FAIRMONT STATE UNIVERSITY

Stacks

COMP 2270 – Data Structures

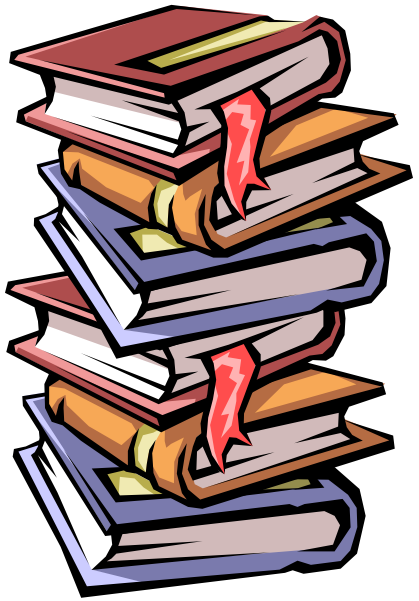
Fall 2014

Dr. Mahmood Hossain

FAIRMONT STATE UNIVERSITY

Example

- What do you need to do if you want to get the third book from the top?



FAIRMONT STATE UNIVERSITY

Example

- During program execution, how does the computer keep track of the function calls?

- How does a compiler evaluate an expression?

FAIRMONT STATE UNIVERSITY

What is a Stack?

- A *stack* is a linear data structure
  - it is a list of homogenous items
  - addition and deletion of items can take place only at one end, called the *top* of the stack
  - also known as *last-in-first-out* (LIFO) structure

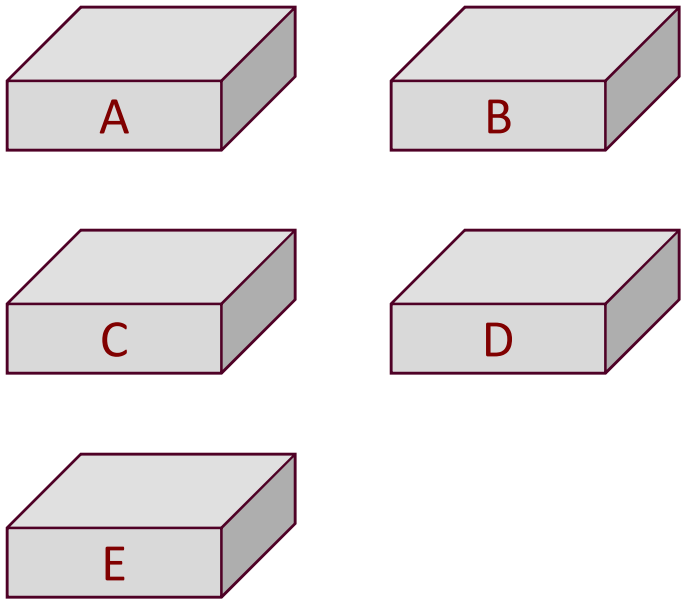
FAIRMONT STATE UNIVERSITY

Basic Operations

- Three basic operations
  - *push*: adding an item at the top of the stack
  - *pop*: removing the item from the top of the stack
  - *stack top*: copying the item at the top of the stack

