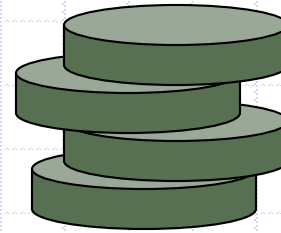
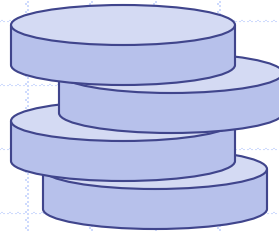
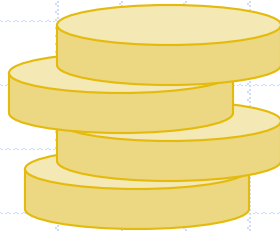
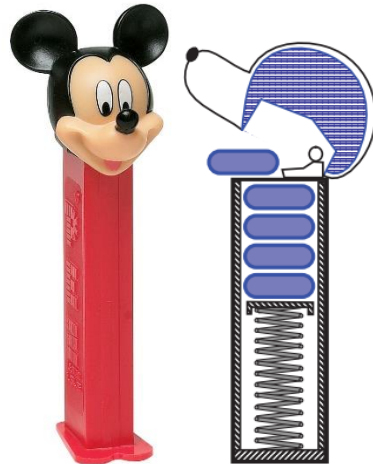


Stacks



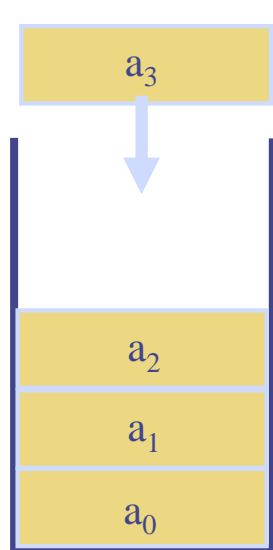
Stack

- ❑ What is a stack?
 - An ordered list where insertions and deletions occur at one end called the top.
 - Also known as last-in-first-out (LIFO) list.
- ❑ Examples

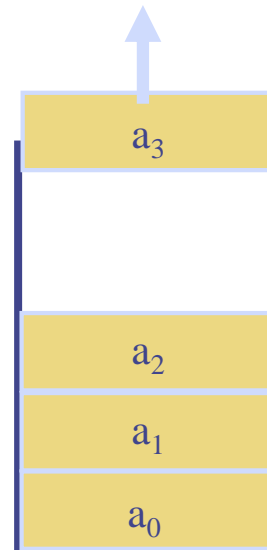


Stack Representation

- Given a stack $S = (a_0, \dots, a_{n-1})$, a_0 is the bottom element, a_{n-1} is the top element, and a_i is on top of element a_{i-1} , $0 < i < n$.



Push (Add)



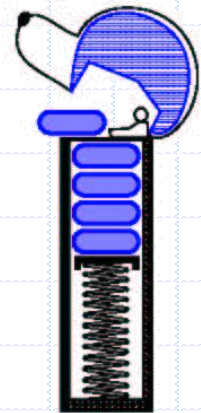
Pop (Delete)

Abstract Data Type (ADT)

- ADT (abstract data type) is an abstraction of a data structure which specifies:
 - Data stored
 - Operations on the data
 - Error conditions associated with operations
- Example: ADT modeling a simple stock trading system
 - The data stored are buy/sell orders
 - The operations supported are
 - ◆ order **buy**(stock, shares, price)
 - ◆ order **sell**(stock, shares, price)
 - ◆ void **cancel**(order)
 - Error conditions:
 - ◆ Buy/sell a nonexistent stock
 - ◆ Cancel a nonexistent order

ADT of Stacks

- ❑ Data
 - Arbitrary objects
- ❑ Operations
 - **push**(object): inserts an element
 - **pop**(): removes the last inserted element
 - object **top**(): returns the last inserted element without removing it
 - integer **size**(): returns the number of elements stored
 - boolean **empty**(): indicates if no elements are stored
- ❑ Exceptions (Error conditions)
 - **pop**() and **top**() cannot be performed if the stack is empty.
 - **push**(object) cannot be performed if the stack is full.



