example1: Parantheses Balance

- Stack can be used to recognize a balanced parentheses.
- Examples of balanced parentheses.

  (a+b), (a/b+c), a/((b-c)*d)

  Open and closed parentheses are properly paired.
- Examples of not balance parentheses.

  ((a+b)*2 and m*(n+(k/2)))

  Open and closed parentheses are not properly paired.

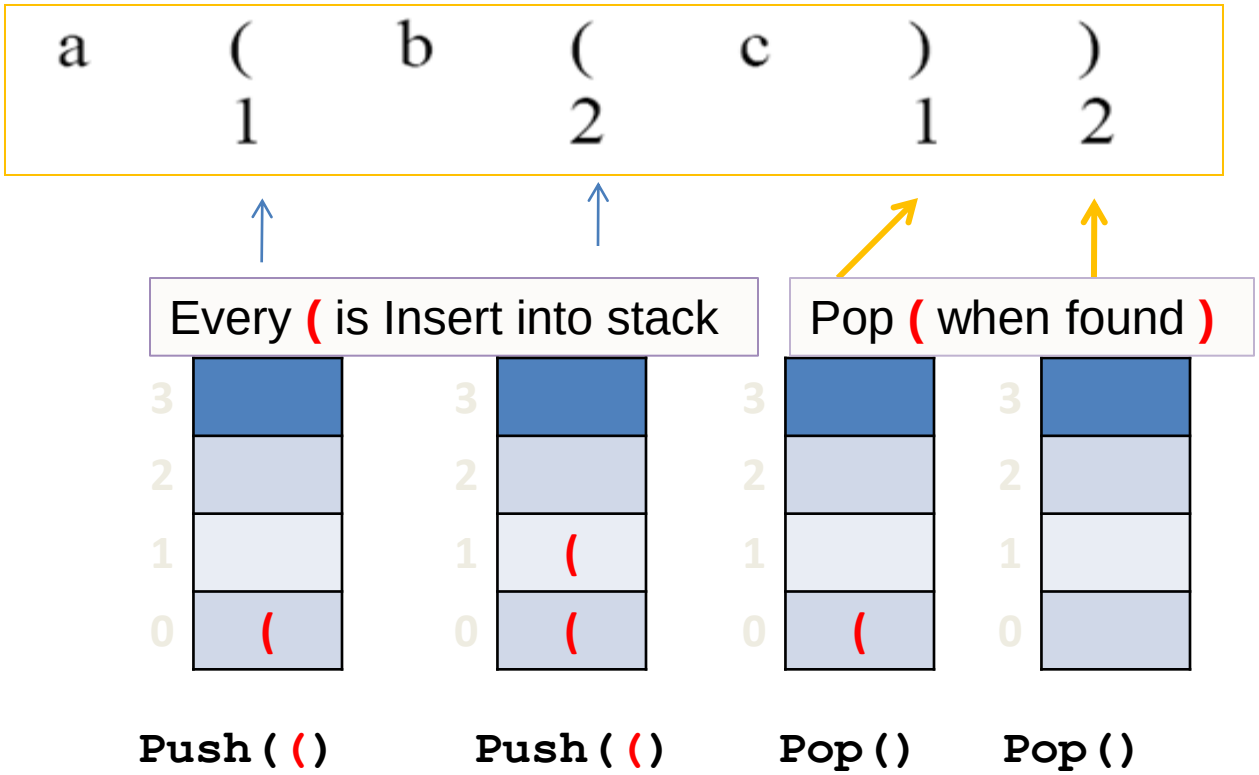
Check for Balanced Parantheses Algorithm

```
create (stack);
continue = true;
while ( not end of input string) && continue
{     ch = getch( );
      if ch = '(' || ch = ')'
        { if ch = '('
             Push(stack, '(');
          else if IsEmpty(stack)
            continue = false;
          else
             Pop(s);
         } // end if
} // end while
if ( end of input && isEmpty(stack);
     cout << "Balanced.." << endl;
else
     cout << "Not Balanced.. " << endl;
```

