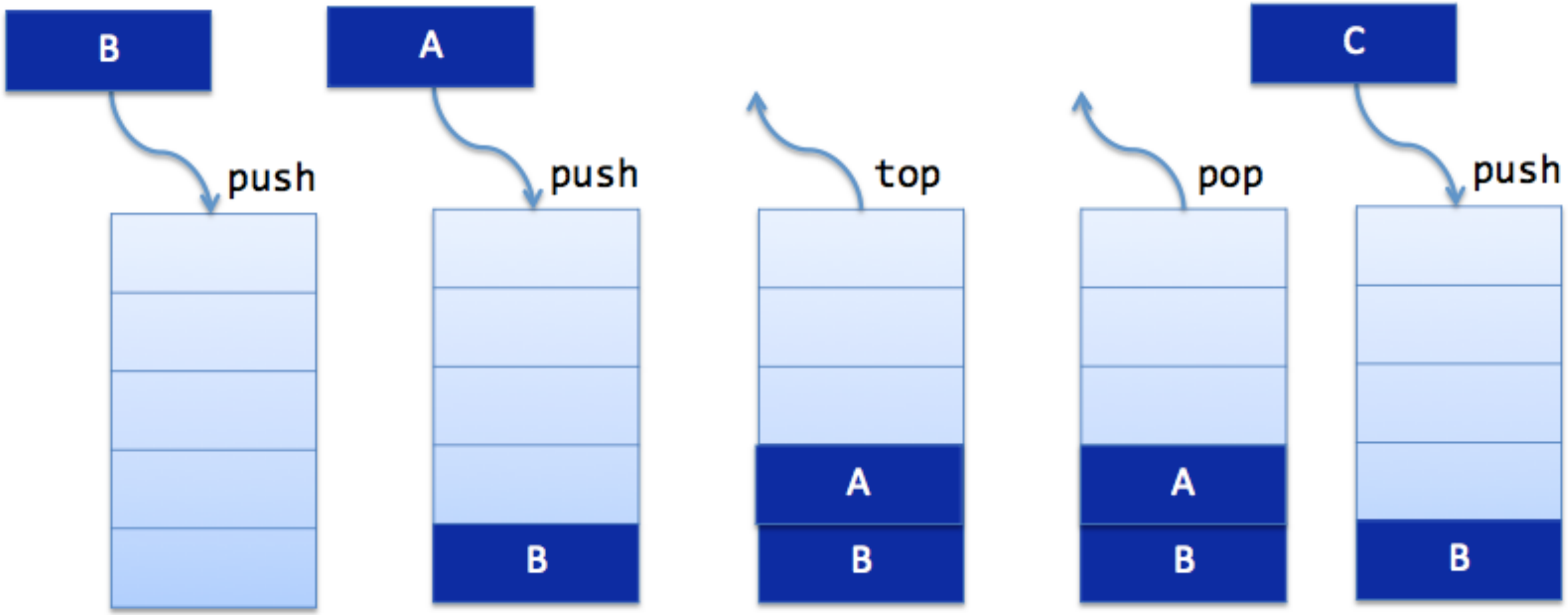
Stacks

CSCI-2270

Elizabeth Boese

Stack



Stack

- LIFO
- Like
  - Dishes
  - Card game discard pile
- Use Cases
  - history for undo operations
  - track function calls and returns
  - verify expression validity (balanced brackets...)
  - evaluate postfix expressions

Stack

Operations

- isEmpty
- isFull *(esp for array-based implementations)*
- push
- pop
- peek (or top)