Stack Interface in C++

- ❑ C++ interface corresponding to our Stack ADT
- ❑ Uses an exception class `StackEmpty`
- ❑ Different from the built-in C++ STL class `stack`

```
template <typename E>
class Stack {
public:
    int size() const;
    bool empty() const;
    const E& top() const
        throw(StackEmpty);
    void push(const E& e);
    void pop() throw(StackEmpty);
}
```

No change in
member variables
(for compiler only)

More info about const:

<http://tw.tonytuan.org/2010/03/c-constconst-pointer-pointer-to-const.html>

Applications of Stacks

- ❑ Page-visited history in a web browser
- ❑ Undo sequence in a text editor
- ❑ Chain of function calls in the C++ run-time system
- ❑ Infix to postfix conversion
- ❑ Postfix expression evaluation
- ❑ Parenthesis matching
- ❑ HTML tag matching
- ❑ Maze solution finding

Not in textbook

Not in textbook

Not in textbook

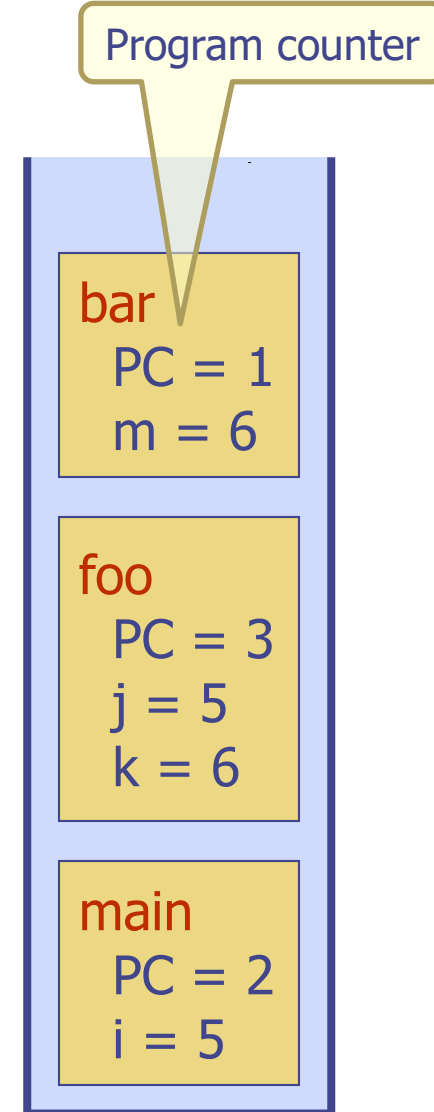
C++ Run-Time Stack

- ❑ The C++ run-time system keeps track of the chain of active functions with a stack
- ❑ When a function is called, the system pushes on the stack a frame containing
 - Local variables and return value
 - Program counter, keeping track of the statement being executed
- ❑ When the function ends, its frame is popped from the stack and control is passed to the function on top of the stack
- ❑ Allows for **recursion**

```
main() {  
    int i = 5;  
    foo(i);  
}
```

```
foo(int j) {  
    int k;  
    k = j+1;  
    bar(k);  
}
```

```
bar(int m) {  
    ...  
}
```

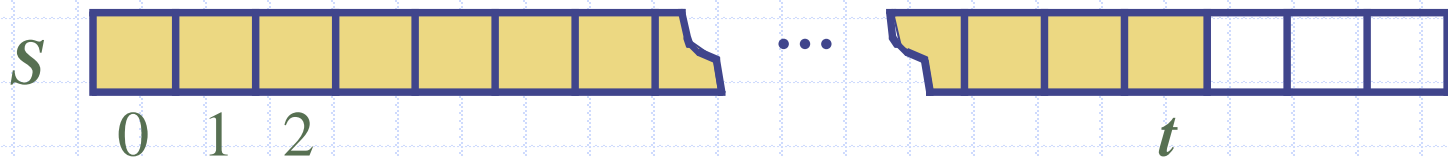


Array-based Stack

- A simple way of implementing the Stack ADT uses an array
- We add elements from left to right
- A variable keeps track of the index of the top element

