CS-2001
Data Structures
Spring 2022
Introduction to Stack ADT

Mr. Muhammad Yousaf

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

Roadmap

Previous Lecture

- Introduction to stack ADT
  - Applications
  - Library implementation
  - Array-based implementation
  - Linked list based Implementation

Today

- Postfix Notation
- Prefix Notation
- Postfix Expression evaluation and use of stack

Introduction to Stack

Stack ADT data

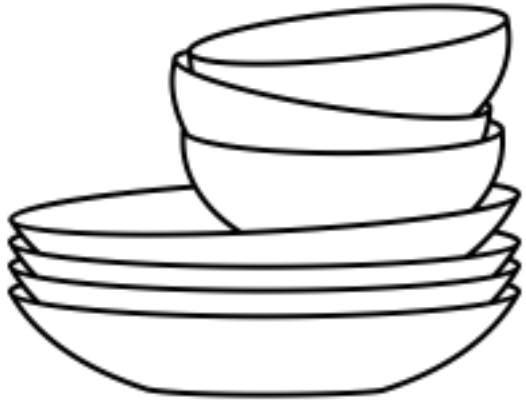
- A stack is a special kind of list
  - Insertion and deletions takes place at one end called top

- Other names
  - Last In First Out (LIFO) Structure
  - First In Last Out (FILO) Structure

Stack Examples

- Books on floor

- Dishes on a shelf / Dish rack



Stack ADT – Operations (2)

- MAKENULL(S)
  - Make Stack S be an empty stack
- TOP(S)
  - Return the element at the top of stack S
- POP(S)
  - Remove the top element of the stack
- PUSH(S,x)
  - Insert the element x at the top of the stack
- EMPTY(S)
  - Return true if S is an empty stack and return false otherwise

Array-based Implementation

Array Implementation – Code (1)

```
class IntStack
{
   private:
      int *stackArray;
      int stackSize;
      int top;

   public:
      IntStack(int);
      ~IntStack( );
      bool push(int);
      bool pop(int &);
      bool isFull();
      bool isEmpty();
};
```

Array Implementation – Code (2)

- Constructor

```
IntStack::IntStack(int size) //constructor
{
   stackArray = new int[size];
   stackSize = size;
   top = -1;
}
```

- Destructor

```
IntStack::~IntStack(void) //destructor
{
   delete [] stackArray;
}
```

Array Implementation – Code (3)

- `isFull` function

```
bool IntStack::isFull(void)
{
   if (top == stackSize - 1)
      return true;
   else
      return false;
   // return (top == stackSize-1);
}
```

- `isEmpty` function

```
bool IntStack::isEmpty(void)
{
   return (top == -1);
}
```

Array Implementation – Code (4)

- `push` function inserts the argument `num` onto the stack

```
bool IntStack::push(int num)
{
   if (isFull())
   {
      cout << "The stack is full.\n";
      return false;
   }

   top++;
   stackArray[top] = num;
   return true;
}
```

Array Implementation – Code (5)

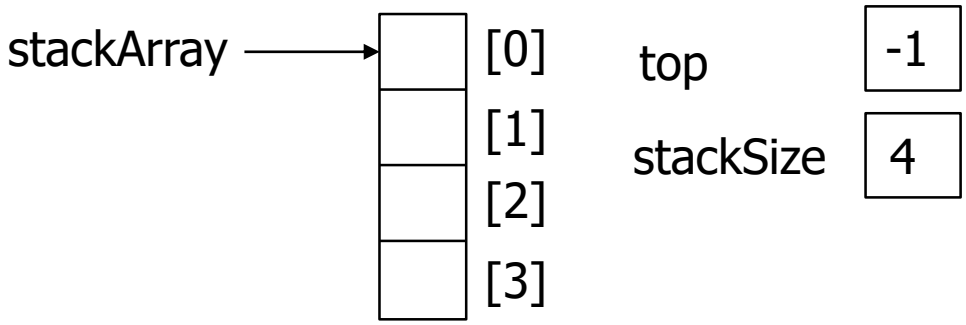
- `Pop` function removes the value from top of the stack and returns it as a reference

```
bool IntStack::pop(int &num)
{
   if (isEmpty())
   {
      cout << "The stack is empty.\n";
      return false;
   }

   num = stackArray[top];
   top--;
   return true;
}
```