FAIRMONT STATE UNIVERSITY

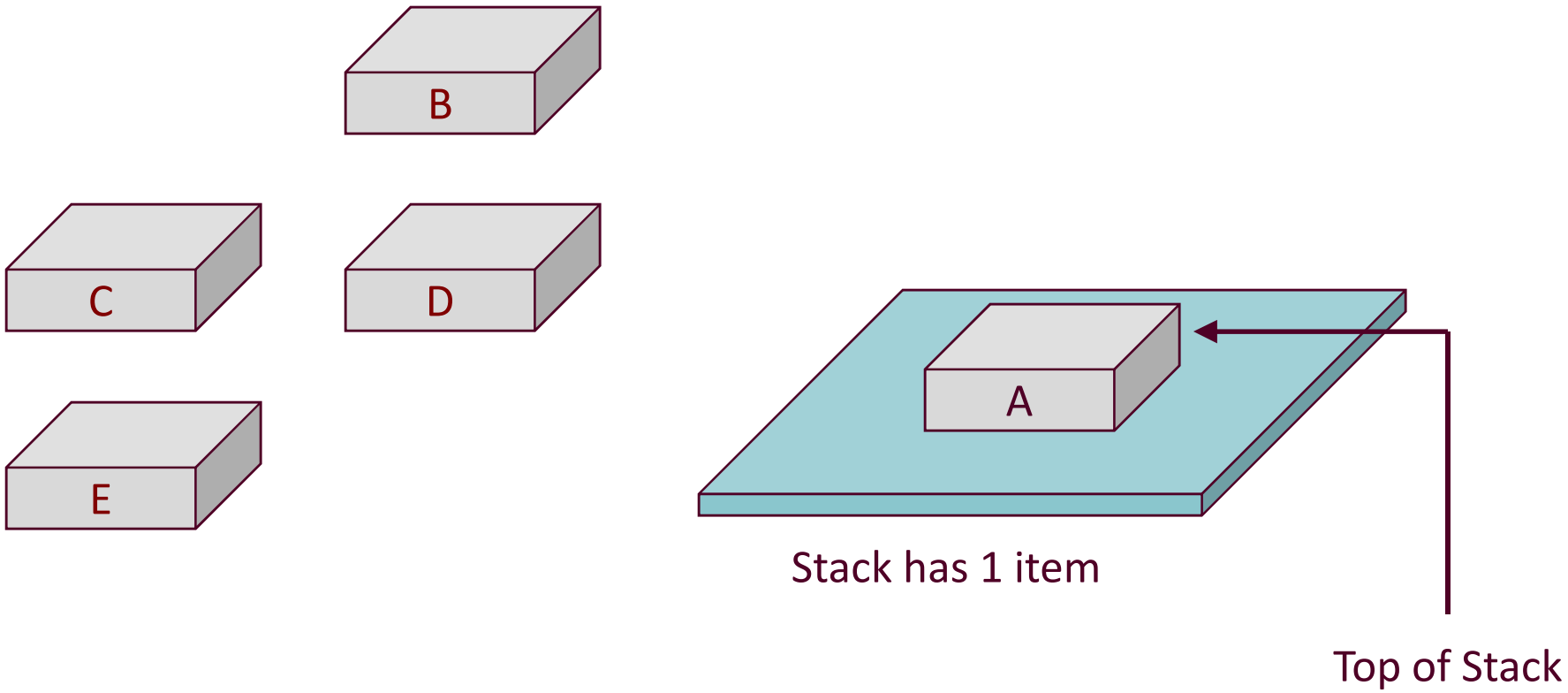
Example - Push



Stack is Empty

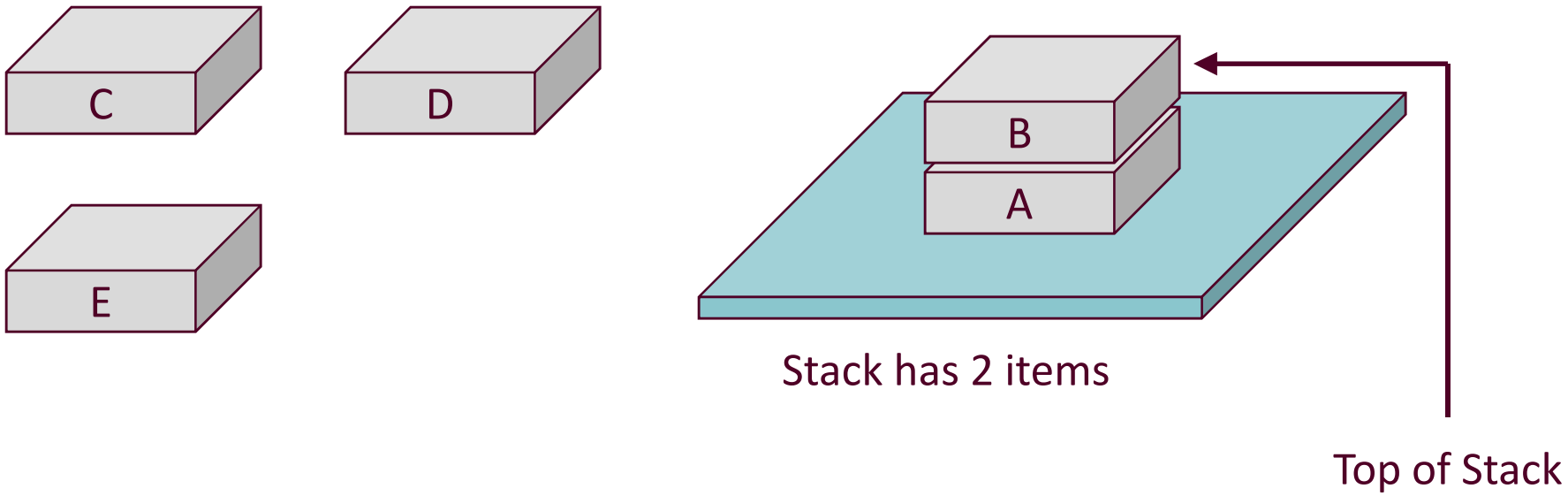
FAIRMONT STATE UNIVERSITY

Example - Push



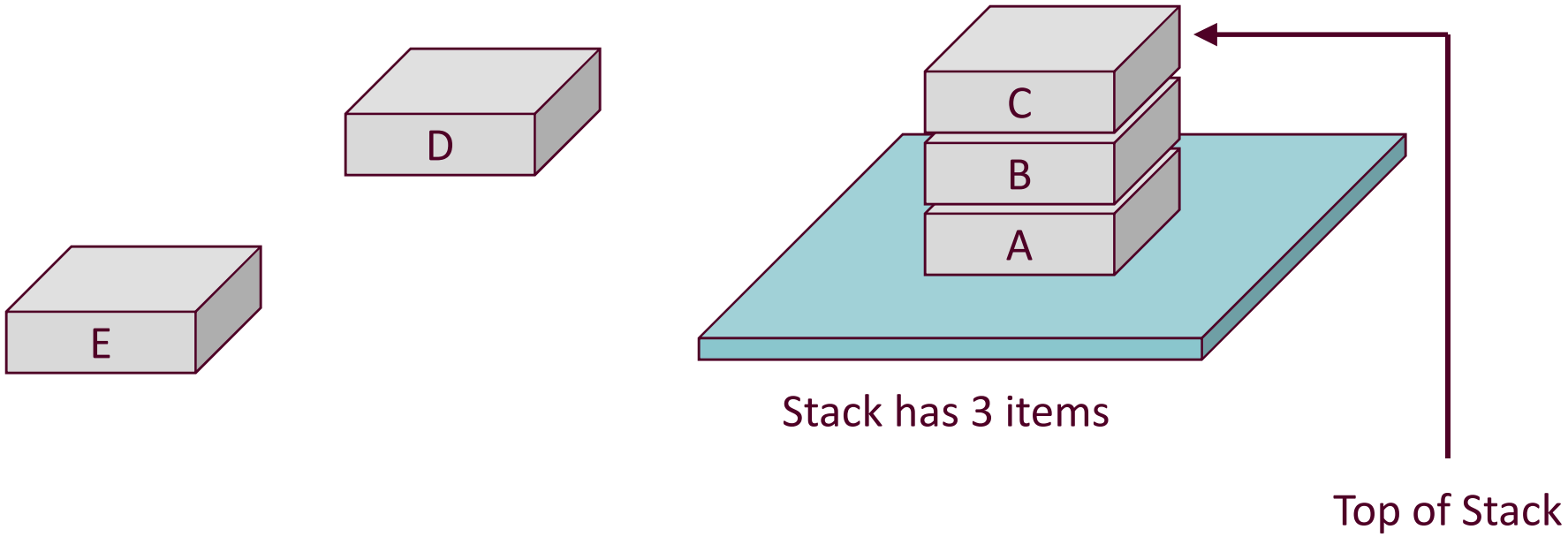
FAIRMONT STATE UNIVERSITY

Example - Push



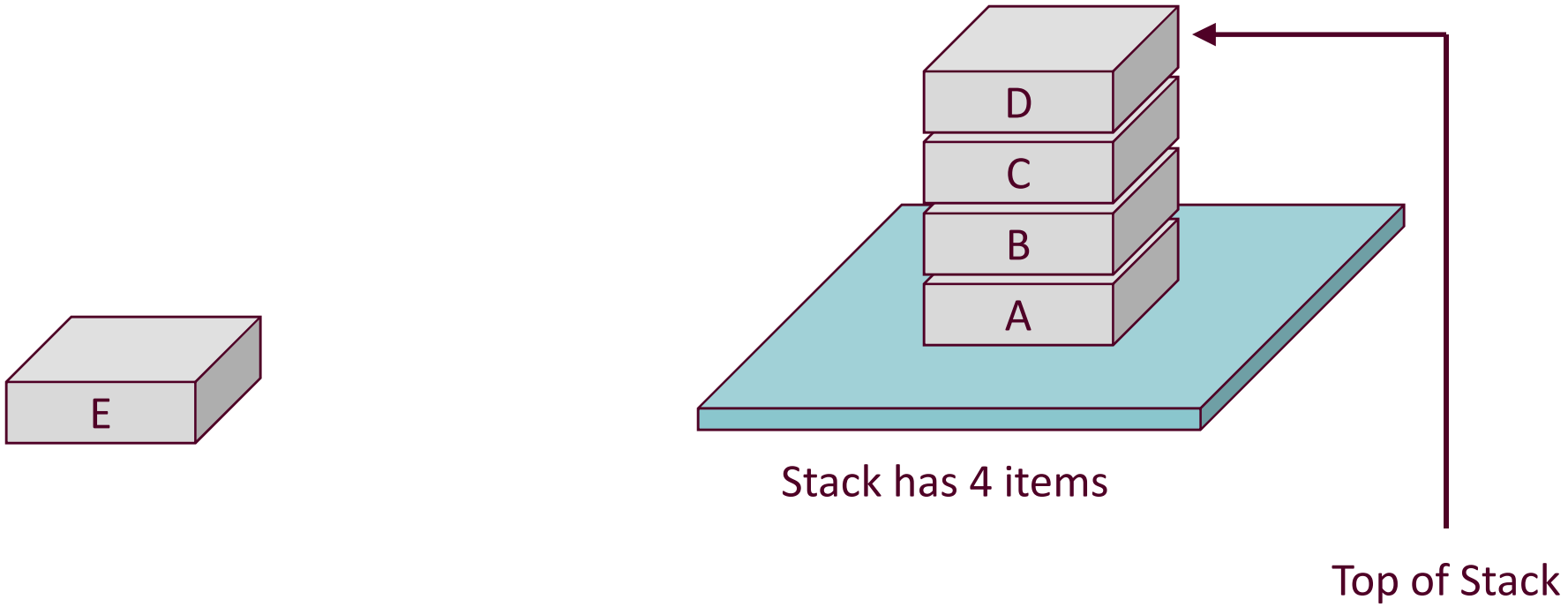
FAIRMONT STATE UNIVERSITY

Example - Push



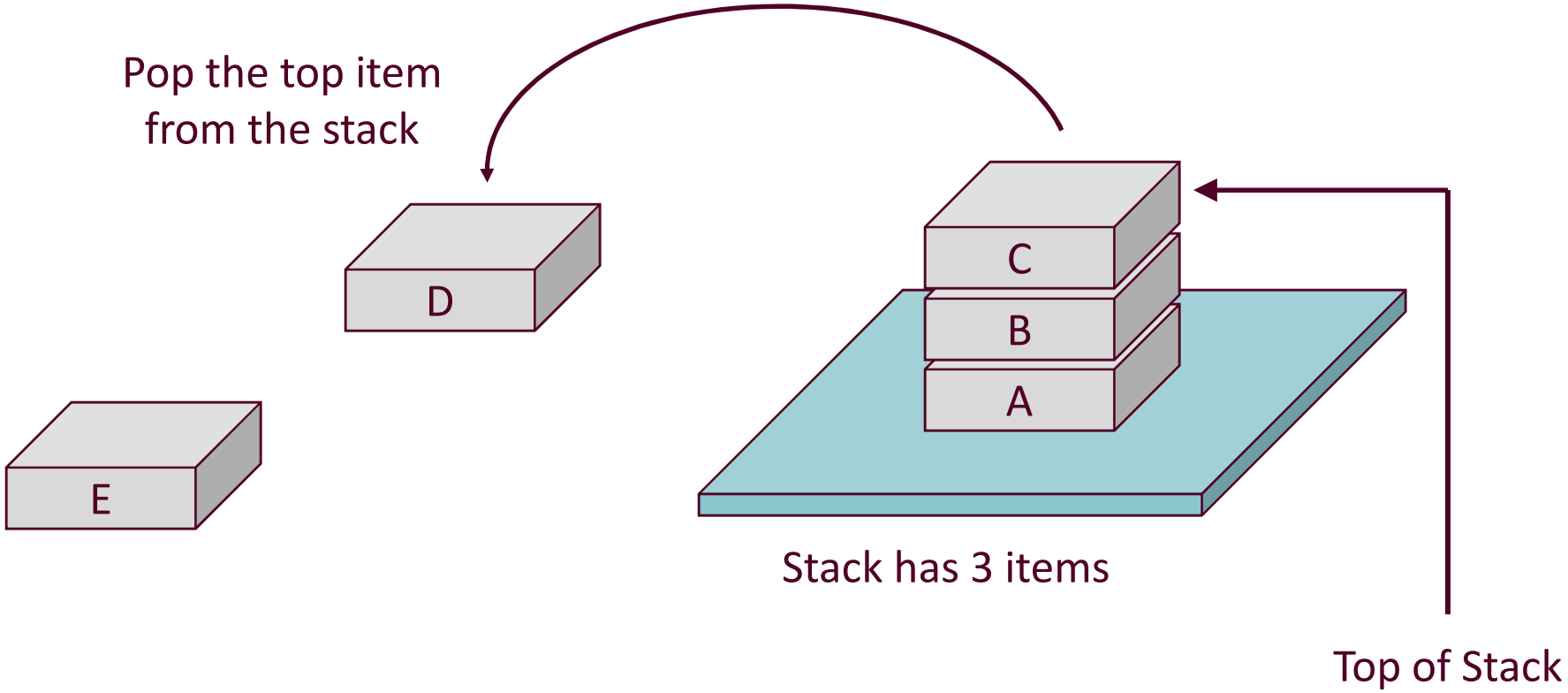
FAIRMONT STATE UNIVERSITY

Example - Push



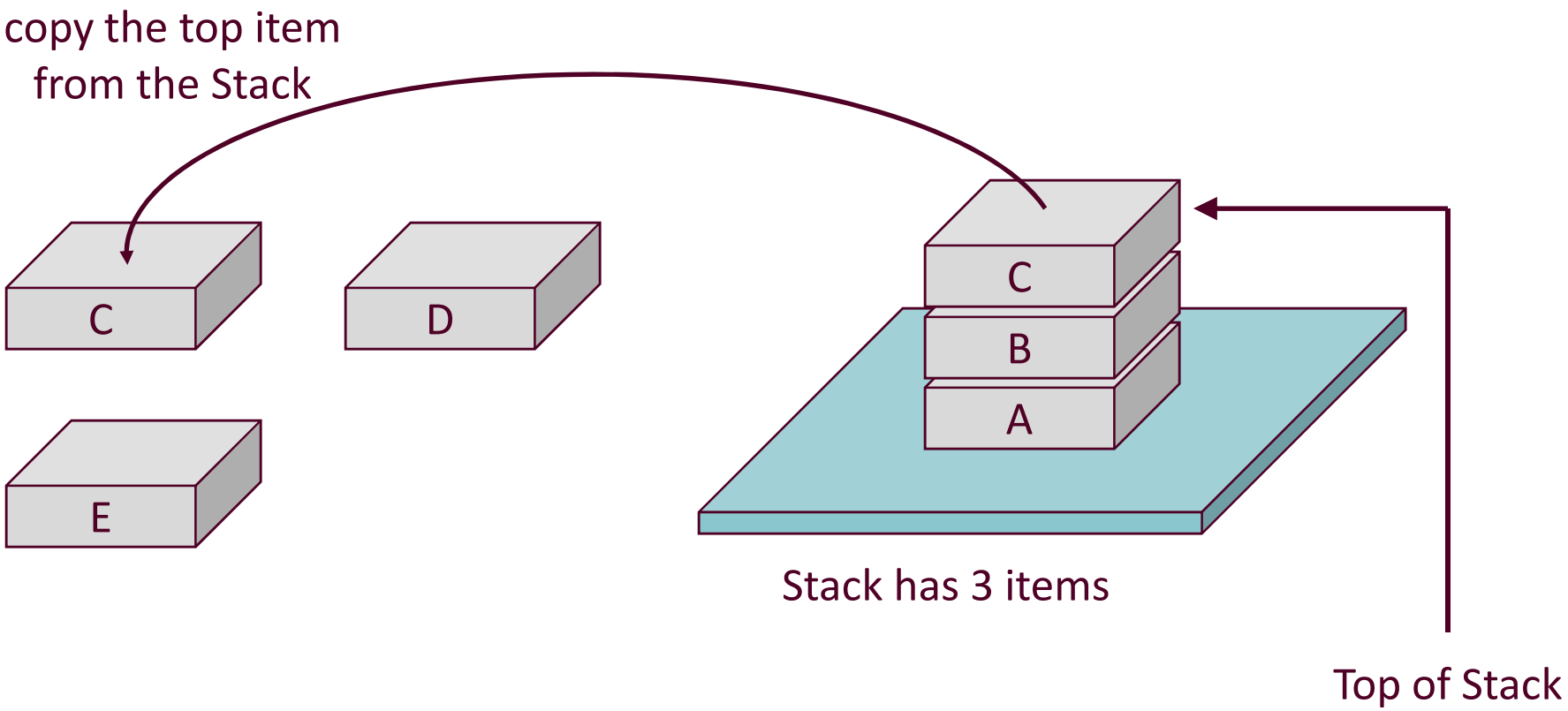
FAIRMONT STATE UNIVERSITY

Example - Pop



FAIRMONT STATE UNIVERSITY

Example – StackTop



FAIRMONT STATE UNIVERSITY

Overflow and Underflow

- Stack overflow
  - occurs when one tries to push an item onto an already full stack
- Stack underflow
  - occurs when one tries to pop/copy an item from an empty stack

FAIRMONT STATE UNIVERSITY

Implementing the Stack as an ADT

- Choose an internal data representation for the items in the stack (e.g., array or linked list)
- Implement the stack operations
  - initialize the stack
  - destroy the stack
  - push into stack
  - pop from stack
  - copy from stack top
  - is stack empty
  - is stack full
  - count # of items

FAIRMONT STATE UNIVERSITY

Implementation of Stacks as Arrays

- Stack items are stored in an array
  - the first item of the stack is stored in the first array position, the second item of the stack in the second array position, and so on
- A variable called *top* should be used to keep track of the top of the stack
  - *top* is typically the index of the top item in the stack
  - when does *top* has a value of -1?
  - what does *top*+1 represent?