Check for Balanced Parantheses Algorithm

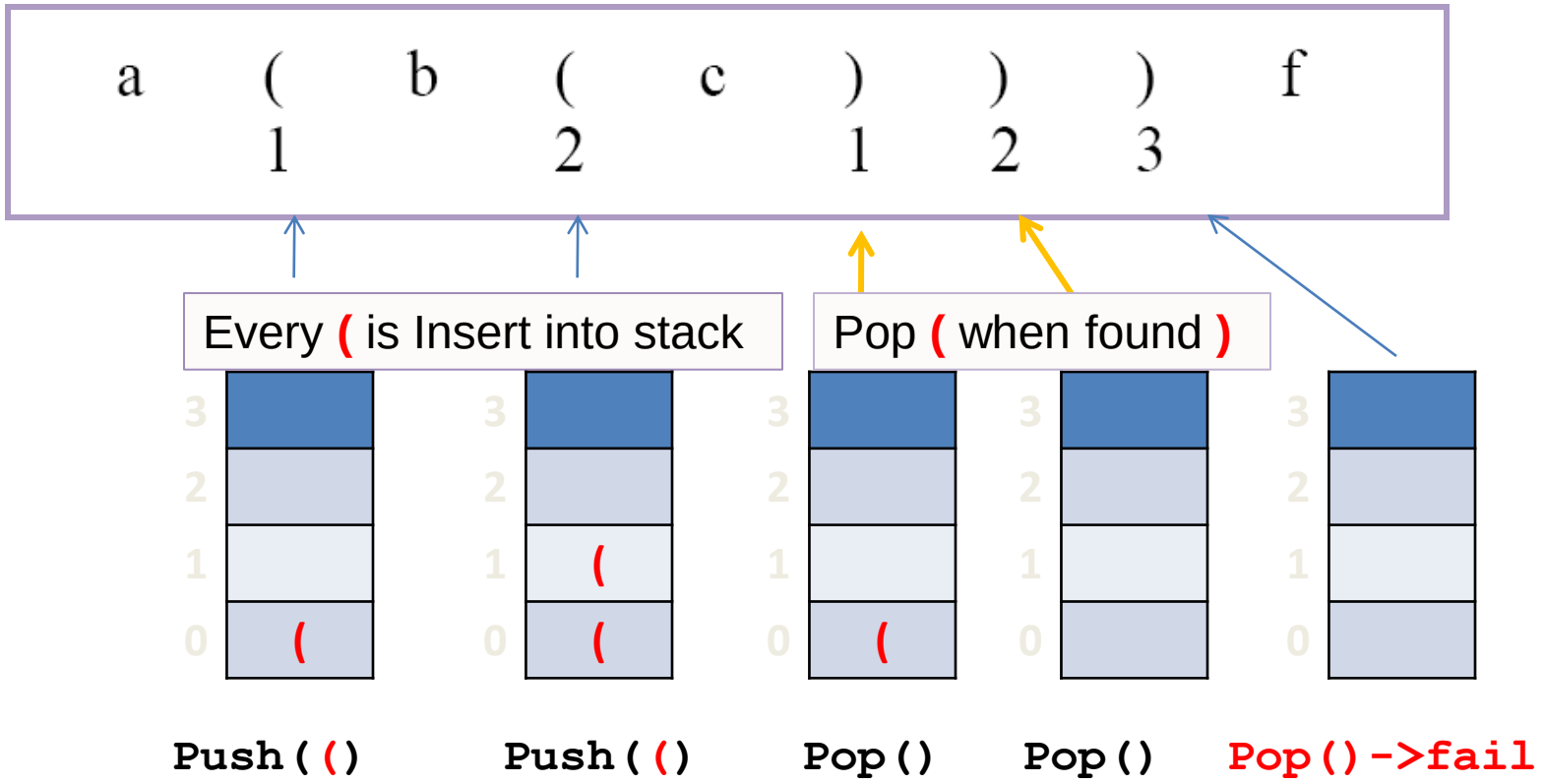
- Every '(' read from a string will be pushed into stack.
- The open parentheses '(' will be popped from a stack whenever the closed parentheses ')' is read from string.
- An expression have balanced parentheses if :
  - Each time a ")" is encountered it matches a previously encountered "(".
  - When reaching the end of the string, every "(" is matched and stack is finally empty.
- An expression does NOT have balanced parentheses if :
  - When there is still ')' in input string, the stack is already empty.
  - When end of string is reached, there is still '(' in stack.