Stack

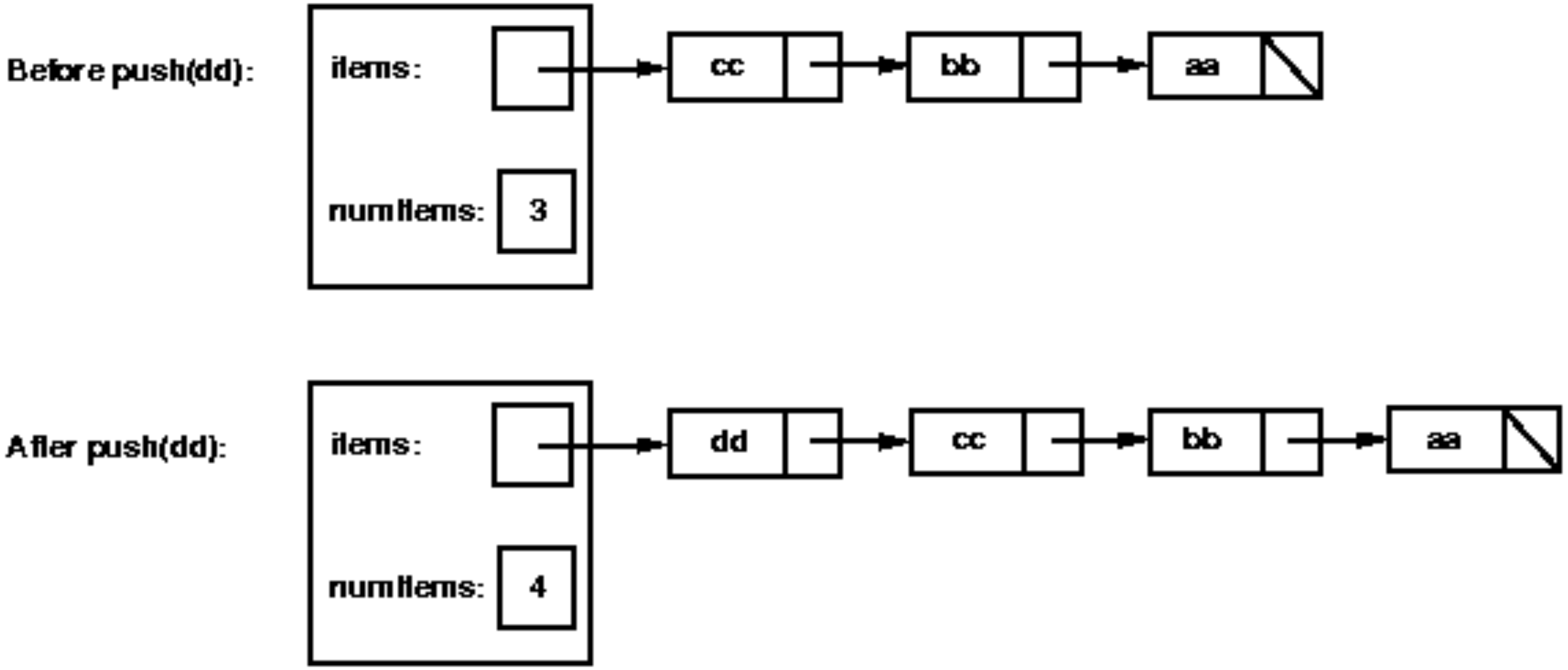
Implementations

- Linked-List
- Array

Stack

Linked List

– push

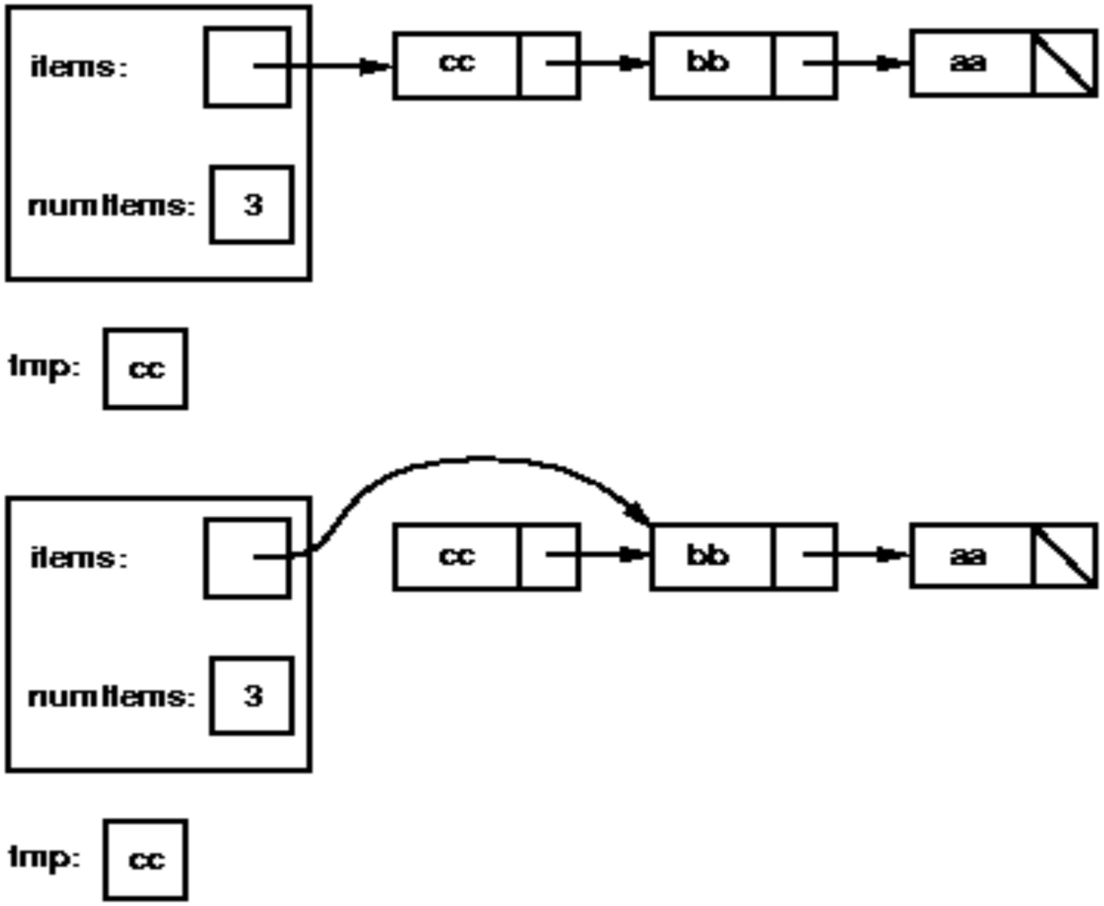


http://pages.cs.wisc.edu/~vernon/cs367/notes/5.STACKS-AND-QUEUES.html

Stack

Linked List

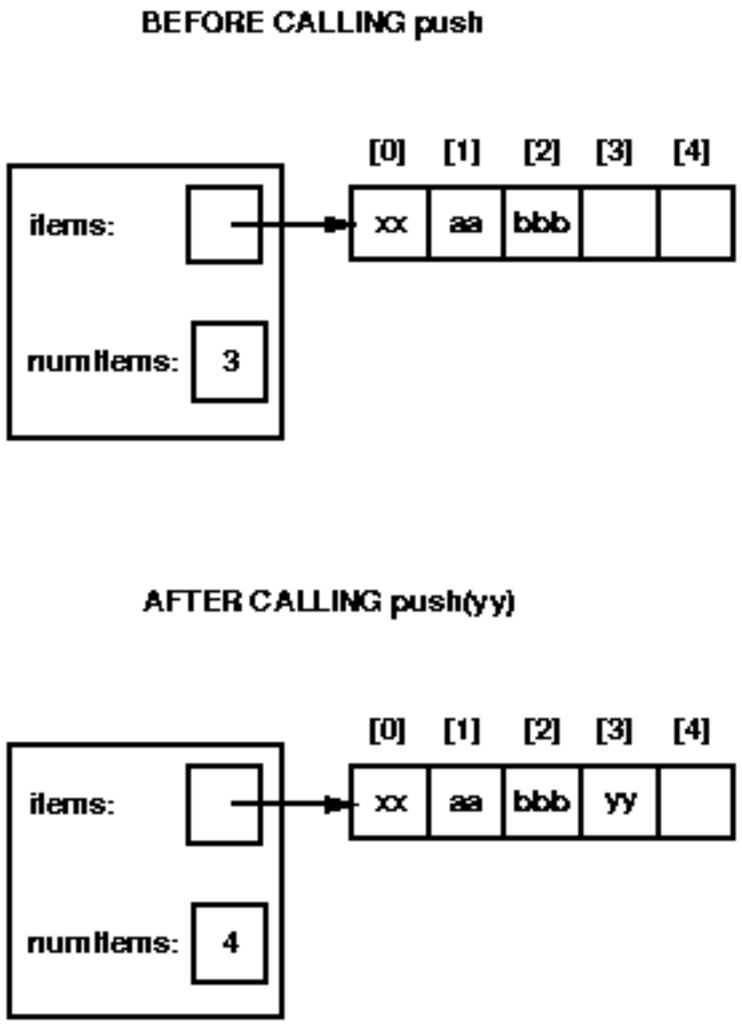
– pop



http://pages.cs.wisc.edu/~vernon/cs367/notes/5.STACKS-AND-QUEUES.html

Stack

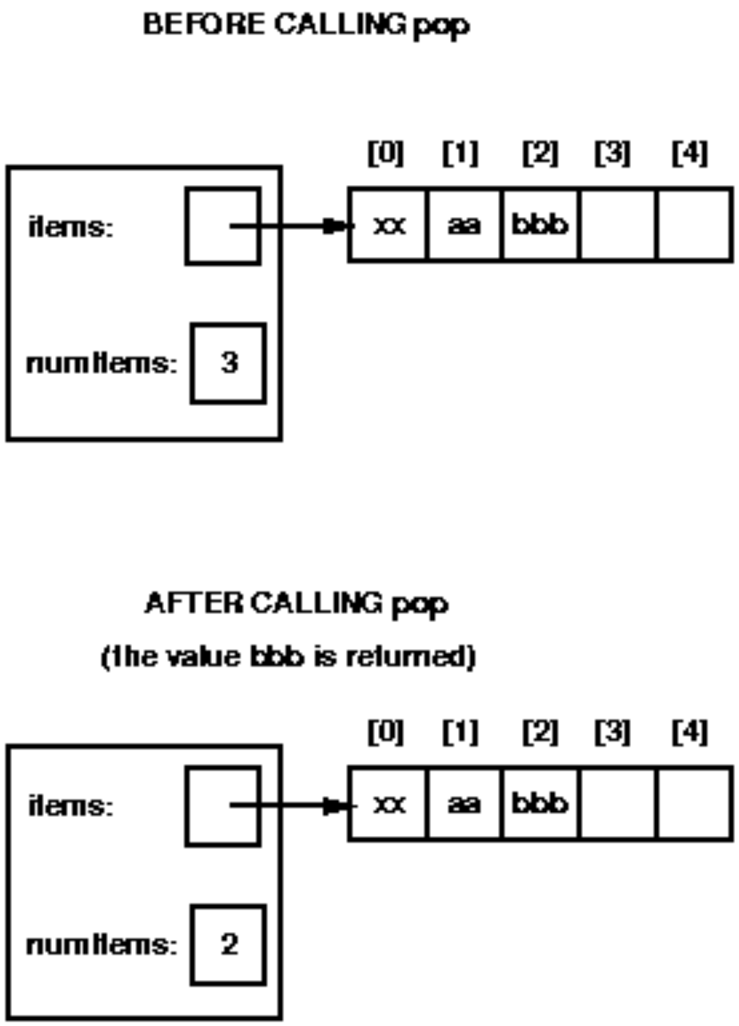
- Array
  - push



http://pages.cs.wisc.edu/~vernon/cs367/notes/5.STACKS-AND-QUEUES.html

Stack

- Array
  - pop



http://pages.cs.wisc.edu/~vernon/cs367/notes/5.STACKS-AND-QUEUES.html

Stack

- Linked-List
  - Fill in the following table, using Big-O notation to give the worst and average-case times for each of the stack methods for a stack of size N.

OPERATION	WORST-CASE TIME	AVERAGE-CASE TIME
isEmpty		
push		
pop		
peek		

Stack

- Array
  - Fill in the following table, using Big-O notation to give the worst and average-case times for each of the stack methods for a stack of size N.

OPERATION	WORST-CASE TIME	AVERAGE-CASE TIME