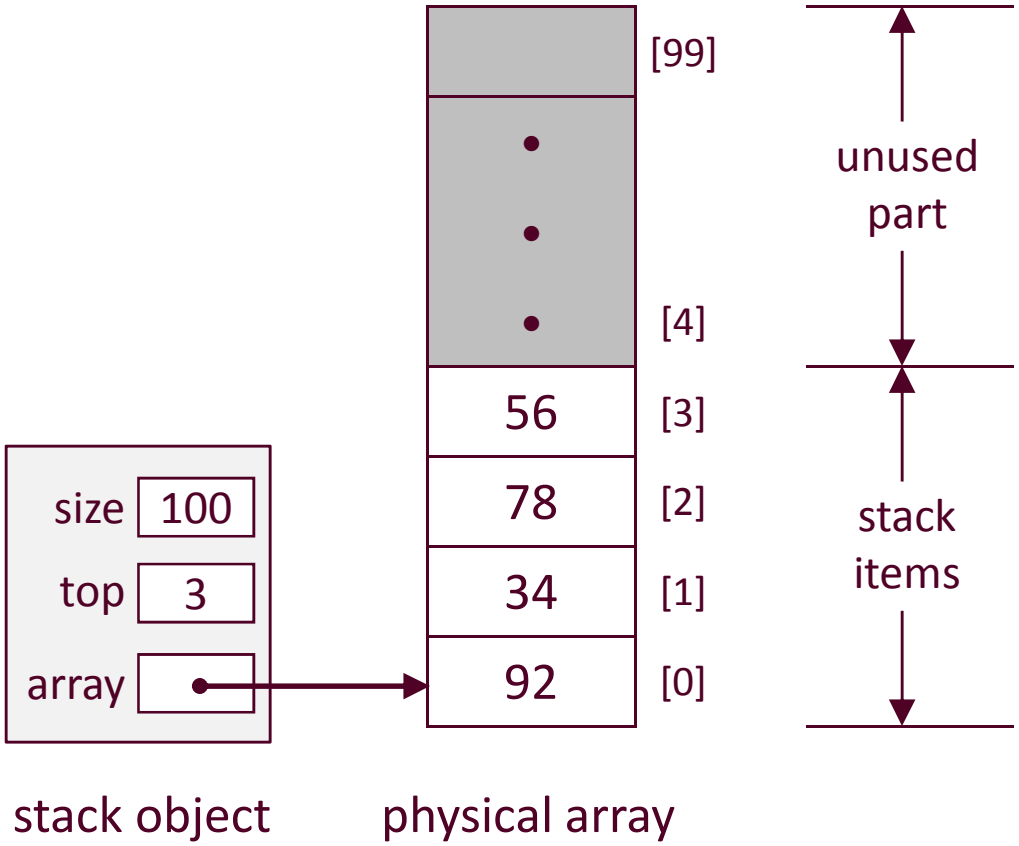
FAIRMONT STATE UNIVERSITY

The **Stack** ADT

```
class Stack
{
   private:
      int* array; // each item is an int
      int size, top;
   public:
      Stack(int s = 100);
      ~Stack();
      bool push(int dataIn);
      bool pop(int& dataOut);
      bool stackTop(int& dataOut);
      bool isEmpty();
      bool isFull();
      int  getCount();
};
```

FAIRMONT STATE UNIVERSITY

A Stack Type Object



FAIRMONT STATE UNIVERSITY

Member Functions

```
// constructor
Stack::Stack(int s)
{



}

// destructor
Stack::~Stack()
{



}
```

FAIRMONT STATE UNIVERSITY

Member Functions

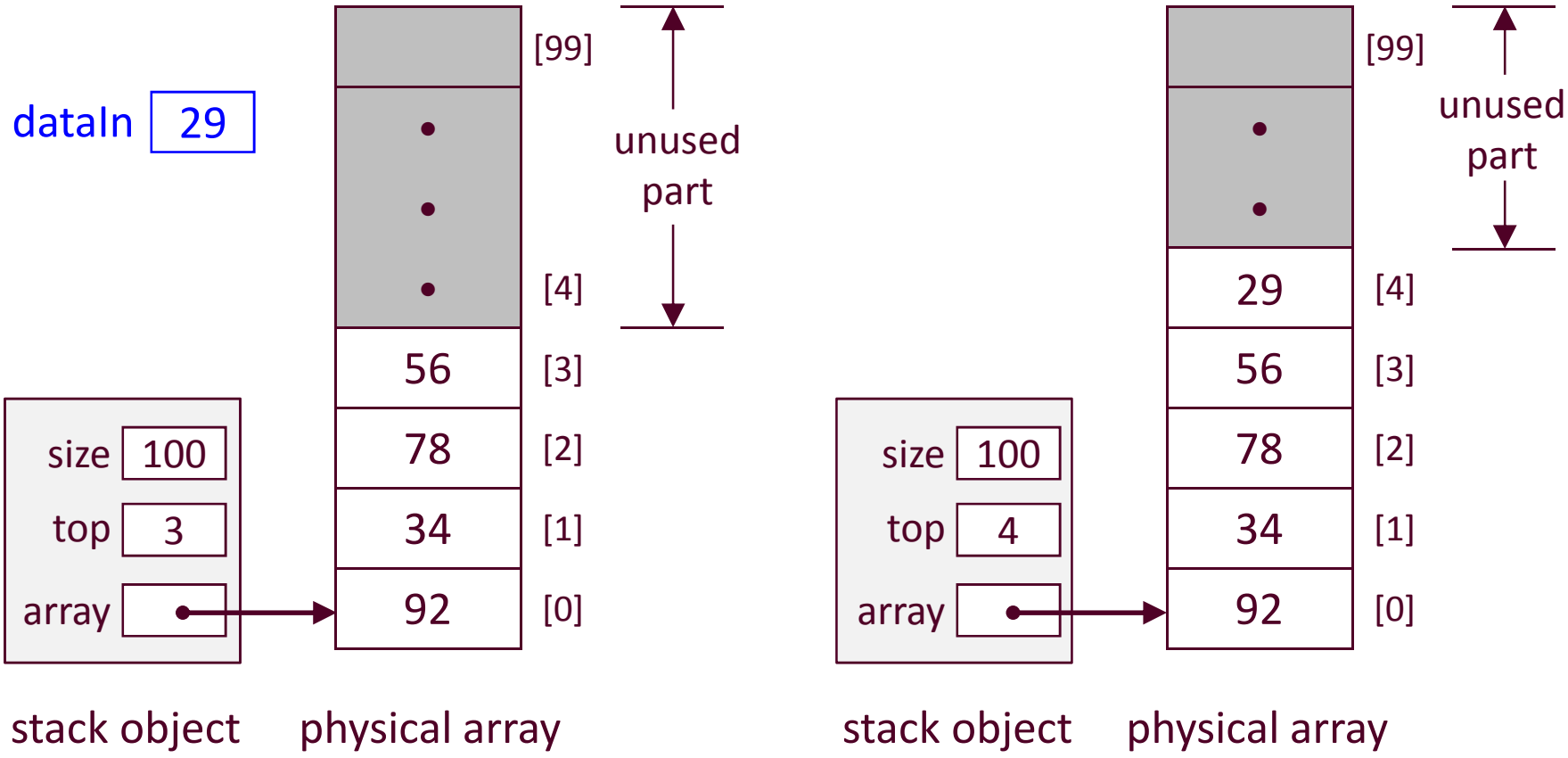
```
bool Stack::push(int dataIn)
{
   // should take care of overflow




}
```