Using Stack (1)

```
int main()
{
   IntStack stack(4);
```

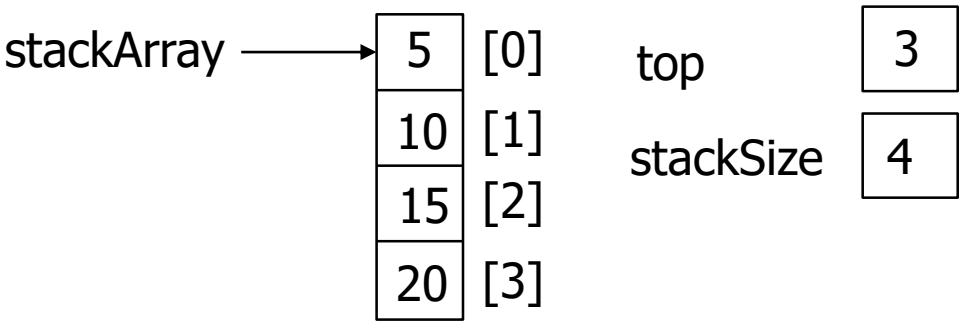



```
}
```

Using Stack (2)

```
int main()
{
   IntStack stack(4);
   int catchVar;

   cout << "Pushing Integers\n";
   stack.push(5);
   stack.push(10);
   stack.push(15);
   stack.push(20);
```




```
}
```

Using Stack (3)

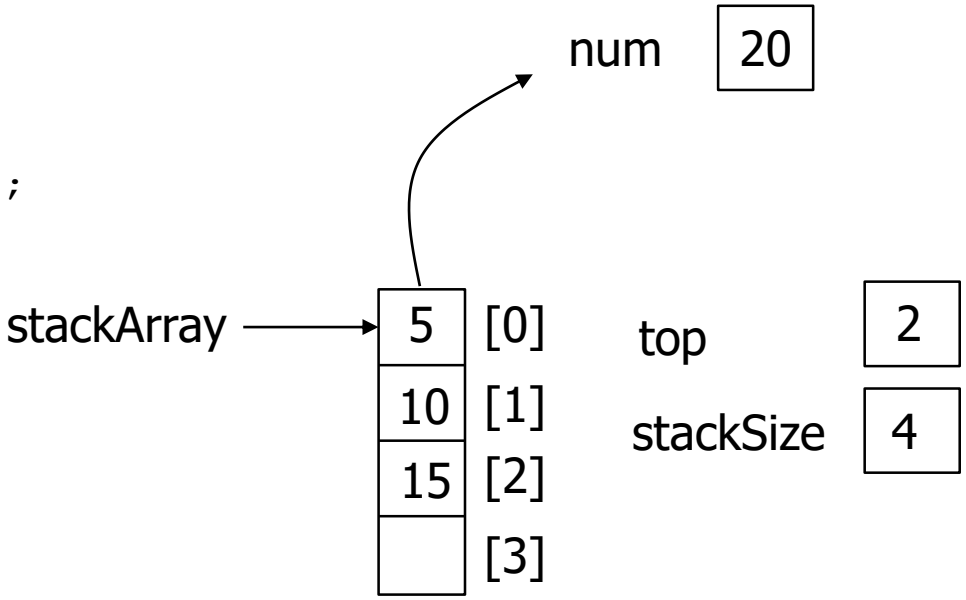
```
int main()
{
   IntStack stack(4);
   int catchVar;

   cout << "Pushing Integers\n";
   stack.push(5);
   stack.push(10);
   stack.push(15);
   stack.push(20);

   cout << "Popping...\n";
   stack.pop(catchVar);
   cout << catchVar << endl;




}
```



Using Stack (4)

```
int main()
{
   IntStack stack(4);
   int catchVar;

   cout << "Pushing Integers\n";
   stack.push(5);
   stack.push(10);
   stack.push(15);
   stack.push(20);

   cout << "Popping...\n";
   stack.pop(catchVar);
   cout << catchVar << endl;
   stack.pop(catchVar);
   cout << catchVar << endl;
   stack.pop(catchVar);
   cout << catchVar << endl;
   stack.pop(catchVar);
   cout << catchVar << endl;
   return 0;
}
```

