Data Structures

Fall 2023

9. Applications of Stack

Applications

- The stack is a very simple data structure
  - Given any problem, if it is possible to use a stack, this significantly simplifies the solution

- Many applications
  - Dealing with undo/redo operations
  - Parsing Code
  - Matching parenthesis
  - Validating algebraic expressions
  - Tracking function calls

Balancing Symbols

- Compilers need to check programs for syntax errors
- How to check whether everything is balanced?

```
int main {
       while (true) {
              if (condition) {
                     if (condition) {
                            \\ statement
                     } else {
                            \\ statement
                     }
              }
       }
}
```

Balancing Symbols

- Check if the following brackets are balanced:

{(){(){(){}{}}}}

- **Algorithm:**

```
• Create an empty stack
• Repeat
    • Read next symbol
    • If symbol is open bracket, Push to stack
    • If symbol is close bracket
        • Compare new symbol with top of stack
        • If top is open bracket of same type, Pop stack
• If stack is empty at the end -> Balanced
```

Palindromes

- Strings that read the same forwards and backwards
  - `level, radar, 0011100`

- How to check if a string is a palindrome?

- **Algorithm:**

```
• Find length of string. Calculate mid = length/2
• Create an empty stack
• Push all elements until mid – 1
• If length is odd, ignore middle element
• Repeat for remaining elements
    • If element == top of stack, Pop stack
• If stack is empty at the end -> Palindrome
```

Algebraic Expressions

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

Algebraic Expressions

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

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)

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) = -1
  - (3 + 4) × (5 – 6) = -7

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

```
MOVE.L #$2A, D1      ; Load  42 into Register D1
MOVE.L #$100, D2     ; Load 256 into Register D2
ADD D2, D1           ; Add D2 into D1
```

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`
  - 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 */
/* output any remaining operators */
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 */
/* output any remaining operators */
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 */
/* output any remaining operators */
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 */
/* output any remaining operators */
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 */
/* output any remaining operators */
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 */
/* output any remaining operators */
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 */
/* output any remaining operators */
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 */
/* output any remaining operators */
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 */
/* output any remaining operators */
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+	

What If Expression Contains Parenthesis?

- 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 ‘)’
  - `prcd( op, ‘)‘ ) = TRUE` For any operator `op` other than ‘(‘
  - `prcd( ‘)‘ ,op ) = undef` For any operator `op` (an error)

Algorithm to Convert Infix to Postfi

```
prcd( ‘(‘ , op) = FALSE

prcd( op, ‘(‘ ) = FALSE

prcd( op, ‘)‘ ) = TRUE

prcd( ‘)‘ ,op ) = undef
```

```
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 Postfi

```
prcd( ‘(‘ , op) = FALSE

prcd( op, ‘(‘ ) = FALSE

prcd( op, ‘)‘ ) = TRUE

prcd( ‘)‘ ,op ) = undef
```

```
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 Postfi

```
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 */
```

```
prcd( ‘(‘ , op) = FALSE

prcd( op, ‘(‘ ) = FALSE

prcd( op, ‘)‘ ) = TRUE

prcd( ‘)‘ ,op ) = undef
```

Example: (A+B)*C

symb	Postfix string	opstk
(		(
A	A	(

Algorithm to Convert Infix to Postfi

```
prcd( ‘(‘ , op) = FALSE

prcd( op, ‘(‘ ) = FALSE

prcd( op, ‘)‘ ) = TRUE

prcd( ‘)‘ ,op ) = undef
```

```
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 Postfi

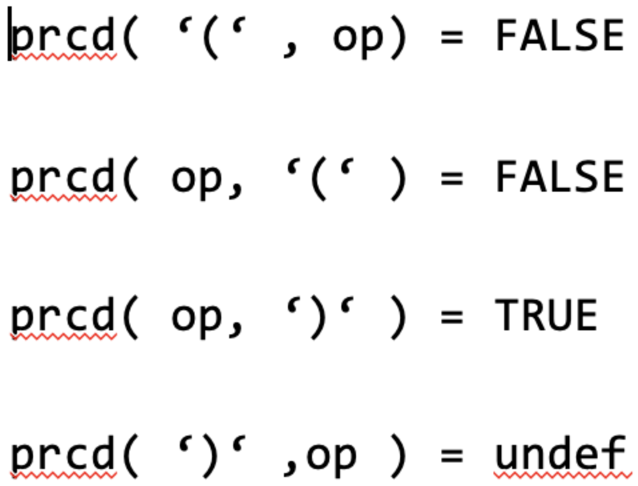
```
prcd( ‘(‘ , op) = FALSE