FAIRMONT STATE UNIVERSITY

Example: Push 29



FAIRMONT STATE UNIVERSITY

Member Functions

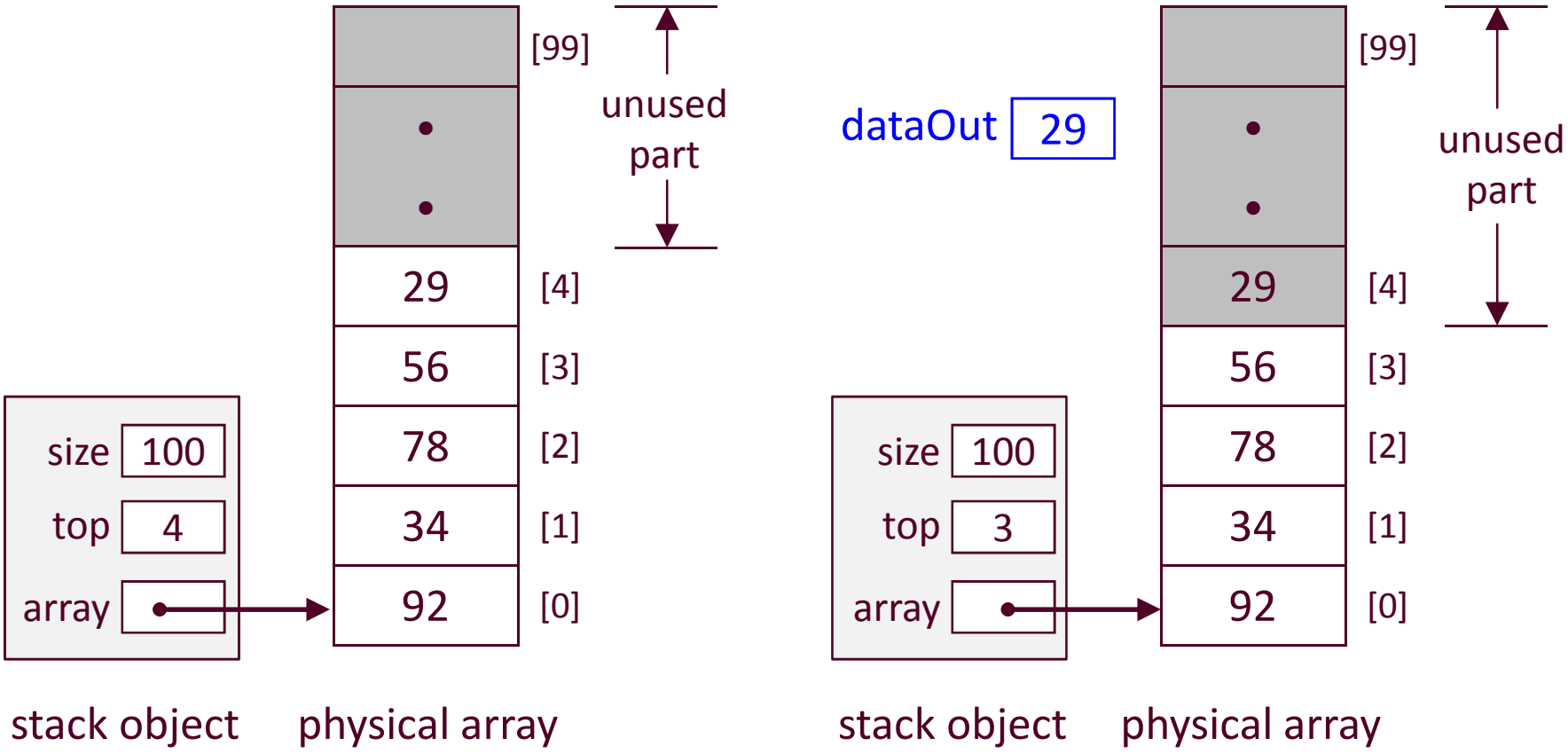
```
bool Stack::pop(int& dataOut)
{
   // should take care of underflow




}
```

FAIRMONT STATE UNIVERSITY

Example: Pop



FAIRMONT STATE UNIVERSITY

Member Functions

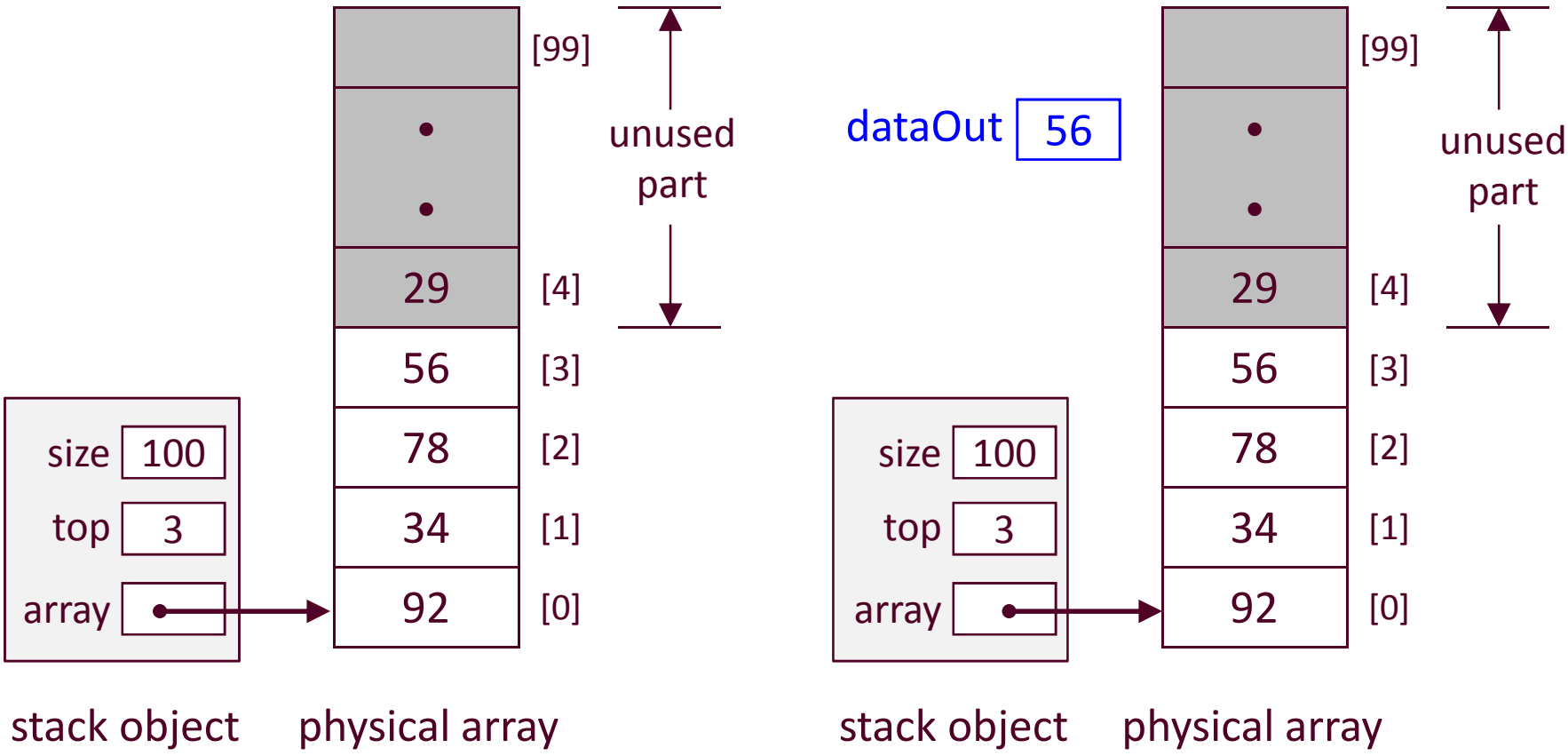
```
bool Stack::stackTop(int& dataOut)
{
   // should take care of underflow




}
```