Algorithm *size()*
return $t + 1$

Algorithm *pop()*
if *empty()* **then**
 throw *StackEmpty*
else
 $t \leftarrow t - 1$
 return $S[t + 1]$



Array-based Stack (cont.)

- ❑ The array storing the stack elements may become full
- ❑ A push operation will then throw a `StackFull` exception

```
Algorithm push(obj)  
  if  $t = S.size() - 1$  then  
    throw StackFull  
  else  
     $t \leftarrow t + 1$   
     $S[t] \leftarrow obj$ 
```



Performance and Limitations

□ Performance

- Let n be the number of elements in the stack
- The space used is $O(n)$
- Each operation runs in time $O(1)$

□ Limitations

- The maximum size of the stack must be defined a priori and cannot be changed
- Trying to push a new element into a full stack causes an implementation-specific exception

Array-based Stack in C++

```
template <typename E>
class ArrayStack {
private:
    E* S; // array holding the stack
    int cap; // capacity
    int t; // index of top element
public:
    // constructor given capacity
    ArrayStack( int c ) :
        S(new E[c]), cap(c), t(-1) { }
```

```
void pop() {
    if (empty()) throw StackEmpty
        ("Pop from empty stack");
    t--;
}
void push(const E& e) {
    if (size() == cap) throw
        StackFull("Push to full stack");
    S[++ t] = e;
}
... (other methods of Stack interface)
```

Example use in C++

```
ArrayStack<int> A;  
A.push(7);  
A.push(13);  
cout << A.top() << endl; A.pop();  
A.push(9);  
cout << A.top() << endl;  
cout << A.top() << endl; A.pop();  
ArrayStack<string> B(10);  
B.push("Bob");  
B.push("Alice");  
cout << B.top() << endl; B.pop();  
B.push("Eve");
```

```
// A = [ ], size = 0  
// A = [7*], size = 1  
// A = [7, 13*], size = 2  
// A = [7*], outputs: 13  
// A = [7, 9*], size = 2  
// A = [7, 9*], outputs: 9  
// A = [7*], outputs: 9  
// B = [ ], size = 0  
// B = [Bob*], size = 1  
// B = [Bob, Alice*], size = 2  
// B = [Bob*], outputs: Alice  
// B = [Bob, Eve*], size = 2
```

* indicates top

Parentheses Matching

- Each “(”, “{”, or “[” must be paired with a matching “)”, “}”, or “]”
 - correct: ()(()){([())}
 - correct: (())(()){([())}}
 - incorrect: (()))}{([())}
 - incorrect: ({ []})}
 - incorrect: (([])

Parentheses Matching Algorithm

Algorithm ParenMatch(X, n):

Input: An array X of n tokens, each of which is either a grouping symbol, a variable, an arithmetic operator, or a number

Output: **true** if and only if all the grouping symbols in X match

Let S be an empty stack

for $i=0$ to $n-1$ **do**

if $X[i]$ is an opening grouping symbol **then**

$S.push(X[i])$

else if $X[i]$ is a closing grouping symbol **then**

if $S.empty()$ **then**

return false {nothing to match with}

if $S.pop()$ does not match the type of $X[i]$ **then**

return false {wrong type}

if $S.empty()$ **then**

return true {every symbol matched}

else return false {some symbols were never matched}

Expression Evaluation


- Expressions are converted into postfix notation for evaluation

- Infix \rightarrow $A/B - C + D * E - A * C$

- Postfix \rightarrow $AB/C - DE * + AC * -$

Operation	Postfix
$T_1 = A / B$	$T_1 C - DE * + AC * -$
$T_2 = T_1 - C$	$T_2 DE * + AC * -$
$T_3 = D * E$	$T_2 T_3 + AC * -$
$T_4 = T_2 + T_3$	$T_4 AC * -$
$T_5 = A * C$	$T_4 T_5 -$
$T_6 = T_4 - T_5$	T_6