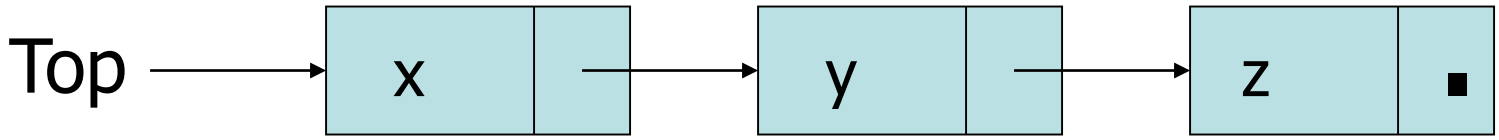
Output:

```
Pushing Integers
Popping...
20
15
10
5
```

Pointer-based Implementation

Pointer-based Implementation of Stacks

- Stack can expand or shrink with each push or pop operation
- Push and pop operate only on the head node, i.e., the first node of the list



Pointer Implementation – Code (1)

```
class node
   {  public:
      int data; node *next;
   };
class Stack
{
   node *top;
   public:
      Stack();
      ~Stack();
      void Push(int newelement);
      bool Pop(int &);
      bool IsEmpty();
      void makeNull();
};
```

Pointer Implementation – Code (2)

- Constructor

```
Stack::Stack()
{
    top = NULL;
}
```

- `IsEmpty` function returns true if the stack is empty

```
bool Stack::IsEmpty()
{
    return (top == NULL);
}
```

Pointer Implementation – Code (3)

- `Push` function inserts a node at the top/head of the stack

```
void Stack::Push(int newelement)
{
    node *newptr = new node;
    newptr->data=newelement;

    newptr->next=top;
    top=newptr;
}
```

Pointer Implementation – Code (4)

- `Pop` function deletes the node from the top of the stack and returns its data by reference

```
bool Stack::Pop(int& returnvalue)
{
    if (IsEmpty())
    {
        cout<<"underflow error";
        return false;
    }

    node* tempptr = top;
    returnvalue = top->data;
    top = top->next;

    delete tempptr;
    return true;
}
```

Pointer Implementation – Code (5)

- Destructor

```
Stack::~Stack()
{
   makeNull(); //deletion is already done in pop function
}
```

- `makeNull()` resets `top` pointer to `NULL`

```
void Stack::makeNull()
{
   int x;
   while( Pop(x) );
}
```

Polish Notations and Use of Stack

Algebraic Expressions

- An algebraic expression is combination of operands and operators
- Operand is a quantity that is operated on
- Operator is a symbol that signifies a mathematical or logical operation

Associativity of Operators

( ) [ ] . -> ++ – –	Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement (see Note 2)	left-to-right
++ – – + – ! ~	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left

Infix, Postfix and Prefix Expressions

- Infix
  - Expressions in which operands surround the operators
  - Example: A+B-C
- Postfix or Reverse Polish Notation (RPN)
  - Operators comes after the operands
  - Example: AB+C-
- Prefix or Polish Notation
  - Operator comes before the operands
  - Example: -+ABC

Example: Conversion From Infix to Postfix (1)

- Infix: A+B*C

- Conversion: Applying the rules of precedence

  A+(B*C) Parentheses for emphasis

  A+(BC*) Convert the multiplication

  ABC*+ Postfix Form

Example: Conversion From Infix to Postfix (2)

- Infix: ((A+B)*C-(D-E)) $ (F+G)

- Conversion: Applying the rules of precedence

  ((AB+)*C-(DE-)) $ (FG+)

  ((AB+C*)-(DE-)) $ (FG+)

  (AB+C*DE--) $ (FG+)

  AB+C*DE- -FG+$

- Exercise: Convert the following to Postfix

  (A + B) * (C – D)

  A / B * C – D + E / F / (G + H)

Infix, Postfix and Prefix Expressions – Examples

Infix	**PostFix**	**Prefix**
A+B	AB+	+AB
(A+B) * (C + D)	AB+CD+*	*+AB+CD
A-B/(C*D^E)	?	?