FAIRMONT STATE UNIVERSITY

Example: StackTop



FAIRMONT STATE UNIVERSITY

Member Functions

```
bool Stack::isEmpty()
{



}

bool Stack::isFull()
{



}
```

FAIRMONT STATE UNIVERSITY

Member Functions

```
int Stack::getCount()
{



}
```

FAIRMONT STATE UNIVERSITY

Assignment

- Write the code that will
  - create a **`Stack`** object called **`myStack`** that can have a maximum of 50 items
  - fill the stack with the items 92, 34, 78, and 56
  - remove and display the top item from stack

FAIRMONT STATE UNIVERSITY

Implementation of Stacks as Linked Lists

- Stack items are stored in a linked list
- In an array representation of a stack
  - *top*+1 indicates the number of stack items
  - *top* indicates the index of the top item of the stack
- In a linked representation
  - *top* gives the address (memory location) of the top item of the stack, i.e., *top* is the head pointer
  - push, pop, and stacktop are done at the beginning of the list

FAIRMONT STATE UNIVERSITY

The **Stack** ADT (Linked Implementation)

```
class Stack
{
   private:
      Node* top;
      int count;
   public:
      Stack();
      ~Stack();
      bool push(int dataIn);
      bool pop(int& dataOut);
      bool stackTop(int& dataOut);
      bool isEmpty();
      bool isFull();
      int  getCount();
};
```

FAIRMONT STATE UNIVERSITY

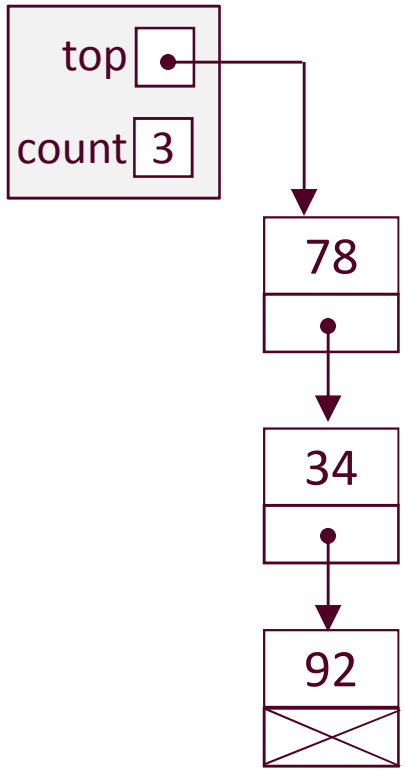
The **Node** Type

```
struct Node
{
   int data;   // item stored at node
   Node *next; // pointer to next node
};
```

FAIRMONT STATE UNIVERSITY

Stack Objects

Empty stack



Non-empty stack

FAIRMONT STATE UNIVERSITY

Member Functions

```
// constructor
Stack::Stack()
{



}

// destructor
Stack::~Stack()
{



}
```

FAIRMONT STATE UNIVERSITY

Member Functions

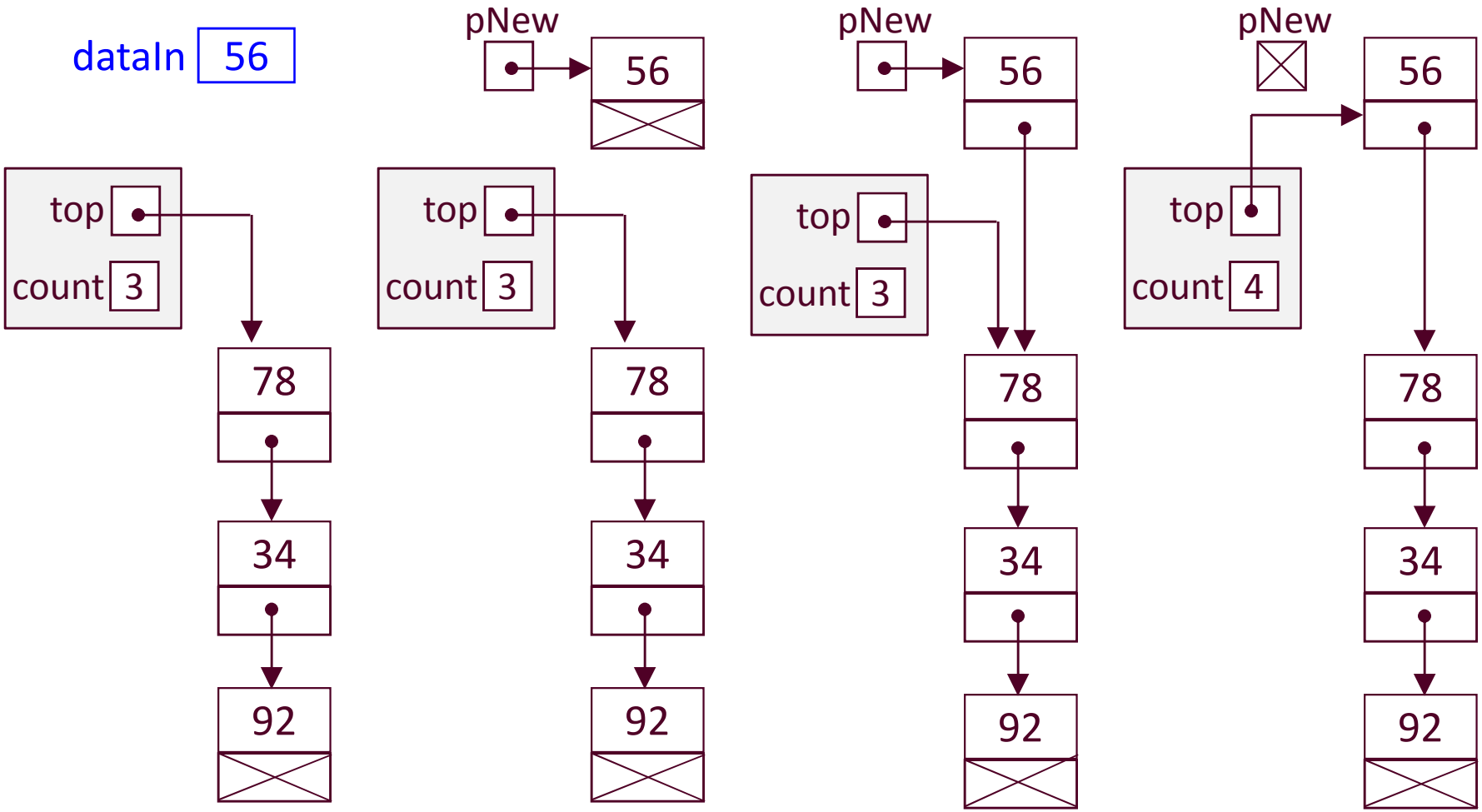
```
bool Stack::push(int dataIn)
{
   // should take care of overflow




}
```