isEmpty		
push		
pop		
peek		

Stack

Parsing HTML

STACK EXAMPLE

Application: Parsing

Most parsing uses stacks

Examples includes:
- Matching tags in XHTML
- In C++, matching
  - parentheses `( . . . )`
  - brackets, and `[ . . . ]`
  - braces `{ . . . }`

Parsing XHTML

The first example will demonstrate parsing XHTML

We will show how stacks may be used to parse an XHTML document

You will use XHTML (and more generally XML and other markup languages) in the workplace

Parsing XHTML

A *markup language* is a means of annotating a document to given context to the text

– The annotations give information about the structure or presentation of the text

The best known example is HTML, or HyperText Markup Language

– We will look at XHTML

Parsing XHTML

XHTML is made of nested

- *opening tags*, e.g., `<some_identifier>`, and
- matching *closing tags*, e.g., `</some_identifier>`

```
<html>
  <head><title>Hello</title></head>
  <body><p>This appears in the <i>browser</
  i>.</p></body>
</html>
```

Parsing XHTML

Nesting indicates that any closing tag must match the most <u>recent</u> opening tag

Strategy for parsing XHTML:

- read though the XHTML linearly
- place the opening tags in a stack
- when a closing tag is encountered, check that it matches what is on top of the stack and

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

`<html>`			

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

`<html>`	`<head>`		

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

`<html>`	`<head>`	`<title>`	

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

`<html>`	`<head>`	`<title>`	

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

<html>	<head>		

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

`<html>`	`<body>`		

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

`<html>`	`<body>`	`<p>`	

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

`<html>`	`<body>`	`<p>`	`<i>`

Parsing XHTML

3.2.5.1

```
<html>
   <head><title>Hello</title></
   head>
   <body><p>This appears in the
   <i>browser</i>.</p></body>
</html>
```