Why Do We Need Prefix and Postfix? (1)

- Normally, algebraic expressions are written using Infix notation
  - For example: (3 + 4) × 5 – 6
- Appearance may be misleading; Infix notations are not as simple as they seem
  - Operator precedence
  - Associativity property
- Operators have precedence: Parentheses are often required
  - (3 + 4) × 5 – 6 = 29
  - 3 + 4 × 5 – 6 = 17
  - (3 + 4) × (5 – 6) = –7
  - 3 + 4 × (5 – 6) = –1

Why Do We Need Prefix and Postfix? (2)

- Infix Expression is Hard To Parse and difficult to evaluate

- Postfix and prefix do not rely on operator priority and are easier to parse
  - No ambiguity and no brackets are required

- Many compilers first translate algebraic expressions into some form of postfix notation
  - Afterwards translate this postfix expression into machine code

Expression Evaluation (Major Challenges)

- We face two problems to evaluate an expression

1. Given an infix expression, convert it into a postfix expressions.
2. Evaluate a postfix expression

Expression Evaluation (Major Challenges)

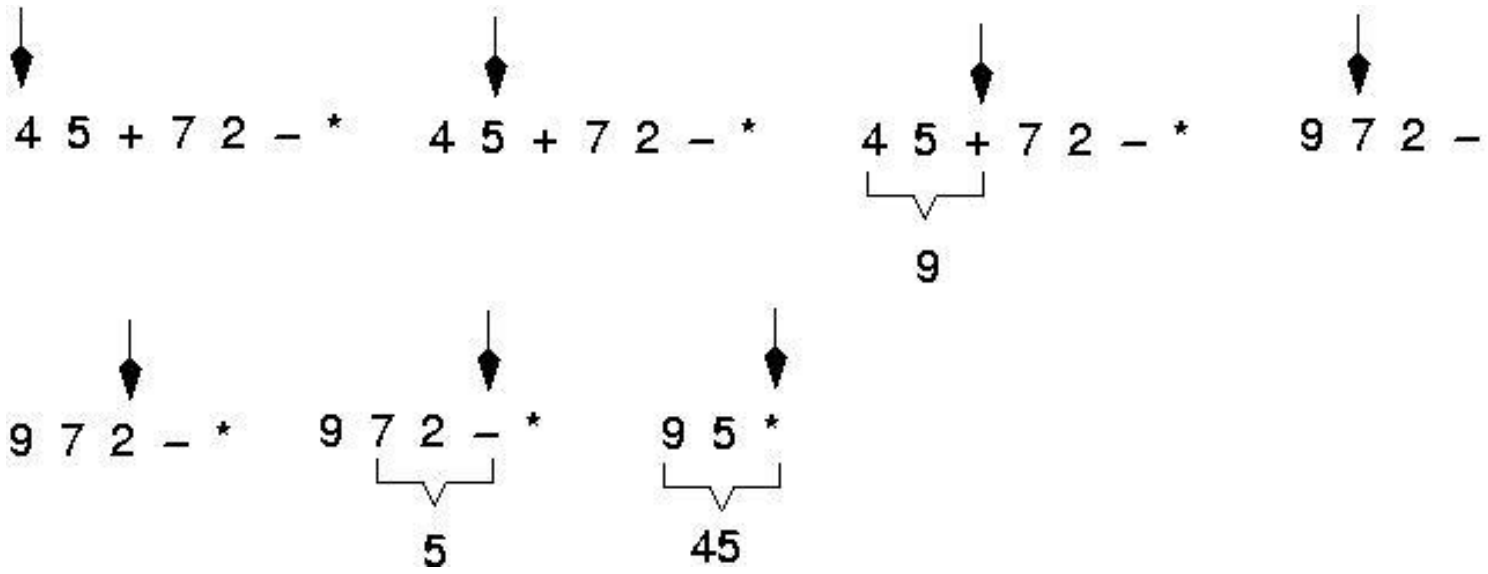
- We face two problems to evaluate an expression