FAIRMONT STATE UNIVERSITY

Example: Push 56



FAIRMONT STATE UNIVERSITY

Member Functions

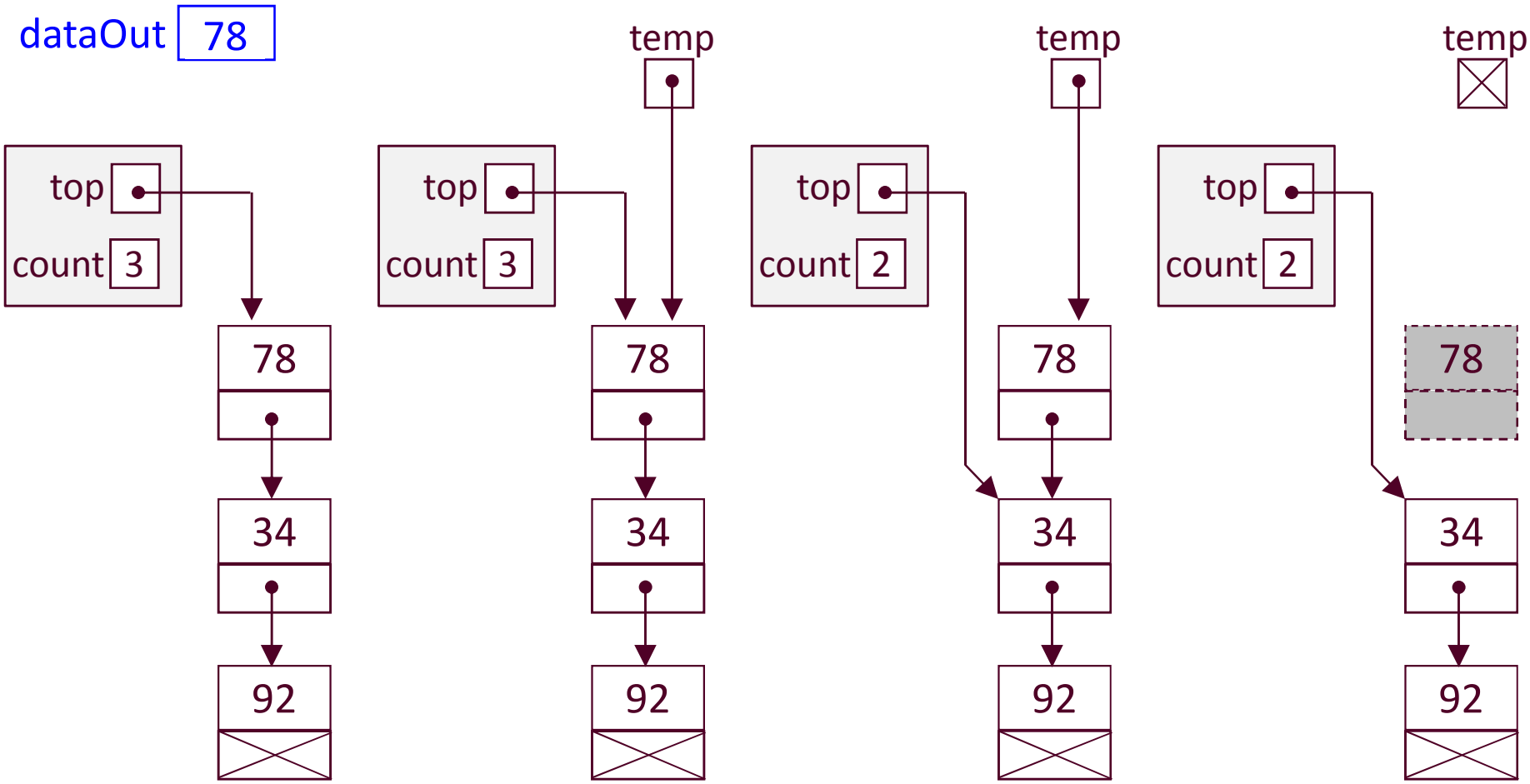
```
bool Stack::pop(int& dataOut)
{
   // should take care of underflow




}
```

FAIRMONT STATE UNIVERSITY

Example: Pop



FAIRMONT STATE UNIVERSITY

Member Functions

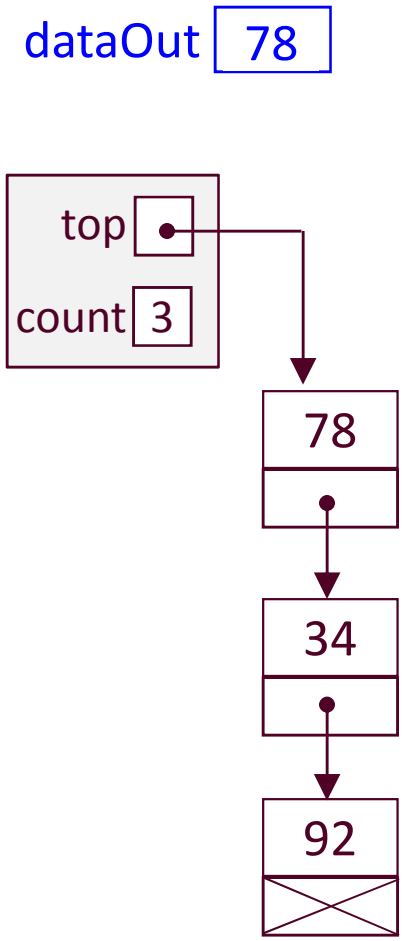
```
bool Stack::stackTop(int& dataOut)
{
   // should take care of underflow




}
```

FAIRMONT STATE UNIVERSITY

Example: StackTop



FAIRMONT STATE UNIVERSITY

Member Functions

```
bool Stack::isEmpty()
{



}

bool Stack::isFull()
{



}
```

FAIRMONT STATE UNIVERSITY

Member Functions

```
int Stack::getCount()
{



}
```

FAIRMONT STATE UNIVERSITY

Array vs. Linked List Implementation

- Overflow
  - can occur with array based implementation
  - does not occur with linked list based implementation (unless the list is really long)
- The variable *top*
  - indicates the index of the top item in the array based implementation
  - indicates the address of the top item in the linked list based implementation

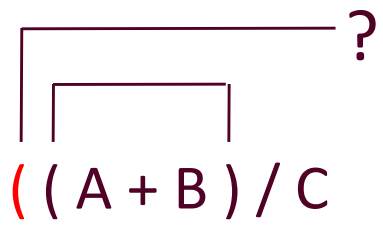
FAIRMONT STATE UNIVERSITY

Array vs. Linked List Implementation

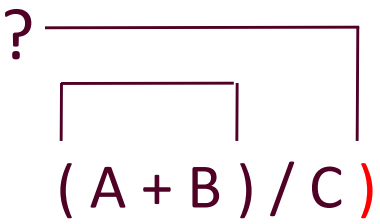
- Complexity of *push*, *pop*, and *stacktop*
  - *O*(1) in both
- Complexity of destructor
  - *O*(1) in array based implementation
  - *O*(*n*) in linked list based implementation
- Complexity of *isEmpty*, *isFull, getCount*
  - *O*(1) in both