`<html>`	`<body>`	`<p>`	`<i>`

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

<html>	<body>	<p>	

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

`<html>`	`<body>`		

Parsing XHTML

```
<html>
  <head><title>Hello</title></
  head>
  <body><p>This appears in the
  <i>browser</i>.</p></body>
</html>
```

`<html>`			

Parsing XHTML

We are finished parsing, and the stack is empty

Possible errors:

- a closing tag which does not match the opening tag on top of the stack
- a closing tag when the stack is empty
- the stack is not empty at the end of the document

HTML

Old HTML required neither closing tags nor nesting

```
<html>
  <head><title>Hello</title></head>
  <body><p>This is a list of topics:
  <ol>                        <!-- para ends with start of list -->
    <li><i>veni                              <!-- implied </li> -->
    <li>vidi                              <!-- italics continues -->
    <li>vici</i>
  </ol>                 <!-- end-of-file implies </body></html> -->
```

Parsers were therefore specific to HTML

– Results: ambiguities and inconsistencies

XML

XHTML is an implementation of XML

XML defines a class of general-purpose *eXtensible Markup Languages* designed for sharing information between systems

The same rules apply for any flavour of XML:

– opening and closing tags must match and be nested

Parsing C++

The next example shows how stacks may be used in parsing C++

Those taking ECE 351 *Compilers* will use this

For other students, it should help understand, in part:

- how a compiler works, and
- why programming languages have the structure they do

Parsing C++

Like opening and closing tags, C++ parentheses, brackets, and braces must be similarly nested:

```
void initialize( int *array, int n ) {
    for ( int i = 0; i < n; ++i ) {
        array[i] = 0;
    }
}
```

(AN UNMATCHED LEFT PARENTHESIS CREATES AN UNRESOLVED TENSION THAT WILL STAY WITH YOU ALL DAY.

http://xkcd.com/859/

Parsing C++

For C++, the errors are similar to that for XHTML, however:

- many XHTML parsers usually attempt to “correct” errors (e.g., insert missing tags)
- C++ compilers will simply issue a parse error:

```
{eceunix:1} cat example1.cpp
#include <vector>
int main() {
        std::vector<int> v(100];
        return 0;
}

{eceunix:2} g++ example1.cpp
example1.cpp: In function 'int main()':
example1.cpp:3: error: expected ')' before ']'
  token
```

QUESTIONS TO PONDER

Questions

1. Execute this code:

```
stack<int> s;
 s.push(1);
 s.push(2);
 s.push(3);
 s.pop( );
```

Suppose that s is represented by a partially filled array.
Draw the state of the private member variables of s after the above code:

```
          _______          __________________________________
     used|       |     data|      |      |      |      |      |
         |_______|         |______|______|______|______|______|
                             [0]    [1]    [2]    [3]    [4]
```

Questions

1. Execute this code with THREE pushes and ONE pop:

```
  stack<int> s;
     s.push(1);
     s.push(2);
     s.push(3);
     cout << s.pop( );
```

Suppose that s is represented by a linked list.

Draw the state of the private member variables of s after the above code:

```
            ________
    head_ptr|        |
            |________|
```

Questions

1. Entries in a stack are "ordered". What is the meaning of this statement?
   A. A collection of stacks can be sorted.
   B. Stack entries may be compared with the '<' operation.
   C. The entries must be stored in a linked list.
   D. There is a first entry, a second entry, and so on.

2. The operation for adding an entry to a stack is traditionally called:
   A. add
   B. append
   C. insert
   D. push

3. The operation for removing an entry from a stack is traditionally called:
   A. delete
   B. peek
   C. pop
   D. remove

Questions

1. Which of the following stack operations could result in stack underflow?
   A. is_empty
   B. pop
   C. push
   D. Two or more of the above answers

2. Which of the following applications may use a stack?
   A. A parentheses balancing program.
   B. Keeping track of local variables at run time.
   C. Syntax analyzer for a compiler.
   D. All of the above.

3. In the linked list implementation of the stack class, where does the push member function place the new entry on the linked list?
   A. At the head
   B. At the tail
   C. After all other entries that are greater than the new entry.
   D. After all other entries that are smaller than the new entry.

Questions

1. Consider the following pseudocode:

```
declare a stack of characters
while ( there are more characters in the word to read )
{
  read a character
  push the character on the stack
}
while ( the stack is not empty )
{
  write the stack's top character to the screen
  pop a character off the stack
}
```

What is written to the screen for the input "carpets"?