

Lecture # 6

Stack: Array or List

- Since both implementations support stack operations in constant time, any reason to choose one over the other?
- Allocating and deallocating memory for list nodes does take more time than preallocated array.
- List uses only as much memory as required by the nodes; array requires allocation ahead of time.
- List pointers (head, next) require extra memory.
- Array has an upper limit; List is limited by dynamic memory allocation.

Implementation

Issues/Discussion in C/C++

- Stack – push() operation
- Stack – pop() operation
- Stack – Is_Full() operation
- Stack – Is_Empty() operation

Uses of Stack

Use of Stack

- Example of use: prefix, infix, postfix expressions.
- Consider the expression $A+B$: we think of applying the *operator* “+” to the *operands* A and B.
- “+” is termed a *binary operator*. it takes two operands.
- Writing the sum as $A+B$ is called the *infix* form of the expression.

Prefix, Infix, Postfix

- Two other ways of writing the expression are

$+ A B$	<i>prefix</i>
$A B +$	<i>postfix</i>

- The prefixes “pre” and “post” refer to the position of the operator with respect to the two operands.

Prefix, Infix, Postfix

- Consider the infix expression

$$A + B * C$$

- We “know” that multiplication is done before addition.

- The expression is interpreted as

$$A + (B * C)$$

- Multiplication has *precedence* over addition.

Prefix, Infix, Postfix

■ Conversion to postfix

$A + (B * C)$

infix form

$A + (B C *)$

convert multiplication

$A (B C *) +$

convert addition

$A B C * +$

postfix form

Prefix, Infix, Postfix

- Conversion to postfix

$(A + B) * C$

infix form

$(A B +) * C$

convert addition

$(A B +) C *$

convert multiplication

$A B + C *$

postfix form

Precedence of Operators

- The five binary operators are: addition, subtraction, multiplication, division and exponentiation. The order of precedence is (highest to lowest)
- Exponentiation \uparrow
- Multiplication/division $*, /$
- Addition/subtraction $+, -$

Precedence of Operators

- For operators of same precedence, the left-to-right rule applies:

$A+B+C$ means $(A+B)+C$.

- For exponentiation, the right-to-left rule applies

$A \uparrow B \uparrow C$ means $A \uparrow (B \uparrow C)$

Infix to Postfix

Infix

$A + B$

$12 + 60 - 23$

$(A + B) * (C - D)$

$A \uparrow B * C - D + E / F$

Postfix

$A B +$

$12 60 + 23 -$

$A B + C D - *$

$A B \uparrow C * D - E F / +$

Infix to Postfix

- Note that the postfix form an expression does not require parenthesis.
- Consider ' $4+3*5$ ' and ' $(4+3)*5$ '. The parenthesis are not needed in the first but they are necessary in the second. The postfix forms are:

$4+3*5$

$(4+3)*5$

$435*+$

$43+5*$

Evaluating Postfix

- Each operator in a postfix expression refers to the previous two operands.
- Each time we read an operand, we push it on a stack.
- When we reach an operator, we pop the two operands from the top of the stack, apply the operator and push the result back on the stack.

Evaluating Postfix

■ Evaluate: 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
↑	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52