FAIRMONT STATE UNIVERSITY

Application of Stacks – Parsing



Opening parenthesis not matched



Closing parenthesis not matched

FAIRMONT STATE UNIVERSITY

Application of Stacks – Postfix Expressions

- *Infix* expression
  - one we are used to, e.g., **3 + 9**
  - each operator is placed *between* its operands
  - requires parenthesis
- *Postfix* expression
  - each operator follows its operands, e.g., **3 9 +**
  - do not require parenthesis
  - used for evaluating expressions by compilers

FAIRMONT STATE UNIVERSITY

Examples

Infix	Postfix
$a + b * c$	$a\ b\ c\ *\ +$
$a * b + c$	$a\ b\ *\ c\ +$
$(a + b) * c$	$a\ b\ +\ c\ *$
$(a - b) * (c + d)$	$a\ b\ -\ c\ d\ +\ *$
$(a \% b) \wedge c$	$a\ b\ \%\ c\ \wedge$

FAIRMONT STATE UNIVERSITY

Example

- Convert the following into postfix:

$$a * b - c / d$$

$$a + c * d - b$$

FAIRMONT STATE UNIVERSITY

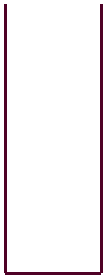
Infix-Postfix Conversion

- Create an empty stack
- Scan the infix expression left-to-right
  - if the next item is an operand, copy it to output expression
  - if the next item is an operator and its precedence is > that of the operator at the top of stack, push it onto the stack
  - if the next item is an operator and its precedence <= that of the operator at the top of stack
    - repeatedly pop the top operators from the stack (as long as their precedence is not lower) and copy them to output expression
    - push the new item (operator) onto the stack
- Continue till there is no more item

FAIRMONT STATE UNIVERSITY

Example

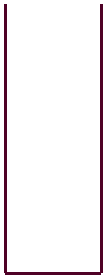
A + B * C – D / E



FAIRMONT STATE UNIVERSITY

Example

+ B * C – D / E

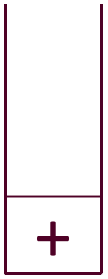


A

FAIRMONT STATE UNIVERSITY

Example

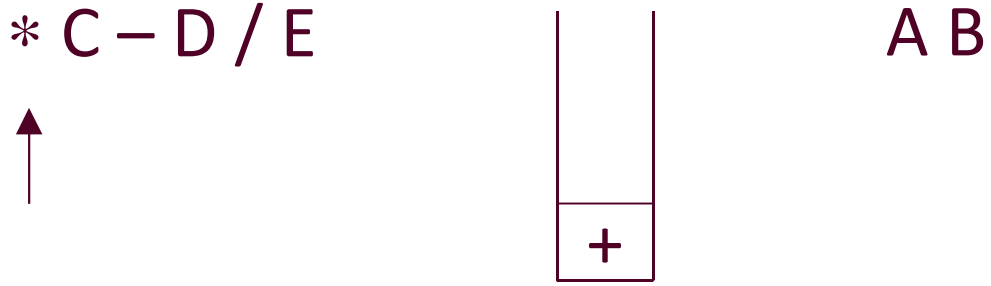
B * C – D / E



A

FAIRMONT STATE UNIVERSITY

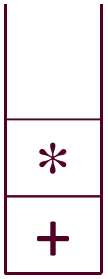
Example



FAIRMONT STATE UNIVERSITY

Example

C – D / E

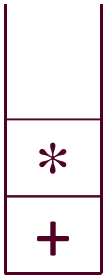


A B

FAIRMONT STATE UNIVERSITY

Example

– D / E

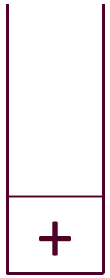


A B C

FAIRMONT STATE UNIVERSITY

Example

– D / E

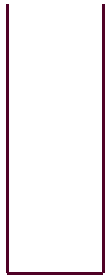


A B C *

FAIRMONT STATE UNIVERSITY

Example

– D / E

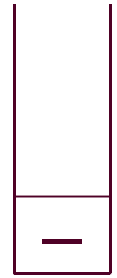


A B C * +

FAIRMONT STATE UNIVERSITY

Example

D / E

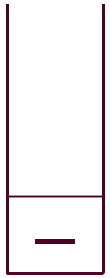


A B C * +

FAIRMONT STATE UNIVERSITY

Example

/ E



A B C * + D

FAIRMONT STATE UNIVERSITY

Example

E

A B C * + D

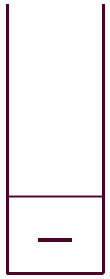
FAIRMONT STATE UNIVERSITY

Example

A B C * + D E

FAIRMONT STATE UNIVERSITY

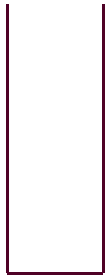
Example



A B C * + D E /

FAIRMONT STATE UNIVERSITY

Example



A B C * + D E / –

FAIRMONT STATE UNIVERSITY

Evaluating Postfix Expressions

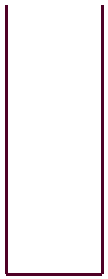
- Create an empty stack
- Scan the postfix expression left-to-right
  - if the next item is an operand, push it onto the stack
  - if the next item is an operator, pop the top two items, perform the operation, and push the result back to the stack
- Continue till there is no more item

FAIRMONT STATE UNIVERSITY

Example

- Evaluate the postfix expression 2 4 + 5 * using a stack.

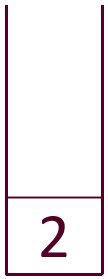
  2 4 + 5 *



FAIRMONT STATE UNIVERSITY

Example

2 4 + 5 * Push 2



FAIRMONT STATE UNIVERSITY

Example

2 4 + 5 * Push 4

FAIRMONT STATE UNIVERSITY

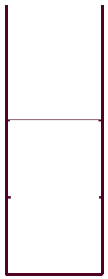
Example

2 4 + 5 *

Pop

popVal1 = 4

popVal2 = 2



FAIRMONT STATE UNIVERSITY

Example

2 4 + 5 *

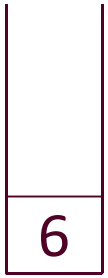
Push 6

popVal1 = 4

popVal2 = 2

pushVal = popVal2 + popVal1

= 2 + 4 = 6



FAIRMONT STATE UNIVERSITY

Example

2 4 + 5 * Push 5

FAIRMONT STATE UNIVERSITY

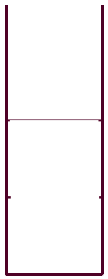
Example

2 4 + 5 *

Pop

popVal1 = 5

popVal2 = 6

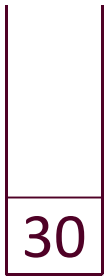


FAIRMONT STATE UNIVERSITY

Example

2 4 + 5 *

Push 30



popVal1 = 5

popVal2 = 6

pushVal = popVal2 * popVal1

= 6 * 5 = 30

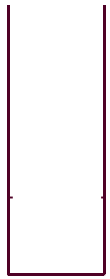
FAIRMONT STATE UNIVERSITY

Example

2 4 + 5 *

Pop

result = 30



FAIRMONT STATE UNIVERSITY

Assignment

- Show how you will evaluate the postfix expression 5 9 3 – 2 / * using a stack.