  1. Conversion of infix to postfix
  2. Evaluation of postfix expression and calculate the solution

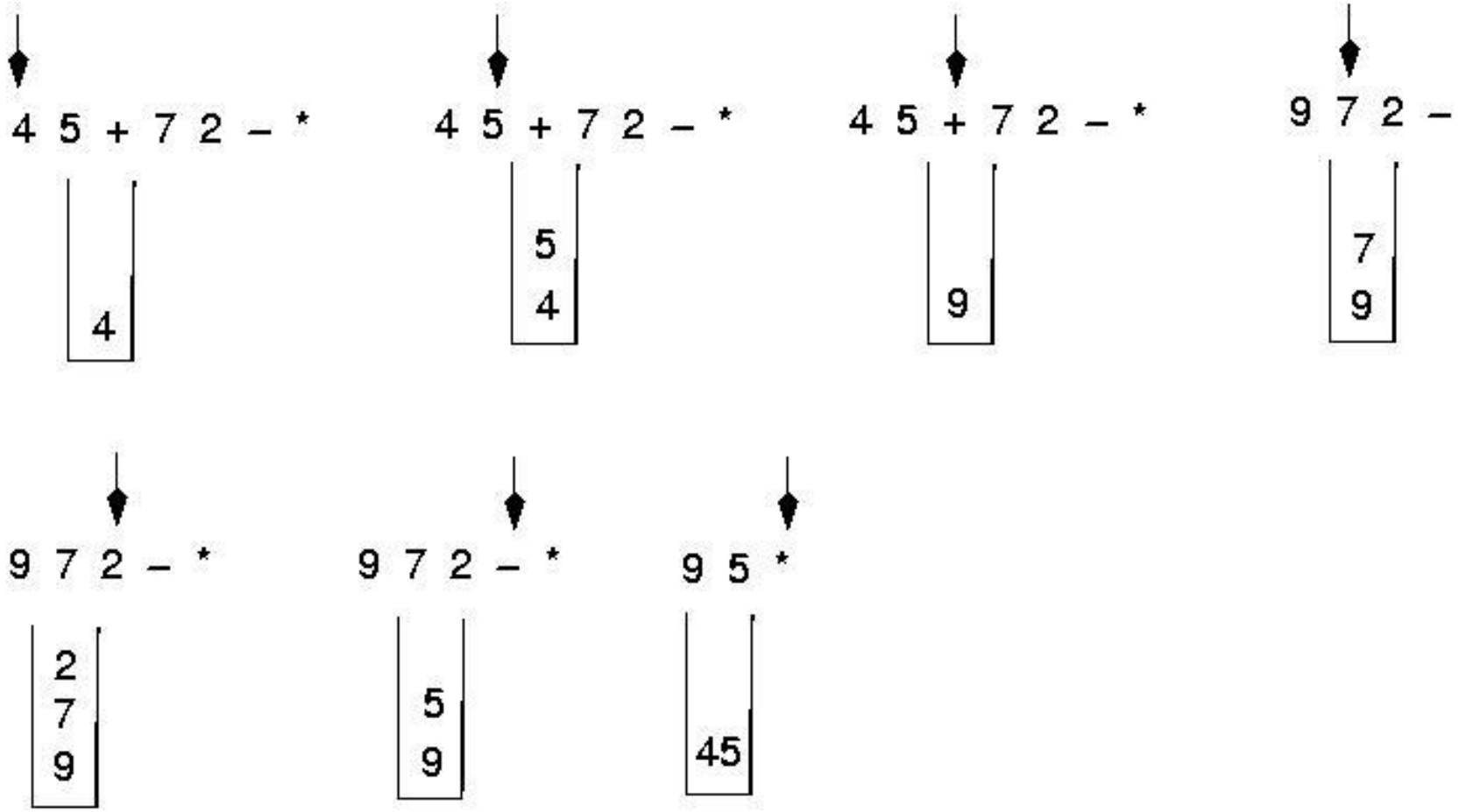
- Let's address the second problem first