prcd( op, ‘(‘ ) = FALSE

prcd( op, ‘)‘ ) = TRUE

prcd( ‘)‘ ,op ) = undef
```

```
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 Postfi



```
prcd( ‘(‘ , op) = FALSE

prcd( op, ‘(‘ ) = FALSE

prcd( op, ‘)‘ ) = TRUE

prcd( ‘)‘ ,op ) = undef
```

```
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 Postfi

```
prcd( ‘(‘ , op) = FALSE

prcd( op, ‘(‘ ) = FALSE

prcd( op, ‘)‘ ) = TRUE

prcd( ‘)‘ ,op ) = undef
```

```
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 Postfi

```
prcd( ‘(‘ , op) = FALSE

prcd( op, ‘(‘ ) = FALSE

prcd( op, ‘)‘ ) = TRUE

prcd( ‘)‘ ,op ) = undef
```

```
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 Postfi

prcd(‘(‘ , op) = FALSE

prcd(op, ‘(‘) = FALSE

prcd(op, ‘)‘) = TRUE

prcd(‘)‘ ,op) = undef

```
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 – Rules

- 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
- 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

Conversion of Infix Expression to Postfix – Practice

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

symb	Postfix string	opstk

Conversion of Infix Expression to Postfix – Practice

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

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+$	

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

Evaluating a Postfix Expression

```
opndstk = the empty stack
/* 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)
      push(opndstk, symb)
   else {
      /* symb is an operator */
      opnd2 = pop(opndstk);
      opnd1 = pop(opndstk);
      value = result of applying symb
              to opnd1 and opnd2;
      push(opndstk, value);
   } /* end else */
} /* end while */
return (pop(opndstk));
```

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

Evaluating a Postfix Expression

```
opndstk = the empty stack
/* 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)
      push(opndstk, symb)
   else {
      /* symb is an operator */
      opnd2 = pop(opndstk);
      opnd1 = pop(opndstk);
      value = result of applying symb
              to opnd1 and opnd2;
      push(opndstk, value);
   } /* end else */
} /* end while */
return (pop(opndstk));
```

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

symb	opnd1	opnd2	value	opndstk
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, 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)	?	?

Any Question So Far?

Use of Stack in Function Calls (1)

- When a function begins execution an activation record is created to store the current execution environment for that function

- Activation record contains all the necessary information about a function call, including
  - Parameters passed by the caller function
  - Local variables
  - Content of the registers
  - (Callee) Function’s return value(s)
  - Return address of the caller function
    - ➢ Address of instruction following the function call

Use of Stack in Function Calls (2)