Example for Balance Parantheses



Expression **a(b(c))** have balance parentheses since when end of string is found the stack is empty.

Example for ImBalanced Parantheses



Expression **a(b(c))) f** does not have balance parentheses => the third) encountered does not has its match, the stack is empty.

Algebraic expression

- One of the compiler's task is to evaluate algebraic expression.
- Example of assignment statement:

$$y = x + z * (w / x + z * (7 + 6))$$

- Compiler must determine whether the right expression is a syntactically legal algebraic expression before evaluation can be done on the expression.
- 3 algebraic expressions are :

  Infix, prefix and postfix

Infix Expression

- The algebraic expression commonly used is infix.
- The term infix indicates that every binary operators appears between its operands.
- Example 1:

A	+	B
operan	operator	operan

- Example 2: A + B * C

A + (B * C)
(a + b) * c

- To evaluate infix expression,

  the following rules were applied:

  1. Precedence rules.
  2. Association rules (associate from *left to right*)
  3. Parentheses rules.

Prefix and Postfix Expressions

- Alternatives to infix expression
- Prefix : Operator appears before its operand.
- Example:

```
+ a b
+ a * b c
* + a b c
```

- Postfix : Operator appears after its operand.
- Example:

```
a b +
a b c * +
a b + c *
```

Infix, prefix and postfix

Infix	Prefix	Postfix
a + b	+ a b	a b +
a + (b * c)	+ a * b c	a b c * +
(a + b) * c	* + a b c	a b + c *

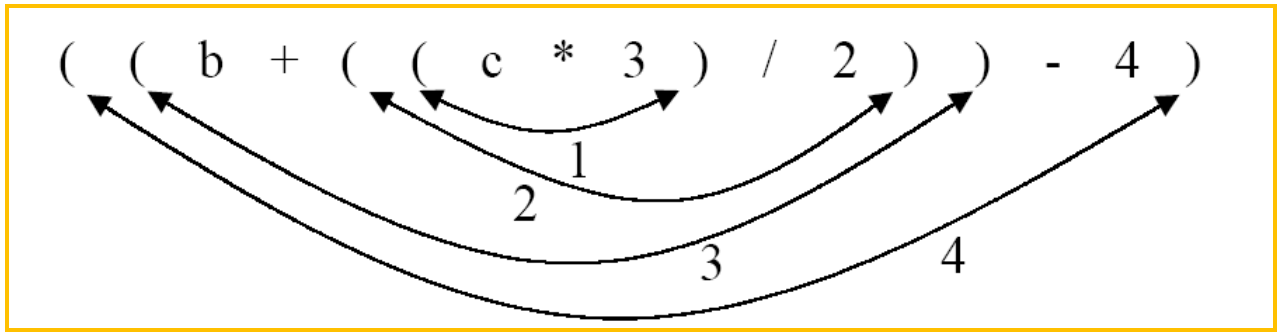
The advantage of using prefix and postfix is that we don't need to use precedence rules, associative rules and parentheses when evaluating an expression.