Postfix Expression Evaluation

Example: Postfix Expressions Evaluation



Example: Postfix Expressions Evaluation and Use of Stack



Example: Postfix Expressions Evaluation and Use of Stack

Quick Exercise

- What does the following postfix expression evaluate to?
  6 3 2 + *

  A. 18
  B. 36
  C. 24
  D. 11
  E. 30

Evaluating a Postfix Expression

```
Let stack be a new Stack object
/* scan the input string reading one element */
/* at a time into symb */
while (not end of input) {
   symb = next input character;
   if (symb is an operand)
      stack. push(symb)
   else {
      /* symb is an operator */
        stack. pop(opnd2);
        stack. pop(opnd1);
        result = result of applying symb
                 to opnd1 and opnd2;
      stack. push(result);
   } /* end else */
} /* end while */
stack. pop(final_result); //add final result to final_result
```

Each operator in postfix string refers to the previous two operands in the string.

Evaluating a Postfix Expression

Example Postfix Expression:
6 2 3 + - 3 8 2 / + * 2 $ 3 +

```
Let stack be a new Stack object
/* scan the input string reading
   one element */
/* at a time into symb */
while (not end of input) {
   symb = next input character;
   if (symb is an operand)
      stack. push(symb)
   else {
      /* symb is an operator */
         stack. pop(opnd2);
         stack. pop(opnd1);
         result = result of
                applying symb
              to opnd1 and opnd2;
      stack. push(result);
   } /* end else */
} /* end while */
stack. pop(final_result);
```

symb	opnd1	opnd2	result	stack

Evaluating a Postfix Expression

Example Postfix Expression:
6 2 3 + - 3 8 2 / + * 2 $ 3 +

```
Let stack be a new Stack object
/* scan the input string reading
   one element */
/* at a time into symb */
while (not end of input) {
   symb = next input character;
   if (symb is an operand)
      stack. push(symb)
   else {
      /* symb is an operator */
         stack. pop(opnd2);
         stack. pop(opnd1);
         result = result of
                applying symb
              to opnd1 and opnd2;
      stack. push(result);
   } /* end else */
} /* end while */
stack. pop(final_result);
```

symb	opnd1	opnd2	result	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

Infix to Postfix Conversion Using Stack

Conversion of Infix Expression to Postfix – Rules If Expression Does not Contain Parenthesis

- Token is an operand
  - Append it to the end of postfix string
- Token is an operator, *, /, +, or –
  - First remove any operators already on the opstk OR stack that have higher or equal precedence and append them to the postfix string
  - Push the token on the opstk OR stack
- Input expression has been completely processed
  - Any operators still on the opstk OR stack should be removed and appended to the end of the postfix string

- Example: 4 - 2 + 6 * 2

Conversion of Infix Expression to Postfix

- Precedence function
  - `prcd(op1, op2)`
  - `op1` and `op2` are characters representing operators

- Precedence function returns `TRUE`
  - If `op1` has precedence over `op2` **`(OR)`** `op1 is same as op2`
- Otherwise function returns `FALSE`

- Examples
  - `prcd('*','+')` returns `TRUE`
  - `prcd('+','+')` returns `TRUE`
  - `prcd('+','*')` returns `FALSE`

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        push(opstk, symb);
    } /* end else */
} /* end while */
/* add remaining operators to string*/
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: A+B*C

symb	Postfix string	opstk
A	A	

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        push(opstk, symb);
    } /* end else */
} /* end while */
/* add remaining operators to string*/
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: A+B*C

symb	Postfix string	opstk
A	A	
+	A	+

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        push(opstk, symb);
    } /* end else */
} /* end while */
/* add remaining operators to string*/
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: A+B*C

symb	Postfix string	opstk
A	A	
+	A	+
B	AB	+

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        push(opstk, symb);
    } /* end else */
} /* end while */
/* add remaining operators to string*/
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: A+B*C

symb	Postfix string	opstk
A	A	
+	A	+
B	AB	+
*	AB	+ *

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        push(opstk, symb);
    } /* end else */
} /* end while */
/* add remaining operators to string*/
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: A+B*C

symb	Postfix string	opstk
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        push(opstk, symb);
    } /* end else */
} /* end while */
/* add remaining operators to string*/
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: A+B*C

symb	Postfix string	opstk
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *
	ABC*	+