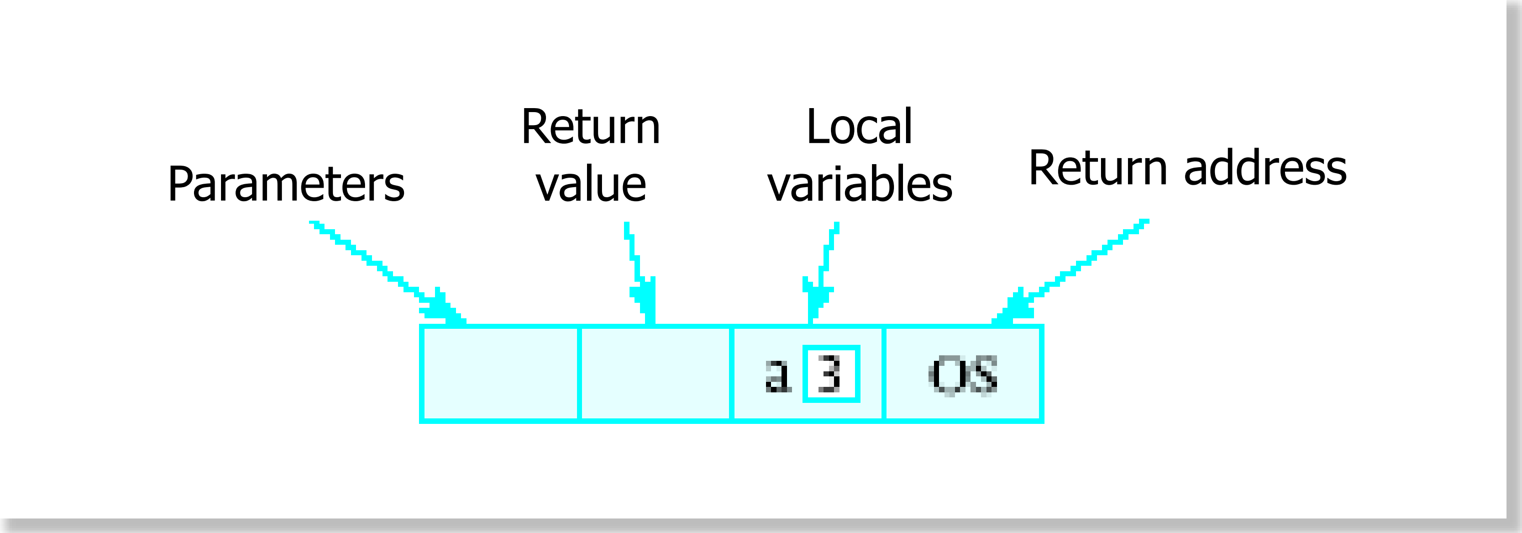
- Each invocation of a function has its own activation record
- Recursive/Multiple calls to the functions require several activation records to exist simultaneously
- A function returns only after all functions it calls have returned Last In First Out (LIFO) behavior
- A program/OS keeps track of all the functions that have been called using run-time stack

Runtime Stack Example (1)

```
void main(){
   int a=3;
   f1(a); // statement A
   cout << endl;
}

void f1(int x){
   cout << f2(x+1); // statement B
}

int f2(int p){
   int q=f3(p/2); // statement C
   return 2*q;
}

int f3(int n){
   return n*n+1;
}
```

Runtime Stack

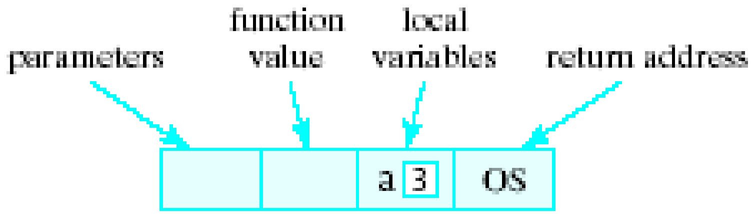
- When a function is called …
  - Copy of activation record pushed onto run-time stack
  - Arguments copied into parameter spaces
  - Control transferred to starting address of body of function



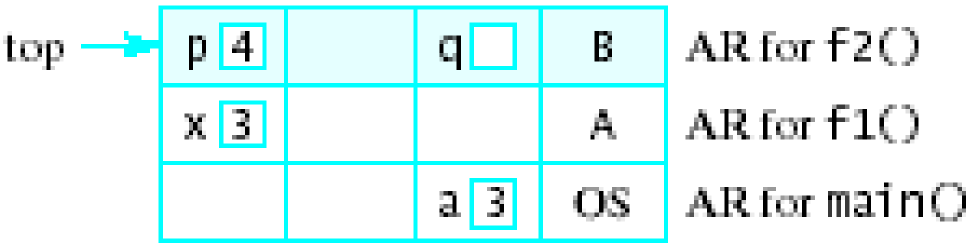
OS denotes that when execution of main() is completed, it returns to the operating system

Runtime Stack Example (2)

```
void main(){
   int a=3;
   f1(a); // statement A
   cout << endl;
}

void f1(int x){
   cout << f2(x+1); // statement B
}

int f2(int p){
   int q=f3(p/2); // statement C
   return 2*q;
}

int f3(int n){
   return n*n+1;
}
```



Function call f2 (x + 1)


top
p 4
q
B
AR for f2()
x 3
A
AR for f1()
a 3
OS
AR for main()