Converting Infix to Prefix

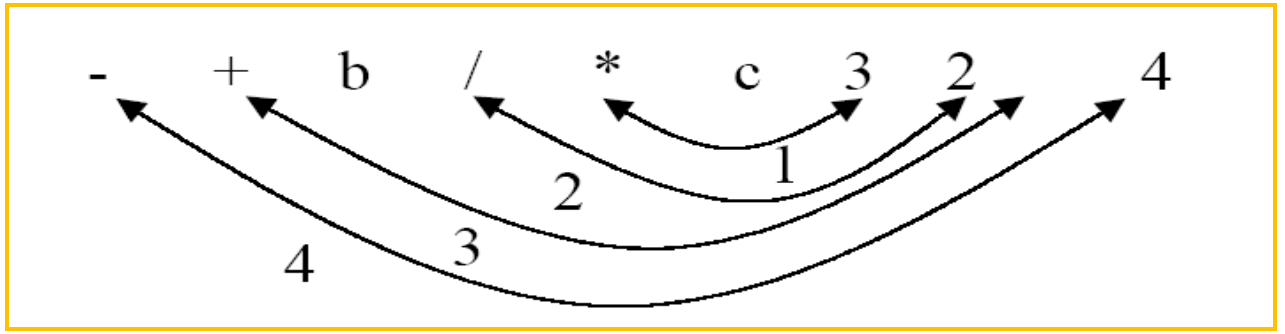
Steps to convert infix expression to prefix:

b + c * 3 / 2 - 4

STEP 1 : Determine the precedence.



STEP 2



Converting Infix to Prefix

Another Example:

1. a + b = (a + b)
   = + a b

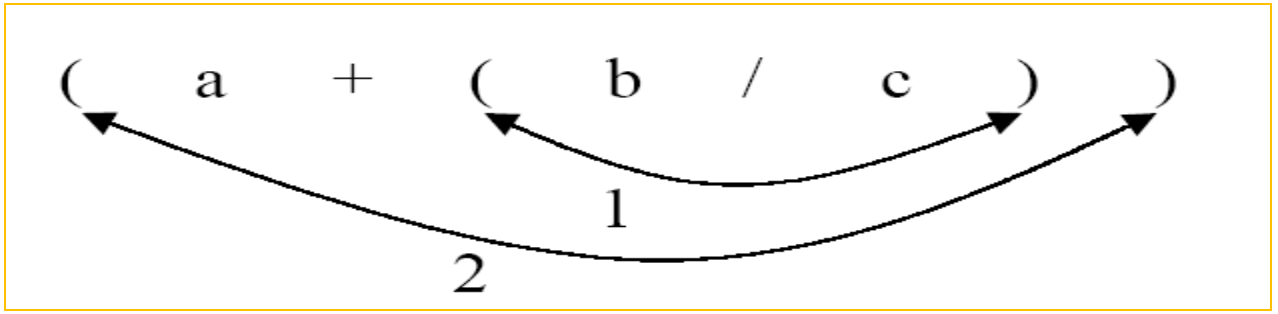
2. a + (b * c) = (a + (b * c))
   = + a * b c

Converting Infix to Postfix

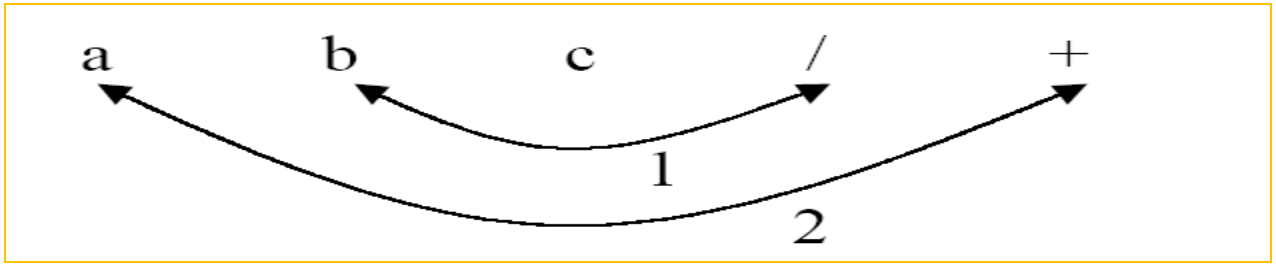
Steps to convert infix expression to postfix:

a + b / c

STEP 1



STEP 2



Converting Infix to Postfix

1. a + b = (a + b)
   = a b +

2. a + b * c = (a + (b * c))
   = a b c * +

3. a + b * (c – d) / (p – r)
   = a + (b * (c – d)) / (p – r)
   = (a + ((b * (c – d)) / (p – r)))
   = a b c d - * p r - / +