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        push(opstk, symb);
    } /* end else */
} /* end while */
/* add remaining operators to string*/
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: A+B*C

symb	Postfix string	opstk
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *
	ABC*	+
	ABC*+	

Algorithm to Convert Infix to Postfix – Practice

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        push(opstk, symb);
    } /* end else */
} /* end while */
/* add remaining operators to string*/
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: A*B+C

symb	Postfix string	opstk

Algorithm to Convert Infix to Postfix – Practice

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        push(opstk, symb);
    } /* end else */
} /* end while */
/* add remaining operators to string*/
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: A*B+C

symb	Postfix string	opstk
A	A	
*	A	*
B	AB	*
+	AB*	+
C	AB*C	+
	AB*C+	

Conversion of Infix Expression to Postfix – Rules If Expression Contains Parenthesis

- Token is an operand
  - Append it to the end of postfix string
- Token is a left parenthesis
  - Push it on the opstk
- Token is a right parenthesis
  - Pop the opstk until the corresponding left parenthesis is removed
  - Append each operator to the end of the postfix string
  - Pop the left parenthesis from the stack [opstk] and discard it as well
- Token is an operator, *, /, +, or –
  - Push it on the opstk
  - First remove any operators already on the opstk that have higher or equal precedence and append them to the postfix string
- Input expression has been completely processed
  - Any operators still on the opstk can be removed and appended to the end of the postfix string

What If Expression Contains Parenthesis? Required Algorithmic changes

- Precedence function `prcd(op1, op2)` has to be modified
  - `prcd( '(' , op) = FALSE` For any operator `op`
  - `prcd( op, '(' ) = FALSE` For any operator `op` other than ')'
    - In short, whenever a **'('** is encountered → Push it onto stack
    - FALSE will ensure that by terminating the *while-loop*
  - `prcd( op, ')' ) = TRUE` For any operator `op` other than '('
    - So, that we can pop all the operators until a starting parathesis is not encountered from the stack
  - `prcd( ')' ,op ) = undef` For any operator `op` (an error)
  - As you will never push closing parenthesis in the stack, so, this case will never be encountered

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        if ( empty(opstk)|| symb != ')' )
            push(opstk, symb);
        else //pop the parenthesis & discard it
            topsymb = pop(opstk);
    } /* end else */
} /* end while */
while (!empty(opstk) ) { // remaining ops
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: (A+B)*C

symb	Postfix string	opstk

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        if ( empty(opstk)|| symb != ')' )
            push(opstk, symb);
        else //pop the parenthesis & discard it
            topsymb = pop(opstk);
    } /* end else */
} /* end while */
while (!empty(opstk) ) { // remaining ops
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: (A+B)*C

symb	Postfix string	opstk
(		(

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        if ( empty(opstk)|| symb != ')' )
            push(opstk, symb);
        else //pop the parenthesis & discard it
            topsymb = pop(opstk);
    } /* end else */
} /* end while */
while (!empty(opstk) ) { // remaining ops
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: (A+B)*C

symb	Postfix string	opstk
(		(
A	A	(

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        if ( empty(opstk)|| symb != ')' )
            push(opstk, symb);
        else //pop the parenthesis & discard it
            topsymb = pop(opstk);
    } /* end else */
} /* end while */
while (!empty(opstk) ) { // remaining ops
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: (A+B)*C

symb	Postfix string	opstk
(		(
A	A	(
+	A	(+

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        if ( empty(opstk)|| symb != ')' )
            push(opstk, symb);
        else //pop the parenthesis & discard it
            topsymb = pop(opstk);
    } /* end else */
} /* end while */
while (!empty(opstk) ) { // remaining ops
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: (A+B)*C

symb	Postfix string	opstk
(		(
A	A	(
+	A	(+
B	AB	(+

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        if ( empty(opstk)|| symb != ')' )
            push(opstk, symb);
        else //pop the parenthesis & discard it
            topsymb = pop(opstk);
    } /* end else */
} /* end while */
while (!empty(opstk) ) { // remaining ops
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: (A+B)*C

symb	Postfix string	opstk
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        if ( empty(opstk)|| symb != ')' )
            push(opstk, symb);
        else //pop the parenthesis & discard it
            topsymb = pop(opstk);
    } /* end else */
} /* end while */
while (!empty(opstk) ) { // remaining ops
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: (A+B)*C

symb	Postfix string	opstk
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	
*	AB+	*