Postfix Notation

A yellow speech bubble with a black outline and a tail pointing towards the top right. Inside the bubble, the word "Quiz!" is written in red text.

- ❑ Advantages of postfix notation
 - No need to use parentheses
 - No need to consider precedence of operators
- ❑ Two questions
 - How to change infix notation into postfix notation?
 - How to evaluate an expression in postfix notation?



Infix to Postfix Conversion by Hand

- Three-pass algorithm:

(1) Fully parenthesize expression

$a / b - c + d * e - a * c \rightarrow$

$(((((a / b) - c) + (d * e)) - a * c))$

(2) All operators replace their corresponding right parentheses.

$(((((a / b) - c) + (d * e)) - (a * c)))$

(3) Delete all parentheses.

$ab/c-de*+ac*-$

Rules of Conversion

- Two rules
 - Operators are taken out of the stack as long as their precedence is higher than or equal to the precedence of the incoming operator.
 - The left parenthesis is placed in the stack whenever it is found in the expression, but it is unstacked only when its matching right parenthesis is found.



Example: Infix to Postfix

The order of operands in infix and postfix are the same.

$a+b*c \rightarrow abc*+$

Token	Stack			Output
	[0]	[1]	[2]	
a				a
+	+			
b	+			b
*	+	*		
c	+	*		c
				*+



Example: Infix to Postfix

$a*(b+c)/d \rightarrow abc+*d/$

Quiz!

Token	Stack			Output
	[0]	[1]	[2]	
a				a
*	*			
(*	(
b	*	(b
+	*	(+	
c	*	(+	c
)	*			+
/	/			*
d	/			d
				/

More Examples

Quiz!

Infix	Postfix
$2+3*4$	$234*+$
$a*b+5$	$ab*5+$
$(1+2)*7$	$12+7*$
$a*b/c$	$ab*c/$
$(a/(b-c+d))*(e-a)*c$	$abc-d+/ea-*c*$
$a/b-c+d*e-a*c$	$ab/c-de*+ac*-$

Walk-through Examples

Quiz!

- $a/b-c+d*e-a*c \rightarrow ab/c-de*+ac*-$
- $(a/(b-c+d))*(e-a)*c \rightarrow abc-d+/ea-*c*$

More Examples

Quiz!

$$\square (a+b)*(c-d)/(e+f) \rightarrow ab+cd-*ef+ /$$

$$\square (a+b)*(c-d)/((e-f)*(g+h)) \rightarrow ab+cd-*ef-gh+*/$$



Evaluating postfix expressions

$O(n)$

- Evaluation process
 - Make a single left-to-right scan of the expression.
 - Place the operands on a stack until an operator is found.
 - Remove, from the stack, the correct numbers of operands for the operator, perform the operation, and place the result back on the stack.



Example of Evaluating postfix expressions

Example: $6\ 2\ /\ 3\ -\ 4\ 2\ *\ +$

Token	Stack		
	[0]	[1]	[2]
6	6		
2	6	2	
/	6/2		
3	6/2	3	
-	6/2-3		
4	6/2-3	4	
2	6/2-3	4	2
*	6/2-3	4*2	
+	6/2-3+4*2		

Walk-through Examples

- $a/b-c+d*e-a*c \rightarrow ab/c-de^*+ac^*-$
- $(a/(b-c+d))^*(e-a)^*c \rightarrow abc-d+/ea-*c^*$



Animation

- Animation
 - Infix to postfix conversion
 - Postfix evaluation