Evaluating Postfix Expression

- Postfix expression can be evaluated easily using stack.
- Stack operations, such as push(), pop() and isEmpty() will be used to solve this problem.
- Steps to evaluate postfix expression :

1. Convert infix to postfix expression.
2. Evaluate postfix using stack.

Converting Infix to Postfix Algorithm

```
create(s);
while (not end of infix input)
{       ch = getch();  // next input character
        if (ch is operand) add ch to postfix notation;
        if (ch = '(') push(ch)
        if (ch = ')')
        {       chpop = pop();
                while (chpop != '(')
                {       add chpop to postfix notation;
                        chpop = pop();
                }
        }
     if (ch is operator)
     {   while (!isEmpty() &&
             (precedence(stacktop()) >= precedence(ch)))
         {   chpop = pop();
             add chpop to postfix notation;
         }
         push(ch);
     }
}
while (!isEmpty())
{   ch = pop();
    add ch to postfix notation;
}
```

Converting Infix to Postfix

A + B * C – D / E

infix	stack	postfix
A + B * C – D / E	#	
+ B * C – D / E	#	A
B * C – D / E	# +	A
* C – D / E	# +	A B
C – D / E	# + *	A B C
– D / E	# + *	A B C * +
D / E	# -	A B C * + D
/ E	# -	A B C * + D
E	# - /	A B C * + D E
	# - /	A B C * + D E
	#	A B C * + D E / -

Converting Infix to Postfix :

A * B – (C + D)+ E

infix	stack	postfix
A * B – (C + D) + E	#	
* B – (C + D) + E	#	A
B – (C + D) + E	# *	A
– (C + D) + E	# *	A B
(C + D) + E	#	A B *
C + D) + E	# -	A B *
+ D) + E	# - (	A B *
D) + E	# - (	A B * C
) + E	# - (+	A B * C
+ E	# - (+	A B * C D
E	# -	A B * C D +
	#	A B * C D + -
	# +	A B * C D + -
	# +	A B * C D + - E
	#	A B * C D + - E +

Steps to Evaluate Postfix Expression

1. If char read from postfix expression is an operand, push operand to stack.
2. If char read from postfix expression is an operator, pop the first 2 operand in stack and implement the expression using the following operations:
   - **pop(opr1) dan pop(opr2)**
   - **result = opr2 operator opr1**
3. Push the result of the evaluation to stack.
4. Repeat steps 1 to steps 3 until end of postfix expression

- Finally, At the end of the operation, only one value left in the stack. The value is the result of postfix evaluation.

Evaluating Postfix Expression

```
Create Stack
while (not end of postfix notation)
{     ch = getch()
if (ch is operand)
            push (ch)
      else
      {     operan1 = pop()
            operan2 = pop()
            result = operan2 ch operan1
            push(result)
      }
}
result = pop()
```

Evaluating Postfix Expression : 2 4 6 + *

postfix	Ch	Opr	Opn1	Opn2	result	stack
2 4 6 + *						
4 6 + *	2					2
6 + *	4					2 4
+ *	6					2 4 6
*	+	+	4	2	6	2 10
	*	*	10	2	20	20

Evaluating Postfix Expression : 2 7 * 18 - 6 +

postfix	Ch	Opr	Opn1	Opn2	result	stack
2 7 * 18 – 6 +						
7 * 18 – 6 +	2					2
* 18 – 6 +	7					2 7
18 – 6 +	*	*	7	2	14	14
– 6 +	18					14 18
6 +	–	–	18	14	-4	-4
+	6					-4 6
	+	+	6	-4	2	2

Conclusion

- Stack is a simple structure but it is very powerful.
- Stacks can be used to decide whether a sequence of parantheses is well balanced.
- Stack also can be used to evaluate algebraic expression.