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        if ( empty(opstk)|| symb != ')' )
            push(opstk, symb);
        else //pop the parenthesis & discard it
            topsymb = pop(opstk);
    } /* end else */
} /* end while */
while (!empty(opstk) ) { // remaining ops
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: (A+B)*C

symb	Postfix string	opstk
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	
*	AB+	*
C	AB+C	*

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        add symb to the postfix string
    else {
        while (!empty(opstk) && prcd(stacktop(opstk),symb) ) {
            topsymb = pop(opstk);
            add topsymb to the postfix string;
        } /* end while */
        if ( empty(opstk)|| symb != ')' )
            push(opstk, symb);
        else //pop the parenthesis & discard it
            topsymb = pop(opstk);
    } /* end else */
} /* end while */
while (!empty(opstk) ) { // remaining ops
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

Example: (A+B)*C

symb	Postfix string	opstk
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	
*	AB+	*
C	AB+C	*
	AB+C*	

Conversion of Infix Expression to Postfix – Practice

- Example: ((A-(B+C)) *D) $ (E+F)

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
          add symb to the postfix string
    else {
          while (!empty(opstk)
                    && prcd(stacktop(opstk),symb) ) {
              topsymb = pop(opstk);
              add topsymb to the postfix string;
          } /* end while */
          if ( empty(opstk) || symb != ')' )
                    push(opstk, symb);
          else  //pop the paranthesis and discard it
                    topsymb = pop(opstk);
    } /* end else */
} /* end while */
/* output any remaining operators */
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

symb	Postfix string	opstk

Conversion of Infix Expression to Postfix – Practice

- Example: ((A-(B+C)) *D) $ (E+F)

```
opstk = the empty stack;
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
          add symb to the postfix string
    else {
          while (!empty(opstk)
                    && prcd(stacktop(opstk),symb) ) {
              topsymb = pop(opstk);
              add topsymb to the postfix string;
          } /* end while */
          if ( empty(opstk) || symb != ')' )
                    push(opstk, symb);
          else  //pop the paranthesis and discard it
                    topsymb = pop(opstk);
    } /* end else */
} /* end while */
/* output any remaining operators */
while (!empty(opstk) ) {
    topsymb = pop(opstk);
    add topsymb to the postfix string;
} /* end while */
```

symb	Postfix string	opstk
(		(
(		((
A	A	((
-	A	((-
(	A	((-(
B	AB	((-(
+	AB	((-(+
C	ABC	((-(+
)	ABC+	((-
)	ABC+-	(
*	ABC+-	(*
D	ABC+-D	(*
)	ABC+-D*	
$	ABC+-D*	$
(	ABC+-D*	$(
E	ABC+-D*E	$(
+	ABC+-D*E	$(+
F	ABC+-D*EF	$(+
)	ABC+-D*EF+	$
	ABC+-D*EF+$	

Infix to Prefix Conversion

Conversion To Prefix Expression (1)

- An Infix to Prefix Conversion Algorithm
  - Reverse the infix string
    - Adjust parenthesis, i.e., make every '(' as ')' and every ')' as '('
  - Perform infix to postfix algorithm on reversed string
  - Reverse the output postfix expression to get the prefix expression

- Example: (A + B) * (B – C)
  -)C – B(*)B + A(→ **(C – B) * (B + A)** Reverse infix string
  - C B - B A + * Perform infix to postfix conversion
  - * + A B - B C Reverse postfix to get prefix expression

Conversion To Prefix Expression (2)

- Example: (A+B^C)*D+E^5
  - 5^E+D*)C^B+A(→ **5^E+D*(C^B+A)** Reverse infix string
  - 5E^DCB^A+*+ Perform infix to postfix conversion
  - +*+A^BCD^E5 Reverse postfix to get prefix expression

Any Question So Far?